

INTERNATIONAL STANDARD

**ISO/IEC
15068-2**

**ANSI/IEEE
Std 1387.2**

First edition
1999-03-15

Information technology — Portable Operating System Interface (POSIX®) system administration —

Part 2: Software administration

*Technologies de l'information — Administration du système de l'interface du
système opératoire portable (POSIX®) —*

Partie 2: Administration du logiciel



Reference number
ISO/IEC 15068-2:1999(E)
ANSI/IEEE
Std 1387.2, 1995 edition

IECNORM.COM : Click to view the full PDF of ISO/IEC 15068-2:1999

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

**Copyright © 1999 by the Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street
New York, NY 10017, USA
All rights reserved.**

Abstract: This standard is part of the POSIX[®] series of standards for applications and user interfaces to open systems. It defines a software packaging layout, a set of information maintained about software, and a set of utility programs to manipulate that software and information.

Keywords: data processing, open systems, operating system, packaging, portable application, POSIX[®], software, system administration, utilities

POSIX is a registered trademark of the Institute of Electrical and Electronics Engineers, Inc.

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

Copyright © 1999 by the
Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 1996.
Printed in the United States of America.

ISBN 0-7381-1568-1

No part of this publication may be reproduced in any form,
in an electronic retrieval system or otherwise,
without the prior written permission of the publisher.

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

International Standard ISO/IEC 15068-2:1999(E)

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

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75% of the national bodies casting a vote.

International Standard ISO/IEC 15068-2 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC22, *Programming languages, their environments and system software interfaces*.

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*
- *Part 4: Print administration*

Annexes A to D of 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

IECNORM.COM : Click to view the full PDF of ISO/IEC 15068-2:1999

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

but not yet installed.

Software packaging layout

This part of ISO/IEC 15068 defines the organization of software on a distribution medium, the information held about that software, and the way in which such information is represented. This enables both portability of software distributions across systems of different architecture, and the use of different media to distribute software (including both file system and serial image forms).

Information kept about software

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

Utilities to administer software

This part of ISO/IEC 15068 defines a utility to convert software into the packaging layout, known as a distribution. This part of ISO/IEC 15068 also contains utilities to examine the information in a distribution, copy software from one distribution to another, install software from a distribution, remove software from a distribution, and verify the integrity of a distribution. There are also utilities for configuring installed software, verifying the integrity of installed software, examining and modifying the information held about installed software, and for removing installed software from a system. This provides administrators with a consistent method of dealing with software across all conforming systems.

Distributed software administration

This part of ISO/IEC 15068 defines the concepts, and the utility syntax and behaviors, for managing software in a distributed environment. This includes the concept of different software administration roles (developer, packager, manager, source, target, and client). Different utilities involve different roles, and different roles may be distributed across multiple systems within a single command execution.

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

1) The numbers in curly brackets correspond to those of the references in 1.2. When preceded by a "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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 15068-2:1999

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

(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

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 downgrade:** 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.

615 2.3 Dependencies on Other Standards

616 2.3.1 Features Inherited From POSIX.1

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

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

625 2.3.1.1 File System

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

630 2.3.1.2 Environment Variables

631 The existence of environment variables in general is assumed, as well as **PATH**,
632 **LANG**, **LC_ALL**, **LC_CTYPE**, **LC_MESSAGES**, **LC_TIME**, and **TZ**, in particu-
633 lar.

634 2.3.1.3 Data Interchange Format

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

638 2.3.2 Features Inherited From POSIX.2

639 This subclause describes some of the features provided by POSIX.2 {3} that are
640 assumed to be globally available to all systems conforming to this part of ISO/IEC
641 15068. This subclause does not attempt to detail all of the POSIX.2 {3} features
642 that are required by all the utilities and control scripts defined in this part of
643 ISO/IEC 15068; additional functionality required may be found in the utility
644 descriptions and in 4.1.6.1.

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 15068-2:1999

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 <i>media.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 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*.

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	<bundle directory>
<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.

492 *vendor_tag* A short identifying name of the vendor that supplied the bundle.
493 This attribute shall be used to identify a vendor object containing
494 additional attributes describing the vendor.
495 This attribute is used to distinguish bundles, allowing more than
496 one vendor to produce a bundle with the same value of the *tag*
497 attribute.

498 3.9 Filesets

499 The fileset class is used to define a set of software files. The fileset is the smallest
500 level of software that can be managed by the tasks defined in this standard.

501 The fileset class inherits attributes from the software common class. Filesets con-
502 tain the actual files and control_files that make up the software product. A partic-
503 ular fileset object is identified within a product by the *tag* attribute.

504 NOTE: A fileset is strictly contained within the product. There cannot be more than one fileset in
505 the product with the same tag. A fileset cannot be in more than one product. However, a product
506 may be referenced by more than one bundle.

507 **Table 3-9 – Attributes of the Fileset Class**

Attribute	Length	Permitted Values	Default Value
<i>control_directory</i>	Undefined	Filename character string	< <i>fileset.tag</i> >
<i>corequisites</i>	Undefined	List of dependency_specs	Empty list
<i>exerequisites</i>	Undefined	List of dependency_specs	Empty list
<i>is_kernel</i>	8	One of: true, false	false
<i>is_locatable</i>	8	One of: true, false	true
<i>is_reboot</i>	8	One of: true, false	false
<i>location</i>	Undefined	Pathname character string	< <i>product.directory</i> >
<i>media_sequence_number</i>	Undefined	List of <i>media.sequence_number</i> values	1
<i>prerequisites</i>	Undefined	List of dependency_specs	Empty list
<i>revision</i>	64	Filename character string	None
<i>state</i>	16	One of: configured, installed, corrupt, removed, available, transient	None
<i>control_files</i>	Undefined	List of <i>control_file.tag</i> values	Empty List
<i>files</i>	Undefined	List of <i>file.path</i> values	Empty List

528 **3.9.1 Fileset Attributes**

529 The attributes listed in Table 3-9 and described in the following, along with the
530 attributes listed in Table 3-6, characterize each instance of the fileset class:

531	<i>control_directory</i>	
532		The name of the fileset control directory below which the
533		control_files for the fileset are stored within an exported catalog.
534		See 5.1.
535	<i>control_files</i>	A list of the values of the <i>tag</i> attribute for the control_files in the
536		fileset.
537	<i>corequisites</i>	A list of dependency_specs for software required to be
538		installed and configured for this fileset to work.
539		Dependencies shall be considered when copying, installing,
540		configuring, verifying, and removing software. See 4.3.7.2,
541		4.4.7.2, 4.5.7.2, 4.9.7.2, and 4.10.7.2.
542		The software specified by the dependency_spec shall be com-
543		plete in order for the dependency to be resolved successfully. See
544		<i>all_filesets</i> in 3.7.
545	<i>exrequisites</i>	A list of dependency_specs for software required not to be
546		installed when this fileset is installed.
547		Dependencies shall be considered when installing, configuring,
548		verifying, and removing software. See 4.3.7.2, 4.5.7.2, 4.9.7.2,
549		and 4.10.7.2.
550		No part of the software specified by the dependency_spec may
551		be installed in order for this dependency to be resolved success-
552		fully.
553	<i>files</i>	A list of the values of the <i>path</i> attribute for the files in the fileset.
554	<i>is_kernel</i>	A Boolean value indicating the fileset requires a kernel rebuild.
555	<i>is_locatable</i>	A Boolean value indicating if the fileset may be re-located during
556		installation.
557	<i>is_reboot</i>	A Boolean value indicating the host on which the fileset is
558		configured should be re-booted.
559	<i>location</i>	Specifies the location below which relocatable files are stored.
560		This attribute is only valid for filesets in installed software. It
561		differs from the <i>product.directory</i> attribute only if relocation was
562		specified during installation. See 4.5.7.3.1.
563	<i>media_sequence_number</i>	
564		Identifies the <i>media.sequence_number</i> for the medium on which
565		the files for this fileset is found.
566		If a single fileset spans multiple media, this attribute identifies a
567		list of <i>media.sequence_number</i> values, identifying all of the
		media on which the fileset is found. In that case, the order of the
		list shall be interpreted as the order in which to read the media.
		See 3.2, 3.3, and 5.3.

568 *prerequisites* A list of **dependency_specs** for software required to be
569 installed prior to the installation of this fileset and configured
570 prior to the configuration of this fileset.

571 Dependencies shall be considered when copying, installing,
572 configuring, verifying, and removing software. See 4.3.7.2,
573 4.4.7.2, 4.5.7.2, 4.9.7.2, and 4.10.7.2.

574 The software specified by the **dependency_spec** shall be com-
575 plete in order for the dependency to be resolved successfully. See
576 *all_filesets* in 3.7.

577 Circular definitions should be avoided within package
578 definitions. Behavior when circular definitions are encountered
579 is implementation defined.

580 *revision* Defines the revision of the fileset.

581 It is used for presentation purposes and for resolving software
582 specifications.

583 *state* An indication of the current status of the fileset.

584 This attribute may have one of the following values: **config-**
585 **ured**, **installed**, **corrupt**, **removed**, **available**, and
586 **transient**.

587 **3.10 Subproducts**

588 Subproducts are groupings of filesets and subproducts within a single product.
589 Subproducts do not contain filesets or subproducts within the name space of the
590 subproduct, but instead refer to them. A subproduct can refer to another subpro-
591 duct. A subproduct or fileset can be referenced by more than one subproduct.

592 The subproduct class shall inherit the attributes of the software common class.

593 A particular subproduct object is named, and identified within a product, by the
594 *tag* attribute. The values of the *tag* attribute of all subproducts and filesets shall
595 be unique within a product.

596 Subproduct definitions are copied or installed when any fileset specified in the
597 contents of the subproduct is copied or installed with **swcopy** or **swinstall**
598 respectively. They remain installed until explicitly removed or until all of their
599 contents are removed.

600 **Table 3-10 – Attributes of the Subproduct Class**

Attribute	Length	Permitted Values	Default Value
<i>contents</i>	Undefined	List of <i>tag</i> values	Empty list

3.10.1 Subproduct Attributes

The attributes listed in Table 3-10 and described in the following, along with the attributes listed in Table 3-6, characterize each instance of the subproduct class:

contents A list of *tag* values that defines the list of filesets and subproducts grouped into this subproduct.

3.11 Software_Files

Software_file is the common class that files and control_files inherit from. A *software_file* is a file as defined in POSIX.1 [2].

Table 3-11 – Attributes of the Software_Files Common Class

Attribute	Length	Permitted Values	Default Value
<i>cksum</i>	16	Integer character string	None
<i>compressed_cksum</i>	16	Integer character string	None
<i>compressed_size</i>	16	Integer character string	None
<i>compression_state</i>	16	One of: uncompressed, compressed, not_compressible	uncompressed
<i>compression_type</i>	64	Filename character string	Empty string
<i>revision</i>	64	Portable character string	Empty string
<i>size</i>	16	Integer character string	None
<i>source</i>	Undefined	Pathname character string	None

3.11.1 Software_File Common Attributes

The attributes listed in Table 3-11 and described in the following, characterize each instance of the *software_file* class, and are inherited by each instance of the files and control_files classes:

cksum An integer character string representing a 32-bit cyclic redundancy check (CRC) identical to that returned in the first field of the output of the **cksum** utility, as defined in 4.9 of POSIX.2 [3].

compressed_cksum

Indicates the **cksum** CRC of the compressed software file in the same manner as the *cksum* attribute.

This attribute may be used to verify the integrity of a compressed file, and to help determine if a file to be copied is already present at the target.

compressed_size

Indicates the size of the compressed software file in the same manner as the *size* attribute.

645		This attribute can be used for computation of disk space analysis
646		when the file will remain compressed after a copy.
647	<i>compression_state</i>	
648		Indicates which one of the following conditions is true:
649		— Uncompressed but permitted to be compressed in a distribu-
650		tion (if this attribute has the value uncompressed or if no
651		value is supplied for the attribute)
652		— Already compressed (if this attribute has the value
653		compressed)
654		— Uncompressed and not permitted to be compressed in a dis-
655		tribution (if this attribute has the value
656		not_compressible)
657	<i>compression_type</i>	
658		Specifies the compression method used to compress the file if the
659		value of the <i>compression_state</i> attribute is compressed .
660		The values supported for <i>compression_type</i> are implementation
661		defined. The way in which an implementation uses this value to
662		implement or execute the compression or uncompression of a file
663		is undefined.
664	<i>revision</i>	Describes a string indicating the revision level of the file.
665	<i>size</i>	Indicates the size of the software file in bytes.
666		This attribute has the same values and meaning as <i>st_size</i> 5.6.1
667		of POSIX.1 {2}.
668	<i>source</i>	When used in a PSF, this attribute specifies the pathname of the
669		file or control_file to be placed in the distribution by the swpack-
670		age utility.

671 3.12 Files

672 Files are the actual files and directories that make up the fileset. Many of the file
673 attributes (such as *owner*, *group*, and *mode*) are derived from, and dependent
674 upon, a POSIX.1 {2} file system.

675 The file class inherits attributes from the *software_file* common class.

676 A particular file object is identified within a fileset by the *path* attribute. When a
677 file is located on a distribution, the *path* attribute indicates the intended installa-
678 tion location of the file. The value of the *path* attribute is also the path below the
679 storage directory for that fileset within file storage structure of the distribution
680 (see 5.1.2). While a file is installed (in an *installed_software* object), the *path* attri-
681 bute indicates the actual location of the file. This path is relative to the root direc-
682 tory for that *installed_software* object.

683 For regular files, the value of the *size* attribute is the actual file size in bytes. For
symbolic links, this is the string length of the *link_source* attribute. For hard
links, directories, and block and character special files, this is always zero. These
types are set to zero since the actual space required by these types depends on the

684 file system. An implementation should consider the impact of these types as part
685 of disk space analysis.

686 The *cksum* attribute only has meaning for a file with type of regular file.

687 **Table 3-12 – Attributes of the File Class**

Attribute	Length	Permitted Values	Default Value
<i>path</i>	Undefined	Pathname character string	None
<i>gid</i>	16	Integer character string	Undefined
<i>group</i>	Undefined	Filename character string	Empty string
<i>is_volatile</i>	8	One of: true, false	false
<i>link_source</i>	Undefined	Pathname character string	None
<i>major</i>	16	Portable character string	None
<i>minor</i>	16	Portable character string	None
<i>mode</i>	16	Octal character string	None
<i>mtime</i>	16	Integer character string	None
<i>owner</i>	Undefined	Filename character string	Empty string
<i>type</i>	8	One of: f, d, h, s, p, b, c	f
<i>uid</i>	16	Integer character string	Undefined

703 3.12.1 File Attributes

704 The attributes listed in Table 3-12 and described in the following, along with the
705 attributes listed in Table 3-11, characterize each instance of the file class:

- 706 *gid* The numeric group id of the file.
707 This attribute has the same values and meaning as *st_gid*, 5.6.1
708 of POSIX.1 {2}.
- 709 *group* The group name of the file.
710 This attribute has the same values and meaning as *gr_name*,
711 9.2.1 of POSIX.1 {2}.
- 712 *is_volatile* Indicates a file whose contents can change, or that can be
713 removed after it has been installed.
- 714 *link_source* The pathname of the target of the link.
715 This attribute only has meaning if the file type is a hard or sym-
716 bolic link.
- 717 *major* This attribute only has meaning if the file type is character or
718 block special file.
719 This attribute has the same values and meaning as the *devmajor*
720 field in the **tar** archive specified in 10.1.1 of POSIX.1 {2}.
- 721 *minor* This attribute only has meaning if the file type is character or
722 block special file.
This attribute has the same values and meaning as the *devminor*
field in the **tar** archive specified in 10.1.1 of POSIX.1 {2}.

723	<i>mode</i>	An octal representation of the permissions bits of the file.
724		This attribute has the same values and meaning as <i>st_mode</i> ,
725		5.6.1 of POSIX.1 {2}, except that this attribute has no meaning if
726		the file type is a hard or symbolic link.
727	<i>mtime</i>	The time of the last data modification of the file.
728		This attribute has the same values and meaning as <i>st_mtime</i> ,
729		5.6.1 of POSIX.1 {2}.
730	<i>owner</i>	The name of the owner of the file.
731		This attribute has the same values and meaning as <i>pw_name</i> ,
732		9.2.2 of POSIX.1 {2}.
733	<i>path</i>	The pathname of the file.
734	<i>type</i>	Supported file types are those described in 5.6.1.1 of POSIX.1 {2},
735		plus hard link and symbolic link.
736		The permitted values of this attribute are the following: f (regu-
737		lar file), d (directory), h (hard link), s (symbolic link), p [named
738		pipe (FIFO)], b (block special device), and c (character special
739		device).
740	<i>uid</i>	The numeric user id of the file.
741		This attribute has the same values and meaning as <i>st_uid</i> , 5.6.1,
742		of POSIX.1 {2}.

743 3.13 Control_Files

744 Control_files can be scripts, data files, or **INFO** files. The product and fileset **INFO**
745 files in the software packaging layout are included as control_files. Control scripts
746 are the vendor-supplied scripts executed at various steps by the software adminis-
747 tration utilities.

748 The control_file class inherits attributes from the software_file common class.

749 A particular control_file object is identified within a product or fileset by the *tag*
750 attribute. The *path* attribute is the storage location of the file relative to the con-
751 trol directory. For distributions, the control directory is the directory in the
752 software packaging layout where the control_files are stored. For
753 installed_software objects, this control directory location is undefined.

754 3.13.1 Control_File Attributes

755 The attributes listed in Table 3-13 and described in the following, along with the
756 attributes listed in Table 3-11, characterize each instance of the control_file class:

757	<i>interpreter</i>	The name of the interpreter used to execute those control_files
758		that are executed as part of the utilities defined in this part of
759		ISO/IEC 15068.

Table 3-13 – Attributes of the Control File Class

Attribute	Length	Permitted Values	Default Value
<i>tag</i>	64	Filename character string	None
<i>interpreter</i>	Undefined	Filename character string	sh
<i>path</i>	Undefined	Filename character string	None
<i>result</i>	16	One of: none , success , failure , warning	none

Within a distribution, a value for this attribute other than **sh** implies that the distribution is neither a Strictly Conforming POSIX.7.2 Distribution nor a Conforming POSIX.7.2 Distribution. Such a distribution may be a Conforming POSIX.7.2 Distribution with Extensions. See 1.3.2.

path The filename of the control_file.

Multiple control_file entries can have the same value of the *path* attribute. This implies that the same script is executed in different steps within the execution of a utility.

result Contains the result of the execution of the control script.

This attribute is only valid for control_files in installed_software. A complete list of legal results is contained in Table 3-13.

tag The identifier of the control_file.

All control files are loaded and maintained within the distribution and installed software catalogs by the utilities defined in this part of ISO/IEC 15068. These utilities execute control scripts with particular tags at various steps in the execution of the utility. The values for the *control_file.tag* attribute for which this part of ISO/IEC 15068 defines behavior are as follows: **request**, **response**, **checkinstall**, **preinstall**, **postinstall**, **unpreinstall**, **unpostinstall**, **verify**, **fix**, **checkremove**, **preremove**, **postremove**, **configure**, **unconfigure**, and **space**.

IECNORM.COM : Click to view the full PDF of ISO/IEC 15068-2:1999

Section 4: Software Administration Utilities

The Software Administration Utilities are the utilities that shall be implemented in all systems conforming to this part of ISO/IEC 15068.

4.1 Common Definitions for Utilities

This part of ISO/IEC 15068 defines the following utilities: **swask**, **swcopy**, **swconfig**, **swinstall**, **swlist**, **swmodify**, **swpackage**, **swremove**, and **swverify**. These utilities share many common definitions and behaviors. This section describes those common definitions and behaviors. These utilities conform to the utility syntax guidelines in 2.10.2 of POSIX.2 {3}.

4.1.1 Synopsis

The following is the general synopsis format for the utilities:

```
<sw_utility> [ -d ||| -r ] [-p] [-u] [-a attribute] [-c catalog]
               [-s source] [-f file] [-t targetfile] [-x option=value]
               [-X options_files] [software_selections] [@ targets]
```

4.1.2 Description

The utilities all operate on *software_selections* in source or target software_collections or both.

4.1.3 Options

Each of the utilities in this standard does not support all of the options shown as follows. Each utility shall support the options indicated in its synopsis subclause and those indicated after the description of the options in this subclause. All options can be repeated. Except where otherwise stated within this part of ISO/IEC 15068, the behavior for repeated options is undefined. In addition to those shown below, the **-W** (capital-W) option shall be reserved for implementation extensions. See 2.10.2 of POSIX.2 {3}.

-a attribute

Used to specify the attributes on which the utility shall operate.

This option can be used multiple times to specify a set of attributes.

Applies to **swlist**, and **swmodify**.

- 29 **-c** *catalog*
30 Used to specify a file with the software definition file syntax or direc-
31 tory with the exported catalog structure.
- 32 This is where software catalog information (metadata) is to be stored
33 to or retrieved from. If this information fits into one file, then the
34 catalog can be a file, otherwise it shall be a directory. See section 5
35 and 5.2.
- 36 Applies to **swask**, **swconfig**, **swinstall**, **swlist**, and **swmo-**
37 **dify**.
- 38 **-d** Indicates to the utility that the operation is on a distribution instead
39 of installed_software.
- 40 Applies to **swlist**, **swmodify**, **swremove**, and **swverify**.
- 41 **-f** *file* Reads the list of *software_selections* from *file*.
- 42 If this option is specified multiple times, all the software specified by
43 each file shall be included in the operation. All of the software
44 specified by using this option, as well as all the software specified
45 directly as arguments to the utility, shall be included in the opera-
46 tion.
- 47 The file shall contain one software selection per line where a software
48 selection uses the syntax for **software_spec** defined in 4.1.4.1.
49 Blank lines shall be ignored. Within the file, the # (pound) character
50 shall act as a comment character. On any line containing a # (pound)
51 character, all characters that follow the # (pound) character up to,
52 but excluding, the next **<newline>**, shall be ignored.
- 53 Applies to all utilities.
- 54 **-p** Previews the operation without making any permanent modifications
55 to the target.
- 56 An implementation should run any control scripts that are executed
57 as part of the selection or analysis phase of the command being pre-
58 viewed, but shall not run any that are executed in the execution
59 phase.
- 60 This option can be used with any or all of the other options to under-
61 stand the impact of an operation before performing it.
- 62 Applies to **swconfig**, **swcopy**, **swinstall**, **swmodify**, **swpack-**
63 **age**, and **swremove**.
- 64 **-r** Indicates to the utility that the operation is on an installed_software
65 object located at an alternate root, instead of either a distribution or
66 the installed_software object located at **/**.
- 67 Applies to **swinstall**, **swlist**, **swmodify**, **swremove**, and
68 **swverify**.

- 69 **-S** *source*
70 Specifies the software source for the operation.
- 71 For **swinstall**, **swask**, and **swcopy** a source can be specified using
72 the syntax in 4.1.4.2. For **swpackage**, the source shall be a product
73 specification file.
- 74 Applies to **swask**, **swcopy**, **swinstall**, and **swpackage**.
- 75 **-t** *targetfile*
76 Reads the list of targets from *targetfile*.
- 77 If this option is specified multiple times, all the targets specified by
78 each file shall be included in the operation. All of the targets
79 specified by using this option, as well as all the targets specified
80 directly as arguments to the utility, shall be included in the operation.
81
- 82 The file shall contain one target per line, where a target uses the syntax
83 for **software_collection_spec** defined in 4.1.4.2. Blank
84 lines shall be ignored. Within the file, the # (pound) character shall
85 act as a comment character. On any line containing a # (pound)
86 character, all characters that follow the # (pound) character up to,
87 but excluding, the next <newline>, shall be ignored.
- 88 Applies to **swconfig**, **swcopy**, **swinstall**, **swlist**, **swmodify**,
89 **swpackage**, **swremove**, and **swverify**.
- 90 **-u** This is the option used to specify undo or delete behavior to a utility.
- 91 Applies to **swconfig** and **swmodify**.
- 92 **-X** *option=value*
93 Used to override the value of an extended option in the defaults file.
- 94 The extended options supported are described in 4.1.5.2. This option
95 can be specified multiple times. If any extended option is defined
96 more than once, the precedence rules from 4.1.5.3.1 shall be used.
- 97 Applies to all utilities.
- 98 **-X** *options_file*
99 Used to override the defaults specified in the system defaults file.
- 100 The options supported are described in 4.1.5.2. This option can be
101 specified multiple times. If any extended option from any file is
102 defined more than once, the precedence rules from 4.1.5.3.1 shall be
103 used.
- 104 The file shall have the format defined in 4.1.5.3.
- 105 Applies to all utilities.

106 **4.1.3.1 Non-interactive Operation**

107 All utilities except **swask** are by default non-interactive. The **swinstall** and
108 **swconfig** utilities also define interactive modes for executing **request** scripts
independent of the **swask** utility.

109 The way in which **swinstall**, **swcopy**, and **swpackage** utilities handle multi-
110 ple volumes for sources or targets is implementation defined.

111 4.1.4 Operands

112 There are two types of operands that may be specified on the command line —
113 *software_selections* and *targets*. The *software_selections* refer to the software
114 objects (bundles, products, subproducts and filesets) on which to be operated. The
115 targets refer to the target software_collections where the software selections are
116 applied. These two operand types shall be separated by the @ operand. With the
117 exception of **swpackage**, the behavior of all utilities defined in this part of
118 ISO/IEC 15068 is undefined if no *software_selections* are provided.

119 4.1.4.1 Software Specification and Logic

120 The following specifies the syntax for software selections in utilities
121 (**software_spec**) and in dependency specifications (**dependency_spec**). This
122 syntax shall be applied by the utilities to search a software_collection catalog for
123 software. See 2.1.2 for the grammar conventions for this syntax. Note that the
124 tokens shown below are defined in 2.2.2.68, 2.2.2.37, and 2.2.2.92.

125 NOTE: For examples of the use of specifications in this section, see Annex C.

```

126 %token      FILENAME_CHARACTER_STRING      /* as defined in 2.2.2.37 */
127 %token      NEWLINE_STRING                  /* as defined in 2.2.2.61 */
128 %token      PORTABLE_CHARACTER_STRING       /* as defined in 2.2.2.68 */
129 %token      SOFTWARE_PATTERN_MATCH_STRING   /* as defined in 2.2.2.92 */
130 %token      WHITE_SPACE_STRING              /* as defined in 2.2.2.110 */

131 %start  software_selections
132 %%

133 software_selections : software_selections ws software_spec
134                     | software_spec
135                     ;

136 software_spec      : bundle_software_spec
137                     | product_software_spec
138                     ;

139 bundle_software_spec : bundle_qualifier version
140                     | bundle_qualifier '.' product_qualifier version
141                     ;

142 bundle_qualifier    : bundle_qualifier '.' bundle_tag
143                     | bundle_tag
144                     ;

145 product_software_spec : product_qualifier version
146                     ;

147 product_qualifier   : product_tag subproduct_qualifier fileset_qualifier
148                     ;

149 subproduct_qualifier : /* empty */
150                     | subproduct_qualifier '.' subproduct_tag

```

```

149         | '.' subproduct_tag
150         ;

151     fileset_qualifier      : /* empty */
152         | '.' fileset_tag
153         ;

154     bundle_tag            : sw_pattern
155         ;

156     product_tag           : sw_pattern
157         ;

158     fileset_tag           : sw_pattern
159         ;

160     subproduct_tag        : sw_pattern
161         ;

162     version               : /* empty */
163         | ','
164         | version_qualifier
165         ;

166     version_qualifier     : version_qualifier ver_item
167         | ver_item
168         ;

169     ver_item              : ',' ver_id '='
170         | ',' ver_id '=' sw_pattern
171         | ',' 'r' rel_op dotted_string
172         ;

173     sw_pattern            : SOFTWARE_PATTERN_MATCH_STRING
174         ;

175     ver_id                : 'r' | 'a' | 'v' | 'l' | 'q'
176         ;

177     rel_op                : '==' | '!=' | '>=' | '<=' | '<' | '>'
178         ;

179     dotted_string         : FILENAME_CHARACTER_STRING
180         ;

181     ws                    : WHITE_SPACE_STRING
182         ;

183     %start dependency_spec
184     %%

185     dependency_spec       : dependency_spec '|' software_spec
186         | software_spec
187         ;

```

188 If the **software_spec** identifies a bundle, product or subproduct software object, then all filesets contained within that object are included as part of that specification. For software selections, this means that all of these filesets are

189 included. For dependency specifications, this means that all of these filesets are
190 needed in order to meet the dependency.

191 If a **software_spec** identifies a set of filesets that is less than the entire set of
192 filesets within a bundle or product, the **software_spec** identifies a partial bundle or product.
193

194 Only the specified strings shall be used to generate a **software_spec**. Blanks
195 shall not appear between items. The **sw_pattern** and **dotted_string** shall be
196 enclosed in quotes if they contain blanks or commas. The **bundle_tag**,
197 **product_tag**, **subproduct_tag**, and **fileset_tag** shall consist of one or
198 more characters from the filename character set, with the exception that the following three characters **.**, **:** (period, comma, and colon) shall not be used.
199

200 Searching a software_collection catalog for software using a **software_spec**
201 yields a list of zero or more software objects that match the **software_spec**.
202 The rules to be used in the search shall be the following:

- 203 (1) The **software_spec** shall be compared against software in the software
204 collection. The leftmost **sw_pattern** of the **software_spec** is
205 matched against the *tag* attribute of all bundles and products in the
206 software collection. All objects that match are initially included for consideration. If the **sw_pattern** does not match any bundle or product, no objects are included.
207
208

209 The version specified in the **software_spec** shall be compared against
210 the *revision*, *architecture*, *vendor_tag*, *location*, and *qualifier* attributes of
211 the objects matching the leftmost **sw_pattern**. If any **ver_id** in the
212 **software_spec** does not match its corresponding attribute, that object
213 is removed from consideration. If the same **ver_id** is given more than
214 once, all the comparisons specified are performed and all shall succeed to
215 be considered a match.

216 **Table 4-1— Software_spec Version Identifiers**

ver_id	Attribute
r	revision
a	architecture
v	vendor_tag
l	location
q	qualifier

224 An implementation may define additional **ver_id** items along with the
225 attributes and objects to which they apply.

226 For each object still included for consideration, each successive
227 **sw_pattern**, left to right, is applied to the bundles, products, subproducts and filesets within that object. The same **sw_pattern** may match
228 multiple bundle, product, subproduct, and fileset objects. If any
229 **sw_pattern** does not match any objects within the current object, the
230 current object is removed from consideration. If a fileset matches a
231 **sw_pattern** but there is still an unmatched **sw_pattern** in the
232 **software_spec**, that fileset is not selected.
233

When there are no more **sw_patterns** left in the **software_spec**, all the objects identified by the rightmost **sw_pattern** of the

- 234 **software_spec** are included in the list of software that match the
235 **software_spec**.
- 236 (2) The comparison performed when the operator is = shall be a software
237 pattern match as described in 2.2.2.92. If the **ver_id** is specified and
238 the value is an empty string, then the comparison is successful only if the
239 corresponding attribute is not specified. See 4.1.4.1.1.
- 240 (3) When **rel_op** is used, the comparison shall be performed on the
241 specified attribute by dividing it into segments separated by the . (period)
242 character.
- 243 NOTE: As defined in the syntax for **ver_item**, this comparison is required for *revision*
244 only and any other comparison is undefined.
- 245 If there is no period in an attribute, it contains one segment. The seg-
246 ments shall be compared with the corresponding segments of the
247 **dotted_string**. If all characters in both segments to be compared are
248 decimal digit characters (0-9), the comparison shall be based on the
249 decimal numeric value of the segments, starting with the leftmost seg-
250 ment.
- 251 NOTE: Leading zeros are acceptable in such segments.
- 252 If either segment includes any character other than a decimal digit char-
253 acter, a string comparison shall be made to determine the relation.
254 String comparisons shall be made using, as a collation sequence, the
255 order of characters in IRV {1}. If one operand has fewer segments than
256 the other, the unmatched segments shall be compared against the value
257 0 (zero).
- 258 (4) When applied to software in installed software collections, use of either
259 the **l** (location) or **q** (qualifier) **ver_id** shall cause comparison with the
260 value of the *location* or *qualifier* attribute respectively for each product or
261 bundle in the **installed_software** object.
- 262 For distributions, use of either the **l** (location) or **q** (qualifier) **ver_id**
263 shall be ignored for the purpose of comparisons. Although not used for
264 comparisons, the location and qualifier **ver_ids** are used by the **swin-**
265 **stall** utility as the *location* attribute for installing the software, and the
266 *qualifier* attribute for the software, respectively.
- 267 (5) When software selections are applied to a source or target, and a
268 **software_spec** resolves to more than one software object, then the
269 **software_spec** shall be considered ambiguous. An ambiguous selec-
270 tion may be elective or incidental. An elective ambiguous selection occurs
271 when a **sw_pattern** in a **software_spec** contains a wildcard charac-
272 ter or when the version contains a **rel_op**, or when the **sw_pattern** is
273 missing. In all other cases the selection is an incidental ambiguous selec-
274 tion. An incidental ambiguous selection is only valid for **swlist**, and for
275 other utilities generates an event.
276 (SW_ERROR: SW_SELECTION_NOT_FOUND_AMBIG)
- 277 (6) If the **software_spec** begins with a *bundle.tag* definition, then that
278 bundle definition is copied or installed with **swcopy** or **swinstall**.
Thus, a **software_spec** that matches one or more bundles can be used
with all other utilities, but only if they were explicitly installed or copied.

279 However, all subproduct definitions are copied or installed independently
280 of whether they were explicitly selected. Thus, a **software_spec** that
281 matches subproducts can always be used on existing products.

282 For both bundles and subproducts, if some part of their contents exist, then the
283 selection can be found; therefore, any operation will succeed. If none of their con-
284 tents exist, then the selection cannot be found; therefore, the operation will fail
285 (for those selections not found) as defined in the individual *Extended Description*
286 subclause of each utility.

287 **4.1.4.1.1 Fully Qualified Software_spec**

288 A fully qualified **software_spec** is one in which no fields contain a shell pattern
289 match string and all version-distinguishing attributes are specified as
290 "**ver_id**=<value>" (if a value is supplied) or as "**ver_id**=" (if no value is sup-
291 plied or if the value supplied is an empty string). Note that a fully qualified
292 **software_spec** always identifies a software object unambiguously.

293 When a **software_spec** is generated by **swlist**, only the following *tags* are
294 included: the product *tag* for products, the bundle *tag* for bundles, the product and
295 fileset *tag* for filesets, and the product and subproduct *tag* for subproducts.

296 **4.1.4.1.2 Software Compatibility**

297 Products contain attributes (*os_name*, *os_version*, *os_release*, and *machine_type*)
298 related to the *uname()* function defined by 4.4.1 of POSIX.1 {2}. These attributes
299 shall be used by the **swinstall**, **swconfig**, and **swverify** utilities to deter-
300 mine if software is compatible with a target host. A product is considered compa-
301 tible with a target host if each of the *uname* attributes of the product contains a
302 pattern in its definition that matches the corresponding values returned by the
303 *uname()* function on the target host. If any of these attributes is undefined, it is
304 considered to match any target host. The compatibility test applies to all com-
305 ponents of a product, including subproducts and filesets.

306 Bundles, like products, possess *uname* attributes. The values of the bundle
307 *uname* attributes determine the compatibility of the bundle in conjunction with
308 the corresponding attributes of products within the bundle. A product specified as
309 part of a bundle shall be considered compatible if both the product and bundle
310 *uname* attributes designate that the software is compatible. As with products, if
311 any of these attributes is undefined, it is considered to match any target host.

312 **4.1.4.2 Source and Target Specification and Logic**

313 Source and target *software_collections* are specified using the following syntax.
314 See 2.1.2 for the grammar conventions for this syntax.

```
315       %token               HOST_CHARACTER_STRING       /* as defined in 2.2.2.43 */
316       %token               PATHNAME_CHARACTER_STRING /* as defined in 2.2.2.64 */
317       %start    target
318       %%
```

```
319       target               : software_collection_spec
320                            ;
```

```
software_collection_spec : HOST_CHARACTER_STRING ':' PATHNAME_CHARACTER_STRING
```



```

321         | HOST_CHARACTER_STRING ':'
322         | HOST_CHARACTER_STRING
323         | PATHNAME_CHARACTER_STRING
324         ;

325     %start    source
326     %%

327     source      : software_collection_spec
328                 ;

```

329 The **:** (colon) is required if both the host and pathname are specified, or if the host
 330 portion starts with a **/** (slash). The pathname portion shall be an absolute path.
 331 The colon is not allowed by itself.

332 The **HOST_CHARACTER_STRING** portion refers to the implementation-defined
 333 identifier for a host. If it is not specified, then the local host is assumed.

334 The **PATHNAME_CHARACTER_STRING** portion refers to the `software_collection`
 335 `path` attribute (the location on the host of the distribution or `installed_software`
 336 object).

337 When the **PATHNAME_CHARACTER_STRING** is not specified for `installed_software`,
 338 the directory **/** is used. A **PATHNAME_CHARACTER_STRING** other than **/** for an
 339 `installed_software` object is referred to as an alternate root directory. When the
 340 **PATHNAME_CHARACTER_STRING** is not specified for source distributions, the
 341 value of the `distribution_source_directory` default option is used. When the
 342 **PATHNAME_CHARACTER_STRING** is not specified for target distributions, the
 343 value of the `distribution_target_directory` default option is used.

344 For `installed_software` objects, the value of the `installed_software_catalog` option
 345 is used to further clarify which installed software object is actually being targeted.
 346 Multiple `installed_software` objects may share the same path attribute, but they
 347 have separate catalog information because they are distinct objects. The
 348 `installed_software path` attribute, prefixed to the value of the
 349 `installed_software_catalog` option, forms the key for the object into the catalog
 350 information. Use of the `installed_software_catalog` is independent of the **-c**
 351 option.

352 An implementation shall support source and target distributions in the directory
 353 format described in 5.3 for all utilities. An implementation shall support a source
 354 distribution in the serial format for **swask**, **swinstall**, and **swcopy** utilities.
 355 An implementation shall support a target distribution in the serial format for
 356 **swlist**, **swcopy**, and **swpackage**. Whether data on an existing target distribu-
 357 tion in serial format is overwritten or merged is implementation defined. An
 358 implementation need not support a target distribution in the serial format for
 359 **swverify**, **swremove**, and **swmodify**. Unless otherwise stated, support for
 360 serial distributions shall include support for both extended **tar** and extended
 361 **cpio** archives. See 5.3. The format of these archives is defined in Section 10. of
 362 POSIX.1 {2}.

363 4.1.5 External Influences

364 4.1.5.1 Defaults and Options Files

365 The defaults file allows setting of system wide defaults for extended options that
366 define information (location of files and other objects), behavior, and policy control
367 items for the utilities defined in this part of ISO/IEC 15068. The location of the
368 defaults file is implementation defined. An implementation may define separate
369 defaults files for each task. These options also may be specified for each user in
370 the manager role in the file **\$HOME/.swdefaults**.

371 4.1.5.2 Extended Options

372 The utilities in this part of ISO/IEC 15068 support the following extended options
373 as noted. If a default value is defined, it is listed after the = (equal sign).

374 *allow_downdate=false*

375 Controls the ability to replace a fileset with one of a lower revision.

376 If *allow_downdate=false*, do not allow installation of a lower revision
377 of a fileset that is already installed at a higher revision in this loca-
378 tion.

379 If *allow_downdate=true*, allow installation of a lower revision of a
380 fileset.

381 Applies to **swinstall**.

382 *allow_incompatible=false*

383 Controls the ability to install software that is not compatible with the
384 underlying operating system, as defined in 4.1.4.1.2).

385 If *allow_incompatible=false*, do not allow incompatible software to be
386 operated on if the installed_software path is **/**.

387 If *allow_incompatible=true*, then attempt the operation.

388 Applies to **swinstall**, **swconfig**, and **swverify**.

389 *allow_multiple_versions=false*

390 Controls the ability to configure multiple versions of a product.

391 If *allow_multiple_versions=false*, do not attempt to configure a
392 second version of a fileset if one is already configured. If
393 *allow_multiple_versions=true*, then attempt the operation.

394 Applies to **swconfig**.

395 *ask=false*

396 Controls the ability to execute **request** scripts for selected software.

397 If *ask=false*, the utilities shall not run any **request** scripts for
398 selected software. The behavior of **swask** for *ask=false*, is
399 undefined.

400 If *ask=true*, the utilities shall execute all **request** scripts for
selected software after resolving selections, but before initiating
analysis on the targets. This is the default value for **swask**.

401 If *ask=as_needed*, the utilities shall execute any **request** scripts for
402 selected software that does not already have a **response** file in the
403 control directory where the script would be executed. The location of
404 this control directory depends on whether the **-c** option has been set.

405 Applies to **swask**, **swconfig**, and **swinstall**.

406 *autoreboot=false*

407 Controls automatic rebooting of the target host. If *autoreboot=false*,
408 do not automatically reboot the target host, even if a fileset installed
409 requires a reboot to take effect.

410 If *autoreboot=true*, automatically reboot the target host if a fileset
411 requiring a reboot is installed.

412 Applies to **swinstall**.

413 *autorecover=false*

414 Controls automatic recovery if an error occurs during install, as
415 specified in 4.5.7.3.8.

416 If *autorecover=false* and an install error occurs, no error recovery
417 shall be provided at all, not even as an extension to this part of
418 ISO/IEC 15068. Consequently, no attempt shall be made to restore
419 the original state of the system prior to install. The value of the
420 fileset *state* attribute shall be set to **corrupt**.

421 If *autorecover=true* and an error occurs, implementations shall, for
422 any fileset having an install error, execute the **unpostinstall**
423 script (if the **postinstall** script had been run) and the **unprein-**
424 **stall** script, restore the files within the fileset from a copy saved
425 prior to the failed install, and the restore the value of the *state* attri-
426 bute. After recovery of the applicable filesets, installation can con-
427 tinue with the rest of the filesets in that product and the rest of the
428 products in the software selections.

429 NOTE: Since failure prior to executing the **preinstall** script should have no side
430 effects, a failure implies that the **unpreinstall** script requires execution.

431 Applies to **swinstall**.

432 *autoselect_dependencies=as_needed*

433 Controls automatic dependency selection.

434 If *autoselect_dependencies=true*, (the default for all utilities except
435 **swinstall** and **swcopy**), prerequisite and corequisite dependencies
436 shall be autoselected if possible during the selection phase.
437 Autoselection of a dependency is done using the software selection
438 logic found in 4.1.4.1. These dependencies are then operated on as if
439 they were selected explicitly.

440 If *autoselect_dependencies=as_needed*, (the default for **swinstall**
441 and **swcopy**), then autoselected dependencies shall only be operated
442 upon if the dependency is not already met on the target. This value
443 only applies to **swcopy** and **swinstall**.

If *autoselect_dependencies=false*, then no dependencies shall be
autoselected for operation. For install and copy, if the dependencies

444 are not already met on the target, an error shall occur when
445 *enforce_dependencies = true*.

446 Applies to **swask**, **swconfig**, **swcopy**, **swinstall**, and **swver-**
447 **ify**.

448 *autoselect_dependents = false*
449 Controls automatic dependency selection.

450 If *autoselect_dependents = true*, dependent software (software that
451 depends on this software) shall be autoselected if possible during the
452 selection phase. This dependent software shall be operated upon
453 unless the dependency can be met by other software on the target. If
454 dependent software exists that cannot still meet its dependencies
455 through other unselected software, then an error shall occur.

456 If *autoselect_dependents = false*, no dependent software shall be
457 autoselected.

458 Applies to **swconfig**, and **swremove**.

459 *check_contents = true*
460 Controls verification of file contents.

461 If *check_contents = true*, then **swverify** shall check the *mtime*, *size*,
462 and *cksum* attributes of files.

463 If *check_contents = false*, then **swverify** shall not check the attri-
464 butes.

465 This applies to both distribution and installed_software files.

466 Applies to **swverify**.

467 *check_permissions = true*
468 Controls verification of file permissions.

469 If *check_permissions = true*, then **swverify** shall check the *owner*,
470 *uid*, *group*, *gid*, *mode* attributes of files, and the *major* and *minor*
471 attributes of device files.

472 If *check_permissions = false*, then **swverify** shall not check the
473 attributes.

474 This only applies to installed_software files.

475 Applies to **swverify**.

476 *check_requisites = true*
477 Controls verification of fileset requisites.

478 If *check_requisites = true*, then **swverify** shall check the *prere-*
479 *quisites*, *corequisites*, and *exrequisites* attributes of files.

480 If *check_requisites = false*, then **swverify** shall not check the attri-
481 butes.

482 This applies to both distribution and installed_software.

Applies to **swverify**.

483 `check_scripts=true`
484 Controls the running of the **verify** script.

485 If `check_scripts=true`, then **swverify** shall run the vendor-supplied
486 **verify** script for each fileset when operating on installed_software
487 objects. When the **-F** option of **swverify** is used, the vendor-
488 supplied **fix** script is also executed.

489 If `check_scripts=false`, then **swverify** shall not not run the scripts.

490 Applies to **swverify**.

491 `check_volatile=false`
492 Controls check of volatile files.

493 If `check_volatile=true`, then **swverify** shall include files whose
494 **is_volatile** attribute is set to **true** in its check of files and their
495 attributes.

496 If `check_volatile=false`, then **swverify** shall not include volatile
497 files. This is useful to eliminate potentially “spurious” reports from
498 **swverify** when the only file changes are those to files known in
499 advance to be volatile.

500 Applies to **swverify**.

501 `compress_files=false`
502 Controls whether uncompressed files are to be compressed in the
503 target distribution, as specified by the value of `compression_type`.

504 If `compress_files=true`, then all files except those that have a
505 **compression_state** of **not_compressible** shall be compressed, or
506 shall remain compressed.

507 If `compress_files=false`, uncompressed files shall not be compressed,
508 and the status of any compressed file shall be determined by the
509 value of `uncompress_files`.

510 Applies to **swcopy**.

511 `compression_type=implementation_defined_value`
512 Specifies the compression type used to compress the software files.

513 The values supported for `compression_type` are implementation
514 defined.

515 The way in which an implementation uses this value to implement or
516 execute the compression or uncompression of a file is undefined.

517 Applies to **swcopy**.

518 `defer_configure=false`
519 Controls automatic configuration at install.

520 If `defer_configure=false`, software being installed shall also be
521 configured when the root directory is **/**.

522 If `defer_configure=true`, then the software is installed but not
 configured, and may require configuration (using **swconfig**) before
 being used.

523 Applies to **swinstall**.

524 *distribution_source_directory = implementation_defined_value*
525 Specifies the default distribution directory.

526 When a source specification does not contain a path specification, the
527 value of this extended option shall be used as the default source
528 distribution directory. When a source specification does contain a
529 path specification, it shall be used.

530 Applies to **swask**, **swcopy**, and **swinstall**.

531 *distribution_target_directory = implementation_defined_value*
532 Specifies the default distribution target.

533 When a target specification does not contain a path specification, the
534 value of this extended option shall be used as the default distribution
535 target. When a target specification does contain a path specification,
536 it shall be used. For **swpackage**, this shall be used only when
537 *media_type = directory*.

538 Applies to **swcopy**, **swlist**, **swmodify**, **swpackage**, **swremove**,
539 and **swverify**.

540 *distribution_target_serial = implementation_defined_value*
541 Specifies the default distribution target.

542 When a target specification does not contain a path specification and
543 *media_type = serial*, the value of this extended option shall be used as
544 the default distribution target. When a target specification does con-
545 tain a path specification, it shall be used.

546 Applies to **swpackage**.

547 *enforce_dependencies = true*
548 Controls the enforcement of dependency specifications.

549 If *enforce_dependencies = true*, no utility except **swremove** and the
550 unconfigure option of **swconfig** shall proceed unless necessary
551 dependencies have been selected, or already exist in the proper state
552 on the target. The **swremove** utility and the unconfigure portion of
553 the **swconfig** utility shall not proceed if operating on the selected
554 software leaves dependent software with their dependencies
555 unresolved beyond what existed before the utility was executed.

556 If *enforce_dependencies = false*, then all utilities shall proceed even if
557 some dependencies are not met. Enforcement of dependencies is
558 independent of whether or not they were autoselected.

559 Applies to **swconfig**, **swcopy**, **swinstall**, **swremove**, and
560 **swverify**.

561 *enforce_dsa = true*
562 Controls the handling of disk space analysis errors.

563 If *enforce_dsa = true*, the implementation-defined error handling pro-
564 cedure shall be invoked when the disk space analysis indicates there
is not enough disk space.

565 If *enforce_dsa=false*, then the operation shall be attempted even if
 566 disk space analysis indicated a problem.

567 Applies to **swcopy**, **swinstall**, and **swpackage**.

568 *enforce_locatable=true*
 569 Controls the handling of errors when relocating a non-relocatable
 570 fileset.

571 If *enforce_locatable=true*, an error shall be generated if an attempt
 572 is made to relocate a non-relocatable fileset.

573 If *enforce_locatable=false*, an attempt shall be made to relocate the
 574 fileset in any case.

575 Applies to **swinstall** and **swverify**.

576 *enforce_scripts=true*
 577 Controls the handling of errors generated by scripts.

578 If *enforce_scripts=true*, the implementation-defined error handling
 579 procedure shall be invoked when the vendor-supplied scripts return
 580 an error.

581 If *enforce_scripts=false*, all script errors shall be treated as warn-
 582 ings, and the utility shall attempt to continue operation.

583 Applies to **swinstall** and **swremove**.

584 *files*
 585 Lists the pathnames of file objects to be added or deleted.

586 If *files='file1 file2 file3 ...'*, then catalog information for those files
 587 shall be added or deleted. When files are added, the attributes of the
 588 file are retrieved from the actual file on the installed file system. File
 589 objects being added or deleted can also be specified in the **INFO** file
 590 format. There is no supplied default.

591 Applies to **swmodify**.

592 *follow_symlinks=false*
 593 Controls the following of symbolic links

594 If *follow_symlinks=false*, then do not follow any symbolic links that
 595 may exist in the packaging source.

596 If *follow_symlinks=true*, then attempt to follow symbolic links.

597 Applies to **swpackage**.

598 *installed_software_catalog=implementation_defined_value*
 599 Specifies installed software catalog.

600 This extended option, along with the *installed_software_path* attri-
 601 bute, defines the logical *installed_software* object upon which the
 602 utility is operating. This extended option is resolved relative to the
 603 **PATHNAME_CHARACTER_STRING** portion of the *targets* operand.
 604 See 4.1.4.2.

This option allows an implementation to define where the catalog
 information is stored. This option also allows multiple logical

605 installed_software objects to share the
606 **PATHNAME_CHARACTER_STRING** where the software is installed.

607 Applies to **swask**, **swconfig**, **swinstall**, **swlist**, **swmodify**,
608 **swremove**, and **swverify**.

609 *logfile=implementation_defined_value*
610 Specifies the location of the the logfile for the management role.

611 Logfile structure for all roles, logfile locations for other roles, and the
612 effect of this option on logfile location is implementation defined.

613 Applies to all utilities except **swlist**.

614 *loglevel=1*
615 Controls the amount of output sent by the utility to log files (not to
616 stdout and stderr).

617 See 4.1.6.5.

618 Applies to all utilities except **swlist**.

619 *media_capacity=0*
620 The storage capacity in megabytes of the output media.

621 A value of 0 (zero) indicates an infinite capacity.

622 Applies to **swpackage**.

623 *media_type=directory*
624 The default media type.

625 If *media_type=directory*, the distribution is located in the value of
626 the *distribution_target_directory* option.

627 If *media_type=serial*, the distribution is located in the value of the
628 *distribution_target_serial* option.

629 Applies to **swpackage**.

630 *one_liner=implementation_defined_value*
631 Specifies attributes to list.

632 The *one_liner* option specifies the attributes to list by default when
633 neither **-v** and **-a** *attribute* options are specified. Only attributes
634 that apply to each object listed are included for that object. At least
635 one of the *tag* attribute (of products, subproducts, filesets and control
636 scripts) or the *path* attribute (of files) shall be included. The order of
637 attributes in the output listing need not be the order of the attributes
638 specified in this option. The listing format used by *one_liner* is
639 undefined.

640 Applies to **swlist**.

641 *reconfigure=false*
642 Controls reconfiguring of software.

643 If *reconfigure=false*, do not reconfigure software if it is already in the
644 **configured** state. If *reconfigure=true*, reconfigure the software
even if it is already in the **configured** state.

645 Applies to **swconfig**.

646 *recopy=false*

647 Controls copying of filesets.

648 If *recopy=false*, do not copy a fileset that is already available on the

649 target at the same version.

650 If *recopy=true*, then copy the fileset in any case.

651 Applies to **swcopy**.

652 *reinstall=false*

653 Controls reinstallation of filesets.

654 If *reinstall=false*, do not install a fileset that already has the same

655 version already installed.

656 If *reinstall=true*, then reinstall the fileset even if this version is

657 already installed.

658 Applies to **swinstall**.

659 *select_local=true*

660 Controls default selection of target.

661 If *select_local=true*, and no targets are specified, then the local host

662 shall be selected as the target.

663 If *select_local=false*, then the local host shall not be automatically

664 included.

665 Applies to all utilities except **swask** and **swpackage**.

666 *software*

667 Specifies a default set of **software_selections** for the utility.

668 Applies to all utilities in this part of ISO/IEC 15068.

669 *targets*

670 Specifies a default set of *targets* for the utility.

671 See the *select_local* option.

672 Applies to all utilities in this part of ISO/IEC 15068 except **swpack-**

673 **age**.

674 *uncompress_files=false*

675 Controls whether compressed files are to be uncompressed in the tar-

676 get distribution, as specified by the value of the *compression_type*

677 attribute of the file.

678 If *uncompress_files=false*, all files with a *compression_state* attribute

679 value of **compressed** shall remain compressed, and the status of

680 uncompressed files shall be determined by the value of

681 *compress_files*.

682 If *uncompress_files=true*, all compressed files shall be uncompressed

683 before being written to the target distribution.

Applies to **swcopy**.

684 *verbose=1*
685 Controls the amount of output sent by the utility to stdout and
686 stderr, but not to log files.
687 For values that are non-negative integers, an increase in *verbose*
688 shall not decrease the information sent stdout and stderr. All
689 implementations shall support the values 0 (zero) and 1 (one). If
690 *verbose=0*, nothing shall be written to either stdout or stderr. The
691 effect of other values of *verbose* is undefined. See also 4.1.6.3 and
692 4.1.6.4.
693 Applies to all utilities in this part of ISO/IEC 15068.

694 4.1.5.3 Extended Options Syntax

695 The syntax is the same for options specified on the command line and for those
696 specified in the options file. See 2.1.2 for the grammar conventions for this syn-
697 tax. Individual options use this syntax as follows:

698 %token FILENAME_CHARACTER_STRING /* as defined in 2.2.2.37 */
699 %token PORTABLE_CHARACTER_STRING /* as defined in 2.2.2.68 */
700 %token SHELL_TOKEN_STRING /* as defined in 2.2.2.80 */
701 %token WHITE_SPACE_STRING /* as defined in 2.2.2.110 */

702 %start software_option
703 %%

704 software_option : command_qualifier keyword '=' value
705 ;

706 command_qualifier : /* empty */
707 | command '.'
708 ;

709 value : multi_value
710 | single_value
711 ;

712 multi_value : value ws single_value
713 | single_value
714 ;

715 single_value : SHELL_TOKEN_STRING
716 ;

717 command : FILENAME_CHARACTER_STRING
718 ;

719 keyword : FILENAME_CHARACTER_STRING
720 ;

721 ws : WHITE_SPACE_STRING
722 ;

723 With respect to this syntax, the following apply:

724 **command**725 A **keyword** prefixed by the **command** name applies to that utility only.726 If no prefix exists, then the **keyword** applies to all the utilities that support
727 it.728 **keyword**729 Names the option or operand being defined, e.g., *allow_incompatible* for
730 **swinstall**, and *verbose* for all the utilities.731 The allowable characters for a **keyword** are as defined in the filename
732 character set, plus the - (hyphen) character.733 **value**734 Assigns the value to the **keyword**.735 All extended options are single valued except those that contain lists of
736 **software_specs** or **software_collection_specs**. Quoting of
737 strings and escaping of characters shall be handled as specified in 3.2 and
738 3.3 of POSIX.2 {3}.739 When specified on the command line, multiple option specifications can be
740 included after a single -x option if included in quotes and separated by white
741 space. Multiple -x options can also be used.742 For option and defaults files, blank lines and all comment text shall be ignored.
743 Comment text is any sequence of characters beginning with a # (pound) character
744 that is neither escaped nor quoted, and continues through the end of that line.745 If the white space between single values contains a <newline>, either it shall be
746 escaped or the entire value shall be quoted.

747 The following are examples of this syntax:

748 **loglevel=1**
749 **allow_incompatible=false**
750 **autoselect_dependencies="as_needed"**
751 **software="Foo, r=1.2, a=hp-ux bar, a=Aix_3.2"**
752 **targets="hosta:/ hostb hostc:"**753 **software="Foo, r=1.2, a=hp-ux**
754 **bar, a=Aix_3.2"**755 **targets="hosta:/**
756 **hostb**
757 **hostc:**
758 **"**759 **4.1.5.3.1 Precedence for Option Specification**760 Multiple option or operand specifications have a precedence that defines which
761 specifications are used.762 Only the option specifications with the highest level of precedence are used for
763 each option and operand. The precedence is the following, in increasing order:

- 764 (1) System defaults file
-
- (2) User defaults file

765 (3) Options file

766 (4) Command line options and operands

767 If there are multiple instances of options at any particular level, then the follow-
768 ing rules apply:

769 — If both **keyword** and **command.keyword** exist in the set of defaults or
770 options files for this level, the **command** uses the latter, more specific,
771 definition.

772 — All values for *software* and *targets* options from all levels are included in
773 the resulting *software_selections* and *target_selections* for the command.

774 — For options besides *software* and *targets*, the behavior, when multiple or
775 conflicting specifications are made, is undefined. This rule applies to
776 options such as **-S source** where implementations may choose to assign a
777 logical interpretation to multiple source specification. The same rule
778 applies to options that are mutually exclusive.

779 4.1.5.4 Standard Input

780 4.1.5.5 Input Files

781 The sections describing the **swpackage** and **swmodify** utilities specifies addi-
782 tional input files specific to those utilities.

783 4.1.5.6 Access and Concurrency Control

784 An implementation of this part of ISO/IEC 15068 shall allow a user to create,
785 modify, delete, and access a catalog that describes a software object located where
786 it is permissible for that user to respectively create, modify, delete, and access
787 files. Other authorization, authentication and concurrency control requirements
788 and mechanisms are undefined within this part of ISO/IEC 15068. This part of
789 ISO/IEC 15068 does provide event definitions that an implementation can use for
790 access and concurrency control errors.

791 If the user of a utility does not have the proper authorization to run a utility,
792 access a *software_collection*, or access software objects within that utility, the tar-
793 get may generate an event.

794 (SW_ERROR: SW_ACCESS_DENIED)

795 If the concurrency control mechanism prevents simultaneous operation on a
796 software collection or software object, the target may generate an event.

797 (SW_ERROR: SW_CONFLICTING_SESSION_IN_PROGRESS)

798 If the command will proceed anyway, then the target may generate an event.

799 (SW_WARNING: SW_CONFLICTING_SESSION_IN_PROGRESS)

800 If the concurrency control mechanism fails for other reasons, the target may gen-
801 erate an event.

802 (SW_ERROR: SW_SOC_LOCK_FAILURE)

803 **4.1.5.7 Environment Variables**

804 Environment variables are a feature of this part of ISO/IEC 15068 inherited from
805 POSIX.1 {2}. The following environment variables shall affect the execution of all
806 the utilities defined in this part of ISO/IEC 15068:

807 **LANG** This variable shall determine the locale to use for the
808 locale categories when both **LC_ALL** and the correspond-
809 ing environment variable (beginning with **LC_**) do not
810 specify a locale.

811 See 2.6 of POSIX.2 {3}.

812 **LC_ALL** This variable shall determine the locale to be used to over-
813 ride any values for locale categories specified by the set-
814 tings of **LANG** or any environment variables beginning
815 with **LC_**.

816 **LC_CTYPE** This variable shall determine the interpretation of
817 sequences of bytes of text data as characters (e.g., single-
818 versus multibyte characters in values for vendor-defined
819 attributes).

820 **LC_MESSAGES** This variable shall determine the language in which mes-
821 sages should be written.

822 **LC_TIME** This variable shall determine the format of dates
823 (*create_date* and *mod_date*) when displayed by **swlist**.

824 It should also be used by all utilities when displaying
825 dates and times in stdout, stderr, and logging.

826 **TZ** This variable shall determine the time zone for use when
827 displaying dates and times.

828 **4.1.6 External Effects**829 **4.1.6.1 Control Script Execution and Environment**

830 The utilities defined in this part of ISO/IEC 15068 shall cause control files to be
831 interpreted according to the following rules:

832 (1) If no value is set for the *control_file.interpreter* attribute, or if the value is
833 set to either the empty string or **sh**, the script shall be interpreted by the
834 POSIX.2 {3} shell.

835 (2) If the value of *control_file.interpreter* is set to a value other than the
836 empty string or **sh**, then the utility shall determine the availability of the
837 interpreter in an operating system dependent fashion equivalent to
838 searching **PATH** for an executable file with a filename equivalent to the
839 value of *control_file.interpreter*. If the interpreter is determined to be
840 available, the control file shall be interpreted using that interpreter.

841 (3) If no interpreter is available, then a return code value of **1** (one) shall be
 presumed for the script, and all other actions defined for that return code
 shall be assumed. See 4.2, 4.3, 4.5, 4.9, and 4.10.

842 During the execution of each such control script, the following environment vari-
843 ables shall be defined for the environment of the `control_script`:

844 **SW_CATALOG**

845 The value of the *installed_software.catalog* attribute indicating the
846 location or identification of the catalog relative to the
847 **SW_ROOT_DIRECTORY**.

848 **SW_CONTROL_DIRECTORY**

849 The directory where the executing script is located.

850 This directory shall be readable from within control script execution
851 and shall be writable from commands within control scripts when the
852 **request** script is being executed. All **control_files** shall be
853 readable by any control script.

854 **SW_CONTROL_TAG**

855 The *tag* of the script being executed.

856 This allows the `control_script` to tell what *tag* is being executed when
857 the actual script path is defined for more than one *tag*.

858 **SW_LOCATION**

859 The base directory where the product or fileset will be installed or is
860 already installed.

861 This is the value of the *location* attribute.

862 **SW_PATH**

863 A **PATH** that, at least, contains all utilities defined by the
864 POSIX.2 {3}.

865 NOTE: POSIX.2 {3}, in 7.8 and B.10.1, requires and defines the C function *confstr()*
866 that obtains such a **PATH**.

867 **SW_ROOT_DIRECTORY**

868 The *installed_software.path* attribute of the installed software object
869 within which the software containing this `control_file` shall be
870 installed.

871 This is the directory relative to which all operations with the script
872 shall be performed.

873 NOTE: For example, if this normally has a value of `/`, but if a proxy install is done
874 to a target directory `/mnt/test/`, this shall have the value of `/mnt/test/`.

875 **SW_SESSION_OPTIONS**

876 The pathname of a file containing the value of every option defined
877 for the software utility being executed, using the options syntax
878 described in 4.1.5.3.

879 The option syntax shall be restricted such that the command prefix
880 shall not be used, there shall be no spaces on either side of the `=`
881 (equal sign), and multiple valued options shall have the values
882 quoted.

This environment variable allows scripts to retrieve any options and
values for this command other than the ones provided explicitly via
environment variables. When the file pointed to by

883 **SW_SESSION_OPTIONS** is made available to **request** scripts,
 884 the *targets* option shall contain a list of
 885 **software_collection_specs** for all targets specified for the
 886 command. When the file pointed to by **SW_SESSION_OPTIONS** is
 887 made available to other scripts, the *targets* option shall contain the
 888 single **software_collection_spec** for the targets on which the
 889 script is being executed.

890 An implementation should ensure that each
 891 **software_collection_spec** contained in the value of the *targets*
 892 option is the same between invocations of commands. This will help
 893 ensure that any per-target information stored by the **request** script
 894 can be located by the subsequent scripts.

895 **SW_SOFTWARE_SPEC**

896 The value of the fully qualified **software_spec** identifying the
 897 software object containing this control script.

898 See 4.1.4.1.1.

899 **4.1.6.1.1 Control Script Stdout and Stderr**

900 The scripts may send information, particularly about reasons for error conditions
 901 to stdout and stderr. The utilities shall log stdout and stderr to the logfile of the
 902 role executing the script.

903 **4.1.6.1.2 Control Script Return Code**

904 The scripts shall return with a return code of 0 (zero), 1, or 2. Additionally, **chec-**
 905 **kininstall**, **checkremove**, **configure**, and **unconfigure** scripts may return
 906 with a return code of 3. The return codes 4 through 31 (inclusive) are reserved for
 907 future use. The meaning of these return codes is shown in the following table:

908 **Table 4-2 – Script Return Codes**

910 Return Code	Effect of Return Code	Status
911 0	The script executed successfully. The utility will proceed normally.	SW_NOTE
912 1	The script had an error. The utility shall generate an error event and implement the error procedure defined for this script type.	SW_ERROR
913 2	The script had a warning. The utility will generate a warning event and continue.	SW_WARNING
914 3	The script is forcing a deselection of this product or filesset. The utility will generate a note and skip this product or filesset during any further processing.	SW_NOTE
915 4-31	Reserved.	

922 All scripts, with the exception of the **request** script, shall be non-interactive.

923 An implementation can define behaviors for additional script return codes. Any
 924 such behavior is implementation defined.

Return codes with no behavior defined by either this part of ISO/IEC 15068 or the implementation should be treated using the behavior associated with return code

2.

4.1.6.2 Asynchronous Events

The following are the set of events generated by the utilities defined in this part of ISO/IEC 15068. These events are generated during the course of a execution of a utility. See 4.1.6.5.

The event codes and their numeric values are listed in Table 4-4 through Table 4-7, inclusive.

NOTE: Not all events are generated by each utility. For example, events related to script execution only apply to **swinstall**, **swask**, **swremove** and **swverify**. The specific events generated by each utility are defined in the appropriate subclause for that utility.

Each event generated also has a severity status associated with it. The event status that can occur for each event is also listed. Event status also has a numeric value, as described in Table 4-3, Table 4-4, Table 4-5, Table 4-6, and Table 4-7. In addition, all numeric values between 0 and 255 (inclusive) are reserved, either for use in this part of ISO/IEC 15068 (as described in the accompanying tables) or for use in future revisions of this part of ISO/IEC 15068.

Table 4-3 – Event Status

Status	Effect of Event	Value
SW_NOTE	The operation continues normally	0
SW_ERROR	implementation-defined error handling procedure is invoked	1
SW_WARNING	The operation continues normally	2

A command shall not have an exit code of zero if any **SW_ERROR** event occurred during the course of a command.

The descriptions in the following tables describe the conditions that lead to this event, and the set of possible event status values for the event. The tables also include “Manager info” and “Target info”, describing the additional information that may be logged for manager and target role event logging, respectively. See 4.1.6.5.

Table 4-4 lists general source and target role events. The way in which some of these events are generated (if at all) may be different for different implementations. Table 4-5 lists the source and target role events related to initialization of a session and ending a session. The way in which some of these events are generated (if at all) may be different for different implementations. Table 4-6 lists the source and target role events related to the analysis phase of the commands. Some of these are also related to the execution phase of the commands. Table 4-7 lists the target role events related to the execution phase.

4.1.6.3 Stdout

Events with a status of **SW_NOTE** shall, if permitted by the value *verbose*, be directed to stdout. Manager role events shall, if permitted by the value *verbose*, be directed to stdout. Nothing shall be written to stdout if *verbose*=0 (zero). The

Table 4-4 – General Error Events

Event Code	Event Status	Description	Value
SW_ILLEGAL_STATE_TRANSITION	SW_ERROR	The manager is requesting a phase out of order. Manager info: target. Target info: current phase.	1
SW_BAD_SESSION_CONTEXT	SW_ERROR	The manager has contacted the wrong target, or this is not a valid manager for this session. Manager info: target. Target info: information about the initiator.	2
SW_ILLEGAL_OPTION	SW_ERROR	An illegal or unrecognized option was sent. Manager info: target, number of options. Target info: option names and values.	3
SW_ACCESS_DENIED	SW_ERROR	The user has insufficient privilege to perform the requested operation. Manager info: target. Target info: information about the initiator.	4
SW_MEMORY_ERROR	SW_ERROR	The target role had a memory allocation error (e.g., out of swap). Manager info: target. Target info: reasons for error.	5
SW_RESOURCE_ERROR	SW_ERROR	The target role had a resource allocation error such as maximum number of processes reached, maximum number of files open, etc. Manager info: target. Target info: reasons for error.	6
SW_INTERNAL_ERROR	SW_ERROR	The target role had an internal implementation error. Manager info: target. Target info: reasons for error.	7
SW_IO_ERROR	SW_ERROR	An I/O error occurred while performing this command. Manager info: target. Target info: reasons for error.	8

writing of any target role events to stdout is undefined.

4.1.6.4 Stderr

If any events with a status of **SW_ERROR** or **SW_WARNING** occur on a target role, this information shall be communicated to the management role. In addition, at least a single message for that target shall, if permitted by the value *verbose*, be directed to the stderr of the management role. Nothing is written to stderr if *verbose=0* (zero).

The sending of any additional messages to stderr of the management role is undefined.

See *verbose* in 4.1.5.2.

Table 4-5 – Session Events

Event Code	Event Status	Description	Value
SW_AGENT_INITIALIZATION_FAILED	SW_ERROR	Failed to initialize a target session. Manager info: target. Target info: reasons for error.	10
SW_SERVICE_NOT_AVAILABLE	SW_ERROR	The target role is not accepting new requests. Manager info: target. Target info: reasons for error.	11
SW_OTHER_SESSIONS_IN_PROGRESS	SW_WARNING	There are other sessions in progress that may affect the results of this command. Manager info: target, number of sessions. Target info: information about other sessions.	12
SW_SESSION_BEGINS	SW_NOTE	The command begins on the target. Manager info: target. Target info: information about the initiator of the command.	28
SW_SESSION_ENDS	SW_NOTE SW_WARNING SW_ERROR	The command ends on the target successfully, with warnings, or with errors. Manager info: target. Target info: none.	29
SW_CONNECTION_LIMIT_EXCEEDED	SW_ERROR	The limit of source or target role sessions on this host has already been reached. Manager info: target, number of sessions. Target info: number of sessions, limit.	30
SW_SOC_DOES_NOT_EXIST	SW_ERROR	The requested target or source software collections does not exist. Manager info: target. Target info: reasons for error.	31
SW_SOC_IS_CORRUPT	SW_ERROR	The software_collection exists, but the information is corrupt. Manager info: target. Target info: reasons for error.	32
SW_SOC_CREATED	SW_NOTE	The target software_collection did not previously exist and was created. Manager info: target. Target info: none.	34
SW_CONFLICTING_SESSION_IN_PROGRESS	SW_ERROR SW_WARNING	A conflicting session is in progress that will prevent this operation (error), or cause its results to possibly be invalid (warning). Manager info: target. Target info: information about other sessions.	35
SW_SOC_LOCK_FAILURE	SW_ERROR	cannot set the proper access control to this source or target. Manager info: target. Target info: reasons for error.	36

Table 4-5 - Session Events (concluded)

Event Code	Event Status	Description	Value
SW_SOC_IS_READ_ONLY	SW_ERROR SW_NOTE	The software_collection is a read only source for a read source or target (note), or is a target to be modified (error). Manager info: target. Target info: none.	37
SW_SOC_IS_REMOTE	SW_ERROR SW_NOTE	The software_collection is on a remote file system. (Whether note or error is implementation defined). Manager info: target. Target info: none.	38
SW_SOC_INCORRECT_MEDIA_TYPE	SW_ERROR	The distribution is an incorrect type for the command (e.g., a tape for swremove). Manager info: target. Target info: reasons for error.	39
SW_SOC_IS_SERIAL	SW_NOTE	The distribution has a serial format (e.g., a tape). Manager info: target. Target info: none.	40
SW_SOC_INCORRECT_TYPE	SW_ERROR	The software_collection is of the wrong type (distribution or installed_software) for the operation. Manager info: target. Target info: target type.	41
SW_CANNOT_OPEN_LOGFILE	SW_ERROR	Cannot open logfile to log the software_collection events. Manager info: target. Target info: reasons for error.	42
SW_SOC_AMBIGUOUS_TYPE	SW_ERROR	The software collection is inadequately specified for the operation. Manager info: target. Target info: reason for error	49
SW_TERMINATION_DELAYED	SW_NOTE	The target role is currently analyzing or executing a command and will terminate the session once completed. Manager info: target. Target info: none.	50
SW_CANNOT_INITIATE_REBOOT	SW_WARNING	The target role failed to initiate the reboot operation of an install command and requires manual reboot. Manager info: target. Target info: reasons for error.	51

4.1.6.5 Logging

The management role and target role each log events. The way in which logging is implemented, including the location of the logfiles, is implementation defined.

Which messages, if any, are placed in the source role logfile is undefined.

All implementations shall support the values **0** (zero), **1** (one) and **2**. If *loglevel*=0, nothing shall be written to log files. The target role shall log all events, except file level events, when *loglevel*=1. The target role shall redirect, to the logfile, stderr and stdout from control scripts when *loglevel*=1. When *loglevel*=2, the target role shall log file level events. For values which are non-

Table 4-6 – Analysis Phase Events

Event Code	Event Status	Description	Value
SW_ANALYSIS_BEGINS	SW_NOTE	The analysis phase begins on the target. Manager info: target. Target info: none.	52
SW_ANALYSIS_ENDS	SW_NOTE SW_WARNING SW_ERROR	The analysis phase ends on the target. The analysis may have succeeded, had warnings, and/or errors. Manager info: target. Target info: none.	53
SW_EXREQUISITE_EXCLUDE	SW_NOTE	One or more filesets were excluded automatically as the software identified as exrequisites was also specified to be selected. Manager info: target, number of filesets. Target info: software_specs	56
SW_CHECK_SCRIPT_EXCLUDE	SW_NOTE	One or more checkinstall or checkremove scripts have caused the software to be unselected and excluded from further processing. Manager info: target, number of filesets. Target info: software_specs	57
SW_CONFIGURE_EXCLUDE	SW_NOTE	One or more configure or unconfigure scripts have caused the software to be unselected and excluded from further processing. Manager info: target, number of filesets. Target info: software_specs	58
SW_SELECTION_IS_CORRUPT	SW_ERROR	The software selection was found, but its state was corrupt or transient . Manager info: target, number of selections Target info: software_specs	59
SW_SOURCE_ACCESS_ERROR	SW_ERROR	Failure contacting or retrieving information from the source. Manager info: target. Target info: reasons for error.	60
SW_SOURCE_NOT_FIRST_MEDIA	SW_ERROR	The source does not have a media number of 1 (needed for retrieval of the INDEX). Manager info: target, current media number. Target info: current media number.	61

negative integers, an increase in *loglevel* shall not decrease the information logged for a given role. All other behavior regarding logging is undefined.

See *loglevel* in 4.1.5.2.

Table 4-6 - Analysis Phase Events (continued)

Event Code	Event Status	Description	Value
SW_SELECTION_NOT_FOUND	SW_ERROR SW_NOTE	One or more software selections cannot be found. This is an error for install or copy; otherwise, a note. Manager info: target, number of selections. Target info: software_specs	62
SW_SELECTION_NOT_FOUND_RELATED	SW_ERROR SW_NOTE	One or more software selections cannot be found as specified, but another version exists. This is an error for install or copy; otherwise, a note. Manager info: target, number of selections. Target info: software_specs	63
SW_SELECTION_NOT_FOUND_AMBIG	SW_ERROR	One or more software selections cannot be unambiguously determined. Manager info: target, number of selections. Target info: software_specs	64
SW_FILESYSTEMS_NOT_MOUNTED	SW_ERROR SW_WARNING	One or more file systems on the target are not mounted. Manager info: target, number of file systems. Target info: file system names.	65
SW_FILESYSTEMS_MORE_MOUNTED	SW_WARNING	One or more file systems mounted are not in file system table. Manager info: target, number of file systems. Target info: file system names.	66
SW_HIGHER_REVISION_INSTALLED	SW_ERROR SW_WARNING	One or more filesets have a higher revision already installed. Whether error or warning is controlled by <i>allow_downdate</i> option. Manager info: target, number of filesets. Target info: software_specs	67
SW_NEW_MULTIPLE_VERSION	SW_ERROR SW_NOTE	One or more products would create a new version in an installation. Whether error or warning is controlled by <i>allow_multiple_versions</i> option. Manager info: target, number of products. Target info: software_specs	68

4.1.7 Extended Description

See the individual utility descriptions for the complete extended descriptions of each task. This subclause lists the steps common to the utilities.

There are three phases in the utilities as follows. Targets may be processed in any order or in parallel.

- (1) Selection phase —

The user specifications are resolved, including source, target, and

Table 4-6 - Analysis Phase Events (continued)

Event Code	Event Status	Description	Value
SW_EXISTING_MULTIPLE_VERSION	SW_ERROR SW_WARNING SW_NOTE	The command is operating on an existing multiple version of one or more products. If trying to install two versions into one location, generate an event. Warning or note controlled by <i>allow_multiple_versions</i> option. Manager info: target, number of products. Target info: software_specs	69
SW_DEPENDENCY_NOT_MET	SW_ERROR SW_WARNING	One or more dependencies cannot be met. Whether error or warning is controlled by <i>enforce_dependencies</i> option. Manager info: target, number of filesets. Target info: software_specs, dependency_specs	70
SW_NOT_COMPATIBLE	SW_ERROR SW_WARNING	One or more products are incompatible for this target. Whether error or warning is controlled by <i>allow_incompatible</i> option. Manager info: target, number of products. Target info: software_specs	71
SW_CHECK_SCRIPT_WARNING	SW_WARNING	One or more checkinstall, checkremove or verify scripts had a warning. Manager info: target, number of filesets. Target info: software_specs	72
SW_CHECK_SCRIPT_ERROR	SW_ERROR	One or more checkinstall, checkremove or verify scripts failed. Manager info: target, number of filesets. Target info: software_specs	73
SW_DSA_INTO_MINFREE	SW_ERROR SW_WARNING	Disk space analysis is over the minimum free limit, but not the overall limit on the target. Whether error or warning is controlled by <i>enforce_dsa</i> option. Manager info: target, number of file systems. Target info: file system names, amount over the minimum free.	74

software selections.

(2) Analysis phase —

The utility attempts to discover conditions that may cause failure when operating on the selected software

(3) Execution phase —

The actual operations on the software objects take place

Table 4-6 - Analysis Phase Events (continued)

Event Code	Event Status	Description	Value
SW_DSA_OVER_LIMIT	SW_ERROR SW_WARNING	Disk space analysis is over the absolute limit. Whether error or warning is controlled by <i>enforce_dsa</i> option. Manager info: target, number of file systems. Target info: file system names, amount over the limit.	75
SW_DSA_FAILED_TO_RUN	SW_ERROR SW_WARNING	Disk space analysis had an internal error and failed to run. Whether error or warning is controlled by <i>enforce_dsa</i> option. Manager info: target, number of filesets. Target info: software_specs	76
SW_SAME_REVISION_INSTALLED	SW_NOTE	One or more filesets have the same revision and are being reinstalled because <i>reinstall=true</i> or <i>recopy=true</i> . Manager info: target, number of filesets. Target info: software_specs	77
SW_ALREADY_CONFIGURED	SW_NOTE	One or more filesets are already configured. Whether they are reconfigured is controlled by <i>reconfigure</i> option. Manager info: target, number of filesets. Target info: software_specs	78
SW_SKIPPED_PRODUCT_ERROR	SW_NOTE	One or more filesets will be skipped because of another error within their product. (Error handling is implementation defined). Manager info: target, number of filesets. Target info: software_specs	79
SW_SKIPPED_GLOBAL_ERROR	SW_NOTE	One or more filesets will be skipped because of a global error (such as disk space analysis failure) within the analyze phase. (Error handling is implementation defined). Manager info: target, number of filesets. Target info: software_specs	80
SW_FILE_IS_REMOTE	SW_WARNING SW_NOTE	One or more files would be created or removed on a remote file system. (Policy for loading remote files is implementation defined). Manager info: target, number of files. Target info: file paths.	81

When a utility initiates a session on a target, generate an event.
(SW_NOTE: SW_SESSION_BEGINS)

When the session completes, generate an event.
(SW_NOTE: SW_SESSION_ENDS)

Table 4-6 - Analysis Phase Events (concluded)

Event Code	Event Status	Description	Value
SW_FILE_IS_READ_ONLY	SW_WARNING	One or more files will not be attempted to be created or removed on a read only file system. Manager info: target, number of files. Target info: file paths.	82
SW_FILE_NOT_REMOVABLE	SW_WARNING	One or more files could not be removed (e.g., text busy, or non-empty directories). Manager info: target, number of files. Target info: file paths.	83
SW_FILE_WARNING	SW_WARNING	One or more files had warnings in analysis or execution. Manager info: target, number of files. Target info: file paths.	84
SW_FILE_ERROR	SW_ERROR	One or more files had errors in analysis or execution. Manager info: target, number of files. Target info: file paths.	85
SW_NOT_LOCATABLE	SW_WARNING SW_ERROR	A fileset is not locatable. Controlled by the <i>enforce_locatable</i> option. Manager info: target, number of filesets. Target info: software_specs	86
SW_SAME_REVISION_SKIPPED	SW_NOTE	One or more filesets have the same revision and are being skipped because <i>reinstall=false</i> or <i>recopy=false</i> . Manager info: target, number of filesets. Target info: software_specs	87

4.1.7.1 Selection Phase

This subclause summarizes the common tasks in the selection phase. Errors and warnings are listed along with the tasks.

4.1.7.1.1 Starting a Session

On invocation, each command processes options as defined in 4.1.5.3.1.

The command shall exit if the user does not have appropriate privilege. Since implementation of the security scheme is unspecified within this part of ISO/IEC 15068, appropriate privilege is implementation defined. An implementation may generate the following events at any point in the execution of the command if they are applicable to that implementation:

(SW_ERROR: SW_ACCESS_DENIED)
(SW_ERROR: SW_ILLEGAL_OPTION)
(SW_ERROR: SW_MEMORY_ERROR)
(SW_ERROR: SW_RESOURCE_ERROR)
(SW_ERROR: SW_INTERNAL_ERROR)
(SW_ERROR: SW_TERMINATION_DELAYED)
(SW_ERROR: SW_IO_ERROR)

Table 4-7 – Execution Phase Events

Event Code	Event Status	Description	Value
SW_EXECUTION_BEGINS	SW_NOTE	The execution phase begins on the target. Manager info: target. Target info: none.	88
SW_EXECUTION_ENDS	SW_NOTE	The execution phase ends on the target. Manager info: target. Target info: none.	89
SW_SELECTION_SKIPPED_IN_ANALYSIS	SW_WARNING		
	SW_ERROR		
	SW_NOTE	One or more selections will not be included for execution because they were determined to be skipped in analysis. Manager info: target, number of filesets. Target info: software_specs	90
SW_SELECTION_NOT_ANALYZED	SW_ERROR	One or more software selections were found, but were not analyzed. Manager info: target, number of filesets. Target info: software_specs	91
SW_WRONG_MEDIA_SET	SW_ERROR	The source media current being used is not the same as that used for analysis. Manager info: target. Target info: information about current media and needed media.	92
SW_NEED_MEDIA_CHANGE	SW_NOTE	The target needs the next media. (Interactive support for media change is implementation defined). Manager info: target, needed media sequence number. Target info: needed media sequence number.	93
SW_CURRENT_MEDIA	SW_NOTE	The current media sequence number for the target Manager info: target, current media sequence number. Target info: current media sequence number.	94
SW_PRE_SCRIPT_WARNING	SW_WARNING	One or more preinstall , preremove , unpreinstall , or fix scripts had a warning. Manager info: target, number of scripts. Target info: software_spec , script <i>tag</i>	95

4.1.7.1.2 Specifying Targets

A target is specified using the syntax in 4.1.4.2. Each target shall pass the following validation checks. If any of these checks fail, the command shall invoke the implementation-defined error handling procedure.

- If the target role was unable to initialize the session on the target host, generate an event.
(SW_ERROR: SW_AGENT_INITIALIZATION_FAILED)
- If administrative access is denied by the target role on the target, generate an event.

Table 4-7 - Execution Phase Events (continued)

Event Code	Event Status	Description	Value
SW_PRE_SCRIPT_ERROR	SW_WARNING SW_ERROR	One or more preinstall , preremove , unpreinstall , or fix scripts failed. Manager info: target, number of scripts. Target info: software_spec , script <i>tag</i>	96
SW_FILESET_WARNING	SW_WARNING	One or more filesets had a warning. Manager info: target, number of filesets. Target info: software_specs	97
SW_FILESET_ERROR	SW_ERROR	One or more filesets had an error. Manager info: target, number of filesets. Target info: software_specs	98
SW_POST_SCRIPT_WARNING	SW_WARNING	One or more postinstall , postremove , or unpostinstall scripts had a warning. Manager info: target, number of filesets. Target info: software_specs	99
SW_POST_SCRIPT_ERROR	SW_WARNING SW_ERROR	One or more postinstall , postremove , or unpostinstall scripts failed. Manager info: target, number of filesets. Target info: software_specs	100
SW_POSTKERNEL_WARNING	SW_WARNING	The postkernel kernel build script had a warning. Manager info: target. Target info: reasons for error.	101
SW_POSTKERNEL_ERROR	SW_ERROR	The kernel failed to build. Manager info: target. Target info: reasons for error.	102
SW_CONFIGURE_WARNING	SW_WARNING	One or more configure or unconfigure scripts had a warning. Manager info: target, number of scripts. Target info: software_specs , script <i>tag</i>	103

(SW_ERROR: SW_ACCESS_DENIED)

— Except for **swinstall**, **swcopy**, and **swpackage**, if the target does not exist on a host, generate an event.

(SW_ERROR: SW_SOC_DOES_NOT_EXIST)

— For **swinstall**, **swcopy**, and **swpackage**, if the target directory does not exist on a host, it is created with default attributes and an event is generated.

(SW_NOTE: SW_SOC_CREATED)

— If the target is corrupt, generate an event.

(SW_ERROR: SW_SOC_IS_CORRUPT)

— If the target is the wrong type (**installed_software** or **distribution**) for the target type the user specified (with a **-r** or **-d** option respectively), generate an event.

(SW_ERROR: SW_SOC_INCORRECT_TYPE)

Table 4-7 - Execution Phase Events (concluded)

Event Code	Event Status	Description	Value
SW_CONFIGURE_ERROR	SW_WARNING SW_ERROR	One or more configure or unconfigure scripts failed. Manager info: target, number of scripts. Target info: software_specs , script <i>tags</i>	104
SW_DATABASE_UPDATE_ERROR	SW_ERROR	An update to the catalog information for installed software or distributions failed. Manager info: target. Target info: reasons for error.	105
SW_REQUEST_WARNING	SW_WARNING	One or more request scripts had a warning. Manager info: software_specs	106
SW_REQUEST_ERROR	SW_ERROR	One or more request scripts failed. Manager info: software_specs	107
SW_COMPRESSION_FAILURE	SW_ERROR	A file could not be compressed or uncompressed. Manager info: target. Target info: filepath.	112
SW_FILE_NOT_FOUND	SW_ERROR	File is missing from the source or target software collection. Manager info: target, number of files. Target info: file path.	113
SW_FILESET_BEGINS	SW_NOTE	The execution phase of a fileset begins. Manager info: target. Target info: software_spec	117
SW_CONTROL_SCRIPT_BEGINS	SW_NOTE	The execution of a control script begins. Manager info: target. Target info: software_spec , control script <i>tag</i>	118
SW_FILE_BEGINS	SW_NOTE	The execution phase of file begins. Manager info: target. Target info: file path.	119

— If both a distribution object and an installed software object exist at the location specified in the target, and neither the **-d** nor the **-r** option is specified, generate an event.

(SW_ERROR: SW_SOC_AMBIGUOUS_TYPE)

— If the target is a serial distribution, generate an event.

(SW_NOTE: SW_SOC_IS_SERIAL)

— If the command is **swcopy** or **swpackage**, a serial distribution shall be overwritten by default. If the command is **swremove**, **swmodify**, or **swverify**, and the implementation does not support these on a serial distribution, generate an event.

(SW_ERROR: SW_SOC_INCORRECT_MEDIA_TYPE)

— If the target is not able to open the implementation-defined logfile, generate an event.

(SW_ERROR: SW_CANNOT_OPEN_LOGFILE)

- 1520 — If the operation needs to modify the target, and it is on a read-only media or
1521 file system, generate an event.
1522 (SW_ERROR: SW_SOC_IS_READ_ONLY)
- 1523 — An implementation may generate the following events, if they are applica-
1524 ble to that implementation, when validating a target:
1525 (SW_ERROR: SW_SERVICE_NOT_AVAILABLE)
1526 (SW_WARNING: SW_OTHER_SESSIONS_IN_PROGRESS)
1527 (SW_ERROR: SW_CONNECTION_LIMIT_EXCEEDED)
1528 (SW_NOTE: SW_SOC_IS_REMOTE)
1529 (SW_ERROR: SW_FILESYSTEMS_NOT_MOUNTED)
1530 (SW_ERROR: SW_FILESYSTEMS_MORE_MOUNTED)

1531 4.1.7.1.3 Specifying the Source

1532 This subclause only applies to **swcopy** and **swinstall**. The source contains
1533 software organized in the software packaging layout. A target can be specified
1534 using the syntax in 4.1.4.2.

1535 If source is specified, then it is resolved in the context of the management role. If
1536 source is not specified, then a default value is supplied as defined in 4.1.5.3.1.

1537 A source shall satisfy the following validation checks:

- 1538 — If the source does not exist or if it cannot be accessed, generate an event.
1539 (SW_ERROR: SW_SOURCE_ACCESS_ERROR)
- 1540 — If administrative access is denied by the source role for that source, gen-
1541 erate an event.
1542 (SW_ERROR: SW_ACCESS_DENIED)
- 1543 — Obtain information on what software is available from the source. If the
1544 information cannot be retrieved, or if an problem occurs while processing it,
1545 generate an event.
1546 (SW_ERROR: SW_SOURCE_ACCESS_ERROR)
- 1547 — If the source is a serial medium, and the *media.sequence_number* is not 1,
1548 generate an event.
1549 (SW_ERROR: SW_SOURCE_NOT_FIRST_MEDIA)
- 1550 NOTE: Only the first medium has the catalog information on it.

1551 4.1.7.1.4 Software Selections

1552 Software selections can be specified on the command line or in an input file using
1553 the syntax in 4.1.4.1.

1554 4.1.7.2 Analysis Phase

1555 This subclause summarizes the common operations and events in the analysis
1556 phase. The analysis phase occurs before the execution phase, and involves execut-
1557 ing checks to determine whether or not the execution should be attempted.

1558 When the analysis phase begins, generate an event.
(SW_NOTE: SW_ANALYSIS_BEGINS)

- 1559 To begin the analysis phase, the management role (the host on which the utility
1560 was invoked) communicates the selection information to each target in the target
1561 list. The target role accesses the source (for **swinstall** or **swcopy**) or target
1562 (for other utilities) to get the product catalog information for the software selec-
1563 tions. The product catalog information includes control script information in the
1564 *control_files* attribute of filesets within each product.
- 1565 — An implementation may generate any of the following events, if they are
1566 applicable to that implementation, when attempting the analysis or execu-
1567 tion phase.
1568 (SW_ERROR: SW_AGENT_INITIALIZATION_FAILED)
1569 (SW_ERROR: SW_ILLEGAL_STATE_TRANSITION)
1570 (SW_ERROR: SW_BAD_SESSION_CONTEXT)
 - 1571 — If the target role cannot access the source, generate an event.
1572 (SW_ERROR: SW_SOURCE_ACCESS_ERROR)
 - 1573 — If an implementation supports access control for particular operations for
1574 particular software objects, and if access is denied for any software object,
1575 generate an event.
1576 (SW_ERROR: SW_ACCESS_DENIED)
 - 1577 — If a fileset has an error for which there is not a more specific event defined,
1578 generate the generic event.
1579 (SW_ERROR: SW_FILESET_ERROR)
 - 1580 — If a fileset has a warning for which there is not a more specific event
1581 defined, generate the generic event.
1582 (SW_WARNING: SW_FILESET_WARNING)
 - 1583 — An implementation may generate the following events as applicable to the
1584 error handling procedures for that implementation:
1585 (SW_NOTE: SW_SKIPPED_PRODUCT_ERROR)
1586 (SW_NOTE: SW_SKIPPED_GLOBAL_ERROR)
1587 (SW_WARNING: SW_FILE_IS_READ_ONLY)
- 1588 See each utility section for the analysis operations specific to each utility. When
1589 the analysis phase ends, generate an event.
1590 (SW_NOTE: SW_ANALYSIS_ENDS)

1591 **4.1.7.3 Execution Phase**

- 1592 This subclause summarizes the common operations and events in the execution
1593 phase.
- 1594 When the execution phase begins, generate an event.
1595 (SW_NOTE: SW_EXECUTION_BEGINS)
- 1596 The execution phase proceeds through the steps defined for each utility. The fol-
1597 lowing events are common to all utilities:
- 1598 — When the execution phase executes a script, generate an event.
1599 (SW_NOTE: SW_CONTROL_SCRIPT_BEGINS)
 - 1600 — When the execution phase begins the key operation on a fileset (such as
loading or removing), generate an event.
(SW_NOTE: SW_FILESET_BEGINS)

- 1601 — When the execution phase begins the key operation on a file (such as load-
1602 ing or removing), generate an event.
1603 (SW_NOTE: SW_FILE_BEGINS)
- 1604 — If at any time there is an error rebuilding any catalog files, generate an
1605 event.
1606 (SW_ERROR: SW_DATABASE_UPDATE_ERROR)
- 1607 — If a fileset has an error for which there is not a more specific event defined,
1608 generate the generic event.
1609 (SW_ERROR: SW_FILESET_ERROR)
- 1610 — If a fileset has a warning for which there is not a more specific event
1611 defined, generate the generic event.
1612 (SW_WARNING: SW_FILESET_WARNING)
- 1613 — For **swinstall** and **swcopy** from a distribution that spans multiple
1614 media, an implementation may generate the following events to convey
1615 needed media change information. An implementation may, but need not,
1616 provide such support for other utilities.
1617 (SW_ERROR: SW_WRONG_MEDIA_SET)
1618 (SW_NOTE: SW_NEED_MEDIA_CHANGE)
1619 (SW_NOTE: SW_CURRENT_MEDIA)
- 1620 — An implementation may generate the following events for software that will
1621 not be executed due to analysis results for that software.
1622 (SW_NOTE: SW_SELECTION_SKIPPED_IN_ANALYSIS)
1623 (SW_ERROR: SW_SELECTION_NOT_ANALYZED)
- 1624 See each utility section for the execution operations specific to each utility. When
1625 the execution phase completes, generate an event.
1626 (SW_NOTE: SW_EXECUTION_ENDS)

1627 4.1.7.3.1 Fileset State Transitions

1628 A conforming implementation shall maintain the *state* attribute of each fileset to
1629 identify the condition and validity of that package. A conforming implementation
1630 shall use these and only these states as valid values of the *fileset.state* attribute.

1631	configured	Indicates the fileset in an <code>installed_software</code> object has 1632 been configured.
1633		This state applies to filesets in <code>installed_software</code> 1634 objects.
1635	installed	Indicates that the specified fileset has been installed 1636 successfully.
1637	corrupt	Indicates that the most recent attempt to handle the 1638 fileset was not successful and any recovery actions that 1639 were attempted were similarly unsuccessful.
1640		Software can transition from this state via the swin- 1641 stall , swcopy , or swremove utilities. Other 1642 implementation-defined methods may also exist for transitioning from this state.

1643		This state applies to filesets in distributions and
1644		installed_software objects.
1645	removed	This state indicates that the files for the fileset has been
1646		removed but the information remains.
1647		The default behavior when removing software with
1648		swremove is to also remove its information (metadata)
1649		from the catalog. An implementation may define a
1650		means for removing software, while maintaining the
1651		catalog information. The catalog information can be
1652		separately removed with the swmodify utility.
1653		This state applies to filesets in distributions and
1654		installed_software objects.
1655	available	Indicates the fileset is present in the distribution and
1656		may be operated on (copied, installed, etc.) using the
1657		appropriate utilities.
1658		This state applies to filesets in a distribution.
1659	transient	Indicates that the fileset is currently being acted upon
1660		by one of the utilities that modify software files, thus
1661		the state of the software is not well defined.
1662		This state should be replaced by another before the util-
1663		ity completes. The presence of this state in the
1664		software_collection when no utility is running indicates
1665		that a utility was previously interrupted (power failure,
1666		kill, etc.) and was not able to record a final state indica-
1667		tion into the software_collection catalog. In such case,
1668		the implementation-defined recovery procedures can be
1669		used to restore the product to a another state.
1670		This state applies to filesets in distributions and
1671		installed_software objects.

1672 4.1.8 Exit Status

1673 The utility shall return one of the following exit codes:

1674 **Table 4-8 – Return Codes**

Return Code	Definition
0	The utility was successful on all targets
1	The utility failed on all targets
2	The utility failed on some targets

1680 The exit status for the **swpackage** utility is different since it is not a distributed
1681 utility.

1682 4.1.9 Consequences of Errors

1683 Utilities can operate on multiple software objects contained in multiple targets.
1684 Whether an error impacts a particular software object, all software objects in the
1685 target, or all targets, is implementation defined. One exception to this is the
1686 minimum error recovery procedure described in 4.5.7.3.8 that describes fileset
1687 level recovery during install.

1688 4.1.9.1 Error Conditions

1689 The conditions leading to errors are described in 4.1.6.2 and 4.1.7.

1690 4.2 swask — Ask for user responses

1691 4.2.1 Synopsis

1692 **swask** [-c *catalog*][-f *file*][-s *source*][-t *targetfile*][-x *option=value*]
1693 [-X *options_file*][*software_selections*][@*targets*]

1694 4.2.2 Description

1695 The **swask** utility runs the interactive software **request** scripts for the software
1696 objects selected. These scripts store the responses to the **response** files for later
1697 use by the **swinstall** and **swconfig** utilities.

1698 The **swinstall** and **swconfig** can also run the interactive **request** scripts
1699 directly.

1700 4.2.3 Options

1701 The **swask** utility supports the following options. Where there is no description,
1702 the description in 4.1 applies.

1703 -c *catalog*

1704 Specifies the pathname to an exported catalog structure, below which
1705 the **response** files created by the **request** scripts are stored for
1706 later use.

1707 If the -c *catalog* option is omitted, the utility shall use the source
1708 catalog, <*distribution.path*>/**catalog**, as the directory in which to
1709 store the **response** files. Hence, these **response** files will be the
1710 ones available for use by the control scripts executed by **swinstall**.

1711 -f *file*

1712 -s *source*

- 1713 **-t** *targetfile*
- 1714 **-x** *option=value*
- 1715 **-X** *options_file*

1716 **4.2.4 Operands**

1717 The **swask** utility supports the *software_selections* operand described in 4.1.
1718 Specifying values for the *targets* operand does not imply operations on the target
1719 role. The values are simply written as a list of targets for the *targets* option in the
1720 file made available to the **request** script via the **SW_SESSION_OPTIONS**
1721 environment variable. See 4.1.6.1.

1722 **4.2.5 External Influences**

1723 See 4.1 for descriptions of external influences common to all utilities.

1724 **4.2.5.1 Extended Options**

1725 The **swask** utility supports the following extended options. The description in 4.1
1726 applies.

- 1727 *autoselect_dependencies = true*
- 1728 *distribution_source_directory = implementation_defined_value*
- 1729 *distribution_source_serial = implementation_defined_value*
- 1730 *logfile = implementation_defined_value*
- 1731 *loglevel = 1*
- 1732 *ask = true*
- 1733 *software*
- 1734 *verbose = 1*

1735 **4.2.6 External Effects**

1736 See 4.1.

1737 **4.2.7 Extended Description**

1738 See 4.1 for general information. There are two phases in the **swask** utility, as fol-
1739 lows:

- 1740 (1) Selection phase
- 1741 (2) Execution phase

1742 4.2.7.1 Selection Phase

1743 The software selections apply to the software available from the distribution
1744 source if the **-s** option was specified. Otherwise, the software selections apply to
1745 software that has been already installed on the targets. Each specified selection is
1746 added to the selection list after it passes the following checks:

1747 — If the **-s** option is specified and the selection is not available from the
1748 source, generate an event. If the **-S** option is not specified and the selection
1749 is not available from the target, generate an event.
1750 (SW_ERROR: SW_SELECTION_NOT_FOUND)

1751 — If a unique version can not be identified, generate an event.
1752 (SW_ERROR: SW_SELECTION_NOT_FOUND_AMBIG)

1753 Add any dependencies to the selection list if *autoselect_dependencies* = *true*.

1754 4.2.7.2 Execution Phase

1755 For each selected software that has a **request** control script:

1756 — If it does not already exist, build the necessary control directories of the
1757 exported catalog structure to hold the **response** file for that software
1758 object.

1759 NOTE: If there are multiple product versions selected, they shall have separate control
1760 directories as specified in the software packaging layout. Update the global **INDEX** in the
1761 exported catalog structure so the appropriate version can be identified later.

1762 — Set the value of the **SW_CONTROL_DIRECTORY** environment variable
1763 to the directory below which the **request** script writes the **response** file.
1764 It may be set to the control directory where the **response** file will eventu-
1765 ally be held, or it may be set to another, temporary, directory.

1766 If a response file can be found from one of the following sources, searched in
1767 the order specified, then the implementation shall ensure that the response
1768 file is contained within the directory pointed to by
1769 **SW_CONTROL_DIRECTORY**. The means for doing this (e.g., copy, link,
1770 symlink) is undefined.

1771 (1) If **-c** *catalog* was specified, any **response** file already existing below
1772 that catalog

1773 (2) If **-s** was not specified, any **response** file from the catalog of the tar-
1774 get or targets specified

1775 (3) If **-s** was specified, any **response** file already existing in the source
1776 catalog

1777 — If *ask* = *true*, execute the **request** script. If *ask* = *as_needed*, execute the
1778 **request** script only if a **response** file does not already exist in the control
1779 directory. The **request** script shall write a single **response** file in the
1780 control directory defined by the supplied environment variable
1781 **SW_CONTROL_DIRECTORY**.

1782 NOTE: The **swinstall** and **swconfig** utilities will distribute the **response** file to the
targets for use by that targets control scripts.

- 1783 — If a **request** script returns an error and *enforce_scripts* = *true*, generate
- 1784 an event and invoke the implementation-defined error handling pro-
- 1785 cedures.
- 1786 (SW_ERROR: SW_REQUEST_ERROR)
- 1787 — If a **request** script returns an error and *enforce_scripts* = *false*, gen-
- 1788 erate an event.
- 1789 (SW_WARNING: SW_REQUEST_ERROR)
- 1790 — If a **request** script returns a warning, generate an event.
- 1791 (SW_WARNING: SW_REQUEST_WARNING)

1792 4.2.8 Exit Status

1793 See 4.1.

1794 4.2.9 Consequences of Errors

1795 See 4.1.

1796 4.3 swconfig — Configure software

1797 4.3.1 Synopsis

1798 **swconfig** [-p][-u][-c *catalog*][-f *file*][-t *targetfile*][-x *option=value*]
 1799 [-X *options_file*][*software_selections*][@ *targets*]

1800 4.3.2 Description

1801 The **swconfig** command configures, unconfigures, and reconfigures installed
 1802 software on the target *hosts* specified on the command line for execution on those
 1803 hosts.

1804 Configuration primarily involves executing vendor-supplied **configure** and
 1805 **unconfigure** scripts. These scripts configure or unconfigure the installed
 1806 software. They are only executed on the target hosts that are intended to actually
 1807 run the software. Software can be configured more than once by rerunning the
 1808 **configure** scripts.

1809 Configuration can also be done as part of the **swinstall** and **swremove** utilities.

1810 4.3.3 Options

1811 The **swconfig** utility supports the following options. Where there is no descrip-
 1812 tion, the description in 4.1 applies.

1813 **-c** *catalog*
 1814 If this option is specified, then use the exported catalog structure at
 1815 this path as the source of the **response** files.

 1816 If *ask=true* or *ask=as_needed*, the control directories in the exported
 1817 catalog structure are used for both the eventual source of the
 1818 **response** files, and the control directory where the **request**
 1819 scripts are executed in order to create any needed **response** files.

 1820 **-f** *file*
 1821 **-p**
 1822 **-t** *targetfile*
 1823 **-u** Undo configuration

 1824 This option tells the **swconfig** utility to unconfigure the software,
 1825 instead of configuring it.

 1826 **-x** *option=value*
 1827 **-X** *options_file*

1828 **4.3.4 Operands**

1829 The **swconfig** utility supports the *software_selections* and *targets* operands
 1830 described in 4.1.

1831 **4.3.5 External Influences**

1832 See 4.1 for descriptions of external influences common to all utilities.

1833 **4.3.5.1 Extended Options**

1834 The **swconfig** utility supports the following extended options. The description in
 1835 4.1 applies.

1836 *allow_incompatible=false*
 1837 *allow_multiple_versions=false*
 1838 *ask=false*
 1839 *autoselect_dependencies=true*
 1840 *autoselect_dependents=false*
 1841 *enforce_dependencies=true*
 1842 *installed_software_catalog=implementation_defined_value*
 1843 *logfile=implementation_defined_value*
 1844 *loglevel=1*
 1845 *reconfigure=false*

1846 `select_local = true`
 1847 `software`
 1848 `targets`
 1849 `verbose = 1`

1850 **4.3.6 External Effects**

1851 See 4.1.

1852 **4.3.7 Extended Description**

1853 See 4.1 for general information. There are three key phases in the **swconfig**
 1854 utility:

- 1855 (1) Selection phase
- 1856 (2) Analysis phase
- 1857 (3) Execution phase

1858 **4.3.7.1 Selection Phase**

1859 Software selections apply to the software installed on the target. Each specified
 1860 selection is added to the selection list after it passes the following checks:

- 1861 — If the selection is not found, generate an event.
 1862 (SW_WARNING: SW_SELECTION_NOT_FOUND)
- 1863 — If the selection is not found at that product location but it does exist in
 1864 another location, generate an event.
 1865 (SW_WARNING: SW_SELECTION_NOT_FOUND_RELATED)

1866 Add any dependencies to the selection list if *autoselect_dependencies = true* and the
 1867 task is **configure**. Add any dependents to the selection list if the
 1868 *autoselect_dependents = true* and the task is **unconfigure**.

1869 If *ask = true* then execute the **request** scripts for the selected software as
 1870 described in the **swask** utility. See 4.2.7.

1871 **4.3.7.2 Analysis Phase**

1872 The following checks are made:

- 1873 — If configuring, check for compatibility of selections and, if the software is
 1874 not compatible, generate an event as follows. See 4.1.4.1.2.
- 1875 — If *allow_incompatible = true*, generate an event.
 1876 (SW_WARNING: SW_NOT_COMPATIBLE)
- 1877 — If *allow_incompatible = false*, generate an event.
 1878 (SW_ERROR: SW_NOT_COMPATIBLE)
- If configuring and the state is **corrupt** or **transient**, generate an event.
 (SW_ERROR: SW_SELECTION_IS_CORRUPT)

- 1879 — If the state is already configured when configuring, or not configured when
1880 unconfiguring, generate an event. This event is independent of whether the
1881 software will be reconfigured or not, which in turn is controlled by the
1882 *reconfigure* option.
1883 (SW_NOTE: SW_ALREADY_CONFIGURED)
- 1884 — If configuring this software will create a new configured version of a fileset
1885 (see 2.2.2.109,) then generate an event as follows.
 - 1886 — If *allow_multiple_versions*=*true*, generate an event.
1887 (SW_WARNING: SW_NEW_MULTIPLE_VERSION)
 - 1888 — If *allow_multiple_versions*=*false*, generate an event.
1889 (SW_ERROR: SW_NEW_MULTIPLE_VERSION)
- 1890 — If the dependencies are not in the **configured** state and have not been
1891 autoselected to be configured, generate an event as follows.
 - 1892 — If *enforce_dependencies*=*false*, generate an event.
1893 (SW_WARNING: SW_DEPENDENCY_NOT_MET)
 - 1894 — If *enforce_dependencies*=*true*, generate an event.
1895 (SW_ERROR: SW_DEPENDENCY_NOT_MET)

1896 4.3.7.3 Execution Phase

1897 The sequential relationship between the configure operations is shown in the fol-
1898 lowing list. Products are ordered by prerequisite dependencies if any. Fileset
1899 operations are also ordered by any prerequisites.

- 1900 (1) Configure each product.
 - 1901 (a) Run **configure** script for the product.
 - 1902 (b) Configure each fileset in the product.
 - 1903 [1] Run the **configure** script for the fileset.
 - 1904 [2] Update the result of the script (*control_file*). Update the state
1905 of the fileset to **configured** in the database for the
1906 *installed_software* object.

1907 If there is no **configure** script for a fileset the state of the fileset is still changed
1908 as specified above.

1909 4.3.7.3.1 Unconfigure Operations

1910 For unconfigure, the order of the products and filesets within the products is the
1911 reverse of the order of products and filesets for configure. The operations are:

- 1912 (1) Unconfigure each product.
 - 1913 (a) Unconfigure each fileset in the product.
 - 1914 [1] Run the **unconfigure** script for the fileset.
 - 1915 [2] Update the result of the script. Update the state of the fileset
in the product to **installed** in the database for the
installed_software object.

1916 (b) Run the **unconfigure** script for the product.

1917 If there is no **unconfigure** script for a fileset the state of the fileset is still
1918 changed as specified above.

1919 4.3.7.3.2 Executing configure Scripts

1920 In this operation, **swconfig** executes vendor-supplied **configure** or **unconfig-**
1921 **ure** scripts.

1922 The **configure** scripts shall not be interactive; however, they may access the
1923 **response** file generated by an interactive **request** script. If any **response** file
1924 has been generated, it will exist in the directory pointed to by
1925 **SW_CONTROL_DIRECTORY**.

1926 — If a **configure** script returns an error and *enforce_scripts=true*, generate
1927 an event and invoke the implementation-defined error handling procedures.
1928 (SW_ERROR: SW_CONFIGURE_ERROR)

1929 — If a **configure** script returns an error and *enforce_scripts=false*, generate
1930 an event.
1931 (SW_WARNING: SW_CONFIGURE_ERROR)

1932 — If the **configure** script returns a warning, generate an event.
1933 (SW_WARNING: SW_CONFIGURE_WARNING)

1934 — If a **configure** script has a return code of 3, generate an event, and
1935 exclude the fileset (or all filesets within the product for a product level
1936 script) from any state change between **configured** and **installed**.
1937 (SW_NOTE: SW_CONFIGURE_EXCLUDE)

1938 4.3.8 Exit Status

1939 See 4.1.

1940 4.3.9 Consequences of Errors

1941 See 4.1.

1942 4.4 swcopy — Copy distribution

1943 4.4.1 Synopsis

1944 **swcopy** [-p] [-f *file*] [-s *source*] [-t *targetfile*] [-x *option=value*]
1945 [-X *options_file*] [*software_selections*] [@ *targets*]

1946 **4.4.2 Description**

1947 The **swcopy** utility copies or merges software from any source distribution to a
1948 target distribution on the local host, or to the *targets* specified on the command
1949 line. The distribution can then be accessed by the **swinstall** utility as a source.

1950 **4.4.3 Options**

1951 The **swcopy** utility supports the following options. Where there is no description,
1952 the description in 4.1 applies.

- 1953 **-f** *file*
- 1954 **-p**
- 1955 **-s** *source*
- 1956 **-t** *targetfile*
- 1957 **-x** *option=value*
- 1958 **-X** *options_file*

1959 **4.4.4 Operands**

1960 The **swcopy** utility supports the *software_selections* and *targets* operands
1961 described in 4.1.

1962 Whether data on an existing target distribution in serial format is overwritten or
1963 merged is implementation defined.

1964 **4.4.5 External Influences**

1965 See 4.1 for descriptions of external influences common to all utilities.

1966 **4.4.5.1 Extended Options**

1967 The **swcopy** utility supports the following extended options. The description in
1968 4.1 applies.

- 1969 *autoselect_dependencies* = *as_needed*
- 1970 *compress_files* = *false*
- 1971 *compression_type* = *implementation_defined_value*
- 1972 *distribution_source_directory* = *implementation_defined_value*
- 1973 *distribution_target_directory* = *implementation_defined_value*
- 1974 *enforce_dependencies* = *true*
- 1975 *enforce_dsa* = *true*
- 1976 *logfile* = *implementation_defined_value*

1977 *loglevel=1*
 1978 *recopy=false*
 1979 *select_local=true*
 1980 *software*
 1981 *targets*
 1982 *uncompress_files=false*
 1983 *verbose=1*

1984 **4.4.6 External Effects**

1985 See 4.1.

1986 **4.4.7 Extended Description**

1987 See 4.1 for general information. The following are the three key phases in the
 1988 copy utility:

- 1989 (1) Selection phase
- 1990 (2) Analysis phase
- 1991 (3) Execution phase

1992 **4.4.7.1 Selection Phase**

1993 If a software selection has dependency specifications on other software and
 1994 *autoselect_dependencies=true*, this software shall be automatically selected using
 1995 the method described in 4.1.4.1. The resulting selection shall be unambiguous to
 1996 be effective. This automatically selected software is then copied along with the
 1997 rest of the selected software. If *autoselect_dependencies=if_needed*, then
 1998 automatically selected software is only copied if the dependency is not already
 1999 met.

2000 The **swcopy** utility supports corequisite and prerequisite dependencies for
 2001 autoselection, but treats them equally (no ordering considerations like **swin-**
 2002 **stall**). The **swcopy** utility ignores exrequisites.

2003 For **swcopy** each selection added to the selected software list shall satisfy the fol-
 2004 lowing validation checks. If any of these checks result in an error, the selection
 2005 shall not be added to the list.

- 2006 — If the selection is not available from the source, generate an event.
 2007 (SW_ERROR: SW_SELECTION_NOT_FOUND)
- 2008 — If a unique version can not be identified, generate an event.
 2009 (SW_ERROR: SW_SELECTION_NOT_FOUND_AMBIG)

2010 No compatibility filtering is done for **swcopy**.

2011 4.4.7.2 Analysis Phase

2012 The target role uses the file size information obtained from the source to deter-
2013 mine whether or not the copy utility proceeds on the given target. If any target
2014 fails any of the analysis operations, the software attempted to be copied is deter-
2015 mined by the implementation-defined error handling procedures.

2016 The target role checks the following requirements:

- 2017 — If the target distribution does not exist on a host, create the path to the tar-
2018 get with default attributes and generate an event.
2019 (SW_NOTE: SW_SOC_CREATED)
- 2020 — Check that selected filesets are not the same version as already available.
2021 If they are the same version, generate an event as follows.
 - 2022 — If *recopy=false*, note that these filesets will be skipped by generating an
2023 event.
2024 (SW_NOTE: SW_SAME_REVISION_SKIPPED)
 - 2025 — If *recopy=true*, note that they will be recopied by generating an event.
2026 (SW_NOTE: SW_SAME_REVISION_INSTALLED)
- 2027 — Verify that the needed dependencies of the filesets are met. If the needed
2028 dependencies are not met, generate an event as follows.
 - 2029 — If *enforce_dependencies=true*, generate an event.
2030 (SW_ERROR: SW_DEPENDENCY_NOT_MET)
 - 2031 — If *enforce_dependencies=false*, generate an event.
2032 (SW_WARNING: SW_DEPENDENCY_NOT_MET)
- 2033 — Check that there is enough free disk space on the target file system to copy
2034 the selected products.
 - 2035 — If there is not enough disk space and *enforce_dsa=true*, generate an
2036 event.
2037 (SW_ERROR: SW_DSA_OVER_LIMIT)
 - 2038 — If there is not enough disk space and *enforce_dsa=false*, generate an
2039 event.
2040 (SW_WARNING: SW_DSA_OVER_LIMIT)

2041 How disk space analysis is implemented is undefined. However, an implementa-
2042 tion shall account at least for the sizes of the files and control_files being copied.

2043 4.4.7.3 Execution Phase

2044 As the result of a **swcopy**, products and bundles, if specified, are copied to the tar-
2045 get, which is a distribution software object.

2046 When creating serial distributions, an implementation shall support one or both of
2047 the POSIX.1 {2} extended **cpio** or extended **tar** archive formats. Whether an
2048 implementation supports writing both archive formats or only one, and which of
2049 the formats is supported if only one, is implementation defined.

The relationship between the fileset loading and state transitions for **swcopy** is shown in the following list.

- 2050 (1) Copy each product.
- 2051 (a) Create the distribution catalog information for the product and its
- 2052 contained subproducts.
- 2053 (b) Copy each fileset in the product.
- 2054 [1] Create the distribution catalog information for the fileset, set-
- 2055 ting the state to **transient**.
- 2056 [2] Load the files for the fileset.
- 2057 [3] Update the state of the fileset to **available**.
- 2058 (2) Copy each bundle.
- 2059 (a) Create the distribution catalog information for the bundle.

2060 4.4.7.3.1 File Loading

2061 In this step, **swcopy** copies the files from the source onto the target.

2062 If a file load fails for any other reason such as a lost connection to the remote

2063 source or tape eject, then the fileset load fails. The following are the errors that

2064 can occur during the file loading step:

- 2065 — If an error or warning occurs while loading a file onto a target, an event is
- 2066 generated and, for errors, the implementation-defined error handling pro-
- 2067 cedures invoked.
- 2068 — Whether remote files are loaded is implementation defined. If the file is on
- 2069 a remote file system, generate an event as follows.
- 2070 — If the file was loaded, generate an event.
- 2071 (SW_NOTE: SW_FILE_IS_REMOTE)
- 2072 — If the file was not loaded, generate an event.
- 2073 (SW_WARNING: SW_FILE_IS_REMOTE)
- 2074 — If the error was a source access problem, the user may attempt to correct
- 2075 the problem and start over with the fileset that had the failure.
- 2076 (SW_ERROR: SW_SOURCE_ACCESS_ERROR)

2077 4.4.7.3.2 Compression

2078 If *compress_files=true* is specified, then the files shall be compressed as follows in

2079 copying to the target distribution:

- 2080 — **INDEX** and **INFO** files shall not be compressed.
- 2081 — All files that have the *compression_state* attribute undefined or its value set
- 2082 to **uncompressed** shall be compressed. If the file cannot be compressed,
- 2083 generate an event.
- 2084 (SW_ERROR: SW_COMPRESSION_FAILURE)
- 2085 — All files that have the value of the *compression_state* attribute set to
- 2086 **not_compressible** shall be copied as is.
- A source file that is already compressed, and has the value of its
- compression_type* attribute equal to the value of the *compression_type*
- extended option, shall be copied as is.

2087 — If a source file is already compressed, and the value of its *compression_type*
2088 attribute is different than the value of the *compression_type* extended
2089 option, the behavior is undefined. Unless the implementation can success-
2090 fully uncompress the file and then compress it with the correct type, gen-
2091 erate an event.
2092 (SW_ERROR: SW_COMPRESSION_FAILURE)

2093 If *uncompress_files=true*, the files shall be uncompressed as follows in copying to
2094 the target distribution:

2095 — All files with a *compression_state* attribute value of **compressed** shall be
2096 uncompressed as part of the copy. If the file cannot be uncompressed, gen-
2097 erate an event.
2098 (SW_ERROR: SW_COMPRESSION_FAILURE)

2099 — All other files shall be copied as is.

2100 If neither *compress_files=true* nor *uncompress_files=true*, then the files shall be
2101 copied as is. Behavior when both are set to true is undefined.

2102 4.4.7.3.3 Copying Into Existing Products

2103 When a fileset is copied into an existing target product, the attributes of this exist-
2104 ing product may be affected as follows. If an attribute exists in the product from
2105 which the fileset came, the value of this attribute is set in the target product. Any
2106 attributes in the target product not found in the product from which the fileset
2107 came are left unaltered. It is possible for attributes to be set multiple times as
2108 filesets from different products are copied into the target product.

2109 4.4.8 Exit Status

2110 See 4.1.

2111 4.4.9 Consequences of Errors

2112 See 4.1.

2113 4.5 swinstall — Install software

2114 4.5.1 Synopsis

2115 **swinstall** [-p][-r][-c *catalog*][-f *file*][-s *source*][-t *targetfile*]
2116 [-x *option=value*][-X *options_files*][*software_selections*][@ *tar-*
2117 *gets*]

2118 4.5.2 Description

2119 The **swinstall** utility installs software from a distribution to installed_software
2120 objects on the targets. It may also configure the software. The software is not
2121 necessarily available for use until after it has been configured.

2122 4.5.3 Options

2123 The **swinstall** utility supports the following options. Where there is no descrip-
2124 tion, the description in 4.1 applies.

2125 **-c** *catalog*

2126 If this option is specified, then use the exported catalog structure at
2127 this path as the source of the **response** files.

2128 If *ask=true* or *ask=as_needed*, the control directories in the exported
2129 catalog structure are used for both the eventual source of the
2130 **response** files, and the control directory where the the **request**
2131 scripts are executed in order to create any needed **response** files.

2132 **-f** *file*

2133 **-p**

2134 **-r**

2135 **-s** *source*

2136 **-t** *targetfile*

2137 **-x** *option=value*

2138 **-X** *options_file*

2139 4.5.4 Operands

2140 The **swinstall** utility supports the *software_selections* and *targets* operands
2141 described in 4.1.

2142 The utility supports software selection operands with [**l=location**] part of the
2143 syntax, designating the *product.location* directory that replaces the
2144 *product.directory* attribute when installing the software.

2145 4.5.5 External Influences

2146 See 4.1 for descriptions of external influences common to all utilities.

2147 4.5.5.1 Extended Options

2148 The **swinstall** utility supports the following extended options. The description
2149 in 4.1 applies.

2150 *allow_downdate=false*

2151 *allow_incompatible=false*
2152 *ask=false*
2153 *autoreboot=false*
2154 *autorecover=false*
2155 *autoselect_dependencies=as_needed*
2156 *defer_configure=false*
2157 *distribution_source_directory=implementation_defined_value*
2158 *enforce_dependencies=true*
2159 *enforce_locatable=true*
2160 *enforce_scripts=true*
2161 *enforce_dsa=true*
2162 *installed_software_catalog=implementation_defined_value*
2163 *logfile=implementation_defined_value*
2164 *loglevel=1*
2165 *reinstall=false*
2166 *select_local=true*
2167 *software*
2168 *targets*
2169 *verbose=1*

2170 **4.5.6 External Effects**

2171 See 4.1.

2172 **4.5.7 Extended Description**

2173 See 4.1. for general information. The following are the three key phases in the
2174 **swinstall** utility:

- 2175 (1) Selection phase
- 2176 (2) Analysis phase
- 2177 (3) Execution phase

2178 **4.5.7.1 Selection Phase**

2179 Multiple versions of a software product can exist from the source, distinguished by
2180 their respective “version distinguishing attributes” (*revision*, *architecture*, and
2181 *vendor_tag*). If the method described in 4.1.4.1 results in an ambiguous selection,
2182 the following method is used to identify a single version:

- 2183 — If *allow_incompatible=false*, the target **uname** attributes are used to filter
- 2184 the available products to only those that are compatible with the target sys-
- 2185 tems, then the version with the highest possible *product.revision* is chosen
- 2186 from this filtered list. If this filtering and selection of a highest revision
- 2187 does not result in a unique version, then no version is selected. If
- 2188 *allow_incompatible=true*, then only the highest revision is used to try to
- 2189 determine a unique version. In either case, if there is still an ambiguous
- 2190 selection, no version is selected. See 4.1.4.1.2.

- 2191 If a software selection has dependency specifications on other software, and the
- 2192 option *autoselect_dependencies=true*, the dependency software is attempted to be
- 2193 automatically selected using the same method to determine a single version. This
- 2194 automatically selected software is then installed along with the rest of the selected
- 2195 software. If *autoselect_dependencies=as_needed*, then dependency software is
- 2196 attempted to be automatically selected and installed only if the dependency is not
- 2197 already met on the target.

- 2198 If a fileset has an exerequisite on another software object, and that other software
- 2199 object is part of the specified software selection, either explicitly or as part of
- 2200 another selection, then the fileset is excluded. If two filesets have exerequisites on
- 2201 each other, then the behavior is implementation defined.

- 2202 For **swinstall** each selection added to the selected software list shall satisfy the
- 2203 following validation checks. If any of these checks result in an event with a status
- 2204 of **SW_ERROR**, the selection is not added to the list and the implementation-
- 2205 defined handling procedure can be invoked.

- 2206 — If the selection is not available from the source, generate an event.
- 2207 (SW_ERROR: SW_SELECTION_NOT_FOUND)

- 2208 — If a unique version can not be identified, generate an event.
- 2209 (SW_ERROR: SW_SELECTION_NOT_FOUND_AMBIG)

- 2210 — If an attempt is made to select more than one version of a given product
- 2211 targeted for the same location, generate an event.
- 2212 (SW_ERROR: SW_EXISTING_MULTIPLE_VERSION)

- 2213 — If *allow_incompatible=true*, then for each target where the software
- 2214 selected is incompatible with that target (see 4.1.4.1.2), generate an event.
- 2215 (SW_WARNING: SW_NOT_COMPATIBLE)

- 2216 — If *allow_incompatible=false*, then for each target where the software
- 2217 selected is incompatible with that target (see 4.1.4.1.2), generate an event.
- 2218 The implementation-defined error handling procedure shall be invoked.
- 2219 (SW_ERROR: SW_NOT_COMPATIBLE)

- 2220 — Check if a non-default product location has been specified and, if so, gen-
- 2221 erate an event as follows.

- 2222 — If *enforce_locatable=true*, generate an event.
- 2223 (SW_ERROR: SW_NOT_LOCATABLE)

- 2224 — If *enforce_locatable=false*, generate an event.
- 2225 (SW_WARNING: SW_NOT_LOCATABLE)

- If the software is excluded, generate an event.
- (SW_NOTE: SW_EXREQUISITE_EXCLUDE)

2226 If *ask=true*, then execute the software **request** scripts for the selected software
2227 as described in the **swask** utility. See 4.2.7.

2228 4.5.7.2 Analysis Phase

2229 The target role uses the file size information and **checkinstall** scripts obtained
2230 from the source to determine whether or not the install utility proceeds on the
2231 given target. When failures occurs in the disk space analysis and **checkinstall**
2232 scripts, it is implementation defined whether or not to proceed with a partial list
2233 of software selections.

2234 If any target generates an event with a status of **SW_ERROR** during any of the
2235 analysis operations, the software attempted to be installed is determined by the
2236 implementation-defined error handling procedures.

2237 The target role checks the following requirements:

- 2238 — If the target `installed_software` object does not exist on a host, create the
2239 path to the target with default attributes, and generate an event.
2240 (**SW_NOTE**: **SW_SOC_CREATED**)
- 2241 — Check that selected filesets are not the same version as already installed.
2242 If any of the selected filesets have already been installed with the same ver-
2243 sion, generate an event as follows.
 - 2244 — If *reinstall=false*, note that they will be skipped by generating an event.
2245 (**SW_NOTE**: **SW_SAME_REVISION_SKIPPED**)
 - 2246 — If *reinstall=true*, note that they will be reinstalled by generating an
2247 event.
2248 (**SW_NOTE**: **SW_SAME_REVISION_INSTALLED**)
- 2249 — Check that selected filesets are not lower versions of the fileset already
2250 installed on the host. If the any of the selected filesets are already installed
2251 with a higher version, generate an event as follows.
 - 2252 — If *allow_downdate=false*, generate an event.
2253 (**SW_ERROR**: **SW_HIGHER_REVISION_INSTALLED**)
 - 2254 — If *allow_downdate=true*, generate an event.
2255 (**SW_WARNING**: **SW_HIGHER_REVISION_INSTALLED**)
- 2256 — Execute vendor-supplied **checkinstall** scripts to perform product-
2257 specific checks of the target.
 - 2258 — If a **checkinstall** script returns an error, and *enforce_scripts=true*,
2259 generate an event and invoke the implementation-defined error han-
2260 dling procedure.
2261 (**SW_ERROR**: **SW_CHECK_SCRIPT_ERROR**)
 - 2262 — If a **checkinstall** script returns an error and *enforce_scripts=false*,
2263 generate an event.
2264 (**SW_WARNING**: **SW_CHECK_SCRIPT_ERROR**)
 - 2265 — If a **checkinstall** script returns a warning, generate an event.
2266 (**SW_WARNING**: **SW_CHECK_SCRIPT_WARNING**)

- 2267 — If a **checkinstall** script has a return code of 3, generate an event and
 2268 unselect the fileset (or all filesets in the product for a product level
 2269 script).
 2270 (SW_NOTE: SW_CHECK_SCRIPT_EXCLUDE)
- 2271 — Verify that the needed dependencies of the filesets are met. If any of the
 2272 dependencies are not met, generate an event as follows.
- 2273 — If *enforce_dependencies = true*, generate an event.
 2274 (SW_ERROR: SW_DEPENDENCY_NOT_MET)
- 2275 — If *enforce_dependencies = false*, generate an event.
 2276 (SW_WARNING: SW_DEPENDENCY_NOT_MET)
- 2277 — Check that there is enough free disk space on the target file system to
 2278 install the selected products.
- 2279 — If there is not enough disk space and *enforce_dsa = true*, generate an
 2280 event.
 2281 (SW_ERROR: SW_DSA_OVER_LIMIT)
- 2282 — If there is not enough disk space and *enforce_dsa = false*, generate an
 2283 event.
 2284 (SW_WARNING: SW_DSA_OVER_LIMIT)
- 2285 — An implementation may generate an event, as follows, if disk space analysis
 2286 encountered any problems that prevented the analysis.
- 2287 — If *enforce_dsa = true*, generate an event.
 2288 (SW_ERROR: SW_DSA_FAILED_TO_RUN)
- 2289 — If *enforce_dsa = false*, generate an event.
 2290 (SW_WARNING: SW_DSA_FAILED_TO_RUN)
- 2291 How disk space analysis is implemented is undefined. However an implementa-
 2292 tion shall account at least for the sizes of the files and control_files being installed,
 2293 the additional sizes from the vendor-supplied **space** file described in 5.2, and any
 2294 additional space required by the implementation-defined recovery process for sav-
 2295 ing files when *autorecover = true*.

2296 4.5.7.3 Execution Phase

2297 The execution phase is the third part of the installation process, and is entered
 2298 once either the selections have passed the analysis phase with no events with a
 2299 status of **SW_ERROR** or if permitted by the implementation-defined error handling
 2300 procedures.

2301 The relationship between the **preinstall** and **postinstall** scripts, fileset
 2302 loading, and state transitions for **swinstall** is shown in the following list. Pro-
 2303 ducts are ordered by prerequisite dependencies if any. Fileset operations are also
 2304 ordered by any prerequisites.

- 2305 (1) Install each product.
- 2306 (a) Create the installed_software catalog information for the product
 2307 and its contained subproducts.

- 2308 (b) Run the **preinstall** script for the product.
- 2309 (c) Install each fileset in the product.
- 2310 [1] Create the `installed_software` catalog information for the
- 2311 fileset, setting the state to **transient**. Also update the state
- 2312 of any existing fileset that is being updated or downdated to
- 2313 **transient**.
- 2314 [2] Run the **preinstall** script for the fileset.
- 2315 [3] Load the files for the fileset.
- 2316 [4] Run the **postinstall** script for the fileset.
- 2317 [5] Update the results of the scripts. Update the state of the
- 2318 fileset to **installed**. Also set the state of any existing fileset
- 2319 that is being updated or downdated to **removed** or remove the
- 2320 catalog information for that fileset.
- 2321 (d) Run the **postinstall** script for the product.
- 2322 (2) Install each bundle.
- 2323 (a) Create the `installed_software` catalog information for the bundle.
- 2324 (3) Configure each product in accordance with 4.3.7.3.2.
- 2325 Configuration shall be done at this point by the **swinstall** utility only if
- 2326 *defer_configure=false*, the target directory is `/`, and no filesets with the
- 2327 *is_reboot* attribute equal to true have been installed.
- 2328 (a) Run the **configure** script for the product.
- 2329 (b) Configure each fileset in the product.
- 2330 [1] Run the **configure** script for the fileset.
- 2331 [2] Update the result of the script. Update the state of the fileset
- 2332 to **configured** in the catalog for the `installed_software`
- 2333 object.
- 2334 Configuration shall not be executed by **swinstall** if the software creates a multi-
- 2335 ple version, the target directory is not `/`, or if the software is incompatible and
- 2336 *allow_incompatible=false* (see 4.1.4.1.2). In these cases, **swconfig** may be used.
- 2337 If events with a status of **SW_ERROR** are detected during the execution phase, the
- 2338 **swinstall** utility generates the appropriate event, any log entries, and invokes
- 2339 the implementation-defined error handling procedures. For each fileset that
- 2340 failed, the `installed_software` catalog is updated to the state **corrupt**.

2341 4.5.7.3.1 File Location

2342 If an alternate root directory was specified (a value for *installed_software.path*

2343 other than `/`), then the alternate root directory is used as a prefix to the *file.path*

2344 attribute to determine the file location in the file system. See 4.1.4.2.

2345 The *file.path* shall be modified if the product is locatable and a new

2346 *product.location* is specified (using the *l=location* software specification). The

product.directory part of the *file.path* is replaced by the value *product.location*

attribute before a file is placed in the target file system.

2347 NOTE: If a product is locatable (has the *product.is_locatable* attribute set to **true**), all files that
 2348 have the value of *product.directory* as the initial part of their path shall be installed to a new loca-
 2349 tion if one has been specified. The *product.directory* attribute is the base directory for the files that
 2350 are locatable within a specific product.

2351 If a *bundle.location* is specified (using the **l=location** software specification
 2352 when specifying a bundle), then the *bundle.location* shall be prefixed to the loca-
 2353 tion specification for each **software_spec** in the contents of the bundle prior to
 2354 replacement of the *product.directory* part of the *file.path*.

2355 4.5.7.3.2 Preinstall Scripts

2356 In this step of the execution phase, **swinstall** executes product and fileset
 2357 **preinstall** scripts.

2358 — If a **preinstall** script returns an error and *enforce_scripts=true*, gen-
 2359 erate an event and invoke the implementation-defined error handling pro-
 2360 cedures.

2361 (SW_ERROR: SW_PRE_SCRIPT_ERROR)

2362 — If a **preinstall** script returns an error and *enforce_scripts=false*, gen-
 2363 erate an event.

2364 (SW_WARNING: SW_PRE_SCRIPT_ERROR)

2365 — If a **preinstall** script returns a warning, generate an event.

2366 (SW_WARNING: SW_PRE_SCRIPT_WARNING)

2367 Control scripts shall adhere to the specifications in 4.1.6.1.

2368 4.5.7.3.3 File Loading

2369 In this step, **swinstall** loads the files from the source onto the target file system
 2370 according to information obtained from the source distribution. All file types are
 2371 created using the attributes defined for those files in the source distribution. Reg-
 2372 ular files (i.e., those with a *file.type* of **f**) are loaded using the content from the
 2373 source distribution.

2374 If the source file is a regular file or a directory and its path already exists on the
 2375 target file system as a symbolic link, then the symbolic link is followed and the file
 2376 is stored in the path defined by the symbolic link.

2377 If the source file is a symbolic link, then the existing path is replaced by symbolic
 2378 link.

2379 — If there are too many levels of symbolic links, then the file is skipped and
 2380 an event is generated.

2381 (SW_WARNING: SW_FILE_WARNING)

2382 NOTE: It is not the intention of this part of ISO/IEC 15068 to define symbolic links in a
 2383 manner inconsistent with POSIX.1 {2}. However, no approved POSIX standard currently
 2384 contains symbolic links. This definition is a placeholder until such time as an approved
 2385 standard provides the definition. See {B21}.

2386 The file owner and group names are set to the values specified for the *file.owner*
 2387 and *file.gid* attributes for the source file. If the target host does not contain those
 file owner and group names, the file uid and gid shall set to the numeric values
 specified for these attributes for the source file. If no values are specified for these
 attributes, the uid and gid shall be set to the effective uid and gid of the current

2388 process. See 5.2.13.3.

2389 — If the user or group of the file is not defined on the target host, or either of
2390 these attributes are not defined for the file, generate an event.
2391 (SW_WARNING: SW_FILE_WARNING)

2392 — If the *mode* attribute of the file has the set user id on execution (**S_ISUID**)
2393 bit set and either the *user* attribute of the file is not defined on the target
2394 host or the *user* attribute is not specified for the file, the corresponding
2395 mode bit in the file system shall not be set when installing the file and an
2396 event shall be generated. See 5.6.1.2 of POSIX.1 {2}.
2397 (SW_ERROR: SW_FILE_ERROR)

2398 — If the *mode* attribute of the file has the set group id on execution
2399 (**S_ISGID**) bit set and either the *group* attribute of the file is not defined on
2400 the target host or the *group* attribute is not specified for the file, the
2401 corresponding mode bit in the file system shall not be set when installing
2402 the file and an event shall be generated. See 5.6.1.2 of POSIX.1 {2}.
2403 (SW_ERROR: SW_FILE_ERROR)

2404 The value of the *file.mode* attribute on the file is set to the value of the *file.mode*
2405 attribute for the source file. An exception is that directories that already exist are
2406 not modified. If no values are specified for this attribute, the mode shall be set to
2407 the default file creation mode for the current process.

2408 If there is an existing installed file that matches the values supplied in the distri-
2409 bution for the *path*, *cksum*, *date*, and *size* attributes, the file is not reloaded unless
2410 the user has specified that the fileset is being reinstalled.

2411 If a file load fails for any other reason such as a lost connection to the remote
2412 source or tape eject, then the fileset install fails.

2413 The following are problems that may occur during the file load step:

2414 — If a problem occurs while loading a file onto a target, an event is generated
2415 and, for events with a status of **SW_ERROR**, the implementation-defined
2416 error handling procedures invoked. If there are too many levels of symbolic
2417 links, generate an event.
2418 (SW_ERROR: SW_FILE_ERROR)

2419 — Whether remote files are installed is implementation defined. If the file is
2420 on a remote file system, generate an event as follows.

2421 — If the file was loaded, generate an event.
2422 (SW_NOTE: SW_FILE_IS_REMOTE)

2423 — If the file was not loaded, generate an event.
2424 (SW_WARNING: SW_FILE_IS_REMOTE)

2425 — If a file can not be updated because it is busy, or it is a directory, then move
2426 that file to an implementation-defined location and generate an event. How
2427 these files are eventually removed is also implementation defined.
2428 (SW_WARNING: SW_FILE_NOT_REMOVABLE)

2429 — If the source becomes inaccessible for any reason during the process of load-
2430 ing files, generate an event.
(SW_ERROR: SW_SOURCE_ACCESS_ERROR)

2431 If *autorecover*=*true*, then all files that are being updated shall be saved. It is
2432 implementation defined where these files are saved. The saved files for filesets in
2433 each product are removed in an implementation-defined manner at some point
2434 after that product completes the execution phase.

2435 4.5.7.3.4 Compression

2436 When installing files, all compressed files shall be uncompressed as follows as part
2437 of file loading:

- 2438 — All files that have the *compression_state* attribute value of **compressed**
2439 shall be uncompressed, according to the value of *compression_type* attri-
2440 bute. The way in which this is done is implementation defined. If the file
2441 cannot be uncompressed, generate an event.
2442 (SW_ERROR: SW_COMPRESSION_FAILURE)

2443 4.5.7.3.5 Postinstall Scripts

2444 In this step, **swinstall** executes the product and fileset **postinstall** scripts.

- 2445 — If a **postinstall** script returns an error and *enforce_scripts*=*true*, gen-
2446 erate an event and invoke the implementation-defined error handling pro-
2447 cedures.
2448 (SW_ERROR: SW_POST_SCRIPT_ERROR)
- 2449 — If a **postinstall** script returns an error and *enforce_scripts*=*false*, gen-
2450 erate an event.
2451 (SW_WARNING: SW_POST_SCRIPT_ERROR)
- 2452 — If a **postinstall** script returns a warning, generate an event.
2453 (SW_WARNING: SW_POST_SCRIPT_WARNING)

2454 4.5.7.3.6 Kernel Scripts

2455 Special customization and install steps are executed when processing kernel
2456 filesets. Kernel filesets are those for which the value of the *is_kernel* attribute is
2457 **true**, causing **swinstall** to modify the fileset load order and to invoke the *post-*
2458 *kernel* script. Apart from this, **swinstall** has no special functionality for instal-
2459 ling kernels.

2460 The *postkernel* scripts are those specified by the value of the *product.postkernel*
2461 attribute or by the implementation-defined default. The functions invoked by this
2462 *postkernel* script are implementation defined. Examples of use include rebuilding
2463 the kernel or moving a new default kernel into place.

2464 The *postkernel* script shall not be interactive, and shall issue all informative and
2465 error messages to stdout and stderr, which shall be redirected to the log file. In
2466 addition, the *postkernel* script shall provide a standard return value indicating
2467 success (0, meaning zero), error (1), or warning (2).

- 2468 — If the *postkernel* script had an error, invoke the implementation-defined
2469 error handling procedures and generate an event.
2470 (SW_ERROR: SW_POSTKERNEL_ERROR)

If the *postkernel* script had a warning, generate an event.
(SW_WARNING: SW_POSTKERNEL_WARNING)

2471 The kernel filesets are processed before the rest of the filesets. All products are
2472 first processed for their kernel filesets, and then all products are processed for
2473 their non-kernel filesets. The ordering of products and filesets, as follows, also
2474 adheres to prerequisites, just as normal filesets:

- 2475 (1) Install the kernel filesets for each product.
 - 2476 (a) Create the `installed_software` catalog information for the product.
 - 2477 (b) Run the **preinstall** script for the product.
 - 2478 (c) Install each kernel fileset in the product.
 - 2479 [1] Create the `installed_software` catalog information for the
 - 2480 fileset, setting the state to **transient**.
 - 2481 [2] Run the **preinstall** script for the kernel fileset.
 - 2482 [3] Load the files for the kernel fileset.
 - 2483 [4] Run the **postinstall** script for the kernel fileset.
 - 2484 [5] Update the results of the scripts. Update the state of the
 - 2485 fileset to **installed**.
 - 2486 (d) Run the **postinstall** script for the product.
- 2487 (2) Perform steps after installing kernel filesets by calling the zero or more
2488 scripts defined by the *product.postkernel* attributes of each product with
2489 a kernel fileset and the implementation-defined default *postkernel* script
2490 if a product does not define a *product.postkernel* attribute.
- 2491 (3) Install the rest of the filesets for each product as described in 4.5.7.3,
2492 omitting the kernel filesets already installed.
- 2493 (4) After all filesets have been installed, the implementation-defined reboot
2494 procedure is executed on the target host if a fileset with the *is_reboot*
2495 attribute set to true has been installed and the
2496 **SW_ROOT_DIRECTORY** is `/`, and *autoreboot=true*. If rebooting, the
2497 software is not configured. The products will be configured after the
2498 reboot in an implementation-defined manner using the **swconfig** utility.
- 2499 (5) If not rebooting, then configure each product as described in 4.5.7.3,
2500 (including both kernel and non-kernel filesets).

2501 4.5.7.3.7 Rebooting the System

2502 If this step is required, the target role executes the implementation-defined reboot
2503 procedure after all products have been installed. It is performed only when
2504 software is installed that requires a reboot as part of its installation (indicated by
2505 the *is_reboot* fileset attribute).

- 2506 — If the system fails to execute the reboot step, generate an event.
2507 (**SW_ERROR: SW_CANNOT_INITIATE_REBOOT**)

2508 **4.5.7.3.8 Recovery**

2509 Within the execution phase of a particular product (from the product preinstall
2510 step through the product postinstall step), if any **preinstall** script, file loading,
2511 or **postinstall** script fails for a fileset, that fileset shall be deemed to have
2512 failed during install. The failure of a product **postinstall** script shall be con-
2513 sidered the same as if all fileset **postinstall** scripts had failed.

2514 If such a failure occurs and *autorecover* = *false*, no recovery shall be provided for
2515 any filesets deemed to have failed during install, and the fileset *state* attribute of
2516 those filesets shall be set to **corrupt**. No further attempt shall be made to install
2517 such filesets during the current invocation of **swinstall**. Install can proceed on
2518 other filesets that did not fail during install.

2519 If an install failure occurs and *autorecover* = *true*, at least the following minimal
2520 error recovery shall be provided at the fileset level. Additional recovery behavior,
2521 such as recovering the whole product or all products, is implementation defined.
2522 Additionally, if *enforce_dependencies* = *true*, implementations should take into
2523 account other filesets in the product that have a dependency on that failed fileset.

2524 The recovery is initiated at the point of failure, recovering the affected filesets,
2525 then continuing from the point of failure to the remaining filesets.

2526 Recovery involves running **unpostinstall** scripts, restoring files, and running
2527 **unpreinstall** scripts. The relationship between these steps for each product is
2528 shown in the following list:

2529 NOTE: In general, the order used when *autorecover* = *true* is the same as that normally used for
2530 successful steps. The reverse order is used when recovery steps are being executed.

2531 (1) Create the installed_software catalog information for the product.

2532 (2) Run the **preinstall** script for the product.

2533 If the **preinstall** script fails, or if all filesets have failed, run the pro-
2534 duct **unpreinstall** script, remove the catalog information for the pro-
2535 duct, and go on to the next product.

2536 (3) Install each fileset in the product.

2537 (a) Create the installed_software catalog information for the fileset, set-
2538 ting the state of it and the fileset being updated to **transient**.

2539 (b) Run the **preinstall** script for the fileset.

2540 If the **preinstall** script fails, run the **unpreinstall** script for
2541 the fileset, remove the catalog information for the fileset, restore the
2542 state of the fileset being updated, and go on to the next fileset.

2543 (c) Load the files for the fileset.

2544 Before loading any files, save any existing files that will be overwrit-
2545 ten by a file being loaded from the fileset, and then load the files for
2546 the fileset. If the fileset loading fails, restore the saved files for the
2547 fileset, delete all loaded files for which there is no saved file, and
2548 perform the previously described recovery step for this fileset.

(d) Run the **postinstall** script for the fileset.

2549 If the **postinstall** script fails, run the **unpostinstall** script
2550 for the fileset, and perform steps (3b) and (3c) for this fileset.

2551 (e) Update the results of the scripts. Update the state of the fileset to
2552 **installed**.

2553 (4) Run the product **postinstall** script.

2554 If the product **postinstall** script fails, run the product **unpostin-**
2555 **stall** script, and perform each of the previously described recovery
2556 steps for each fileset.

2557 (5) This is the first point in the process where the saved files may be
2558 removed. Remove the catalog information for filesets that were updated,
2559 or set the state of those filesets to **removed**.

2560 **4.5.8 Exit Status**

2561 See 4.1.

2562 **4.5.9 Consequences of Errors**

2563 See 4.1.

2564 **4.6 swlist — List software catalog**

2565 **4.6.1 Synopsis**

2566 **swlist** [**-d** ||| **-r**] [**-v**] [**-a** *attribute*] [**-c** *catalog*] [**-f** *file*] [**-l** *level*]
2567 [**-t** *targetfile*] [**-x** *option=value*] [**-X** *options_file*]
2568 [*software_selections*] [**@** *targets*]

2569 **4.6.2 Description**

2570 The **swlist** utility displays information about software that has been installed on
2571 a system or is in a distribution.

2572 When combined with **swmodify** there is a complete read/write interface to the
2573 **installed_software** and **distribution catalog** information.

2574 **4.6.3 Options**

2575 The **swlist** utility supports the following options. Where there is no description,
2576 the description in 4.1 applies.

2577 **-a** *attribute*

2578 Specifies which attributes to list.

Multiple attributes may be listed by specifying multiple **-a** *attribute*
options. Only attributes that apply to each object listed are included

2579 for that object. When used with the **-v** option, the attributes are in
 2580 the software definition file format. When the **-v** option is not
 2581 specified, then the listing format is undefined (see *one_liner* extended
 2582 option).

2583 In addition to all attribute names defined in this part of ISO/IEC
 2584 15068, three additional items shall be supported by the **-a** *attribute*
 2585 option:

2586 *create_date*
 2587 If this value is specified, **swlist** shall return a
 2588 sequence of characters representing the date associated
 2589 with the *create_time* attribute.

2590 The format of this sequence of characters in the POSIX
 2591 locale shall be equivalent to the default **date** format
 2592 described in 4.15.6.1 of POSIX.2 {3}.
 2593 **date** "+%a %b %e %H:%M:%S %Z %Y"
 2594 The format for other locales is undefined.

2595 *mod_date*
 2596 If this value is specified, **swlist** shall return a
 2597 sequence of characters representing the date associated
 2598 with the *mod_time* attribute.

2599 The format of this sequence of characters in the POSIX
 2600 locale shall be equivalent to the default **date** format
 2601 described in 4.15.6.1 of POSIX.2 {3};
 2602 **date** "+%a %b %e %H:%M:%S %Z %Y"
 2603 The format for other locales is undefined.

2604 *software_spec*
 2605 If this value is specified, **swlist** shall return the fully
 2606 qualified **software_spec** for the object, as defined in
 2607 4.1.4.1.1, instead of listing the identified objects.

2608 The **software_spec** includes the *tag* of the object, the
 2609 *tag* of the associated product (if this object is a fileset or
 2610 subproduct), and the version distinguishing attributes of
 2611 this object or its associated product (if this object is a
 2612 fileset or subproduct).

2613 These additional items can also be used with the *one_liner* extended
 2614 option.

2615 **-c** *catalog*
 2616 Provides a means to list the full catalog structure.

2617 If the **-c** option is specified, output from **swlist** is written to an
 2618 exported catalog structure instead of stdout.

2619 The **-c** option specifies a directory below which the catalog informa-
 2620 tion for the specified objects and attributes are stored. The exported
 2621 catalog structure is used both for distributions and
 2622 installed_software catalog information. See 5.1.1.

- 2623 **-d**
- 2624 **-f** *file*
- 2625 **-l** *level*
- 2626 Specifies level at which to list the objects below the specified
2627 software.
- 2628 Level may have values from the enumerated list **bundle**, **product**,
2629 **subproduct**, **fileset**, **control_file**, **file**. If the **-l level**
2630 option is not included, then only the object at the level directly below
2631 the specified software or software_collection is listed. See Table 4-9.

Table 4-9 – Default Levels

Software Selection	Level Listed
none specified	products
bundle	products
product	filesets
subproduct	filesets
fileset	files

- 2640 If no level is specified for bundle and subproduct specifications, all
2641 the available or currently installed product and fileset objects,
2642 resolved recursively, are listed.
- 2643 Multiple **-l level** options may be used to explicitly control what
2644 objects are included.
- 2645 **-r**
- 2646 **-t** *targetfile*
- 2647 **-v** List all the attribute value pairs of the objects specified.
- 2648 The **-v** option specifies that the format of the output shall be in the
2649 **INDEX** file format, as defined in 5.2. Which attributes and objects
2650 are included is controlled by other options and operands. If the **-a**
2651 option is defined, then only those attributes are listed, otherwise all
2652 attributes are listed. If there is no **-v** option, then the listing format
2653 is undefined (see *one_liner* extended option).
- 2654 **-X** *option=value*
- 2655 **-X** *options_file*

2656 **4.6.4 Operands**

- 2657 The **swlist** utility supports the *software_selections* and *targets* operands
2658 described in 4.1. If no *software_selections* are provided, all software in the catalog
2659 (either distribution or installed software) shall be selected.

2660 **4.6.5 External Influences**

2661 See 4.1 for descriptions of external influences common to all utilities.

2662 **4.6.5.1 Extended Options**

2663 The **swlist** utility supports the following extended options. The description in
2664 4.1 applies.

2665 *distribution_target_directory=implementation_defined_value*

2666 *installed_software_catalog=implementation_defined_value*

2667 *one_liner=implementation_defined_value*

2668 *select_local=true*

2669 *software*

2670 *targets*

2671 **4.6.6 External Effects**

2672 See 4.1 for general information.

2673 **4.6.7 Extended Description**

2674 See 4.1 for general information. The following are the two phases in the **swlist**
2675 utility:

2676 (1) Selection phase

2677 (2) Execution phase

2678 **4.6.7.1 Selection Phase**

2679 If there are no software selections specified, then all software from the catalog is
2680 processed. Otherwise, each selection added to the selected software list shall
2681 satisfy the following validation check:

2682 — If the selection is not available from the catalog file, generate an event.
2683 (SW_ERROR: SW_SELECTION_NOT_FOUND)

2684 Unlike all other utilities in this part of ISO/IEC 15068, **swlist** includes all
2685 software that matches a specification, even if the specification is ambiguous.

2686 **4.6.7.2 Execution Phase**

2687 The attributes for the selections determined from the previous phase are listed in
2688 the formats defined by the options.

2689 **4.6.8 Exit Status**

2690 See 4.1.

2691 **4.6.9 Consequences of Errors**

2692 See 4.1.

2693 **4.7 swmodify — Modify software catalog**

2694 **4.7.1 Synopsis**

2695 **swmodify** [**-d** ||| **-r**] [**-p**] [**-u**] [**-a** *attribute=value*] [**-c** *catalog*]
2696 [**-f** *file*] [**-t** *targetfile*] [**-x** *option=value*] [**-X** *options_file*]
2697 [*software_selections*] [**@** *targets*]

2698 **4.7.2 Description**

2699 The **swmodify** utility provides an object and attribute update, create, and delete
2700 interface to the distribution and installed software catalog information indepen-
2701 dent of the other utilities. When combined with **swlist**, there is a complete
2702 read/write interface to the installed software and distribution catalog information.

2703 **4.7.3 Options**

2704 The **swmodify** utility supports the following options. Where there is no descrip-
2705 tion, the description in 4.1 applies.

2706 **-a** *attribute=value*

2707 As an alternative to using a software definition file format to describe
2708 the file attributes, this option may be used to add or modify a single
2709 attribute (e.g., *is_locatable*). If combined with the **-u** option, this
2710 may be used to delete an attribute.

2711 Only one of the **-c** *catalog* and **-a** *attribute* options may be
2712 specified.

2713 **-c** *catalog*

2714 This option specifies the pathname of the catalog information. If it is
2715 a file, then it shall be a file using the software definition file syntax,
2716 in 5.2.1, that defines the objects and attributes desired to be created
2717 or modified.

2718 If it is a directory, then it shall have the exported catalog structure.
2719 For example, this could be a directory containing the output of the
2720 **swlist -c** command.

2721 Only one of the **-c** *catalog* and **-a** *attribute* options may be
specified.

- 2722 **-d**
- 2723 **-f** *file*
- 2724 **-p**
- 2725 **-r**
- 2726 **-t** *targetfile*
- 2727 **-u** Deletes the objects or attributes specified.
- 2728 **-x** *option=value*
- 2729 **-X** *option_file*

2730 **4.7.4 Operands**

- 2731 The **swmodify** utility supports the *software_selections* and *targets* operands
- 2732 described in 4.1.
- 2733 This utility need not support a target distribution in the serial format.

2734 **4.7.5 External Influences**

- 2735 See 4.1 for descriptions of external influences common to all utilities.

2736 **4.7.5.1 Extended Options**

- 2737 The **swmodify** utility supports the following extended options. The description in
- 2738 4.1 applies.

- 2739 *distribution_target_directory* = *implementation_defined_value*
- 2740 *installed_software_catalog* = *implementation_defined_value*
- 2741 *files*
- 2742 *logfile* = *implementation_defined_value*
- 2743 *loglevel* = 1
- 2744 *select_local* = *true*
- 2745 *software*
- 2746 *targets*
- 2747 *verbose* = 1

2748 **4.7.5.2 Standard Input**

2749 **4.7.5.3 Input Files**

- 2750 The source input files may be in one of the following:
- 2751 — *software definition file*
 Described in 5.2.

2752 — exported catalog structure
2753 Described in 5.2.

2754 Note that this structure may be used to describe the `installed_software`
2755 catalog information. There shall be a separate `product.instance_id` for each
2756 version of the product.

2757 NOTE: An installed version is distinguished by the same attributes as in a distribution,
2758 plus the `location` attribute.

2759 **4.7.6 External Effects**

2760 See 4.1.

2761 **4.7.7 Extended Description**

2762 See 4.1 for general information. The `swmodify` utility consists of the following
2763 three phases:

- 2764 (1) Selection Phase
- 2765 (2) Analysis Phase
- 2766 (3) Execution Phase

2767 **4.7.7.1 Selection Phase**

2768 **4.7.7.1.1 Specifying the Source**

2769 The source selection differs from the general information in 4.1 in that the source
2770 is a catalog file or set of catalog files in the software packaging layout format
2771 instead of a distribution, so there are no access control events for accessing the
2772 catalog file.

2773 — If the file parsing discovers syntax errors or missing but required attri-
2774 butes, generate an event.
2775 (SW_ERROR: SW_SOURCE_ACCESS_ERROR)

2776 **4.7.7.1.2 Software Selections**

2777 If there are no software selections specified, then all software from the catalog is
2778 processed. Otherwise, each selection added to the selected software list shall
2779 satisfy the following validation checks. If any of these checks result in an error,
2780 the selection is not added to the list.

2781 — If the selection is not available from the catalog file, generate an event.
2782 (SW_ERROR: SW_SELECTION_NOT_FOUND)

2783 — If a unique version can not be identified, generate an event.
2784 (SW_ERROR: SW_SELECTION_NOT_FOUND_AMBIG)

2785 **4.7.7.2 Analysis Phase**

2786 See 4.1.

2787 **4.7.7.3 Execution Phase**

2788 The execution phase modifies the target catalog. Certain errors can occur when
2789 modifying the catalog.

2790 — If a file cannot be found in order to look up its attributes for modifying the
2791 catalog, generate an event.
2792 (SW_ERROR: SW_FILE_NOT_FOUND)

2793 More complex rules apply when modifying attributes inherited from the product to
2794 the fileset level. The *filesets* and *is_locatable* attributes are updated only by
2795 **swpackage** and **swmodify**. If a fileset definition is removed with **swmodify**,
2796 the *filesets* attribute shall be updated.

2797 If **swmodify** is used to change a fileset *is_locatable* attribute, then the
2798 corresponding product attribute shall be recalculated.

2799 **4.7.8 Exit Status**

2800 See 4.1.

2801 **4.7.9 Consequences of Errors**

2802 See 4.1.

2803 **4.8 swpackage — Package distribution**

2804 **4.8.1 Synopsis**

2805 **swpackage** [-p] [-f *file*] [-s *psf*] [-x *option=value*] [-X *options_file*]
2806 [*software_selections*] [@ *targets*]

2807 **4.8.2 Description**

2808 The **swpackage** utility packages files from the local host into software objects
2809 that can be managed by the utilities in this part of ISO/IEC 15068 using the
2810 definitions from a PSF. The **swpackage** utility packages software into
2811 distributions that can be installed, copied or otherwise distributed or managed.

2812 **4.8.3 Options**

2813 The **swpackage** utility supports the following options. Where there is no
2814 description, the description in 4.1 applies.

2815 **-f** *file*

2816 **-p**

2817 **-s** *psf*

2818 This option specifies the pathname of the PSF, which describes the
2819 details of the packages that **swpackage** operates on.

2820 **-x** *option=value*

2821 **-X** *options_file*

2822 **4.8.4 Operands**

2823 The **swpackage** utility supports the *software_selections* and *targets* operands
2824 described in 4.1 with one exception. The utility may support only a single, local
2825 distribution target.

2826 If no *software_selections* are provided, all software described by the PSF shall be
2827 selected.

2828 Whether data on an existing target distribution in serial format is overwritten or
2829 merged is implementation defined.

2830 **4.8.5 External Influences**

2831 See 4.1 for descriptions of external influences common to all utilities.

2832 **4.8.5.1 Extended Options**

2833 The **swpackage** utility supports the following extended options. The description
2834 in 4.1 applies.

2835 *distribution_target_directory* = *implementation_defined_value*

2836 *distribution_target_serial* = *implementation_defined_value*

2837 *enforce_dsa* = *true*

2838 *follow_symlinks* = *false*

2839 *logfile* = *implementation_defined_value*

2840 *loglevel* = *1*

2841 *media_capacity* = *0*

2842 *media_type* = *directory*

2843 *psf_source_file* = *psf*

2844 *software*

2845 *verbose = 1*

2846 **4.8.5.2 Product Specification File**

2847 See Section 5.

2848 **4.8.6 External Effects**

2849 See 4.1.

2850 **4.8.7 Extended Description**

2851 The **swpackage** utility consists of the following three phases:

2852 (1) Selection Phase

2853 (2) Analysis Phase

2854 (3) Execution Phase

2855 **4.8.7.1 Selection Phase**

2856 **4.8.7.1.1 Specifying Targets**

2857 The target selection differs from the general information in 4.1 in that there may
2858 be only one target. If the target is a serial distribution, **swpackage** sets default
2859 tape types and sizes as described in 4.8.5.1.

2860 **4.8.7.1.2 Specifying the Source**

2861 The source selection differs from the general information in 4.1 in that the source
2862 shall be a PSF, instead of a distribution. Hence there are no access control events
2863 for accessing the PSF.

2864 The selection phase reads (and parses) the PSF as follows to obtain the informa-
2865 tion from the source PSF:

2866 — The product, subproduct, and fileset structure

2867 — The files contained in each fileset

2868 — The attributes associated with these objects

2869 If the file parsing discovers syntax errors, or missing but required attributes, gen-
2870 erate an event.

2871 (SW_ERROR: SW_SOURCE_ACCESS_ERROR).

2872 **4.8.7.1.3 Software Selections**

2873 If there are no software selections specified, then all software from the PSF is pro-
2874 cessed. Otherwise, each selection added to the selected software list shall satisfy
 the following validation checks. If any of these checks result in an error, the
 selection is not added to the list.

- 2875 — If the selection is not available from the PSF, generate an event.
- 2876 (SW_ERROR: SW_SELECTION_NOT_FOUND)
- 2877 — If a unique version can not be identified, generate an event.
- 2878 (SW_ERROR: SW_SELECTION_NOT_FOUND_AMBIG)

2879 4.8.7.2 Analysis Phase

2880 The analysis phase consists of the following steps:

- 2881 — Check the dependency specifications for irregularities (such as circular
- 2882 prerequisites or missing dependencies). If there are irregularities, generate
- 2883 an event.
- 2884 (SW_WARNING: SW_DEPENDENCY_NOT_MET)
- 2885 — Before a new storage directory is created, check to see if this product ver-
- 2886 sion has the same identifying attributes as an existing product version,
- 2887 namely the same tag, revision, architecture, and vendor_tag. If all the
- 2888 identifying attributes match, then the user is repackaging (modifying) an
- 2889 existing version. If a fileset within that product is being repackaged, gen-
- 2890 erate an event.
- 2891 (SW_NOTE: SW_SAME_REVISION_INSTALLED)
- 2892 — Check the existence and attributes of the control_files and files that the
- 2893 PSF defines. If any are missing, generate an event.
- 2894 (SW_ERROR: SW_FILE_NOT_FOUND)
- 2895 — Check that there is enough free disk space on the target file system to pack-
- 2896 age the selected products.
- 2897 — If there is not enough space and *enforce_dsa=true*, generate an event.
- 2898 (SW_ERROR: SW_DSA_OVER_LIMIT)
- 2899 — If there is not enough space and *enforce_dsa=false*, generate an event.
- 2900 (SW_WARNING: SW_DSA_OVER_LIMIT)

2901 4.8.7.3 Execution Phase

2902 The execution phase packages the source files and information into a product and
2903 creates, or merges the product into, the target distribution.

2904 When creating a serial distribution, an implementation shall support one or both
2905 of POSIX.1 {2} extended **cpio** or extended **tar** archive formats. Whether an
2906 implementation supports writing both archive formats or only one, and which for-
2907 mat is supported if only one, is implementation defined.

2908 When packaging a product, the storage directory within the target distribution is
2909 created/updated directly by **swpackage**. For each unique version of the product,
2910 a directory is created using the defined *product.tag* attribute and a unique
2911 sequence number for all the product versions which use the same tag as specified
2912 in Section 5.

2913 The **swpackage** command generates certain attributes as specified in 5.2.

More complex rules apply when modifying packaging attributes inherited from the product to the fileset level. The *filesets* and *is_locatable* attributes are updated only by **swpackage** and **swmodify**. When packaging, the value of the *filesets*

2914 attribute is set to include all the filesets in the PSF, plus any others that already
2915 exist in the distribution but are not in the PSF. In the latter case, the user is
2916 warned that the PSF is not complete.

2917 If undefined, the *product.is_locatable* attribute is set by **swpackage** if any of the
2918 filesets in the *filesets* list are locatable. If none of the filesets are locatable, or if
2919 that cannot be determined, then the value of the *product.is_locatable* attribute is
2920 set to **false**. If defined, the *product.is_locatable* attribute is used to define the
2921 value of the *is_locatable* attribute for any filesets that do not have *is_locatable*
2922 defined. If the value of the *is_locatable* attribute is defined at both the product
2923 and fileset level, then the fileset definition shall override the product definition.

2924 Certain errors can occur as follows when packaging the files:

2925 — If a file can not be added to the distribution for any reason, generate an
2926 event.
2927 (SW_ERROR: SW_FILE_ERROR)

2928 4.8.8 Exit Status

2929 The **swpackage** utility returns as follows:

2930	0	The products specified in the PSF were successfully packaged onto
2931		the media.
2932	1	An error occurred in parsing the PSF. The media was not modified.
2933	2	An error during the packaging operation. The media has been
2934		modified. Review the log file for details.

2935 4.8.9 Consequences of Errors

2936 See 4.1.

2937 4.9 swremove — Remove software

2938 4.9.1 Synopsis

2939 **swremove** [-d ||| -r][-p][-f *file*][-t *targetfile*][-x *option=value*]
2940 [-X *options_file*][*software_selections*][@ *targets*]

2941 4.9.2 Description

2942 The **swremove** utility performs the opposite function of the install software utility
2943 or the copy software utility. It removes installed software or software stored in a
2944 distribution.

2945 The **swremove** utility removes software installed at the local host or at the tar-
gets specified on the command line. It also removes software from local or remote
distributions.

2946 **4.9.3 Options**

2947 The **swremove** utility supports the following options. Where there is no descrip-
2948 tion, the description in 4.1 applies.

- 2949 **-d**
- 2950 **-f** *file*
- 2951 **-p**
- 2952 **-r**
- 2953 **-t** *targetfile*
- 2954 **-X** *option=value*
- 2955 **-X** *options_file*

2956 **4.9.4 Operands**

2957 The **swremove** utility supports the *software_selections* and *targets* operands
2958 described in 4.1.

2959 This utility need not support a target distribution in the serial format.

2960 **4.9.5 External Influences**

2961 See 4.1 for descriptions of external influences common to all utilities.

2962 **4.9.5.1 Extended Options**

2963 The **swremove** utility supports the following extended options. The description in
2964 4.1 applies.

- 2965 *autoselect_dependents = false*
- 2966 *distribution_target_directory = implementation_defined_value*
- 2967 *enforce_dependencies = true*
- 2968 *enforce_scripts = true*
- 2969 *installed_software_catalog = implementation_defined_value*
- 2970 *logfile = implementation_defined_value*
- 2971 *loglevel = 1*
- 2972 *select_local = true*
- 2973 *software*
- 2974 *targets*
- 2975 *verbose = 1*

2976 **4.9.6 External Effects**

2977 See 4.1.

2978 **4.9.7 Extended Description**

2979 See 4.1 for general information. The **swremove** utility consists of three main
2980 phases as follows:

- 2981 (1) Selection phase
- 2982 (2) Analysis phase
- 2983 (3) Execution phase

2984 **4.9.7.1 Selection Phase**

2985 As opposed to **swinstall**, software selections apply to the target distribution or
2986 installed_software.

2987 Each specified selection shall pass the following checks. If a specification does not
2988 pass a check, the implementation-defined error handling procedure is invoked.

- 2989 — If the selection is not found, generate an event.
2990 (SW_WARNING: SW_SELECTION_NOT_FOUND)
- 2991 — If the selection is not found at that product directory, generate an event.
2992 (SW_WARNING: SW_SELECTION_NOT_FOUND_RELATED)
- 2993 — If a single version of the software is not uniquely identified from the pro-
2994 duct and product attributes specified, generate an event.
2995 (SW_ERROR: SW_SELECTION_NOT_FOUND_AMBIG)

2996 Add any dependent software to the selection list if *autoselect_dependents = true*.

2997 **4.9.7.2 Analysis Phase**

2998 This subclause details the analysis phase. The analysis phase occurs before the
2999 removing of files begins and involves executing checks to determine whether or
3000 not the removal should be attempted. No aspect of the target host environment is
3001 modified, so canceling the removal after these operations has no negative effect.

3002 The target role makes the following checks:

- 3003 — When removing installed software, execute vendor-supplied **checkremove**
3004 scripts to perform product-specific checks of the target.
- 3005 — If a **checkremove** script returns an error and *enforce_scripts = true*, an
3006 event is generated and the implementation-defined error handling pro-
3007 cedure is invoked.
3008 (SW_ERROR: SW_CHECK_SCRIPT_ERROR)
- 3009 — If a **checkremove** script returns an error and *enforce_scripts = false*,
3010 generate an event.
3011 (SW_WARNING: SW_CHECK_SCRIPT_ERROR)

- 3012 — If a **checkremove** script returns a warning, generate an event.
- 3013 (SW_WARNING: SW_CHECK_SCRIPT_WARNING)
- 3014 — If a **checkremove** script has a return code of 3, generate an event and
- 3015 unselect the fileset (or all filesets in the product for a product level
- 3016 script).
- 3017 (SW_NOTE: SW_CHECK_SCRIPT_EXCLUDE)
- 3018 — Verify that the dependencies are met. The **swremove** utility does not
- 3019 remove a fileset if it is required by other filesets that have not been selected
- 3020 for removal or cannot be removed. If a non-selected fileset depends on a
- 3021 selected fileset, generate an event as follows.
- 3022 — If *enforce_dependencies = true*, invoke the implementation-defined error
- 3023 handling procedure and generate an event.
- 3024 (SW_ERROR: SW_DEPENDENCY_NOT_MET)
- 3025 — If *enforce_dependencies = false*, generate an event.
- 3026 (SW_WARNING: SW_DEPENDENCY_NOT_MET)
- 3027 If a software object has been specified for removal and there is a bundle referring
- 3028 to that object that has not also been specified for removal, the behavior is imple-
- 3029 mentation defined. Likewise, if a fileset or subproduct object has been specified
- 3030 for removal, and there is a subproduct referring to that object that has not also
- 3031 been specified for removal, the behavior is implementation defined.

3032 4.9.7.3 Execution Phase

3033 For `installed_software`, the sequential relationship between the **unconfigure**,
3034 **preremove**, and **postremove** scripts, and removing files for **swremove** is
3035 shown in the following list. The **unconfigure** scripts are only run if the target
3036 directory is `/`.

- 3037 (1) Unconfigure each product.
- 3038 If the fileset has been configured more than once, the **unconfigure**
- 3039 script shall unconfigure each instance.
- 3040 (a) Unconfigure each fileset in the product.
- 3041 [1] Run the **unconfigure** script for the fileset.
- 3042 [2] Update the result of the script. Update the state of the fileset
- 3043 in the product to **installed** in the database for the
- 3044 `installed_software` object.
- 3045 (b) Run the **unconfigure** script for the product.
- 3046 (2) Remove each product.
- 3047 (a) Run the **preremove** script for the product.
- 3048 (b) Remove each fileset in the product.
- 3049 [1] Update the state of the fileset to **transient** in the catalog for
- 3050 the `installed_software` object.
- [2] Run the **preremove** script for the fileset.

- 3051 [3] Remove the files for the fileset.
- 3052 [4] Run the **postremove** script for the fileset.
- 3053 [5] Update the results of the scripts. Update the state of the
- 3054 fileset to **removed** in the catalog for the `installed_software`
- 3055 object or remove the catalog information for the fileset.
- 3056 (c) Run the **postremove** script for the product.
- 3057 (d) If the catalog information has been removed for all filesets in the
- 3058 product, an implementation can also remove the catalog information
- 3059 for the product and its contained subproducts.
- 3060 (3) Remove each bundle.
- 3061 (a) Remove the `installed_software` catalog information for the bundle.
- 3062 (4) If the catalog information has been removed for all products and bundles
- 3063 in the `installed_software` object, an implementation can also remove the
- 3064 catalog information for the `installed_software` object.

3065 For each fileset that failed to be removed, the installed software catalog informa-
3066 tion is updated to the state **corrupt**.

3067 4.9.7.3.1 Executing preremove Scripts

3068 In this step of the execution phase, **swremove** executes the software **preremove**
3069 scripts.

- 3070 — If a **preremove** script returns an error and `enforce_scripts=true`, generate
- 3071 an event and invoke the implementation-defined error handling procedures.
- 3072 (SW_ERROR: SW_PRE_SCRIPT_ERROR)
- 3073 — If a **preremove** script returns an error and `enforce_scripts=false`, generate
- 3074 an event.
- 3075 (SW_WARNING: SW_PRE_SCRIPT_ERROR)
- 3076 — If a **preremove** script returns a warning, generate an event.
- 3077 (SW_WARNING: SW_PRE_SCRIPT_WARNING)

3078 When **swremove** is removing software from a distribution, no scripts shall be run.

3079 4.9.7.3.2 File Removing

3080 In this step, **swremove** removes the files from the target. The target role
3081 attempts to remove each file from the target file system according to information
3082 obtained in the `software_selections` sent.

3083 If **swremove** cannot remove a file (either because the file is busy [ETXTBSY], or
3084 for some other reason), the file name and the reason are logged so an administra-
3085 tor can take corrective action.
3086 (SW_WARNING: SW_FILE_NOT_REMOVABLE)

3087 If a filename is a symbolic link, the target is not removed. To achieve this
3088 behavior, the **swremove** utility handles symbolic links according to the following
3089 rules:

- 3090 — If a file was recorded in the catalog as a symbolic link to another file, and it
3091 is still a symbolic link on the file system, remove the symbolic link, but do
3092 not remove the target file.
- 3093 — If a file was recorded in the catalog as a file, but exists as a symbolic link on
3094 the file system, remove the symbolic link, but do not remove the target file.
- 3095 — If the pathname to the file includes a symbolic link, this path is followed
3096 and the correct file is removed.
- 3097 All files that are targets of symbolic links are removed when the fileset to which
3098 they belong is removed.

3099 **4.9.7.3.3 Executing postremove Scripts**

- 3100 In this step of the execution phase, **swremove** executes software **postremove**
3101 scripts.
- 3102 — If a **postremove** script returns an error and *enforce_scripts=true*, gen-
3103 erate an event and invoke the implementation-defined error handling pro-
3104 cedures.
3105 (SW_ERROR: SW_POST_SCRIPT_ERROR)
 - 3106 — If a **postremove** script returns an error and *enforce_scripts=false*, gen-
3107 erate an event.
3108 (SW_WARNING: SW_POST_SCRIPT_ERROR)
 - 3109 — If a **postremove** script returns a warning, generate an event.
3110 (SW_WARNING: SW_POST_SCRIPT_WARNING)

3111 **4.9.7.3.4 Kernel Reconfiguration**

- 3112 If the *is_kernel* attribute of the fileset is true, then a warning message to rebuild
3113 the kernel is displayed and also recorded in the log file. However, **swremove** does
3114 not modify any of the kernel generation files.

3115 **4.9.7.3.5 Removing From a Distribution**

- 3116 The list of operations is simpler for removing filesets from a distribution than for
3117 installed software.

- 3118 (1) Remove each product.
- 3119 (a) Remove each fileset in the product.
 - 3120 [1] Update the state of the fileset to **transient** in the catalog for
3121 the distribution.
 - 3122 [2] Remove the files for the fileset.
 - 3123 [3] Update the state of the fileset to **removed** in the catalog for
3124 the distribution or remove the catalog information for the
3125 fileset.
- 3126 (b) If the catalog information has been removed for all filesets in the
3127 product, an implementation can also remove the catalog information
for the product and its contained subproducts. For each fileset that
failed to be removed, the distribution catalog information is updated
to the state **corrupt**.

- 3128 (2) Remove each bundle.
- 3129 (a) Remove the distribution catalog information for the bundle.
- 3130 (3) If the catalog information has been removed for all products and bundles
- 3131 in the distribution object, an implementation can also remove the catalog
- 3132 information for the distribution object.

3133 **4.9.8 Exit Status**

3134 See 4.1.

3135 **4.9.9 Consequences of Errors**

3136 See 4.1.

3137 **4.10 swverify — Verify software**

3138 **4.10.1 Synopsis**

3139 **swverify** [-d ||| -r] [-F] [-f *file*] [-t *targetfile*] [-x *option=value*]
3140 [-X *options_file*] [*software_selections*] [@ *targets*]

3141 **4.10.2 Description**

3142 The **swverify** utility checks the accuracy of software in distributions and
3143 installed *software*. The utility checks the integrity of directory structures and the
3144 files. Discrepancies are reported on stderr along with a detailed explanation of the
3145 problem.

3146 **4.10.3 Options**

3147 The **swverify** utility supports the following options. Where there is no descrip-
3148 tion, the description in 4.1 applies.

- 3149 **-d**
- 3150 **-f** *file*
- 3151 **-F** Correct problems as well as report them.
- 3152 If *check_permissions=true*, correct the corresponding problems
- 3153 reported.
- 3154 If *check_scripts=true*, correct the corresponding problems reported.
- 3155 The **-F** option only applies to installed software.

3156 **-r**
3157 **-t** *targetfile*
3158 **-X** *option=value*
3159 **-X** *options_file*

3160 **4.10.4 Operands**

3161 The **swverify** utility supports the *software_selections* and *targets* operands
3162 described in 4.1.

3163 This utility need not support a target distribution in the serial format.

3164 **4.10.5 External Influences**

3165 See 4.1 for descriptions of external influences common to all utilities.

3166 **4.10.5.1 Extended Options**

3167 The **swverify** utility supports the following extended options. The description in
3168 4.1 applies.

3169 *allow_incompatible=false*
3170 *autoselect_dependencies=true*
3171 *check_contents=true*
3172 *check_permissions=true*
3173 *check_requisites=true*
3174 *check_scripts=true*
3175 *check_volatile=false*
3176 *distribution_target_directory=implementation_defined_value*
3177 *enforce_dependencies=true*
3178 *enforce_locatable=true*
3179 *installed_software_catalog=implementation_defined_value*
3180 *logfile=implementation_defined_value*
3181 *loglevel=1*
3182 *select_local=true*
3183 *software*
3184 *targets*
3185 *verbose=1*

3186 **4.10.6 External Effects**

3187 See 4.1.

3188 **4.10.7 Extended Description**3189 See 4.1. for general information. The key phases in the **swverify** utility are the
3190 following:

3191 (1) Selection phase

3192 (2) Analysis phase

3193 (3) Execution phase

3194 **4.10.7.1 Selection Phase**3195 Like **swremove**, software selections apply to the software installed (or available in
3196 the case of a distribution).3197 Each specified selection is added to the selection list after it passes the following
3198 checks:

3199 — If the selection is not found, generate an event.

3200 (SW_WARNING: SW_SELECTION_NOT_FOUND)

3201 — If the selection is not found at that product location, but that product exists
3202 at another location, generate an event.

3203 (SW_WARNING: SW_SELECTION_NOT_FOUND_RELATED)

3204 Add any dependencies to the selection list if *autoselect_dependencies* = *true*.3205 **4.10.7.2 Analysis Phase**3206 This subclause details the analysis phase for **swverify**. No aspect of the target
3207 host environment is modified unless the **-F** option is specified. The target role
3208 accesses its *software_collection* catalog to get the information for the selected
3209 software.

3210 The target role makes the following checks:

3211 — For each product that is incompatible with the *uname* attributes of the tar-
3212 get host, generate an event as follows. See 4.1.4.1.2.3213 — If *allow_incompatible* = *false*, generate an event.

3214 (SW_ERROR: SW_NOT_COMPATIBLE)

3215 — If *allow_incompatible* = *true*, generate an event.

3216 (SW_WARNING: SW_NOT_COMPATIBLE)

3217 Applies to installed software.

3218 — For each fileset whose state is other than **installed**, **configured**,
3219 **available**, or **removed**, generate an event.

3220 (SW_WARNING: SW_SELECTION_IS_CORRUPT)

Applies to distributions and installed software.

- 3221 — If a dependency cannot be met, generate an event as follows.
- 3222 — If *enforce_dependencies* = *true*, generate an event.
- 3223 (SW_ERROR: SW_DEPENDENCY_NOT_MET)
- 3224 — If *enforce_dependencies* = *false*, generate an event.
- 3225 (SW_WARNING: SW_DEPENDENCY_NOT_MET)
- 3226 Applies to distributions and installed software.
- 3227 — Executes vendor-supplied **verify** scripts.
- 3228 — If a **verify** script returns an error, generate an event.
- 3229 (SW_ERROR: SW_CHECK_SCRIPT_ERROR)
- 3230 — If a **verify** script returns a warning, generate an event.
- 3231 (SW_WARNING: SW_CHECK_SCRIPT_WARNING)
- 3232 Applies to installed software.
- 3233 — The following file level checks are made:
- 3234 — Check for missing files and directories. For installed software, if
- 3235 *check_volatile* = *false*, then this check shall not be made for files with
- 3236 *file.is_volatile* equal **true**.
- 3237 Applies to distributions and installed software.
- 3238 — Check for files that have been modified.
- 3239 — For distributions, check *size*, *cksum*, and *mtime*.
- 3240 — For installed software, check *mode*, *owner*, *group*, *size*, *cksum*,
- 3241 *mtime*, *revision*, *major*, and *minor*, if defined for that file object.
- 3242 If *check_volatile* = *false*, then these checks shall not be made for files
- 3243 with *file.is_volatile* equal to **true**.
- 3244 — If a file is compressed, then the *compressed_size* and *compressed_cksum*
- 3245 attributes of the file should be checked instead of the *size* and *cksum*
- 3246 attributes.
- 3247 Applies to distributions.
- 3248 — Check symbolic links for correct values.
- 3249 Applies to distributions and installed software.
- 3250 If any of these checks fail for any file, generate an event.
- 3251 (SW_ERROR: SW_FILE_ERROR)

3252 4.10.7.3 Execution Phase

- 3253 If the **-F** option is set, then the execution phase operations are run.

3254 4.10.7.3.1 Executing fix Scripts

- 3255 In this step, **swverify** executes vendor-supplied **fix** scripts if operating on
- 3256 installed software. Scripts are executed in the same order as **verify** scripts.

- 3257 — If a **fix** script returns an error, generate an event.
3258 (SW_ERROR: SW_PRE_SCRIPT_ERROR)
- 3259 — If a **fix** script returns a warning, generate an event.
3260 (SW_WARNING: SW_PRE_SCRIPT_WARNING)

3261 Control scripts shall adhere to the specifications in 4.1.6.1.

3262 **4.10.7.3.2 File Level Fix**

3263 The following file level fixes are made:

- 3264 — Missing directories are created (except volatile unless the *check_volatile*
3265 option is true)
- 3266 — Files that have been modified (except volatile unless the *check_volatile*
3267 option is true) are fixed for *mode*, *owner*, *group*, *major*, and *minor*, as appli-
3268 cable
- 3269 — Symbolic links are recreated to correct their values
- 3270 If any of these fixes fail for any file, generate an event.
3271 (SW_ERROR: SW_FILE_ERROR)

3272 **4.10.8 Exit Status**

3273 See 4.1.

3274 **4.10.9 Consequences of Errors**

3275 See 4.1.

IECNORM.COM : Click to view the full PDF of ISO/IEC 15068-2:1999

Section 5: Software Packaging Layout

This section describes the software packaging layout. The software packaging layout consists of

- (1) The directory structure consisting of the following major components:
 - The exported catalog structure containing software information including software definition files and customize scripts used by the install/update and copy utilities
 - The file storage structure that contains the actual software files for each fileset
- (2) The software definition file formats and the objects and attributes they contain, **INDEX** for software definitions and **INFO** for file and control_file definitions (used by all the utilities defined in this part of ISO/IEC 15068), the PSF for product specification (used by **swpackage**), and the **space** file (used by disk space analysis in install).
- (3) The serial format of the layout containing an archive of files in the directory structure.

This ordering of directories and files shall also apply to distributions on a set of hierarchical file systems that span multiple media.

Thus, two distinct (but related) formats for the software packaging layout are supported by this part of ISO/IEC 15068 — a directory structure format that resides within a POSIX.1 {2} hierarchical file system (disk, CD-ROM, etc.), and a bit stream serial format that resides within a POSIX.1 {2} extended **cpio** or extended **tar** archive.

A Strictly Conforming POSIX 7.2 Distribution shall not contain any other files or directories besides those explicitly entered in the distribution catalog. A Conforming POSIX.7.2 Distribution can contain other files and directories besides those belonging to the distribution.

5.1 Directory Structure

This clause describes the directory structure for the software packaging layout, and how this representation stores the definitions of the software and file objects contained within it. The directory structure is a POSIX.1 {2} hierarchical file system containing files in the software packaging layout.

This part of ISO/IEC 15068 defines a single directory structure for directory and serial distributions. The software packaging layout can be stored in the following two forms:

- 34 — A direct access file system as described in this clause
- 35 — A serial access extended **cpio** or extended **tar** archive as described in 5.3
- 36 of POSIX.1 {2}

37 The same structure offers optimal load performance for serial distributions while
 38 providing a simple structure for directory distributions. This structure shall apply
 39 to each medium if the distribution spans multiple media.

40 The structure supports multiple versions of a product contained within a single
 41 distribution, where versions are distinguished by a unique combination the pro-
 42 duct *tag*, *revision*, *architecture*, and *vendor_tag* attributes.

43 Figure 5-1 shows the directory structure of a software packaging layout located
 44 under a directory *path*.

45	Directory or File	Purpose
46	<code><path>/catalog/</code>	Contains all information about the distribution
47	<code><path>/catalog/INDEX</code>	Global index of distribution and its contents
48	<code><path>/catalog/dfiles/</code>	Contains distribution attributes stored in files
49	<code><path>/catalog/dfiles/...</code>	
50		
51	<code><path>/catalog/<product1>/</code>	Storage for information on the first product
52	<code><path>/catalog/<product1>/pfiles</code>	Contains all product attributes stored in files
53	<code><path>/catalog/<product1>/pfiles/INFO</code>	Control_file information for this product
54	<code><path>/catalog/<product1>/pfiles/<script1></code>	First vendor-defined control_file
55	<code><path>/catalog/<product1>/pfiles/...</code>	Additional vendor-defined control_files
56	<code><path>/catalog/<product1>/<fileset1></code>	Storage for information and scripts on this fileset
57	<code><path>/catalog/<product1>/<fileset1>/INFO</code>	File and control_file information for this fileset
58	<code><path>/catalog/<product1>/<fileset1>/<script1></code>	First vendor-defined control_file
59	<code><path>/catalog/<product1>/<fileset1>/...</code>	Additional vendor-defined control_files
60	<code><path>/catalog/<product1>/<fileset2>/</code>	Storage for information and scripts on the next fileset
61	<code><path>/catalog/<product1>/<fileset2>/...</code>	
62	<code><path>/catalog/<product2>/</code>	Storage for information on the next product
63	<code><path>/catalog/<product2>/...</code>	
64	<code><path>/<product1>/</code>	Storage for this product's filesets
65	<code><path>/<product1>/<fileset1>/</code>	Storage for this fileset's files
66	<code><path>/<product1>/<fileset1>/<file1></code>	Path to first file
67	<code><path>/<product1>/<fileset1>/...</code>	Path to additional files
68	<code><path>/<product1>/<fileset2>/</code>	Storage for next fileset's files
69	<code><path>/<product1>/<fileset2>/files</code>	Actual directory structure of files
70	<code><path>/<product1>/<fileset2>/...</code>	
71	<code><path>/<product2>/</code>	Storage for next product's filesets
72	<code><path>/<product2>/...</code>	

73

Figure 5-1 – Example of Software Packaging Layout

74

75 The directory structure for the software packaging layout is divided into the fol-
 76 lowing two areas:

- 77 — The exported catalog structure, consisting of control directories containing
 the software definition files that describe the products contained in the dis-
 tribution, as well as the software control_files

- 78 — The software file storage structure, consisting of the product and fileset
- 79 storage directories, under which the actual software files for each fileset are
- 80 located

81 **5.1.1 Exported Catalog Structure**

82 The catalog structure describes the software contained in the distribution. It is
 83 organized by product, and each product is organized by fileset. The specific con-
 84 tent is described in the following subclauses.

85 **5.1.1.1 INDEX File**

86 The distribution catalog shall contain a global **INDEX** file as follows:

- 87 — **catalog/INDEX**

88 This **INDEX** file contains the definition of all software objects in the distri-
 89 bution.

90 **5.1.1.2 Distribution Files**

91 The **catalog/dfiles/** directory contains files used to store certain attributes of
 92 the distribution object. The distribution information stored can include the follow-
 93 ing:

- 94 — *<attribute>*

95 A distribution attribute can be stored as a separate file, the file name of
 96 which can be the name of the attribute.

97 **5.1.1.3 Product Catalog**

98 The catalog files for each product are stored under a directory
 99 **catalog/***<product_control_directory>***//**. The way in which the value of each
 100 fileset control directory is determined is defined in 5.1.2.1.

101 **5.1.1.4 Product Control Files**

102 The **catalog/***<product_control_directory>***//pfiles/** directory contains the
 103 control_files for the product object. The product control_files include the follow-
 104 ing:

- 105 — *<attribute>*

106 A product attribute can be stored as a separate file, the file name of which
 107 can be the name of the attribute.

- 108 — **INFO**

109 Contains the definitions for the control_file objects contained within the
 110 product.

- 111 — **checkinstall**
preinstall

...

112 **postremove**

113 The vendor-supplied control scripts for the product.

114 — *<control_file>*

115 All other vendor-defined control_files for this product.

116 **5.1.1.5 Fileset Control Files**

117 The **catalog**/*<product_control_directory>//<fileset_control_directory>* directory
118 contains the control_files for the fileset object. The way in which the value of each
119 fileset control directory is determined is defined in 5.1.2.1. The fileset control_files
120 include the following:

121 — *<attribute>*

122 A fileset attribute can be stored as a separate file, the filename of which can
123 be the name of the attribute.

124 — **INFO**

125 Contains the definitions for the control_file and file objects contained within
126 the fileset.

127 — **checkinstall**

128 **preinstall**

129 ...

130 **postremove**

131 The vendor-supplied control scripts for the fileset.

132 — *<control_file>*

133 All other vendor-defined control_files for this fileset.

134 **5.1.2 File Storage Structure**

135 The second portion of a distribution contains the actual software files contained in
136 each fileset object.

137 The files of each fileset are store in a directory with the name
138 *<fileset_control_directory>* that is itself in a directory called
139 *<product_control_directory>*.

140 Each regular file (ones for which *file.type* is **f**) is stored in a location defined by
141 appending the *file.path* attribute to the path of the fileset file storage directory.

142 This may require the creation of additional directories. Other file types (direc-
143 tories, except as needed to store files; hard links and symbolic links) are not
144 required to exist in the distribution. The POSIX.1 {2} file permissions for files in
145 the file storage area are undefined.

146 **5.1.2.1 Control Directory Names**

147 In the simplest case, the value of the *product.tag* attribute is the name of the pro-
product control directory. The *fileset.tag* attribute is used as the name of the fileset
control directory. The following two conditions complicate this simple naming:

148 (1) Length of the tag attribute exceeds {POSIX_PATH_MAX} of the system
149 where the distribution resides.

150 (2) Name collision with an existing product control directory.

151 Given that multiple versions of a product may be contained in the same
152 distribution, collisions from product control directories named by the tag
153 attribute are common.

154 These conditions are met by defining a *control_directory* attribute for each pro-
155 duct and fileset that is unique within the distribution. The attribute uses the fol-
156 lowing syntax:

157 **%token** FILENAME_CHARACTER_STRING /* as defined in 2.2.2.37 */

158 **%start** control_directory

159 **%%**

160 control_directory : tag_part
161 | tag_part "." instance_id_part
162 ;

163 tag_part : FILENAME_CHARACTER_STRING
164 ;

165 instance_id_part : FILENAME_CHARACTER_STRING
166 ;

167 The **tag_part** may be the product or fileset *tag* attribute, truncated as necessary
168 to meet any filename length restrictions of the operating system.

169 The **instance_id_part** is a string that, when added after the “.” (period),
170 defines a **control_directory** that, for products, is unique within the distribu-
171 tion and, for filesets, is unique within the product. For products, this
172 **instance_id_part** may be the *instance_id* of the product if that *instance_id*
173 was generated considering other products **tag_parts** in addition to *tag* attri-
174 butes.

175 5.2 Software Definition File Format

176 The software definition files contain the software structure and the detailed attri-
177 butes for distributions, installed_software, bundles, products, subproducts,
178 filesets, files, and control_files. While information on installed software is
179 represented in this form as input to, or output from, the various software adminis-
180 tration utilities, the actual storage of this metadata for installed software is
181 undefined. This subclause describes the format of the software definition files as
182 follows:

183 — The **INDEX** file contains the definition of distribution or installed_software
184 objects as well as the software objects contained within those
185 software_collections. The information in this file is primarily used in selec-
186 tion phases of the utilities.

— The **INFO** file contains the definition of the software files and control_files
for a product or fileset within a distribution or installed_software object.
The information in this file is primarily used in analysis and execution

187 phases of the utilities.

188 — The PSF also contains the definition of distribution attributes, software
189 objects, and the software files and control_files for the product and fileset
190 software objects. This file is created by the software vendor and used by the
191 packaging tool to create the distribution, represented by the **INDEX** and
192 **INFO** files, in the software packaging layout.

193 The PSF supports the same syntax as the **INDEX** and **INFO** files. Addi-
194 tional syntactic constructs are supported for specifying files and
195 control_files. This file is used in selection, analysis, and packaging phases
196 of the **swpackage** command.

197 Additionally, there is a **space** file that is created by the software vendor for addi-
198 tional disk space needed for a product or fileset. This file is used in the analysis
199 phase of the **swinstall** command to account for additional disk space required.

200 5.2.1 Software Definition File Syntax

201 The **INDEX** and **INFO** files have essentially the same syntax and semantics as the
202 PSF. One key difference is that the **INDEX** file does not contain control_file and
203 file definitions, the **INFO** file contains only control_file and file definitions, and the
204 PSF file contains all definitions. In a distribution, each product and fileset has a
205 separate **INFO** file.

206 The software specification file syntax is as follows. See 2.1.2 for the grammar con-
207 ventions for this syntax and Annex C for examples of syntax usage.

```
208 %token      FILENAME_CHARACTER_STRING /* as defined in 2.2.2.37 */
209 %token      NEWLINE_STRING             /* as defined in 2.2.2.61 */
210 %token      PATHNAME_CHARACTER_STRING /* as defined in 2.2.2.67 */
211 %token      SHELL_TOKEN_STRING         /* as defined in 2.2.2.80 */
212 %token      WHITE_SPACE_STRING        /* as defined in 2.2.2.110 */
```

```
213 %start software_definition_file
214 %%
```

```
215 software_definition_file : INDEX
216                          | INFO
217                          | PSF
218                          ;
219 INDEX                    : soc_definition
220                          soc_contents
221                          ;
222 INFO                     : info_contents
223                          ;
224 PSF                      : distribution_definition
225                          soc_contents
226                          ;
227 media                    : /* empty */
228                          | media_definition
229                          ;
```

```

229  vendors                : /* empty */
230                          | vendors NEWLINE_STRING vendor_definition
231                          | vendor_definition
232                          ;

233  bundles                 : /* empty */
234                          | bundles NEWLINE_STRING bundle_definition
235                          | bundle_definition
236                          ;

237  products                : /* empty */
238                          | products NEWLINE_STRING product_specification
239                          | product_specification
240                          ;

241  product_specification   : product_definition
242                          | product_contents
243                          ;

244  subproducts             : /* empty */
245                          | subproducts NEWLINE_STRING subproduct_definition
246                          | subproduct_definition
247                          ;

248  filesets                : filesets NEWLINE_STRING fileset_specification
249                          | fileset_specification
250                          ;

251  fileset_specification   : fileset_definition
252                          | fileset_contents
253                          /* fileset contents not valid in INDEX files */
254                          ;

255  control_files           : /* empty */
256                          | control_files NEWLINE_STRING control_file_definition
257                          | control_file_definition
258                          ;

259  files                   : /* empty */
260                          | files NEWLINE_STRING file_definition
261                          | file_definition
262                          ;

263  fileset_contents        : fileset_contents NEWLINE_STRING fileset_content_items
264                          | fileset_content_items
265                          ;

266  fileset_content_items   : control_files
267                          | files
268                          ;

269  info_contents           : info_contents NEWLINE_STRING info_content_items
270                          | info_content_items
271                          ;

272  info_content_items      : control_files
273                          | files
274                          ;

  product_contents        : product_contents NEWLINE_STRING product_content_items
                          | product_content_items

```

```

275                                     ;

276  product_content_items             : control_files
277                                     /* control_files not valid in INDEX files */
278                                     | subproducts
279                                     | filesets
280                                     ;

281  soc_contents                      : soc_contents NEWLINE_STRING soc_content_items
282                                     | soc_content_items
283                                     ;

284  soc_content_items                  : vendors
285                                     | bundles
286                                     | products
287                                     ;

288  soc_definition                     : distribution_definition
289                                     | installed_software_definition
290                                     ;

291  distribution_definition            : software_definition
292                                     media
293                                     ;

294  media_definition                   : software_definition
295                                     ;

296  installed_software_definition      : software_definition
297                                     ;

298  vendor_definition                  : software_definition
299                                     ;

300  bundle_definition                  : software_definition
301                                     ;

302  product_definition                 : software_definition
303                                     ;

304  subproduct_definition              : software_definition
305                                     ;

306  fileset_definition                 : software_definition
307                                     ;

308  control_file_definition            : software_definition
309                                     | extended_definition
310                                     /* extended_definition only valid in PSF files */
311                                     ;

312  file_definition                    : software_definition
313                                     | extended_definition
314                                     /* extended_definition only valid in PSF files */
315                                     ;

316  software_definition                : object_keyword NEWLINE_STRING
317                                     attribute_value_list
318                                     ;

    attribute_value_list              : /* empty */

```

```

319         | attribute_value_list attribute_definition NEWLINE_STRING
320         | attribute_definition NEWLINE_STRING
321         ;
322 attribute_definition      : attribute_keyword WHITE_SPACE_STRING attribute_value
323         ;
324 object_keyword           : FILENAME_CHARACTER_STRING
325         ;
326 attribute_keyword        : FILENAME_CHARACTER_STRING
327         ;
328 extended_definition      : extended_keyword WHITE_SPACE_STRING attribute_value
329         ;
330 extended_keyword          : FILENAME_CHARACTER_STRING
331         ;
332 attribute_value           : attribute_value WHITE_SPACE_STRING single_value
333         | single_value
334         | '<' WHITE_SPACE_STRING PATHNAME_CHARACTER_STRING
335         | '<' PATHNAME_CHARACTER_STRING
336         ;
337 single_value              : SHELL_TOKEN_STRING
338         ;

```

339 The following syntax rules are applicable to software definition files:

- 340 (1) All keywords and values are represented as character strings.
- 341 (2) Each keyword is located on a separate line. Keywords can be preceded by
342 white space (tab, space). White space separates the keyword from the
343 value.
- 344 (3) Comments can be placed on a line by themselves or after the keyword-
345 value syntax. They are designated by preceding them with the # (pound)
346 character. The way in which comments are used in **INDEX** and **INFO** is
347 undefined.
- 348 (4) All object keywords have no values. All attribute keywords have one or
349 more values.
- 350 (5) An attribute value ends on the same line as the keyword with one excep-
351 tion. Attribute values can span lines if and only if the value is prefixed
352 and suffixed with the " (double quote) character.
- 353 (6) When an attribute value begins with < (less than), the remainder of the
354 string value shall be interpreted as a filename whose contents will be
355 used as a quoted string value for the attribute. For **INDEX** files, the
356 filename shall be a path relative to the control directory for that distribu-
357 tion, product, or fileset. For PSF files, the filename shall be a path to a
358 file on the host that contains the file.
- 359 (7) The use of " (double quote) is not required when defining a single line
360 string value that contains embedded white space. Trailing white space
361 shall be removed; embedded white space shall be used. The quotes can be
used.

- 362 (8) The " (double quote), # (pound), and \ (backslash) characters can be
363 included in multi-line string values by "escaping" them with \
364 (backslash).
- 365 (9) The order of attributes is not significant, except that the *layout_version*
366 shall be the first attribute defined in an **INDEX** file for a distribution or
367 installed_software object.

368 5.2.1.1 Keyword and Attribute Semantics

369 The keywords and attribute types have the following semantics:

- 370 (1) The object keywords **distribution**, **installed_software**, **bundle**,
371 **product**, **subproduct**, **fileset**, **control_file**, and **file** each
372 define a new object of that type. The keywords **distribution**,
373 **installed_software**, **product**, and **fileset** also define nested
374 blocks that contain the objects describing the software hierarchy.
- 375 (2) If an attribute is not supplied, then its default value shall be used, unless
376 no default value is permitted.
- 377 (3) Attributes that have Boolean permitted values shall be described by the
378 strings **true** and **false**.
- 379 (4) Attributes that have an enumerated set of permitted values shall be
380 described by one of the enumerated values. Enumerated values shall not
381 contain spaces and shall be case sensitive. In addition, abbreviations of
382 the string shall not be allowed. For example, **conf** is not equivalent to
383 **configured**.
- 384 (5) For attributes whose values are integer character strings, the default
385 value shall be used if the attribute is not supplied. If the first two charac-
386 ters of an integer character string are **0x** (zero followed by a lowercase
387 "x"), then the value shall be interpreted as hexadecimal. Otherwise, if
388 the first character of an integer character string is **0** (zero), then the
389 value shall be interpreted as octal. Attribute values denoting time shall
390 be integer character strings that signify seconds since the Epoch.
- 391 (6) Attributes whose permitted values are lists of *tags* or **software_specs**
392 can be described either by one or more repeating keywords, each listing
393 one or more *tags* or **software_specs** separated by white space (e.g., for
394 *subproduct.contents* or *fileset.prerequisites*), or by blocks of object frag-
395 ments (e.g., product, fileset, and file definitions).
- 396 The former is used when the hierarchy is defined by reference, and the
397 latter is used when the hierarchy is defined by containment. For exam-
398 ple, subproducts and filesets are contained within products, but filesets
399 are referenced by subproducts.
- 400 (7) Attributes that have permitted values of
401 **software_pattern_matching_string** are software pattern match-
402 ing strings as described in 2.2.2.92. For all product attributes related to
403 the *uname* structure (as defined in 4.4.1 of POSIX.1 {2}), an empty string
404 value is treated as equivalent to * (asterisk), implying a universal match.

5.2.1.2 Vendor-Defined Keywords and Attributes

A software definition file can contain keywords (implying attributes) not defined by this part of ISO/IEC 15068. All such keywords in a file not recognized by an implementation shall be preserved (along with their associated values) by being transferred to the resulting **INDEX** or **INFO** files created by **swpackage** or **swcopy**. For any keyword, the keyword itself shall be a filename character string.

The value associated with any keyword shall be processed as an **attribute_value** (see 5.2.1) and thus can be continued across multiple input lines or can reference a file containing the value for the keyword.

Implementations that make use of keywords beyond those described in this part of ISO/IEC 15068 take actions they believe appropriate for those keywords. The handling of any keywords that are both not defined by this part of ISO/IEC 15068, and still recognized by an implementation, is undefined.

5.2.2 Distribution Definition

distribution

layout_version	<i>layout_version</i>
path	<i>path</i>
dfiles	<i>dfiles</i>
pfiles	<i>pfiles</i>
uuid	<i>uuid</i>

INDEX and PSF files can contain distribution definitions. Neither file contains the *path* attribute. Its value is generated dynamically by **swlist**.

The *bundles*, *media*, and *products* attributes are not stored as attributes, but rather as **bundle**, **media**, and **product** definitions. These attributes are not included in **swlist -v** output. Rather, they are generated dynamically only by **swlist -a attribute**.

A PSF does not require a distribution definition. The PSF shall not contain the *uuid* attribute. It is generated dynamically, if needed, by **swcopy** and **swpackage**.

An **INDEX** file shall contain the *layout_version* attribute as the first attribute defined in the file. Distributions that span multiple media shall contain the *uuid* attribute.

5.2.3 Media Definition

media

sequence_number	<i>sequence_number</i>
------------------------	------------------------

INDEX files for distributions can contain media definitions.

An **INDEX** file for a distribution shall contain the *sequence_number* attribute if the distribution spans multiple media.

444 5.2.4 Installed Software Definition

445	installed_software	
446	layout_version	<i>layout_version</i>
447	path	<i>path</i>
448	dfiles	<i>dfiles</i>
449	pfiles	<i>pfiles</i>
450	catalog	<i>catalog</i>

451 The storage of catalogs for installed software is undefined. With the use of the
452 **swlist** utility, the contents of such catalogs may be manifested in exported cata-
453 log form. The rules contained within this subclause shall apply when the contents
454 of an installed software catalog is manifested in exported catalog form.

455 **INDEX** files can contain installed_software definitions. This describes the attri-
456 butes for installed_software objects when listed in exported catalog structure
457 using **swlist**.

458 The *products* and *bundles* attributes are not stored as attributes, but rather as
459 product and bundle definitions. These attributes are not included in **swlist -v**
460 output. Rather, they are generated dynamically only by **swlist -a** attribute.

461 An **INDEX** file does not contain the *path* or *catalog* attributes; they are generated
462 dynamically by **swlist**.

463 An **INDEX** file shall contain the *layout_version* attribute as the first attribute
464 defined in the file.

465 5.2.5 Vendor Definition

466	vendor	
467	tag	<i>tag</i>
468	title	<i>title</i>
469	description	<i>description</i>

470 **INDEX** and PSF files can contain vendor definitions. The *tag* attribute is required
471 for all vendor objects.

472 5.2.6 Bundle Definition

473	bundle	
474	tag	<i>tag</i>
475	architecture	<i>architecture</i>
476	location	<i>location</i>
477	qualifier	<i>qualifier</i>
478	revision	<i>revision</i>
479	vendor_tag	<i>vendor_tag</i>
480	contents	<i>contents</i>
481	copyright	<i>copyright</i>
482	create_time	<i>create_time</i>
483	description	<i>description</i>
484	directory	<i>directory</i>
485	instance_id	<i>instance_id</i>
486	is_locatable	<i>is_locatable</i>
487	machine_type	<i>machine_type</i>
488	mod_time	<i>mod_time</i>
	number	<i>number</i>
	os_name	<i>os_name</i>
	os_release	<i>os_release</i>

489	os_version	<i>os_version</i>
490	size	<i>size</i>
491	title	<i>title</i>

492 **INDEX** and PSF files can contain bundle definitions. The *tag* and *contents* attri-
493 butes are required for all bundles.

494 Neither file contains the *size* attribute. The value of the *size* attribute is gen-
495 erated dynamically based on the sizes of the filesets currently contained within
496 the bundle.

497 An **INDEX** file shall also contain an *instance_id* attribute. The value of the
498 *instance_id* attribute is generated dynamically by **swpackage** or **swcopy**. An
499 **INDEX** file for installed software shall contain a *create_time* attribute and a
500 *mod_time* attribute for each bundle.

501 Only bundle definitions for installed software may contain either the *location* or
502 *qualifier* attributes; bundle definitions for distributions shall not contain either
503 the *location* or *qualifier* attributes.

504 A PSF should not contain either the *location* or *qualifier* attributes; they shall be
505 ignored when parsing the file.

506 5.2.7 Product Definition

507	product	
508	tag	<i>tag</i>
509	architecture	<i>architecture</i>
510	qualifier	<i>qualifier</i>
511	revision	<i>revision</i>
512	vendor_tag	<i>vendor_tag</i>
513	all_filesets	<i>all_filesets</i>
514	control_directory	<i>control_directory</i>
515	copyright	<i>copyright</i>
516	create_time	<i>create_time</i>
517	directory	<i>directory</i>
518	description	<i>description</i>
519	instance_id	<i>instance_id</i>
520	is_locatable	<i>is_locatable</i>
521	postkernel	<i>postkernel</i>
522	location	<i>location</i>
523	machine_type	<i>machine_type</i>
524	mod_time	<i>mod_time</i>
525	number	<i>number</i>
526	os_name	<i>os_name</i>
527	os_release	<i>os_release</i>
528	os_version	<i>os_version</i>
529	size	<i>size</i>
530	title	<i>title</i>

531 **INDEX** and PSF files can contain product definitions. The *tag* and
532 *control_directory* attributes are required for all products.

533 Neither file contains the *size* attribute. The value of the *size* attribute is gen-
534 erated dynamically based on the sizes of the filesets currently contained within
535 the product.

536 An **INDEX** file for installed software shall contain a *create_time* attribute and a
mod_time attribute for each product.

537 The *control_files*, *filesets*, and *subproducts* attributes are not stored as attributes,
538 but rather as **control_file**, **fileset**, and **subproduct** definitions. These
539 attributes are not included in **swlist -v** output. Rather, they are generated only
540 by **swlist -a** attribute.

541 An **INDEX** file shall contain an *instance_id* attribute. The value of the *instance_id*
542 attribute is generated by **swpackage** or **swcopy**. An **INDEX** file shall also con-
543 tain an *all_filesets* attribute in addition to the fileset definitions. This attribute is
544 generated by **swpackage** and represents all filesets defined for the product, as
545 opposed to those that are currently contained within the product. The value of
546 the *filesets* and *all_filesets* attributes may differ, since some originally defined
547 filesets might not be copied or installed. Only product definitions for installed
548 software may contain either the *location* or *qualifier* attributes; product
549 definitions for distributions shall not contain either the *location* or *qualifier* attri-
550 butes.

551 A PSF should not contain either the *location* or *qualifier* attributes; they shall be
552 ignored when parsing the file.

553 5.2.8 Subproduct Definition

554	subproduct	
555	tag	<i>tag</i>
556	contents	<i>subproducts</i>
557	create_time	<i>create_time</i>
558	description	<i>description</i>
559	mod_time	<i>mod_time</i>
560	size	<i>size</i>
561	title	<i>title</i>

562 **INDEX** and PSF files can contain subproduct definitions. The *tag* and *contents*
563 attributes are required for all subproducts.

564 Neither file contains the *size* attribute. The value of the *size* attribute is gen-
565 erated dynamically based on the sizes of the filesets currently contained within
566 the subproduct.

567 An **INDEX** file for installed software shall contain a *create_time* attribute and a
568 *mod_time* attribute for each subproduct.

569 5.2.9 Fileset Definition

570	fileset	
571	tag	<i>tag</i>
572	control_directory	<i>control_directory</i>
573	corequisites	<i>corequisites</i>
574	create_time	<i>create_time</i>
575	description	<i>description</i>
576	exerequisites	<i>exerequisites</i>
577	is_reboot	<i>is_reboot</i>
578	is_kernel	<i>is_kernel</i>
579	is_locatable	<i>is_locatable</i>
580	location	<i>location</i>
581	media_sequence_number	<i>media_sequence_number</i>
582	mod_time	<i>mod_time</i>
	prerequisites	<i>prerequisites</i>
	revision	<i>revision</i>

583	size	<i>size</i>
584	state	<i>state</i>
585	title	<i>title</i>

586 **INDEX** and PSF files can contain fileset definitions. The *tag* and *control_directory*
587 attributes are required for all filesets.

588 The *control_files* and *files* attributes are not stored as attributes, but rather as
589 **control_file** and **file** definitions. These attributes are not included in
590 **swlist -v** output. Rather, they are generated only by **swlist -a attribute**.

591 An **INDEX** file shall contain a *size* attribute for each defined fileset. Fileset
592 definitions for distributions that span multiple media shall contain the
593 *media_sequence_number* attribute. An **INDEX** file for installed_software shall
594 contain a *create_time* and a *mod_time* attribute for each fileset.

595 A PSF should not contain the *location*, *media_sequence_number*, *size*, or *state*
596 attributes; they shall be ignored when parsing the file. The value of the *size* attri-
597 bute is generated dynamically by **swpackage** based on the sizes of the files and
598 *control_files*.

599 5.2.10 Control_File Definition

600	control_file	
601	tag	<i>tag</i>
602	cksum	<i>cksum</i>
603	compressed_cksum	<i>compressed_cksum</i>
604	compressed_size	<i>compressed_size</i>
605	compression_state	<i>compression_state</i>
606	compression_type	<i>compression_type</i>
607	interpreter	<i>interpreter</i>
608	path	<i>path</i>
609	revision	<i>revision</i>
610	size	<i>size</i>
611	source	<i>source</i>
612	result	<i>result</i>

613 **INFO** and PSF files can contain *control_file* definitions.

614 A PSF should not contain the *cksum*, *compressed_cksum*, *compressed_size*,
615 *compression_state*, *compression_type*, *size*, or *result* attributes; they shall be
616 ignored when parsing the file. The values of the *size* and *cksum* attributes are
617 generated dynamically by **swpackage** based on the source file. A PSF shall con-
618 tain a *source* attribute. A PSF can contain a *path* attribute. If it does not,
619 **swpackage** shall use the basename obtained from the value of the *source* attri-
620 bute as the value of the *path* attribute. A PSF can contain a *tag* attribute. If it
621 does not, **swpackage** shall use the basename obtained from the value of the *path*
622 attribute as the value of the *tag* attribute. The **swpackage** utility shall resolve
623 the value of the *tag* attribute after it resolves the value of the *path* attribute.

624 An **INDEX** file shall contain the *tag*, *path*, *cksum*, and *size* attributes. *Control_file*
625 definitions for installed software shall also contain the *result* attribute. **INDEX**
626 files should not contain the *source* attribute; it shall be ignored when parsing the
627 file.

628 The **swpackage** command automatically includes the **INFO** file itself as a control
file and adds the *tag*, *path*, and *size* attributes for it. The value of the *cksum* attri-
bute for the **INFO** *control_file* itself is not defined. An implementation can choose

629 to store certain software object attributes, such as *copyright*, as *control_files*.

630 5.2.11 File Definition

631	file	
632	path	<i>path</i>
633	cksum	<i>cksum</i>
634	compressed_cksum	<i>compressed_cksum</i>
635	compressed_size	<i>compressed_size</i>
636	compression_state	<i>compression_state</i>
637	compression_type	<i>compression_type</i>
638	gid	<i>gid</i>
639	group	<i>group</i>
640	is_volatile	<i>is_volatile</i>
641	link_source	<i>link_source</i>
642	major	<i>major</i>
643	minor	<i>minor</i>
644	mode	<i>mode</i>
645	mtime	<i>mtime</i>
646	owner	<i>owner</i>
647	revision	<i>revision</i>
648	size	<i>size</i>
649	source	<i>source</i>
650	type	<i>type</i>
651	uid	<i>uid</i>

652 **INFO** and PSF can contain file definitions.

653 A PSF shall contain a *source* attribute. A PSF should not contain the *cksum*
654 *compressed_cksum* *compressed_size*, *compression_state*, *compression_type*, or *size*
655 attributes; they shall be ignored when parsing the file. Device files (including the
656 *major* and *minor* attributes) should not be defined in a PSF (but can be added via
657 **swmodify** after being created by a **configure** script); they shall be ignored
658 when parsing the file. The values of the *size* and *cksum* attributes are generated
659 dynamically by **swpackage** based on the source file. A PSF can contain a *path*
660 attribute, otherwise the source is used to defined the path by **swpackage**. A PSF
661 can contain *gid*, *group*, *link_source*, *mode*, *mtime*, *owner*, *type*, and *uid* attributes,
662 otherwise they are retrieved from the source file by **swpackage**. A PSF can con-
663 tain *is_volatile* and *revision* attributes. Automatic determination of the file revi-
664 sion is implementation defined.

665 An **INFO** file should not contain the *source* attribute; it shall be ignored when
666 parsing the file. Table 5-1 shows the required, optional, and non-applicable attri-
667 butes for each of the file types in an **INFO** file. The file types are described in 3.13.
668 Within a fileset, no more than one copy of a file shall be stored with the same
669 path.

670 Within a PSF file, if the same file is defined more than once, the attributes from
671 the last definition shall be used and shall redefine the attributes previously
672 defined. This action shall not cause additional copies of the file to be stored in the
673 distribution. All attributes not specifically listed shall remain unchanged.

Table 5-1 – File Attributes for INFO File

attribute	f	d	h	s	b	c	p
type	R	R	R	R	R	R	R
path	R	R	R	R	R	R	R
size	R	-	-	-	-	-	-
link_source	-	-	R	R	-	-	-
mode	O	O	-	-	O	O	O
owner	O	O	-	-	O	O	O
group	O	O	-	-	O	O	O
uid	O	O	-	-	O	O	O
gid	O	O	-	-	O	O	O
cksum	O	-	-	-	-	-	-
major	-	-	-	-	R	R	-
minor	-	-	-	-	R	R	-
is_volatile	O	O	O	O	O	O	O
mtime	O	-	-	-	-	-	-
revision	O	-	-	-	-	-	-

Key: R: Required O: Optional -: Ignored

5.2.12 Extended Control_File Definitions

checkinstall *source* [path]
preinstall *source* [path]
postinstall *source* [path]
verify *source* [path]
fix *source* [path]
checkremove *source* [path]
preremove *source* [path]
postremove *source* [path]
configure *source* [path]
unconfigure *source* [path]
request *source* [path]
unpreinstall *source* [path]
unpostinstall *source* [path]
space *source* [path]
control_file *source* [path]

A PSF can contain extended control_file definitions. Each control_file definition defines the *source* attribute (the source file) to be used for the script. The keyword (meaning **checkinstall**, **preinstall**, etc.) defines the *tag* of the script, which tells the utilities when to execute the script.

If the optional *path* is supplied, it shall be the file name in the distribution (relative to the control directory for the software containing this script) used to store the file; otherwise, the *control_file.tag* attribute is used as the file name. This also allows a vendor to define one script to be executed for multiple tags. The script can determine the *tag* being executed by the **SW_CONTROL_TAG** environment variable.

If the **control_file** keyword is used, then the basename of the *source* attribute is the *tag* of the control_file.

721 5.2.13 Extended File Definitions

722 A PSF can contain extended file definitions. The **swpackage** utility supports
723 these extended file definition mechanisms:

724 directory mapping

725 A PSF can point the **swpackage** utility at a source directory con-
726 taining the files for the fileset. In addition, a PSF can map this
727 source directory to the appropriate (destination) directory containing
728 this subset of the filesets files.

729 recursive (implicit) file definition

730 If a directory mapping is active, a PSF can direct the **swpackage**
731 utility to include all files (recursively) from within the directory in
732 the fileset.

733 explicit file definition

734 For all or some of the files in the fileset, a PSF can name each source
735 file and destination path with a one line per file syntax.

736 default permission definition

737 For all or some of the files in the fileset, a PSF can define a default
738 set of permissions.

739 excluding files

740 Files that otherwise would be included can be explicitly excluded.

741 including files

742 File definitions may be included from a separate file.

743 These mechanisms can all be used in combination with the others.

744 5.2.13.1 Directory Mapping

745 **directory** *source* [*path*]

746 This syntax defines a *source* directory under which subsequently listed files are
747 located. In addition, the user can map the *source* directory to a *destination* direc-
748 tory under which the packaged files will be located.

749 The destination directory shall be an absolute pathname, and shall be used as a
750 prefix to the *path* attribute for each of the files.

751 The source directory can be either an absolute pathname, or a relative pathname.
752 If relative, the **swpackage** utility interprets it relative to the current working
753 directory in which the utility was invoked.

754 If the *source* directory does not exist, the **swpackage** utility shall generate an
755 error.

756 5.2.13.2 Recursive File Definition

757 **file** *

758 This syntax directs the **swpackage** utility to include every file (and directory)
759 within the current source directory in the fileset. The **swpackage** utility
attempts to include the entire recursive contents of the source directory in the
fileset.

760 The **directory** keyword shall be specified before the **file *** specification is
 761 used. After finishing the recursive processing of the source directory, the
 762 **swpackage** utility processes further specifications with respect to the original
 763 directory.

764 All other attributes for the destination file object are taken from the source file,
 765 unless a **file_permissions** keyword is active. This keyword is described
 766 below.

767 The user can specify multiple **directory** and **file *** pairs to gather all files
 768 from different source directories into a single fileset.

769 5.2.13.3 Explicit File Definition

770 **file** [-t *type*] [-m *mode*] [-o *owner*[,*uid*]] [-g
 771 *group*[,*gid*] [-n] [-v] *source* [*path*]

772 Instead of, or in addition to, the recursive file specification, the user can explicitly
 773 specify the files to be packaged into a fileset.

774 This syntax may be used to redefine an attribute of a previously defined file. All
 775 attributes not specifically listed remain the same.

776 The **directory** keyword can be used to define a source (and destination) for
 777 explicitly specified files. If no **directory** keyword is active, then the full source
 778 path and the absolute destination path (the *path* attribute) shall be specified for
 779 each file.

780 The meaning of each of these fields is as follows:

781 **file** This keyword specifies an existing file or directory, perhaps within
 782 the currently active source directory, to include in the fileset. It can
 783 also specify a directory, hard link, or symbolic link that does not exist
 784 as a source file, but is created when the fileset is installed.

785 *source* When specifying an existing source file, this value defines the path to
 786 it.

787 If this is a relative path, the **swpackage** utility searches for it rela-
 788 tive to the source directory set by the **directory** keyword. If no
 789 source directory is active, the **swpackage** utility searches for it rela-
 790 tive to the current working directory in which the utility was
 791 invoked.

792 All attributes for the destination file object are taken from the source
 793 file, unless a **file_permissions** keyword is active, or the **-m**, **-o**,
 794 or **-g** options are also included in the file specification.

795 When specifying a new directory to be created upon installation
 796 where there is no destination *path* specified, the *source* defines the
 797 path of the installed directory. When specifying a new hard link or
 798 symbolic link to be created upon installation, the *source* defines the
 799 pathname of the installed file to use as the source for the new file.

800 *path* When specifying a new or existing source file, this value defines the
 801 destination path at which the file will be created or installed. If *path*
 is a relative path, the active destination directory set by the **direc-**
 tory keyword shall be prefixed to it. If the path is relative, and no

802 destination directory is active, the **swpackage** utility shall generate
803 an error. If the path is not specified, then the *source* is used as the
804 *path* with the appropriate mapping done with the active destination
805 directory (if any).

806 **-t type** When creating a new directory, hard link or symbolic link (a file in
807 the fileset that does not exist in the source), this option shall be
808 specified to define the file type. The following file types can be
809 created:

810 **d** Create a directory. If only the *source* is specified, it is
811 used as the *path*. Otherwise, the *source* is used to
812 retrieve the attributes for the directory created at *path*.
813 If the path is not specified, or any attributes, then
814 default values of the attributes shall be implementation
815 defined.

816 **h** Create a hard link. Both the *source* and *path* shall be
817 specified. The *source* is the pathname of the installed
818 file object to be used as the source for the new hard link
819 (the *link_source* attribute).

820 **s** Create a symbolic link. Both the *source* and *path* shall
821 be specified. The *source* is the pathname of the installed
822 file object to be used as the source for the new symbolic
823 link (the *link_source* attribute).

824 Files with the types **c** (character special), **b** (block special), and **p**
825 (named pipe | FIFO) are not supported by **swpackage** and **swin-**
826 **stall** and can be created via a **configure** control script. In gen-
827 eral, device files and pipes are created during system configuration
828 on the system actually running the software. Also, there can be files
829 of other types that the **swpackage** utility does not recognize and
830 that shall therefore cause an error.

831 **-m mode** This option defines the (octal) mode for a file or directory.

832 **-o [owner][,uid]**

833 This option defines the name or uid, or both, of the owner of the des-
834 tination file. If only the *owner* is specified, then the *owner* and *uid*
835 attributes are set for the destination file object based on the database
836 of the packaging host. If only the *uid* is specified, it is set as the *uid*
837 attribute for the destination object and no owner name is assigned.
838 If both are specified, each sets the corresponding attribute for the file
839 object. If neither are specified, then the owner and uid of the file
840 shall be used as found in the file system of the packaging host. See
841 4.5.7.3.3.

842 During an installation, the *owner* attribute is used to set the owner
843 name and uid unless the owner name is not defined in the target sys-
844 tem user database. In this case, the value of the *uid* attribute is used
845 to set the uid.

846 **-g [group][,gid]**

847 This option defines the name or gid, or both, of the group of the desti-
nation file. If only the *group* is specified, then the *group* and *gid*
attributes are set for the destination file object based on the database

848 of the packaging host. If only the *group* is specified, and it contains
 849 digits only, it is interpreted as the *gid*, and is set as the *gid* attribute
 850 for the destination object; no group name is assigned to the object. If
 851 both are specified, each sets the corresponding attribute for the file
 852 object. If neither are specified, then the group and *gid* of the file
 853 shall be used as found in the file system of the packaging host. See
 854 4.5.7.3.3.

855 During an installation, the *group* attribute is used to set the group
 856 name and *gid*, unless the group name is not defined in the target sys-
 857 tem group database. In this case, the *gid* attribute is used to set the
 858 *gid*.

859 **-n** This option indicates that the file is not compressible.

860 **-v** The use of **-v** on a source line is used to specify that the file is vola-
 861 tile (contents, attributes or existence can change after installation).

862 5.2.13.4 Default Permission Definition

863 **file_permissions** [-m *mode* | -u *umask*][-o [*owner*[,]][*uid*]] [-g [*group*[,]][*gid*]]

864 A destination file object inherits the mode, owner, and group of the source file.
 865 The **file_permissions** keyword can be specified as follows to set a default per-
 866 mission mask, owner, and group for all the files being packaged into the fileset:

867 **file_permissions**

868 This keyword only applies to the fileset it is defined in. Multiple
 869 **file_permissions** can be specified, and subsequent definitions
 870 simply replace previous definitions.

871 **-m mode** This option defines a default (octal) mode for all file objects.

872 **-u umask**

873 Instead of specifying an octal mode as the default, the user can
 874 specify an octal *umask*() value that gets “subtracted” from the mode
 875 of an existing source file, or applied for each nonexistent file, to gen-
 876 erate the mode of the destination file.

877 By specifying a *umask*() value the user can set a default mode for
 878 executable files, non-executable files, and directories. A specific mode
 879 can be set for any file, as described above.

880 **-o** [*owner*[,]][*uid*]

881 This option defines the name or *uid*, or both, of the owner of the des-
 882 tination file. See the discussion of the **-o** option in 5.2.13.3.

883 **-g** [*group*[,]][*gid*]

884 This option defines the name or *gid*, or both, of the group of the desti-
 885 nation file. See the discussion of the **-g** option in 5.2.13.3.

886 5.2.13.5 Excluding Files

887 **exclude** *source*

888 A file listed after the **exclude** keyword that was previously included, for example
 from a recursive file definition, is excluded from the list of files.

889 If the source specifies a directory, then all files below that directory are excluded.

890 **5.2.13.6 Including Files**

891 **file** < *include_file*

892 The **file** keyword can be used to include definitions for files from a separate
893 *include_file* by specifying a < (less than) character followed by the *include_file*.

894 **5.2.14 Space Control_file**

895 *path* [+ | -] *size*

896 For each *path* listed in the **space** file, the **swinstall** utility shall add the size,
897 in bytes, to the disk space requirements. The size can be positive (reflecting the
898 maximum transient or permanent disk space required for the install), or negative
899 (reflecting space freed by one of the scripts executed by the **swinstall** com-
900 mand). An implementation shall consider positive records and may consider nega-
901 tive records.

902 **5.3 Serial Format and Multiple Media**

903 A distribution in the serial format of the software packaging layout is a bit-stream
904 representation, implemented as a set of POSIX.1 {2} extended **cpio** or extended
905 **tar** archives which contain files in the directory structure of the software packag-
906 ing layout defined in Section 5.

907 A serial distribution can be stored on any serial medium. A serial distribution can
908 also be stored in any file, within the file system, which supports the storing of
909 POSIX.1 {2} extended **cpio** or extended **tar** archives. How a system reads or
910 writes to the different media devices is outside the scope of this part of ISO/IEC
911 15068.

912 Implementations shall support serial distributions if the underlying operating sys-
913 tem supports the **pax** utility, as defined in POSIX.2 {3}, or otherwise supports
914 reading and writing of the extended **tar** and extended **cpio** archives defined in
915 POSIX.1 {2}. If serial distributions are supported, the serial distribution formats
916 supported shall include extended **tar** and extended **cpio**.

917 The distribution is implemented as a set of one or more POSIX.1 {2} extended
918 **cpio** or extended **tar** archives. The archives reside on a set of one or more serial
919 media, or in a file. Each media in a serial distribution shall contain one and only
920 one archive.

921 A distribution may span multiple media in a hierarchical structure. In this case,
922 the set of files on any particular media, including the attributes defined in any
923 software definition files, should be similar to that for a serial archive. In other
924 words, the decision for which files are put on which media should be the same
925 whether the distribution is serial or hierarchical. Space considerations on media
926 may cause some differences.

927 The following are the rules regarding ordering of files within serial distributions.
These rules, including generation of the *fileset.media_sequence_number*, are
implemented by the **swpackage** utility.

- 928 (1) The catalog files (which contain all the information describing the
929 software contained in the distribution), as well as the control scripts, in
930 this relative order.
- 931 (a) The global **INDEX** file, as described in 5.1.1.1
- 932 (b) The distribution files, as described in 5.1.1.2
- 933 (c) The product catalog files, product by product, as described in 5.1.1.3
- 934 [1] The product control files, as described in 5.1.1.4
- 935 [2] The fileset control files, fileset by fileset, as described in 5.1.1.5
- 936 (2) The actual software files, fileset by fileset, as described in 5.1.2
- 937 (a) Prerequisites of filesets before the filesets that depend on them
- 938 (b) Kernel filesets before non-kernel filesets (except where kernel
939 filesets have prerequisites on non-kernel)
- 940 (3) Each medium shall have (as its first file, if a serial medium)
- 941 (a) A global **INDEX** file, **catalog/INDEX** that shall at least contain the
942 *distribution.uuid* and *media.sequence_number* attributes (used to
943 identify a particular media within a particular distribution)
- 944 (4) Each archive shall start at the beginning of the medium. Multiple
945 archives on one medium are not allowed.

946 Additionally, in order to increase the usability of multiple media serial distribu-
947 tions, the following guidelines should be used and in decreasing importance:

- 948 — Each medium should contain complete files wherever possible. If a file is
949 larger than the the capacity defined by the *media_capacity* option, then the
950 behavior is implementation defined.
- 951 — Each medium should contain complete filesets whenever possible or practi-
952 cal. The *fileset.media_sequence_number* attribute is the number of the
953 medium where the fileset begins. If a fileset is larger then the medium size,
954 then the *fileset.media_sequence_number* attribute shall contain the list of
955 *media.sequence_numbers* describing the media that contain this fileset.
- 956 — Each medium should contain complete products whenever possible or prac-
957 tical.
- 958 — Each medium should contain needed dependencies whenever possible or
959 practical.

960 Thus, a conforming implementation shall be able to

- 961 — Read the **INDEX** off of the first medium for the Selection Phase
- 962 — Scan the first medium (and those following as needed) for the necessary
963 catalog files for the Analysis Phase
- 964 — Request the next needed medium for the next needed fileset based on
965 *media_sequence_number* during the Execution Phase
- 966 — Request the next medium when the fileset spans media

967 Note that in all respects, a serial distribution shall conform to the specifications of
968 the extended **cpio** or extended **tar** archives. See 10.1.1 and 10.1.2 of
969 POSIX.1 {2}. This includes, but is not limited to, the following:

970 — Recording format

971 — Character sets

IECNORM.COM : Click to view the full PDF of ISO/IEC 15068-2:1999

Annex A (informative)

Bibliography

- {B1} Desktop Management Task Force *Desktop Management Interface Specification, Version 1.0*, 29 April 1994.¹⁾
- {B2} ISO 639: 1988, *Code for the representation of names of languages*.²⁾
- {B3} ISO/IEC 2022: 1994, *Information technology—Character code structure and extension techniques*.
- {B4} ISO 2047: 1975, *Information processing—Graphical representations for the control characters of the 7-bit coded character set*.
- {B5} ISO 3166: 1993, *Codes for the representation of names of countries*.
- {B6} ISO 4217: 1995, *Codes for the representation of currencies and funds*.
- {B7} ISO/IEC 4873: 1991, *Information technology—ISO 8-bit code for information interchange—Structure and rules for implementation*.
- {B8} ISO/IEC 6429: 1992, *Information technology—Control functions for coded character sets*.
- {B9} ISO/IEC 6937: 1994, *Information technology—Coded graphic character set for text communication—Latin alphabet*.
- {B10} ISO 8601: 1988, *Data elements and interchange formats—Information interchange—Representation of dates and times*.
- {B11} ISO/IEC 8806: 1991, *Information technology—Computer graphics—Graphical Kernel System for Three Dimensions (GKS-3D) language bindings—Part 4:C*
- {B12} ISO 8859, *Information processing—8-bit single-byte coded graphic character sets*. (Parts 1 to 10 published.)
- {B13} ISO/IEC 9899: 1990, *Programming languages—C*.³⁾

1) DMTF documents can be obtained via the World Wide Web from <http://www.dmtf.org/>

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

3) IEC documents can be obtained from the IEC office, 3 rue de Varembe, Case Postale 131, CH-1211, Genève 20, Switzerland/Suisse.

- 33 {B14} ISO/IEC 10164-18: 1997, *Information technology — Open Systems Intercon-*
34 *nection — Systems Management — Part 18: Software Management Func-*
35 *tion.*
- 36 {B15} ISO/IEC 10646-1: 1993, *Information technology—Universal Multiple-Octet*
37 *Coded Character Set (UCS)—Part 1: Architecture and Basic Multilingual*
38 *Plane.*
- 39 {B16} ISO/IEC TR 10000-1: 1992, *Information technology—Framework and taxon-*
40 *omy of International Standardized Profiles—Part 1: General principles and*
41 *documentation framework.*
- 42 {B17} ISO/IEC JTC 1 N1335, *Final Report of ISO/IEC JTC 1 TSG-1 on Stan-*
43 *dards necessary to define Interfaces for Application Portability (IAP).*
- 44 {B18} International Organization for Standardization/Association Française de
45 Normalisation. *Dictionary of Computer Science/Dictionnaire de*
46 *L'Informatique.* Geneva/Paris: ISO/AFNOR, 1989.
- 47 {B19} IEEE Std 100-1992, *IEEE Standard Dictionary of Electrical and Electron-*
48 *ics Terms.*⁴⁾
- 49 {B20} IEEE Std 1003.0-1995, *IEEE Guide to the POSIX® Open Systems Environ-*
50 *ment (OSE).*
- 51 {B21} IEEE P1003.1a/D12, *Draft Revision to Information technology—Portable*
52 *Operating System Interface (POSIX®) Part 1: System Application Program*
53 *Interface (API) [C Language]*⁵⁾
- 54 {B22} IEEE P2003/D7, *Standard for Information Technology—Test Methods for*
55 *Measuring Conformance to POSIX®.*
- 56 {B23} IEEE P2003.2/D11, *Standard for Information Technology—Test Methods*
57 *for Measuring Conformance to POSIX®—Part 2: Shell and Utilities.*
- 58 {B24} RFC 819, Su, Z. and Postel, J. B. *Domain naming convention for Internet*
59 *user applications.*⁶⁾
- 60 {B25} RFC 822, Crocker, D. *Standard for the format of ARPA Internet text mes-*
61 *sages.*
- 62 {B26} RFC 920, Postel, J. B. and Reynolds, J. K. *Domain requirements.*
- 63 {B27} RFC 921, Postel, J. B. *Domain name system implementation schedule —*
64 *revised.*
- 65 {B28} RFC 1123, Braden, R. T. *Requirements for Internet hosts — application*
66 *and support.*
- 67
- 68 4) IEEE publications can be obtained from IEEE Publications, 445 Hoes Lane, P.O. Box 1331,
69 Piscataway, NJ 08855-1331. Telephone: 1 (800) 678-IEEE or +1 (908) 981-1393 (outside US).
- 70 5) Numbers preceded by P are IEEE authorized standards projects that were not approved by the
71 IEEE Standards Board at the time this publication went to press. For information about
72 obtaining drafts, contact the IEEE.
- 73 6) Internet Requests for Comments (RFC) are available from the DDN Network Information
74 Center, SRI International, Menlo Park, CA, USA 94025.

- 75 {B29} RFC 1514, Grillo, P. and Waldbusser, S. *Host Resources MIB*
- 76 {B30} American Telephone and Telegraph Company. *System V Interface*
77 *Definition (SVID), Issues 2 and 3*. Morristown, NJ: UNIX Press, 1986,
78 1989.⁷⁾
- 79 {B31} University of California at Berkeley—Computer Science Research Group.
80 *4.3 Berkeley Software Distribution, Virtual VAX-11 Version*. Berkeley, CA:
81 The Regents of the University of California, April 1986.
- 82 {B32} X/Open Company, Ltd. *X/Open Preliminary Specification, Systems*
83 *Management: Distributed Software Administration*. Reading, UK: X/Open
84 Company, 1995.
- 85 {B33} X/Open Company, Ltd. *X/Open Guide, Systems Management:*
86 *Identification of Management Services*. Reading, UK: X/Open Company,
87 1993.
- 88 {B34} X/Open Company, Ltd. *X/Open Guide, Systems Management: Managed*
89 *Object Guide*. Reading, UK: X/Open Company, 1993.
- 90 {B35} X/Open Company, Ltd. *X/Open CAE Specification, Systems Management:*
91 *Management Protocols API*. Reading, UK: X/Open Company, 1994.
- 92 {B36} X/Open Company, Ltd. *X/Open Guide, Systems Management: Reference*
93 *Model*. Reading, UK: X/Open Company, 1993.
- 94 {B37} X/Open Company, Ltd. *X/Open Preliminary Specification, The Common*
95 *Object Request Broker: Architecture and Specification*. Reading, UK:
96 X/Open Company, 1993.
- 97 {B38} X/Open Company, Ltd. *X/Open Portability Guide, Issue 2*. Amsterdam:
98 Elsevier Science Publishers, 1987.
- 99 {B39} X/Open Company, Ltd. *X/Open Specification, X/Open Portability Guide,*
100 *Issue 3*. Reading, UK: X/Open Company, 1992.
- 101 {B40} X/Open Company, Ltd. *X/Open Specification, XPG4*. Reading, UK:
102 X/Open Company, 1992.

103

104 7) This is one of several documents that represent an industry specification in a related area. The
105 creators of such documents may be able to identify newer versions that may be interesting.

IECNORM.COM : Click to view the full PDF of ISO/IEC 15068-2:1999

Annex B (informative)

Rationale and Notes

B.1 General

B.1.1 Scope

A number of areas are not covered in this part of ISO/IEC 15068. A few things (such as physical media) are truly outside the scope of this part of ISO/IEC 15068. However, some things are listed as either undefined or unspecified due to time and resource constraints as well as the inability to reach consensus. Areas thought to be the subject of future revisions or extensions to this part of ISO/IEC 15068 include the archiving of compressed files, partial product replacement, and end user customization. Standardizing the current content of this part of ISO/IEC 15068 was considered the most essential task, providing a basis for future implementation and for future development of standards.

The requirement for all POSIX.2 {3} utilities and all the file system features of POSIX.1 {2} generated significant discussion. The assertion was made that the underlying operating system need not be fully POSIX.1 {2} or POSIX.2 {3} conformant. With the requirement left as it is, implementation on such systems as DOS, OS/2, MVS, VMS, etc., is feasible. Some guidance, however, should be supplied by implementors of this part of ISO/IEC 15068 to those who write scripts to be packaged in distributions. It is actually those scripts that have significant dependencies on the features of the underlying operating system. Assured portability of scripts is not possible without assurance of an interpreter and utilities. By allowing other interpreters, some concession has been made to assist in managing existing software in the real world. The best portability assumption is that the **checkinstall**, **preinstall**, and **postinstall** scripts should not depend on features beyond those of POSIX.2 {3} or POSIX.1 {2}. The **configure** scripts run only on the systems that actually use the software, hence, they need not be as portable as the **preinstall** and **postinstall** scripts.

This part of ISO/IEC 15068 specifies distributed operations without specifying the mechanism for such. Clearly, this could represent a serious problem for interoperability. It was noted that the existing practices used remote procedure calls, with technologies that are not currently specified as formal standards. References to such documents were recognized as impediments to formal international standardization. The work to specify interoperability for this part of ISO/IEC 15068 is in progress at X/Open and the reader is referred to {B32}.

B.1.2 Normative References

There is no additional rationale provided for this clause.

B.1.3 Conformance

The conformance classes defined in this part of ISO/IEC 15068 are derived somewhat from the examples of POSIX.1 {2} and POSIX.2 {3}, with variations to support unique situations. This part of ISO/IEC 15068 contains no API at all, which (in the minds of many) eliminates the Application Conformance classes of POSIX.1 {2} and POSIX.2 {3}. Implementation Conformance is based on implementation of the utilities defined in this part of ISO/IEC 15068, and on the proper POSIX.1 {2} and POSIX.2 {3} support from the operating system. There is a new conformance class, Distribution Conformance, to allow suppliers of software to package their software in a conformant manner. Distributions have many of the characteristics of applications using POSIX.2 {3}, since the distributions contain executables (presumably shell scripts).

B.1.3.1 Implementation Conformance

This class requires support of all the POSIX.1 {2} and POSIX.2 {3} functionality referenced in this part of ISO/IEC 15068. The requirements from POSIX.1 {2} are primarily for hierarchical file system support, including the file attributes of owner, group, and mode. In addition, the POSIX.2 {3} utilities are required to support portable scripts.

This assures that every Conforming Implementation will be able to install any Strictly Conforming Distribution properly, including the proper settings of file attributes. One might question this need if one is installing software particular to a system that is not POSIX.1 {2} conformant. It is the pervasive ability to serve the software over a distributed file system that makes critical the need for all Conforming Implementations to understand at least one set of well specified operating system behavior. The one set of operating system behavior we have chosen is POSIX.1 {2}. The need for POSIX.2 {3} is primarily driven by the presence of executable control files within distributions. At least one guaranteed mechanism is required to invoke those files, and the shell interpreter was chosen for that purpose. Further, developers of portable scripts need a guarantee of some basic set of utilities with which to work, and the POSIX.2 {3} utilities were chosen for that purpose.

A Conforming Implementation need not include the POSIX.1 {2} and POSIX.2 {3} implementation itself, but shall document how such can be obtained for the systems that the implementation supports. It is reasonable to assume that a given implementation, conformant in the presence of proper POSIX.1 {2} and POSIX.2 {3} support from the operating system, may still operate correctly on some distributions even when the proper operating system support is not present in full or in part.

77 **B.1.3.2 Distribution Conformance**

78 **B.1.3.2.1 Strictly Conforming POSIX.7.2 Distribution**

79 The Strictly Conforming Distribution class is intended to provide the highest
80 degree of portability for a distribution. Conformance to this class guarantees that
81 any conforming implementation can install this software properly.

82 **B.1.3.2.2 Conforming POSIX.7.2 Distribution**

83 The Conforming Distribution class is intended to guarantee that any conforming
84 implementation can copy or install this software properly. This class also allows
85 for additional functionality, which may come either from implementations that
86 can take advantage of additional attributes, or from software being able to store
87 and retrieve that information from any Conforming Implementation.

88 **B.1.3.2.3 Conforming POSIX.7.2 Distribution Using Extensions**

89 This class is intended to allow evolution of this standard, but in an open, con-
90 sistent and well-documented manner. Examples of this are compressed media or
91 bootable serial media. Both of these are features were recognized as important,
92 but upon which consensus was not reached.

93 This class also provides flexibility for distributions needing to conform to other
94 constraints related to the support of POSIX.1 {2} and POSIX.2 {3}. Users among
95 the developers of this part of ISO/IEC 15068 strongly voiced the desire to support
96 interpreters other than **sh**. Support for other interpreters also permits the use of
97 such distributions on systems, such as DOS, which are not conformant with
98 POSIX.1 {2} or POSIX.2 {3}.

99 **B.1.3.2.4 Documentation**

100 There is no additional rationale provided for this subclause.

101 **B.1.4 Test Methods**

102 There is no additional rationale provided for this clause.

103 **B.2 Terminology and General Requirements**

104 **B.2.1 Conventions**

105 **B.2.1.1 Editorial Conventions**

106 There is no additional rationale provided for this subclause.

B.2.1.2 Grammar Conventions

There is no additional rationale provided for this subclause.

B.2.2 Definitions

In defining standards for software administration, the concept of roles is used to specify the way interactions occur in order for software administration to take place. Figure B-1 shows the various roles, including those that are outside the scope of this part of ISO/IEC 15068, as well as those within that scope.

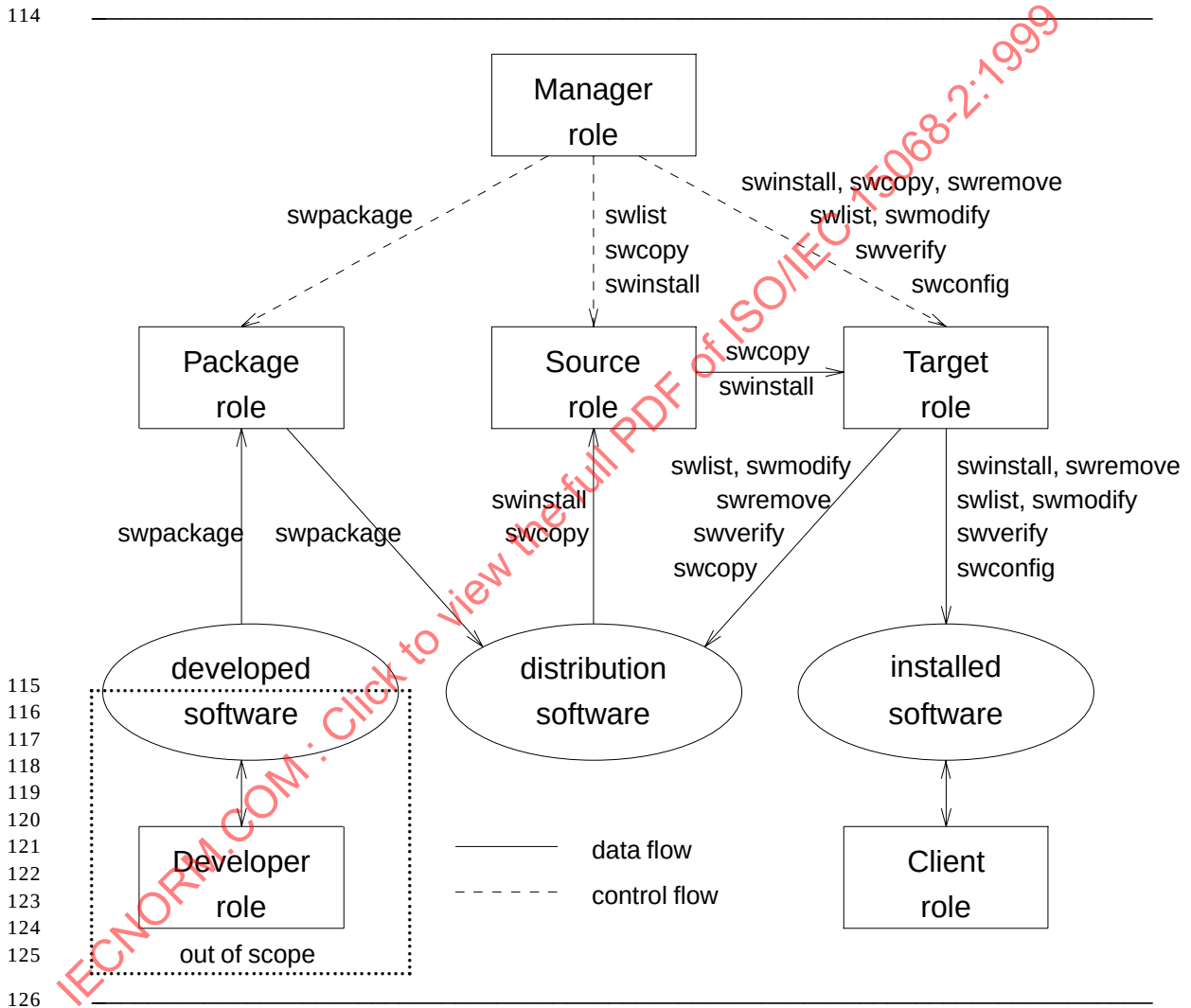


Figure B-1 – Roles in Software Administration

Distributed applications require actions to be performed in more than one place (system or directory). These distributed portions have often been referred to as client and server. Software administration tasks also are often initiated by different users at different times. Since the terms client and server have implementation implications beyond the scope of this part of ISO/IEC 15068, the more

131 neutral concept of role is introduced. This stems from a need to refer to things
132 that occur logically, if not physically, on what might be thought of as a client or
133 server. In the context of this part of ISO/IEC 15068, roles are simply a convenient
134 way of referring to where function is apparent, with no implication for how this is
135 actually implemented.

136 It may be helpful to think of roles as separate processes, one per role, but that is
137 only one possible implementation. Roles may operate on separate systems, or
138 hosts, although all roles may operate on the same host. For example, the pack-
139 ager role creates an initial distribution. When copying this distribution, the
140 source role provides read access to the distribution files, while the target role
141 writes the new copy. This new copy may then be read by another source role for
142 another install or copy.

143 For any implementation, a role consists of the entire set of tasks that may occur
144 within the role. A task is a set of well-defined behaviors and state changes in the
145 managed objects. Tasks are initiated by the system administrator using a specific
146 command in the command line interface (CLI). Tasks are defined by this part of
147 ISO/IEC 15068 in terms of the state changes on the software objects on target
148 hosts.

149 As each task proceeds, different roles are involved. These roles may be realized on
150 a single machine or could involve a different machine for each role.

151 Developer Role Where the software is developed, tested, and main-
152 tained.

153 This role is outside the scope of this part of ISO/IEC
154 15068. In Figure B-1, software is developed by the
155 development role in some environment that results in it
156 being in the developed state.

157 Manager Role Where each task is initiated and is concerned with tak-
158 ing appropriate action at the completion or failure of a
159 task.

160 Manager control is understood as a more common need
161 than target control, so at least that should be supported.
162 For this reason, the manager role sets the options for a
163 task, and each of the target hosts implements those
164 options. So, any extension involves a set of ways to
165 define selected control over particular policies. A design
166 for this has not been pursued beyond recognizing the
167 complexity of the problem.

168 The manager role provides the means of controlling the
169 way software is created, transferred, and installed. In
170 particular it provides an administrative interface to the
171 other roles, enabling their activities to be controlled in a
172 coordinated manner.

173 Packager Role Where software that has been developed is organized in
174 a form suitable for distribution.

The packager role transforms a product from the format
produced by the developer role to the format specified by

175		this part of ISO/IEC 15068 for use by the next stage of
176		the process, the source, and manager roles. The pack-
177		ager role defines the requirements for this transforma-
178		tion to be successful — the input (the product
179		specification file, and the files it describes), the com-
180		mand line interface to initiate the transformation
181		(swpackage) utility, and the output of the packaging
182		task (packaging layout).
183		Two distinct, but related, formats for packaged software
184		are supported by this part of ISO/IEC 15068 — a struc-
185		tured format residing within a POSIX.1 {2} hierarchical
186		file system (such as disk, CD, etc.), and a bit stream
187		representation residing on any serial device or file (such
188		as tape, tar archive file, etc.).
189	Source Role	Where the software exists in a form suitable for distri-
190		bution and hence forms a context for the establishment
191		of a repository of software from which the manager may
192		choose to distribute to the target.
193		Software exists in the source until it is removed by a
194		task initiated by the manager.
195		The source role provides a repository where software
196		may be stored, and provides access for those roles that
197		require the software.
198	Target Role	The target of a task.
199		For example, when installing software, the target is
200		where software is installed after having been delivered
201		from a source. As another example, the target for a
202		copy task command refers to the distribution where pro-
203		ducts are added. For management tasks like removing
204		software, the target refers to either the
205		installed_software objects or the distributions from
206		which software is being removed.
207		Part of the distributed model involves the target role
208		granting permission to the manager role to perform
209		various software administration tasks. Authority for
210		certain classes of tasks may be individually controllable,
211		for example, modifying vs. listing installed products.
212		While it is entirely conceivable that the target may want
213		to restrict the way authorized tasks are performed, it is
214		beyond the scope of this part of ISO/IEC 15068.
215	Client Role	Where the software is actually executed or used, which
216		may be different from where it is actually installed.
217		Software is configured for use on the client.
218		An example is installing for an environment where
		many hosts share software from one system. Diskless
		systems are one example of systems that do such

219 sharing. A manager role initiates the install task with
220 the source role serving the software from the distribu-
221 tion and the target role installing the software on a
222 fileserver. After the installation is complete, then a
223 client role on each client sharing this software performs
224 configuration for the shared software and the client
225 host.

226 It is important to understand the difference between the target and client roles.
227 The client role is where the software is actually used and where configuration of
228 the software takes place, while the target role is where the software is installed.
229 Although in many cases these are the same machines, in some cases they are
230 different and the separation of configuration from installation is important. Each
231 target from the *targets* operand of an install or configure task may identify a tar-
232 get role (if installing but not configuring), a client role (if just configuring), or both
233 (if installing and configuring).

234 Two examples of when these roles refer to different machines are:

- 235 — Proxy install (installing on one system for use by another system) where
236 configuration of the software is done separately, because the target may not
237 have the necessary capabilities or information, or both, for configuring the
238 client.
- 239 — The target is a file server, and there are multiple clients that access the
240 software installed on the file server. Each client may require separate
241 configuration as targets of **swconfig**.

242 Figure B-1 shows a split between the manager role and the other roles. The
243 administrative interface to software administration is provided in the manager
244 role, from which the individual tasks that take place in the other roles are con-
245 trolled.

246 This part of ISO/IEC 15068 defines a set of utilities that is such an administrative
247 interface. These utilities provide basic facilities for controlling the individual
248 tasks. Other management applications may be built that provide much more
249 comprehensive software administration facilities. This part of ISO/IEC 15068
250 defines facilities that enable management applications to control software
251 administration across any number of systems with conforming implementations.

252 One item of note among the general terms is the definition of symbolic link (see
253 2.2.2.99). While not yet standardized (see POSIX.1a {B21}), symbolic links are an
254 entrenched part of existing practice. This part of ISO/IEC 15068 makes no
255 attempt to independently define symbolic links. Rather, the functional charac-
256 teristics of symbolic links are undefined.

B.3 Software Structures

An example of the structure of the software objects for this part of ISO/IEC 15068 is illustrated in Figure B-2.

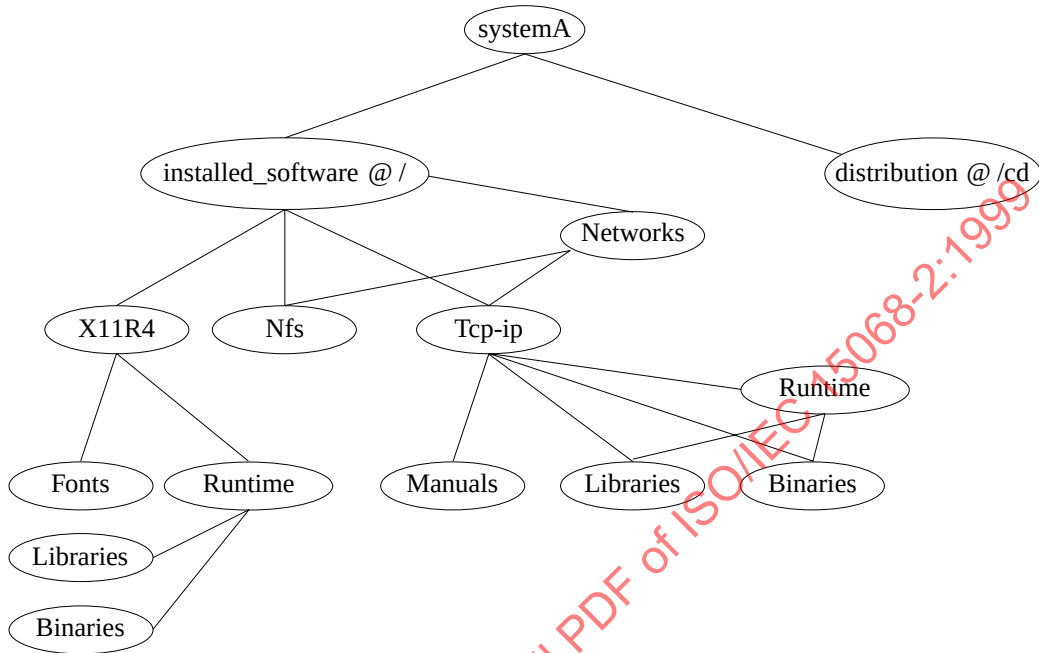


Figure B-2 – Example of Software Structure

At the top of the hierarchy is a host, which is a system that conforms to this part of ISO/IEC 15068. It is the starting point for finding all the software on that system that falls within this part of ISO/IEC 15068. A host contains software_collections:

There are two distinct types of software_collections, as listed in the following, that may exist within a conformant system:

distribution	A distribution consists of software products, in a form ready for installation. A distribution may also contain software bundles. There may be many distributions within a host.
installed_software	An installed_software object consists of products installed from a distribution. An installed_software object may also contain software bundles. There may be other installed_software objects for use by this system or for other systems.

Software is organized into a hierarchy of objects, as described in the following, that are operated on by the utilities defined in this part of ISO/IEC 15068:

product	A product consists of filesets and control scripts, plus all the associated metadata. The content of a product may be specified as a collection of subproducts, filesets, or a
---------	--

283		combination of the two.
284	bundle	A bundle is a grouping of other software objects and is a
285		convenient way to reference a set of software.
286	fileset	A fileset consists of the actual files plus control scripts.
287		Filesets are generally the lowest level of software object
288		that can be operated on by the utilities.
289	subproduct	Subproducts are a grouping of other subproducts, or of
290		filesets, or of some combination, that resolve to a group
291		of filesets. Subproducts are a convenient way to aggregate
292		filesets.
293	The <code>software_files</code> define the files and <code>control_files</code> that are contained in the	
294	software objects that are operated on during a software administration utility.	
295	There are two classes of <code>software_files</code> as described in the following:	
296	control_file	Control_files consist of control scripts and other files
297		that are used in various ways by the utilities. Control
298		scripts are executed by the utilities at various points in
299		a task. Control scripts provide a way to perform steps,
300		in addition to those executed by the utilities, at various
301		points in the task such as preinstall checking, postin-
302		stall customization, configuration, and verification.
303		Either a single script with multiple entry points, or mul-
304		tiiple scripts can be defined.
305		Most control scripts are run on the target, which may be
306		a different architecture than the client on which the
307		software operates. They should, therefore, use
308		POSIX.2 {3} utilities, except where they can determine
309		that they are running on the client.
310		In addition to scripts, other <code>control_files</code> provide input
311		to the control scripts, or to the utilities directly (e.g., the
312		response and space <code>control_files</code>).
313	file	Files are the lowest level of object defined by this part of
314		ISO/IEC 15068. Files contain the attributes describing
315		the file including the contents of the file and its installed
316		location.
317	The <code>distributions</code> and <code>installed_software</code> objects are the sources or targets of a	
318	software administration command. The software objects (products, filesets, bun-	
319	dles, and subproducts) are the objects that are being applied to those targets.	
320	This part of ISO/IEC 15068 describes the structure and the attributes for	
321	<code>software_collections</code> , software objects, and <code>software_files</code> . It also describes the	
322	behaviors for the utilities that operate on these objects. However, these structure	
323	definitions are not managed object classes in the ISO sense because the behaviors	
324	are not described in terms of methods within object classes.	
325	NOTE: Object classes are templates for the creation of object instances. They are analogous to the	
326	definition statements used in programming languages to define data structures that will be created	
	later. Objects contain more than data structures, in that they also possess methods (procedures	
	that are executed by objects). A well-formed object class has methods defined that handle all object	

327 data manipulation, including creation, modification, and listing, so that the actual storage of the
328 data is appropriately hidden from the application using the objects.

329 Figure B-3 shows the components of the software object hierarchy. The contain-
330 ment arrows designate objects that are defined within the context of their contain-
331 ing objects. An object can only exist within one containing object. The identifier of
332 an object (for example, the *tag* attribute of a fileset) only needs to be unique within
333 the scope of the containing object.

334 The reference arrows designate objects that are included when this object is
335 operated on. An object may be referred to by more than one object. Bundles need
336 not refer to entire products, but can refer to individual filesets or subproducts.
337 Fileset and subproduct objects can be referenced directly by bundles by also iden-
338 tifying the product of which the fileset or subproduct is a part.

339 Figure B-4 shows the software administration common classes and the software
340 objects that inherit attributes from these common classes.

341 Interoperability between implementations of this part of ISO/IEC 15068 may be
342 achieved through the definition of methods for the first two of these common
343 classes, *software_collections* and *software*. The *software_collections* are the
344 source and target objects for software administration, while the *software* objects
345 are the objects that are operated on within the context of the *software_collections*.
346 Operations on individual *software_files* independent of operations on *software*
347 objects is undefined.

348 This part of ISO/IEC 15068 also does not define how remote file systems are
349 managed. In the simplest case, each file system is “local” to a single host, and all
350 installations may be directed to the file system through an agent process on that
351 host. Thus, files on a file system are contained within one of the installed
352 software collections contained below that host. On the other hand, an implemen-
353 tation may also choose to allow installation to a remote file system over a remote
354 file system protocol. That is, the target process is running on a host that is
355 different from the one that contains the file system. In this case, the files on that
356 file system may be contained within the same software collection as before, or may
357 be contained within a local software collection. In another implementation, all
358 software collections may be stored within a global naming service instead of below
359 any particular host.

360 An implementation may choose to define a software host object, or manage
361 software as part of a more general host object. The attributes of a host object that
362 are of interest to this part of ISO/IEC 15068 are shown in the following table:

363 The following are the attributes of the hosts that contain *software_collections*
364 managed by this part of ISO/IEC 15068:

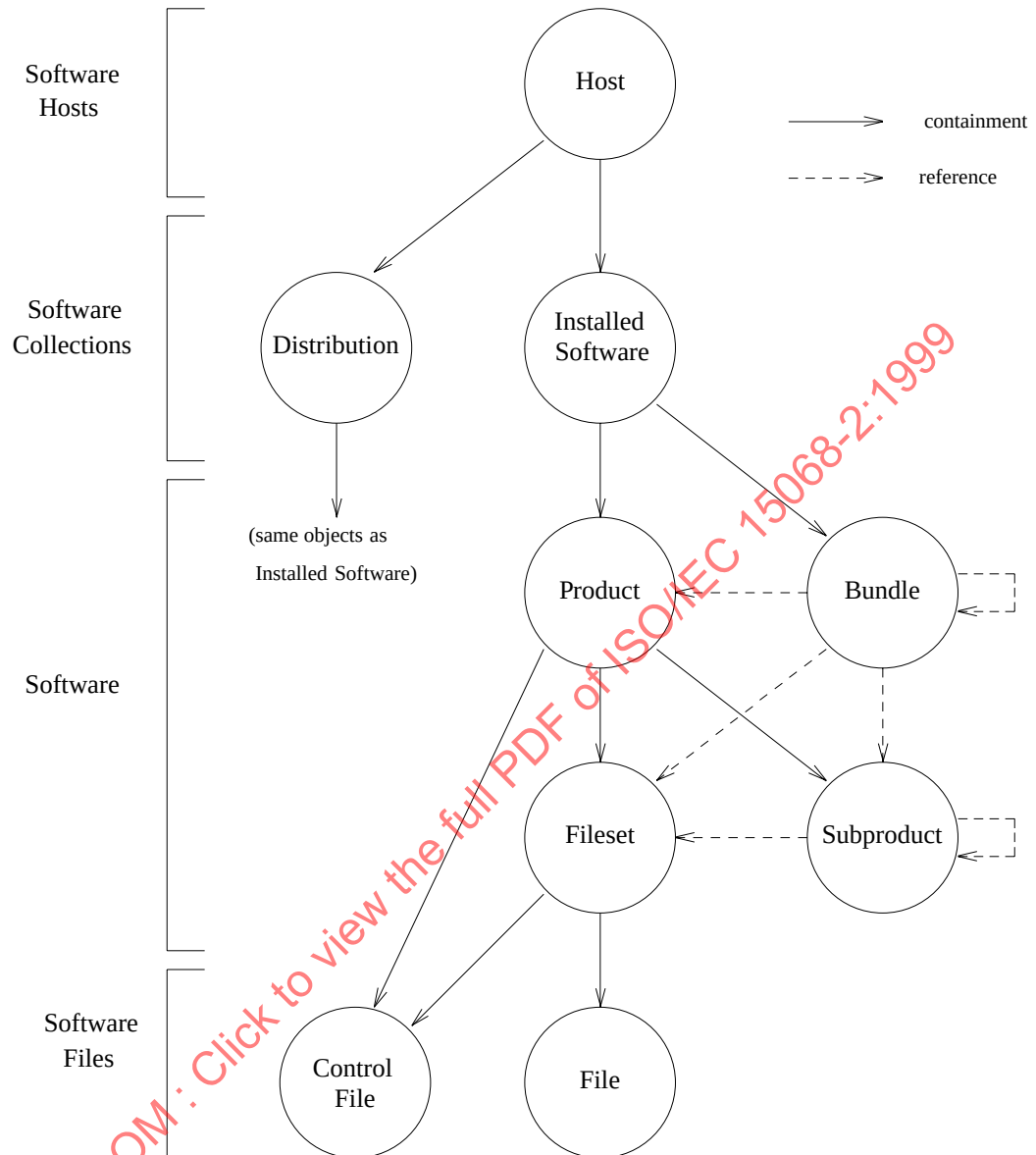
365 *distributions* The list of *distribution.path* attributes for distributions in the
366 software host object.

367 These describe the **PATHNAME** portion of a *software_collection*
368 source or target.

369 *host* Identifier used to specify the host portion of a software source or
370 target.

Identification of a remote host system is dependent on the net-
working services implementation and thus the syntax and

371

372
373374
375376
377

378

Figure B-3 – Software Object Containment

semantics of the host name is undefined within this part of ISO/IEC 15068.

installed_software

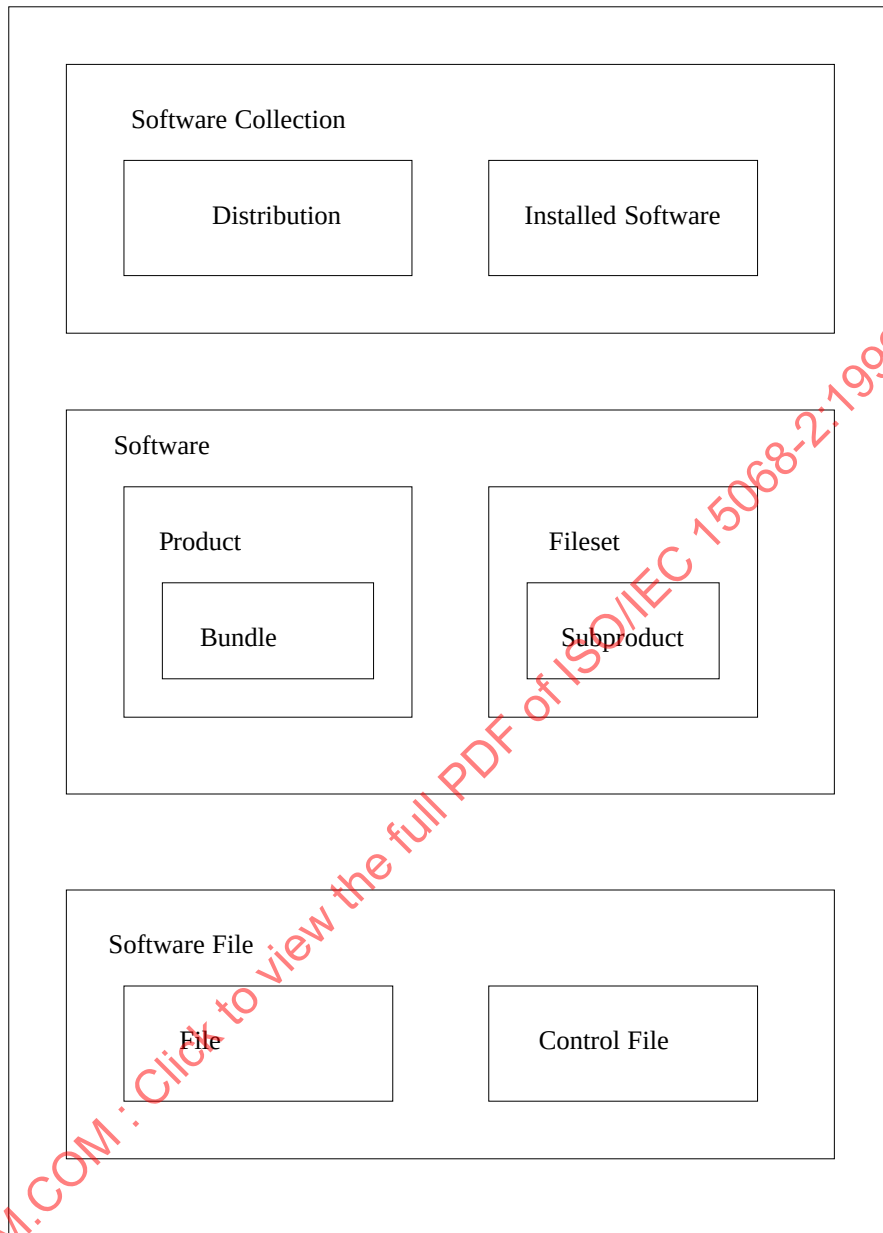
The list of *installed_software.path* and *installed_software.catalog* attributes for *installed_software* objects in the software host object.

These describe the **PATHNAME** portion of a *software_collection* target.

machine_type

Corresponds to the *machine* member of the *uname()* structure

388



389

390

391

Figure B-4 – Software Object Inheritance

defined in 4.4.1 of POSIX.1 {2}.

It is the hardware type on which the system is running.

os_name Corresponds to the *sysname* member of the *uname()* structure defined in 4.4.1 of POSIX.1 {2}.

It is the name of this implementation of the operating system.

os_release Corresponds to the *release* member of the *uname()* structure defined in 4.4.1 of POSIX.1 {2}.

Table B-1 – Possible Attributes of a Host Class

Attribute	Length	Permitted Values	Default Value
<i>host</i>	Undefined	Portable character set	None
<i>os_name</i>	32	Portable character set	None
<i>os_release</i>	32	Portable character set	None
<i>os_version</i>	32	Portable character set	None
<i>machine_type</i>	32	Portable character set	None
<i>distributions</i>	Undefined	List of distribution directories	Empty list
<i>installed_software</i>	Undefined	List of installed_software directories and catalog identifiers	Empty list

It is the release level of the operating system implementation.

os_version Corresponds to the *version* member of the *uname()* structure defined in 4.4.1 of POSIX.1 {2}.

It is the version level of this release of the operating system.

B.3.1 Software_Collection

This class definition exists for convenience in defining the classes that inherit from it. It is not intended that any direct instances of this class be created, but only of the classes that inherit from it.

Multiple versions of products and bundles are possible when subsequent releases of a product or bundle have different revision numbers, and when products or bundles targeted for different machine types or other OS attributes define the architecture attribute differently.

The *layout_version* attribute is the version number of this part of ISO/IEC 15068 to which the distribution conforms. The name of this part of ISO/IEC 15068 (e.g., P1387.2-19xx) was considered but there was concern that the delay between IEEE acceptance and ISO acceptance would make it hard to pick the year correctly. It is not clear when to change the number from 1.0 to 1.1 or even from 1.x to 2.0.

It is possible for an **INDEX** file describing a distribution to contain products with different values of *layout_version*. The software_collection *layout_version* refers only to the format of the distribution attributes and the **product** keyword. After the **product** keyword, the product *layout_version* defines the format of the definitions of all objects within that product.

B.3.2 Distribution

POSIX.1 {2} allows for different pathname and filename sizes. Thus it is possible for a distribution to be created on one system and not be readable or installed on another system (each of which conforms with this part of ISO/IEC 15068) because of differences in their POSIX.1 {2} {NAME_MAX} and {PATH_MAX}. Consideration was given to attributes defining the longest sizes of file names and paths on a distribution, but these were not included since their use could neither ensure failure nor success of installing or copying a particular product from the distribution. Another issue implementors should consider is the maximum name and

438 path that may be contained within a supported archive.

439 The need for the *media_sequence_number* attribute is to number the tapes (or
440 disks or whatever) if a distribution is on more than one of them. If there is only
441 one, then its number is 1.

442 The following attributes at one point were listed as distribution attributes. How-
443 ever, it was determined that the only time it could be guaranteed that these attri-
444 butes were accurate was for an initial distribution definition. As soon as a
445 **swcopy** or **swremove** operation occurred on a distribution, the attributes could
446 be invalid because it would be impossible to modify these attributes in any logical
447 manner based on the operation. It is recognized that these attributes are valuable
448 and many vendors may choose to put them in as vendor extensions.

449 *tag* A short name associated with the distribution, used for selecting
450 the distribution from the command line

451 *title* A longer name used for display purposes.

452 *description* A more detailed description of the contents of the distribution.

453 *revision* A revision associated with the distribution.

454 *media_type* Describes the type of media being used (e.g., CD-ROM, 8 mm,
455 etc.)

456 *copyright* The copyright notice for the distribution.

457 *create_time* The date, in seconds since the Epoch, when the distribution was
458 made.

459 *number* The vendor part number for the distribution.

460 *architecture* A sequence of characters used by a vendor to describe the
461 machine or product.

462 This is presumably more “user friendly” than the values
463 returned by the **uname** utility.

464 Usually distributions will be created upon creation of the first product with
465 **swpackage** or **swcopy**. Usually distributions will be removed as a part of remov-
466 ing the last product with **swremove**. An implementation may choose to provide
467 more explicit control for creation and deletion of empty distributions. The
468 **swcopy** and **swremove** utilities should be used for this purpose. The **swmodify**
469 utility may also be used.

470 **B.3.3 Media**

471 There is no additional rationale provided for this clause.

472 **B.3.4 Installed_Software**

473 This class definition exists for convenience in defining the classes that inherit
474 from it. It is not intended that any direct instances of this class be created, but
475 only of the classes that inherit from it.

476 The `installed_software` catalog may be located by something as simple as a path-
477 name where the catalog is stored as a file, or it could be located in a more compli-
478 cated fashion such as with a key from a directory service used to identify all or
479 part of a database.

480 **B.3.5 Vendor**

481 The *vendor.tag* attribute is intended to distinguish software objects from different
482 vendors that happen to have the same *product.tag*. A vendor should attempt to
483 choose a *vendor.tag* that is unique among all vendors.

484 **B.3.6 Software**

485 This class definition exists for convenience in defining the classes that inherit
486 from it. It is not intended that any direct instances of this class be created, but
487 only of the classes that inherit from it.

488 This standard has defined four related software objects — products, filesets, bun-
489 dles, and subproducts. See Figure B-4. Implementations are encouraged to
490 present these to the user as hierarchy of similar “software” objects, and to actually
491 implement these so that they differ only as needed. That is to say, an implemen-
492 tation should use inheritance from a common class as much as possible. The
493 rationale for the four differently named software objects is as follows:

- 494 — Products and filesets are concepts firmly entrenched in existing practice.
495 All of the many practices that have contributed to this standard have
496 included these two levels. Manageable software objects necessarily includes
497 some files to manage. This is the basis of a software product. Additionally,
498 most application software has both required and optional pieces, so often
499 only a subset of the product may be installed. Thus, a fileset is chosen as a
500 “set of files” and a product is a collection of filesets that have a number of
501 shared attributes, and are distributed in a single distribution (usually from
502 a single vendor).
- 503 — It was agreed that a “recursive notational convenience” was very desirable.
504 Additionally, many (but not all) existing practices had realized the need for
505 various overlapping groupings of software into new “configurations.” Bun-
506 dles and subproducts are merely “macro” or “recursive” products and
507 filesets, respectively. Just as products and filesets are a bit different, the
508 use of bundles and subproducts are a bit different. Bundles provide a way
509 to make products out of existing products or parts of products. Subproducts
510 provide a way to provide selectable units that may overlap in fileset con-
511 tents. For example, a fileset may be part of “runtime” support as well as
512 “development” environment subproducts. Finally, bundles and subproducts
513 are recursive in that they may contain other bundles and subproducts,
514 respectively.
- 515 — The containment of filesets and subproducts within products allows for
516 derived naming of components of a product — that is, a simple *tag* for a
517 component relative to a more complex name (*tag*, *revision*, *vendor_tag*,
architecture) for a product. In addition, this leads to distributions with a
simple directory structure for filesets within products.

The need to localize the following descriptive software and vendor attributes was recognized — *title*, *description* and *copyright*. However, since the existing practice for localization of software information files in portable media is immature, this has been deferred to a possible future revision of this part of ISO/IEC 15068.

Until a future revision of this part of ISO/IEC 15068 addresses localization, one recommended way to internationalize these attributes is to create vendor-defined attributes with the format

keyword.<LANG>

where keyword is "description", "title", or "copyright", and <LANG> is the value of the **LANG** environment variable. An implementation should then recognize if **LANG** is set to a value other than its default and search for a corresponding attribute. If that attribute does not exist, then the default one will be used. For example

product

tag GreatProduct

title "This is great!"

title.FRENCH "C'est magnifique!"

title.GERMAN "Sehr gut!"

description"Long boring paragraph why this is great"

description.FRENCH "..."

description.GERMAN "..."

...

Note that the *tag*, *revision*, and other attributes that affect the defined behavior of the implementation, shall not be internationalized. For this revision of this part of ISO/IEC 15068, this includes all defined attributes except *title*, *description*, and *copyright*.

The size for the software may be larger than that supported by the POSIX.1 {2} *size_t* structure since software can contain many files. It is recommended that an implementation allocate at least 64 b for the internal storage of the software *size* attribute.

An effort has been made to support the attributes needed by the Desktop Management Task Force (DMTF) {B1}. and the Internet Host Resources MIB (RFC 1514) {B29}. The information requested by the two is similar. This discussion only mentions one of the two (DMTF), but is applicable to both.

The DMTF Component ID Group contains the required attributes. The mapping between software attributes and those for the Component ID Group can be derived as shown in the following table:

Table B-2 – Mapping from Software to DMTF Component ID

DMTF Attribute	POSIX.7.2 Attribute	POSIX Example
Manufacturer	vendor_title	Any Computer System, Inc.
Product	title	Widget Maker
Version	revision	A.10.0
Serial Number	number	B1234-13245
Installation	create_date	199306291000.00
Verify	(swverify)	0

Note that *create_date*, shown in Table B-2, is not really an attribute. However, it behaves as as an attribute when executing **swlist -a create_date**.

565 Tying DMTF Verify to **swverify** (and **swlist**) execution on the software object
566 seems closest to the intention of DMTF. DMTF defines the following values:

567	Value	Meaning
568	-----	-----
569	0	an error occurred; check status code
570	1	component does not exist
571	2	verify not supported
572	4	component exists, functionality untested
573	5	component exists, functionality unknown
574	6	component exists, functionality no good
575	7	component exists, functionality good

576 The **swverify** command can generate return codes for DMTF value 6 (return
577 code 1) and value 7 (return code 0). The **swlist** command can be used to test for
578 existence, DMTF 1. A lack of **verify** scripts could be related to DMTF 2 or
579 DMTF 7.

580 B.3.7 Products

581 The value of the *revision* attribute is interpreted as a . (period) separated string,
582 as defined in 3.7 and further in 4.1.4.1. This definition permits the use of such a
583 string, but does not require it. The string can be constructed entirely without the
584 use of periods. An example of the comparison is

585 A1.003.01 < A.004.00 < B.000.00
586 A1_003_01 < A_004_00 < B_000_00
587 First < Second < Third
588 First < Fourth < Second

589 Historically, some implementations computed the value of *instance_id* sequen-
590 tially, while other implementations have used an algorithm based on the product
591 *tag*, *vendor_tag*, and the various machine type attributes. No implementation is
592 specified, other than to guarantee that the *tag* and *instance_id* uniquely identify
593 the product within the distribution or installed_software object. This is to make it
594 easier to specify a particular product when there are other products sharing the
595 same *tag* as would be the case when there are different product instances in a dis-
596 tribution for several machine types or multiple concurrent versions on a host.

597 The *vendor_tag* attribute is intended to be universally unique to distinguish pro-
598 duct and bundle software objects that otherwise would be treated as the same
599 object if the *tag*, *revision*, and *architecture* attributes were the same. Guarantee-
600 ing universal uniqueness is difficult at best, and no need was seen at present to
601 cause the value of *vendor_tag* to be either some sort of machine-generated univer-
602 sally unique value or officially registered.

603 Multiple versions of the “same” product or bundle (ones with the same value for
604 the *tag* attribute) is supported by each version possessing values of the version
605 distinguishing attributes unique within that installed software catalog.

606 The *architecture* attribute should include information related to four *uname()*
607 structure members. The *architecture* attribute is needed for **software_specs**
608 since the patterns used for determining compatibility in the attributes related to
uname() can be somewhat complex and contain patterns, while **software_specs**
themselves can contain patterns.

609 It is recommended that a set of guidelines be used for the architecture attributes
610 to maintain a consistent “syntax” for related architectures. This increases the
611 usability of this field for users selecting software. An example guideline is to order
612 any information contained in the value of the attribute in a consistent way,
613 separated by a consistent delimiter. For example

614 **architecture sunos_4.1_sun4**

615 for a product with the attributes

616 **os_name sunos**
617 **os_rev 4.1.***
618 **os_ver ***
619 **machine_type sun4***

620 Another example is

621 **architecture hp-ux_9_pa-risc**

622 for a product with the attributes

623 **os_name hp-ux**
624 **os_rev 9.*|10.***
625 **os_ver [a..e]**
626 **machine_type 9000/[6..8]???**

627 Product machine attributes describe the target systems on which this product
628 may be installed. Each of these keywords are related to a POSIX.1 {2} *uname()*
629 member and may be defined as a simple string, or a software pattern matching
630 notation. How compatible software is determined depends on whether the pro-
631 ducts are being installed on the system that will be using them, or whether the
632 installation will be used by other systems with perhaps different attributes.

633 If a *uname* attribute is undefined, the behavior is essentially the same as if it were
634 defined to be *** (meaning compatible with all systems).

635 The product directory for an application should be the directory that is part of all
636 paths in the product. Thus, if an application has three filesets that contain files
637 below **/appl/console**, **/appl/agent**, and **/appl/data** respectively, the
638 *product.directory* attribute should be set to **/appl**. If a user relocates the product
639 with a command like

640 **swinstall appl,r=1.0,l=/disk2/appl**

641 then all three filesets have the same location attribute. If the user relocates the
642 product to three different locations

643 **swinstall appl.console,r=1.0,l=/disk1/appl**
644 **swinstall appl.agent,r=1.0,l=/disk2/appl**
645 **swinstall appl.data,r=1.0,l=/disk3/appl**

646 then each fileset will have a different location attribute. There will be three pro-
647 duct instances containing the three filesets (since products versions are dis-
648 tinguished by location), but the user can still identify all three filesets as one with
649 the specification

650 **swverify appl,r=1.0,l=***

651 Alternatively, the user could relate all these locations with the same version
qualifier, such as **"q=current"** as follows:

swinstall appl.console,r=1.0,l=/disk1/appl,q=current

```

652      swinstall appl.agent,r=1.0,l=/disk2/appl,q=current
653      swinstall appl.data,r=1.0,l=/disk3/appl,q=current
654  And subsequently identify all pieces with
655      swverify appl,q=current

```

656 The *postkernel* attribute supports the ability to install one operating system in
 657 proxy (to an alternate root) by another implementation that does not understand
 658 that operating system. All products that contain kernel filesets that will be
 659 installed into the same installed_software object should have the same path
 660 defined. There should be one core OS kernel fileset that includes this path in its
 661 set of files so that it has been installed by the time the *postkernel* script is exe-
 662 cuted.

663 In general, a product with no **preinstall** or **postinstall** scripts is recover-
 664 able. However, if there are **preinstall** or **postinstall** scripts, then
 665 **unpreinstall** and **unpostinstall** scripts shall be provided if any steps need
 666 to be undone to support autorecovery.

667 There was an issue whether dependencies should be an attribute of a product.
 668 The following types of dependencies have been discussed:

- 669 — Fileset to fileset within a product
- 670 — Fileset to (some other) product
- 671 — Fileset in one product to fileset in another product
- 672 — Product to product
- 673 — Product to fileset in some other product
- 674 — Product to fileset in that product (essentially mandatory fileset)

675 The last three dependency types are not necessary if the first three types exist
 676 (which they do), since those dependencies can be specified in terms of the others.
 677 For example, if an entire product depends on a second product, then the second
 678 product can be defined as a dependency for all filesets in the first product.

679 The developers of this part of ISO/IEC 15068 recognized that numerous additional
 680 dependency requirements are possible, particularly for software updates. These
 681 may be handled via **checkinstall** scripts, and can be considered for future revi-
 682 sions of this part of ISO/IEC 15068.

683 The intention behind the inclusion of the *layout_version* attribute within a pro-
 684 duct is that it be required if its value is different than that for its associated
 685 software_collection.

686 **B.3.8 Bundles**

687 Bundles serve two purposes — they allow the software supplier to group different
 688 subsets of products into new configurations or products, and they allow the
 689 software administrator to build useful groups of software (configurations) from
 690 already defined bundles and products.

691 The bundle class does not have *location* or *directory* attributes. This is because
 692 **software_specs** within the bundles can refer to products with different default
 directory attributes or even products that have been relocated.

Bundles have “uname” attributes that only have any value if the bundle aggregate has a different compatibility than that of any of its contents. Besides offering more control to the person defining the bundle, it is useful in a GUI that wants to only display compatible software by default. For example, a bundle may contain one product that operates on a system with an uname attribute of “A” and another product that operates on systems with uname attributes of “A” or “B”. In this case, it might be useful to define the bundle attribute to be “A”. Since it is possible that not all the bundles contents exist in a particular distribution or installed_software object, it may not be possible to determine the compatibility of the bundle in all cases unless the bundle attributes are also defined.

The *vendor_tag* attribute is intended to be universally unique to prevent naming clashes for similarly named products and bundles from different vendors. Guaranteeing universal uniqueness is difficult at best; it was deemed unnecessary at present to cause the value of *vendor_tag* to be either some sort of machine-generated universally unique value or officially registered.

The intention behind the inclusion of the *layout_version* attribute within a bundle is that it be required if its value is different than that for its associated *software_collection*.

The value of the *bundle.contents* attribute is not modified when a location is specified for a bundle, allowing future resolutions of its contents to remain consistent. For example, assume bundles “CAT” and “DOG”, and products “FOO” and “BAR”, all with directory attributes defined as “/”.

```
bundle
    tag CAT
    contents DOG, l=/dog BAR, l=/bar
bundle
    tag DOG
    contents FOO, l=/foo
```

When the bundle “CAT” is installed and relocated to */cat*, the following objects are installed:

```
CAT, l=/cat
DOG, l=/cat/dog
FOO, l=/cat/dog/foo
BAR, l=/cat/bar
```

So, when resolving “CAT, l=/cat” in installed software, applying the proper locations to the *software_specs* in the contents will result in the same *software_specs* in the installed software.

Bundle definitions are only copied or installed when explicitly specified since they are external to the product and not always applicable to the use of the product installed. The creator of a product has no control over what bundles reference it. For example, a product may be a member of numerous bundles, and many of those bundles will likely have nothing to do with the bundles and products chosen to be installed. Also, see B.3.10.

Bundles and subproducts have lists defining their contents that are always copied (*contents* is a static attribute). So, if a partial bundle or product is copied, the value of the *contents* attribute does not change. However, by comparing that attribute to what objects are actually installed, “completeness” of a bundle or subproduct can be determined.

740 **B.3.9 Filesets**

741 The *media_sequence_number* is used for serial distributions to describe which
742 media the archive containing the fileset starts on. There is generally one archive
743 per media, unless a fileset is larger than a media. Each media has a unique
744 sequence number whether it begins an archive or continues a previous one.

745 At one point a fileset *class* attribute existed that could contain the value of recom-
746 mended, mandatory, or optional. The attribute was removed because it was felt
747 that this part of ISO/IEC 15068 could not specify any behavior for the attribute. It
748 would be possible to make a specific fileset mandatory by having all other filesets
749 in that product specify it as either a prerequisite or corequisite.

750 Another way to handle recommended, mandatory, or optional filesets would be to
751 create subproducts with *tags* of the appropriate names. Although this part of
752 ISO/IEC 15068 does not specify any behavior based on the name of *subproduct.tag*,
753 a specific implementation could define behavior as an extension.

754 **B.3.10 Subproducts**

755 Unlike bundles, subproduct definitions (that are internal to a product) are copied
756 or installed when any fileset specified in the *contents* attribute of the subproduct
757 is copied or installed. Products are meant to be sets of related software and are
758 usually created and managed by one person or organization. Additionally, subpro-
759 ducts are normally used to specify useful subsets of filesets within the product,
760 which in turn are useful for dependencies. With subproducts, the “parts make up
761 the whole.”

762 **B.3.11 Software_Files**

763 This class definition exists for convenience in defining the classes that inherit
764 from it. It is not intended that any direct instances of this class be created, but
765 only of the classes that inherit from it.

766 The *compression_type* attribute allows compressing and uncompressing of indivi-
767 dual files during **swcopy**, and uncompressing during **swinstall**. The way in
768 which an implementation uses this attribute is undefined, although the general
769 thought was that this would normally be the name of a
770 compression/uncompression routine with a simple interface. An implementation
771 should be flexible in locating routines specified by *compression_type*, utilizing any
772 or all of the following:

- 773 — Built-in knowledge of the *compression_type* format for compressing and
774 uncompressing
- 775 — The product control directory for a program named in *compression_type*
- 776 — **PATH** on the target system for a program named in *compression_type*

777 No particular compression method is specified in the standard largely because the
778 developers of this part of ISO/IEC 15068 saw no standard for file compression and
779 did not want to specify all of the details of the compression methodology as part of
this part of ISO/IEC 15068. It was generally agreed that to achieve adequate
interoperability, a single method of consensus should be supported by all

implementations. It is likely that the format used by the **gzip** utility is appropriate for all implementations. Each implementation may support any number of other methods.

The interface to the compression routine was also left unspecified. It is recommended that input be taken from stdin and output be directed to stdout, that the routine operate with no option to imply compress, and that a **-u** option imply uncompress. However, specific compression routines may require more complex interfaces.

The group also considered archiving of compressed files, i.e., concatenation or other combination into a single file. The main purpose of this would be to save cluster space on diskette distributions. It was finally decided that the risks for current standardization were too high — especially if an archive extended over more than one diskette — and the issue was left implementation dependent. In implementing this, there should be consideration of the following factors:

- A new *archive_source* attribute to indicate that the file contents are within a named archive.
- Defining a new fileset *archive_type* attribute with values of empty string, **cat**, or the name of an archive routine like **tar**. The type **cat** indicates simple appending to an archive file. If **cat** (or even possibly **tar**) were used, an *archive_offset* attribute would indicate where within the archive the file started. This could be used for fast single-file extraction using either *size* or *compressed_size*.
- Extended options on **swcopy** for *archive_files* and *archive_type* (similar to *compress_files* and *compression_type*). The *uncompress_files=true* option on **swcopy** would both unarchive and uncompress.
- An archiver interface that permitted appending or extracting one file at a time.
- The archiver, like the compressor, could be distributed in the product control directory.

Finally, compression support for **swpackage** was considered, and deemed as unnecessary, since compression can be achieved by copying after packaging. But an implementation can easily add attributes to achieve this function.

B.3.12 Files

The letters chosen for the file *type* attribute are consistent with the syntax of the **find** utility with the **-type** option, as defined in POSIX.2 {3}. Hard links are not specifically mentioned in 5.6.1.1 of POSIX.1 {2}. Symbolic links are not mentioned in POSIX.1 {2} but are included to support existing practice. Work to standardize symbolic links is included in POSIX.1a {B21}.

Implementations running on operating systems that do not support a POSIX.1 {2} file system can interpret the defined attributes in any appropriate way. See 1.3.1.1. Any implementation can extend file attributes with additional attributes appropriate to the file system in question. To avoid confusion when defining new attributes for a particular file system, it might be best to prefix such attributes with a designator of the file system. An example, for a FAT file system, might be

822 the attributes *FAT_Hidden* and *FAT_ReadOnly*.

823 There was some debate whether the *major* and *minor* attributes are appropriate
824 or not since there is no standard that specifies how these files are created. In
825 addition, this part of ISO/IEC 15068 specifies that the serial distribution be in
826 POSIX.1 {2} **cpio** or **tar** format; however these attributes are biased towards **tar**
827 format as opposed to **cpio** format.

828 Other SVR4 supported file types are **f**, **d**, **l**, **s** (like SDU RF, DR, HL, SL); **b**, **c**, **p**
829 (special files and pipes); and **e**, **v**, **x** (editable and volatile files; and exclusive direc-
830 tories).

831 Considered was a size file type (**z**) that was removed in favor of the **space**
832 *control_file* similar to SVR4. An implementation may choose to internally imple-
833 ment a size type or a separate *size_file* object to represent the data from this file.

834 The developers of this part of ISO/IEC 15068 considered an *is_exclusive* (directory)
835 attribute that was removed due to objections that the utilities would remove files
836 that they did not have recorded in their database. Also, this was not a common
837 need, and can be implemented either as vendor extension or by having software
838 **fix** script implement similar functionality.

839 There has been much discussion about compression being handled within the
840 scope of this part of ISO/IEC 15068. Currently there are ways that both implemen-
841 tations and individual software products can handle compression. Compression
842 can be handled through cooperation of the source and target roles, if they are from
843 the same implementation. Software vendors can choose to ship their files
844 compressed and uncompress them as part of the **postinstall** script. They can
845 add a **space** *control_file* to account for the extra space required.

846 A similar need would apply to other post processing, such as for ANDF files that
847 are processed as part of **postinstall** or configuration.

848 Though it may be adequate for protecting against accidental damage, the existing
849 POSIX.2 {3} **cksum** is considered inadequate for virus protection. Implementa-
850 tions may wish to create additional vendor-defined attributes and utility behaviors
851 for this purpose.

852 Each of the prerequisite or corequisite **dependency_specs** in the list is required
853 to resolve successfully in order for dependencies to be met. Also, a
854 **dependency_spec** can contain alternate **software_specs** separated by the |
855 (vertical line) character (see 4.1.4.1). So, if a fileset has a corequisite dependency
856 on software, expressed with a Boolean equation (**A|B|C**)&(**D|E**), this can be
857 specified in a PSF as

858 **corequisite A|B|C**
859 **corequisite D|E**

860 There are files (particularly for OS software such as **/etc/rc** for SVR4 and
861 **autoexec.bat** for DOS) that are modified between software update times.
862 These may be termed modifiable files. Although OS modifiable files are slowly
863 being replaced by mechanisms where applications can simply add their own
864 requirements as separate read-only files in a particular directory, there currently
865 would be some value in supporting features where modifiable files are compared
with the original files to see what changes need to be applied during software
updates. Actually implementing these changes is a more difficult problem since it

requires knowledge of the formats of the files being updated. Similarly, reversing (during **swremove**) changes made to modifiable files (during **swinstall** and **swconfig**) is an exceedingly difficult problem. The existing practice for treating modifiable files is fairly ad-hoc. It was not feasible to address all of the possible needs for updating modifiable files. Instead, it does provide the attribute *is_volatile* for files that may be modified after installation, and leaves the rest of the treatment of modifiable files as either implementation defined, or handled in control scripts. This area may be considered for a future revision of this part of ISO/IEC 15068.

875 **B.3.13 Control Files**

Using *tags* as the identifier of when a script should be executed (independent of the path the script is stored as) allows anywhere from one file per *tag* to one file for all *tags*. A concern on PC or DOS systems is that requiring more than one control script for all *tags* is a space problem. Instead, software vendors might prefer a single master script that took care of all needs. Multiple scripts are also supported, since many software vendors favor this approach over a “mega-script.” However, other vendors may prefer the single script approach, especially to save space if there are many scripts defined for this product that share a lot of the same code.

Control files do not have *mode*, *owner*, *group*, *uid*, *gid*, and *mtime* attributes since they are not necessary for the execution of the control scripts or for the management of these files within the distribution or installed software catalog. However, an implementation shall ensure that they are executable.

The *interpreter* attribute has two uses. It is useful for those who choose not to use the POSIX.2 {3} shell, i.e., **sh**. It is also useful for systems that would not otherwise require POSIX.2 {3}. Those creating distributions and control files are encouraged to use the POSIX.2 {3} shell for portability.

893 **B.4 Software Administration Utilities**

Software administration involves the control of software throughout the software life cycle from the organization or creation of a software object through its installation, maintenance phases, and eventual removal.

The following tasks are identified in this part of ISO/IEC 15068. The defined utilities provide a way of accomplishing these tasks except as noted.

899 **Install software (**swinstall**)**

This task takes software from a source distribution and installs it on a target file system in a form suitable to be configured on this system or another system sharing this software. Parts of software products (subproducts or filesets) can be installed or reinstalled at different times.

In the case where the system on which the software is installed will also be using the software (i.e., it is acting as both a target and client role), configuring the

907 software can be combined with the install software task.

908 **Reinstall software (swinstall)**

909 This task is simply installing the exact same software

910 that was previously already installed.

911 **Configure software (swconfig)**

912 This task takes place on the client role that will be

913 using the installed software. Configuration makes that

914 software ready to use. Configured software can also be

915 reconfigured as required or can be unconfigured (to

916 deactivate a particular version or prepare it for remo-

917 val).

918 **Update software (swinstall)**

919 This task updates the target file system by installing a

920 newer revision of software than is already installed.

921 This is also referred to as upgrading.

922 The new revision of software can be installed in the

923 same location as the current revision. In this case, the

924 software **configure** scripts executed by the configure

925 task need to handle saving or updating the necessary

926 configuration data.

927 The new revision of software can alternatively be

928 installed in a location different than the current revi-

929 sion. In this case, the old revision may be unconfigured

930 by the **unconfigure** script executed as part of the

931 unconfigure task, and the new revision is configured by

932 the **configure** scripts executed as part of the

933 configure task.

934 **Downdate software (swinstall)**

935 This task “downdates” the target file system by instal-

936 ling an older revision of software than is already

937 installed. This is also referred to as “downgrading” or

938 “reverting.”

939 The older revision of software can be installed in the

940 same location as the current revision. In this case, the

941 configuration process of the older version handles the

942 necessary changes in configuration.

943 The older revision of software can alternatively be

944 installed in a location different from that of the current

945 revision. In this case, the new revision can be

946 unconfigured via the unconfigure task, and the older

947 revision can be configured either independently, or as

948 part of install.

949 **Recover software (swinstall)**

950 This task restores the previous version of software (if it

951 exists) in the case where an update, downdate, or rein-

install of software fails. This part of ISO/IEC 15068
defines the minimum required support for automatic

- 952 recovery process in the install task.
- 953 Apply software patch (**swinstall**)
- 954 This task replaces part of a software fileset with a new
- 955 set of files by installing a fileset with those new files in
- 956 the same location as the fileset being patched. This is
- 957 also referred to as fixing software.
- 958 This part of ISO/IEC 15068 currently does not provide
- 959 any special functionality for patching software. Filesets
- 960 related through naming conventions and prerequisites
- 961 can be used. Backup of patched files can be achieved via
- 962 install control scripts.
- 963 Consensus was not reached as to how patches should be
- 964 managed and what level of functionality is required.
- 965 Issues include whether patches are cumulative or com-
- 966 plete, whether they involve partial products or filesets
- 967 (or even files), how volatile files should be managed,
- 968 naming or versioning schemes, and the level of rollback
- 969 support required (number of patch removes that are
- 970 supported for cumulative patches). It was concluded
- 971 that this could not be standardized at this point, but
- 972 that this part of ISO/IEC 15068 does provide a sufficient
- 973 base for implementing patch functionality.
- 974 Remove installed software (**swremove**)
- 975 This task removes software from an installed_software
- 976 object where it previously was installed. Parts of
- 977 software products (subproducts or filesets) can be
- 978 removed at different times.
- 979 If the system where the software is installed was also
- 980 using the software, unconfiguring the software can be
- 981 combined with the remove software task.
- 982 Remove software patch (**swremove**)
- 983 This task removes a patch fileset. This is also referred
- 984 to as rejecting software.
- 985 This part of ISO/IEC 15068 does not provide any special
- 986 functionality for patching software. Filesets related
- 987 through naming conventions and prerequisites can be
- 988 used. Restoring patch files when removing the patched
- 989 can be achieved via remove control scripts.
- 990 Verify the installed software (**swverify**)
- 991 This task checks that software previously installed still
- 992 exists and is intact. If operating on a system that was
- 993 configured to use the software, it can also check that the
- 994 software is configured properly.
- 995 List installed software information (**swlist**)
- 996 This task provides a list of the software that has been
- installed on a target. Options are available to specify
- which software packages are to be listed and to control

- 997 the amount of information provided.
- 998 Fix installed software information (**swmodify, swverify**)
- 999 This task modifies information about software that has
- 1000 been installed on a target. Options are available to
- 1001 specify which software packages, and what information
- 1002 about those packages, are modified.
- 1003 Package software (**swpackage**)
- 1004 This task takes place in the packager role and
- 1005 transforms developed software into the software packag-
- 1006 ing layout suitable for distribution. The metadata that
- 1007 defines the software objects to be packaged is contained
- 1008 in the product specification file (PSF).
- 1009 Copy distribution software (**swcopy**)
- 1010 This task copies distribution software between a source
- 1011 and a target, for subsequent use of that target as a
- 1012 source. Copying software can be used to merge distribu-
- 1013 tions, to distribute products to the installation targets,
- 1014 and then install from that local copy, or to copy part of a
- 1015 distribution to a removable media for physical distribu-
- 1016 tion (as opposed to electronic distribution).
- 1017 Remove distribution software (**swremove**)
- 1018 This task removes products from a target distribution.
- 1019 Check/verify distribution software (**swverify**)
- 1020 This task checks that a target distribution exists and is
- 1021 intact.
- 1022 List distribution information (**swlist**)
- 1023 This task lists source or target distribution information.
- 1024 Options are available to specify which objects in a distri-
- 1025 bution are to be listed, and to control the amount of
- 1026 information provided.
- 1027 Fix distribution information (**swmodify, swverify**)
- 1028 This task modifies information that describes, and is
- 1029 contained within, target distributions. Options are
- 1030 available to specify which objects in a distribution are
- 1031 modified.
- 1032 License installed software (undefined)
- 1033 How software licenses are managed is undefined within
- 1034 this part of ISO/IEC 15068.
- 1035 The task definitions were based on study of existing practice for software adminis-
- 1036 tration. This included presentations on existing practice by many different system
- 1037 vendors and system administrators. From these, a functionally adequate base
- 1038 was selected upon which all parties could build. While it was recognized that this
- 1039 did not address every concern, it was felt that that the utility descriptions (includ-
- 1040 ing detailed behavior), software structure definitions, and media layout, provided
- 1041 an excellent starting point. After comparing various existing practices, these
- choices appeared to be quite similar to other existing practices in many details of these key areas.

1042 Table B-3 below compares and contrasts the SVR4, HP-UX, SCO Unix, and AIX
1043 software management utilities. While this table is neither complete nor current,
1044 it does provide a useful comparison of existing practices.

Table B-3 – Comparison of Some Existing Practices

Functionality	SVR4 pkg*	HP-UX sw*	custom+	AIX
install software	pkgadd	swinstall	custom+	installp
read to spool	pkgtrans (pkgadd -s)	swcopy		
T}	custom+ archive installation	bffcreate		
store interactive session	pkgask	swinstall allows a session to be saved	yes	smit allows a session to be saved
user interaction (interactive scripts)	packages can have interactive scripts (custom- ize, etc.)	customize scripts and check scripts are not interactive	each package may have a configuration script	scripts called by installp may not interactive
check installa- tion	pkgchk	not applicable	yes	lppchk
display software package info	pkginfo	swlist	custom+	lspp
make a package	pkgmk	swpackage	distmaster	yes
generate prelim- inary input to packager	pkgproto	not applicable (swpackage may do some of this)	distmaster	not applicable
remove a pack- age	pkgrm	swremove	custom+	not applicable

B.4.1 Common Definitions for Utilities

B.4.1.1 Synopsis

There is no additional rationale provided for this subclause.

B.4.1.2 Description

There is no additional rationale provided for this subclause.

B.4.1.3 Options

The **-d** option is needed to remove ambiguity for utilities that operate on both distributions and installed software.

The **-r** option is needed for the following reasons. Installing software at / involves a somewhat different set of operations than software installed at an alternate root, as well as a different implied use. Software installed on alternate roots is not configured in the context of the target where the software is installed, but rather in the context of the client actually running the software. Another difference is that the target is not rebooted after installing software that requires

1082 a reboot (the clients of the software need to be rebooted). An alternate root con-
1083 taining operating system software can be thought of as a root to which one could
1084 *chroot()*. From a usability standpoint, it is important that alternate roots are
1085 understood to be different than relocating a software product, or specifying an
1086 alternate catalog for the same root.

1087 Related to the **-s** option, an implementation could define an additional source
1088 syntax to use well-known sources whose existence is available through some sort
1089 of directory service.

1090 The **-s** option could be extended to supported multiple source specifications.
1091 There are several possible ways to interpret multiple sources, including searching
1092 sources sequentially, ignoring all specifications after the first one, using the last
1093 specification, or choosing the “best” source based on criteria such as performance
1094 or ability to reduce network load. It may even be desirable for multiple source
1095 specifications to be interpreted differently for different commands.

1096 An implementation may implement the **-p** option (preview) by simply executing
1097 the command through the analysis phase. Alternatively, an implementation may
1098 emulate the execution phase, listing the operations that would occur, including
1099 listing control scripts that would be run, but not actually performing those opera-
1100 tions. As preview is undefined, other alternatives are possible.

1101 **B.4.1.3.1 Non-interactive Operation**

1102 It is recognized that there may need to be some sort of interaction with the user in
1103 order to handle multiple volumes (e.g., tapes) for sources and targets.

1104 **B.4.1.4 Operands**

1105 Discussion pointed to the fact that the **@** character does not have any applicable
1106 precedence as a separator of operands. It was concluded that the use of **@** in mail
1107 addresses and BSD commands is a bit different. Another point was that having
1108 two lists of operands was not desirable in any case.

1109 On the other hand, the two types of operands are the two key objects upon which
1110 the utilities operate. The syntax is valid according to the utility guidelines from
1111 2.10.2 of POSIX.2 {3}. Distributed utilities extend the problem space that
1112 POSIX.2 {3} has already addressed, thus the need for precedence might be less.
1113 Thus, it was decided that the **@** was acceptable, and perhaps desirable over the
1114 alternatives.

1115 One alternative was to move one or both operands to options (such as **-S** for
1116 software and **-T** for targets). But, it was felt that this was not necessary because
1117 there are already **-f** and **-t** options for files containing lists of operands. Another
1118 point was that listing target operands on the command line was not critical in any
1119 case, as an administrator of many systems would not use either the *@ targets* or
1120 *-T target* syntax.

1121 **B.4.1.4.1 Software Specification and Logic**

1122 Using a less formal grammar convention that defines zero or one item by enclosing
these items in **[]** (brackets) and zero or more repeated items in **{ }** (braces), the
following shows a common subset of the **software_spec** syntax:

```

1123 software_spec : bundle_tags [ product_tags ] [ version ]
1124                | product_tags [ version ]
1125                | '*' [ version ]
1126                ;
1127 bundle_tags    : bundle { '.' bundle }
1128                ;
1129 product_tags   : product
1130                [ '.' subproduct { '.' subproduct } ]
1131                [ '.' fileset ]
1132                ;
1133 version        : { ',r' rel_op revision }
1134                [ ',a=' architecture ]
1135                [ ',v=' vendor_tag ]
1136                [ ',l=' location ]
1137                [ ',q=' qualifier ]
1138                | ',*'
1139                ;
1140 rel_op          : '==' | '!=' | '>=' | '<=' | '<' | '>'
1141                ;

```

The keywords **bundle**, **product**, **subproduct**, and **fileset** refer to the *tag* attributes of those objects. The value of **revision** is usually a dot separated string compared to the value of the *revision* attribute of the first object. The values of **architecture**, **vendor_tag**, **location**, and **qualifier** are usually exact strings or patterns compared to the like-named attributes of the first object. These version attributes can validly be specified like **revision** is, but operators and multiple specifications do not make much sense.

Examples of **software_specs** are

```

1150 *
1151 Networks
1152 Networks.X11
1153 Networks.X11.Runtime,a=*80?86*
1154 X11
1155 X11.Runtime
1156 X11.Runtime,r=4,v=CloneInc
1157 X11.Runtime,r>=4.0,r<5.0
1158 X11,r=4.03.07,l=/usr/X11R4
1159 X11,r=5.00,l=/usr/X11R5,q=latest
1160 X11,*
1161 *,a=*80?86*

```

A **software_spec** shall begin with a bundle or product *tag*. A particular bundle or product object can be determined since they share the same name space (they also have different *instance_id* attributes).

The *location* attribute applied to the product means all filesets in that product in that location. This is the same set of filesets as if the *location* attribute was applied to the filesets.

Since the components of the **version_qualifier** of a **bundle_software_spec** refer to the attributes of bundle objects, there is no way to select one version of a product if more than one version is specified in the *bundle.contents*. Neither the inclusion of multiple versions of a product within a bundle, nor the specifying of partial bundles, is seen as the normal use model, so

1171 having this part of ISO/IEC 15068 limit the flexibility slightly in this area was
1172 deemed as acceptable.

1173 This part of ISO/IEC 15068 permits the use of values other than those defined in
1174 4.1.4.1 for **ver_id**. This part of ISO/IEC 15068 also permits the use of **ver_id** in
1175 conjunction with attributes and objects other than the first listed in a
1176 **software_spec**. This allows additional flexibility for identifying software
1177 objects.

1178 A possible syntax for these vendor extensions include, but are not limited to

1179	ver_id	attribute	object
1180	br	revision	bundle
1181	ba	architecture	bundle
1182	bv	vendor_tag	bundle
1183	bl	location	bundle
1184	pr	revision	product
1185	pa	architecture	product
1186	pv	vendor_tag	product
1187	pl	location	product
1188	fr	revision	fileset
1189	fl	location	fileset

1190 The **ver_id fr** is seen as most useful since it can identify a particular fileset
1191 object within a product where the product may not have a revision, but the fileset
1192 does. Note however, that any object can still be identified with only the attributes
1193 defined in this part of ISO/IEC 15068. For example, if a bundle includes two par-
1194 tial products with the same *tag* value but different revisions or locations, these
1195 partial products could be identified with the standard syntax by excluding the
1196 bundle portion of the **software_spec**. For example,
1197 **bundle.product,pr=1.3** could also be identified by a **software_spec** of
1198 **product,r=1.3**.

1199 In another example, the fileset that could be identified by its revision
1200 (**product.fileset,fr=1.3**) could also be identified by a **software_spec**
1201 (e.g., **product,r=revision**), where revision refers to the product revision,
1202 including possibly the empty string.

1203 Relocation occurs by replacing the *product.directory* part of each file path as it
1204 occurs in the distribution, with the *location* specified and using the resulting path
1205 for installation. This is still relative to the *installed_software* directory described
1206 below. See 4.5.7.3.1 for more information.

1207 Using a **sw_pattern** in a **software_spec** is a way for the user to indicate that
1208 all software objects that match the **software_spec** are to be included. For
1209 example, applying the **software_spec** "*" to **swcopy** means to copy all
1210 software in the distribution. Applying the **software_spec** "Foo,*" to
1211 **swremove** means to remove all versions of Foo.

1212 The behavior for **swlist** is different (by default including all software if none is
1213 specified) because this is the command that is used to find all versions of software,
1214 and because listing cannot negatively affect the state of the *software_collection*.

1215 If using software pattern matching notation characters on the command line, they
shall be escaped or enclosed in single quotes to avoid matching files in the current
working directory.

1216 This specification provides the means to select products and specify dependencies
1217 using a single syntax. The use of the shell-type pattern match specified in 3.13 of
1218 POSIX.2 {3} allows for reasonable specification of sets of values that share such
1219 patterns. Thus, for example, a specification of **"a=HP-UX*"** may be used to
1220 select packages for any of a set of architectures. The specification using the rela-
1221 tional operators provides support for testing the type of release/version
1222 specifications that are frequently used by vendors. In particular, it provides sup-
1223 port for testing when a numeric test is needed (e.g., comparing 2.9 to 2.10 as ver-
1224 sion levels of a product). Additional operators such as **>>** were considered. The
1225 specification of the **>>** operator allows the user to specify the selection of the most
1226 recent (highest version number) of a set of otherwise identical packages. This
1227 exposes to the interface the mechanism used by **swinstall** to select such a pack-
1228 age.

1229 The range of attributes that may be specified allows for selection of packages that
1230 may be needed to support code serving to alternate architectures, or other operat-
1231 ing environments. In addition, it provides the needed support to specify installed
1232 software that may only be distinguished by the location of installation.

1233 Examples of fully qualified **software_specs** are

```
1234     Foo, r=3.0, a=, v=XT
1235     BundleA.Foo, r=1.0, a=, v=XT
1236     Dow.Bar, r=2.0, a=SunOS, v=, l=/opt/foo.2
```

1237 It is possible for bundles to contain **software_specs** that are not fully qualified.
1238 This is not recommended for bundle definitions provided by software vendors
1239 because the results of operations on this bundle may be undesirable for an
1240 administrator. However, there is some flexibility provided by ambiguous software
1241 specs that administrators may want to use.

1242 For example, a bundle with contents **"*.Man"** could be used to manipulate all
1243 **"Man"** filesets or subproducts in all products.

1244 If a vendor includes any wildcards in a **software_spec** in a bundle definition,
1245 then the *vendor_tag* attribute should be included and its value should have no
1246 wildcards, thus limiting the scope of the pattern matching.

1247 The difference between **"F00, v="** and **"F00"** is that the first will only match a
1248 product or bundle **"F00"** where vendor is not defined, while the second will
1249 match a product or bundle **"F00"** with any vendor definition.

1250 **B.4.1.4.2 Source and Target Specification and Logic**

1251 Using a less formal grammar convention that defines zero or one item by enclosing
1252 these items in **[]** (brackets) and zero or more repeated items in **{ }** (braces), the
1253 following shows a common subset of the **software_collection_spec** syntax:

```
1254 software_collection_spec    : [ host ] [ ':' ] [ path ]
```

1255 Examples of distribution **software_collection_specs** are

```
1256 /var/spool/sw
hostA
hostA.cloneinc.com
```

1257 **hostA:/var/spool/sw**
1258 **15.1.94.296**
1259 **15.1.94.296:/depots/applications**

1260 Examples of installed_software **software_collection_specs** are

1261 **/**
1262 **hostA**
1263 **hostA.cloneinc.com**
1264 **hostA:/**
1265 **15.1.94.296**
1266 **15.1.94.296:/exports/applications**

1267 Target distributions in the serial format need not be supported for **swverify**,
1268 **swremove**, and **swmodify** as this requires the implementation to unload the
1269 entire distribution, merge in the changes, then reload it. The user can accomplish
1270 this (and an implementation can implement this) by first copying the distribution
1271 into a directory format, implementing the changes, then copying the distribution
1272 back to the serial media. This operation also could require significant temporary
1273 disk space.

1274 A similar rationale applies to **swcopy**, and **swpackage**, which by default,
1275 overwrite the existing distribution instead of merging in the specified software.

1276 **B.4.1.5 External Influences**

1277 **B.4.1.5.1 Defaults and Options Files**

1278 For SVR4 or similar file system layout, the defaults file may be located in
1279 **/var/adm/sw/defaults**. The use of this location is strongly encouraged.

1280 The difference between “system-level” defaults and “site-level” defaults was dis-
1281 cussed.

1282 The former is provided by the implementation of the utilities and the latter is con-
1283 structed by the administrator. The intent here is for the implementation to
1284 respect any customizations to the system level defaults file, so it can be used for
1285 site policies. It is recommended that implementations “hard code” the defaults as
1286 opposed to relying on the system file containing all definitions, and provide a
1287 means to support new options in future releases without changing the site specific
1288 values in the system defaults file.

1289 **B.4.1.5.2 Extended Options**

1290 For SVR4 or similar file system layout, *distribution_source_directory* may be set
1291 to **/var/spool/sw**. The use of this location is encouraged.

1292 For SVR4 or similar file system layout, *distribution_target_directory* may be set to
1293 **/var/spool/sw**. The use of this location is encouraged.

1294 For SVR4 or similar file system layout, *installed_software_catalog* may be set to
1295 **/var/adm/sw/catalog**. The use of this location is encouraged. The catalog
may simply be a pathname of a directory where the database containing the cata-
log is stored, or may be a key into a directory service specifying a catalog in a file
or database, or any other implementation-defined method of specifying a catalog.

1296 The location of the storage for the catalog itself is implementation defined.

1297 The **swinstall** utility, and other utilities that operate on `installed_software`,
1298 modify the catalog information based on the outcome of the utility. Information
1299 contained within the catalog is resolved in the context of each target.

1300 Originally, it was thought that a catalog would be kept as a flat file in a directory
1301 that could be specified using this option. In the interest of generality, so that
1302 implementors might be allowed to use databases, the *catalog* attribute is now
1303 described as a key. This allows an implementor to either use a flat file or a data-
1304 base or some other form of persistent storage for the information, yet still be able
1305 to separate the address space as desired. The motivation for permitting the
1306 separate address space stems from the following two cases. First, it seems desir-
1307 able to allow ordinary (non-root) users to be able to use **swinstall** to store
1308 software in their own private space. Likely the only real restriction is a potential
1309 lack of write authority to the central storage for the catalog, hence the ability to
1310 create a separate catalog. This also allows a user to manage personal software
1311 with utilities such as **swremove** or other utilities. Second, installations may wish
1312 to deploy stable versions of their software in the normal location, and a test ver-
1313 sion installed in a second location where access may be more tightly controlled.
1314 There may even be other versions installed that are under development. Since
1315 this software may have identical attributes, it is desirable to allow such separate
1316 space for management. Both of these examples show the need for separate
1317 domains of software management.

1318 Two values of *autoselect_dependencies* (*autoselect_dependencies* = *true* and
1319 *autoselect_dependencies* = *as_needed*) support different possible policies by the
1320 user. Having *autoselect_dependencies* = *true* ensures that all targets are kept in
1321 sync, while having *autoselect_dependencies* = *as_needed* prevents the possibility of
1322 updating dependency software to a higher revision unnecessarily.

1323 Autoselection of a dependency across products is possible if a compatible product
1324 version with the highest revision that meets the dependency is unique. In other
1325 words, the same rules apply for dependency selection as for normal selection as
1326 described in 4.1.4.1.

1327 For the *ask* option, the **checkinstall** and **configure** scripts are required to
1328 detect needed **response** files when they are necessary, and return with the
1329 appropriate warning or error.

1330 For the *installed_software_catalog* option, the catalog and the directory together
1331 form a key to identify one `installed_software` object. For example, this would allow
1332 the files on the file system to be split up into different management domains. For
1333 example, OS software, networking software, and application software could be in
1334 three different logical `installed_software` objects, although they are all installed
1335 under the root file system.

1336 **B.4.1.5.3 Extended Options Syntax**

1337 In the interest of having a single common extended option syntax for all the
1338 POSIX system administration standards, the following syntax was agreed upon.
1339 As of this writing, the syntax is a superset of that used by this part of ISO/IEC
1340 15068, IEEE P1387.3, and IEEE P1387.4.

```

1341 %token      FILENAME_CHARACTER_STRING /* as defined in 2.2.2.37 */
1342 %token      NEWLINE_STRING             /* as defined in 2.2.2.61 */
1343 %token      PORTABLE_CHARACTER_STRING  /* as defined in 2.2.2.68 */
1344 %token      SHELL_TOKEN_STRING         /* as defined in 2.2.2.80 */
1345 %token      WHITE_SPACE_STRING         /* as defined in 2.2.2.110 */

```

```

1346 %start  sysadmin_option
1347 %%

```

```

1348 sysadmin_option      : qualifier option operator_value
1349                       ;

```

```

1350 qualifier            : compulsory_qualifier command_qualifier
1351                       ;

```

```

1352 compulsory_qualifier : /* empty */
1353                       | '-' | '='
1354                       ;

```

```

1355 command_qualifier    : /* empty */
1356                       | command '.'
1357                       ;

```

```

1358 option               : keyword op_ws
1359                       ;

```

```

1360 operator_value       : '=='
1361                       | value_qualifier '=' value
1362                       ;

```

```

1363 value_qualifier       : /* empty */
1364                       | '+' | '-'
1365                       ;

```

```

1366 value                : op_ws value ws single_value
1367                       | op_ws single_value
1368                       ;

```

```

1369 single_value         : value_structure
1370                       | SHELL_TOKEN_STRING
1371                       ;

```

```

1372 value_structure      : '{' op_ws value_list op_ws '}'
1373                       ;

```

```

1374 value_list           : /* empty */
1375                       | value_list ws single_value
1376                       | single_value
1377                       ;

```

```

1378 command              : FILENAME_CHARACTER_STRING
1379                       ;

```

```

1380 keyword              : SHELL_TOKEN_STRING
1381                       ;

```

```

1382 op_ws                : /* empty */
1383                       | ws
1384                       ;

```

```

1383  ws                : WHITE_SPACE_STRING
1384                      ;

1385  %start command_line_options
1386  %%
1387  command_line_options : command_line_options ws sysadmin_option
1388                      | sysadmin_option
1389                      ;

1390  %start options_file
1391  %%
1392  options_file         : options_file NEWLINE_STRING option_file_line
1393                      | option_file_line
1394                      ;

1395  option_file_line     : op_ws op_comment
1396                      | op_ws sysadmin_option op_ws op_comment
1397                      ;

1398  op_comment           : /* empty */
1399                      | '#' PORTABLE_CHARACTER_STRING
1400                      ;

```

- 1401 (1) A - (hyphen) qualifier indicates a compulsory behavior while = (equal)
1402 indicates a non-compulsory behavior.
- 1403 (2) For options that support multiple values, values can be added to the
1404 existing list of values by using the += (plus equal) operator. Similarly,
1405 values can be removed by using the -= (hyphen equal) operator. Any
1406 option can be set to the default value by using the == (equal equal) opera-
1407 tor and value combination.
- 1408 (3) A shell token can be an unquoted or quoted string according to the rules
1409 of token recognition rules described in 3.3 of POSIX.2 {3}. For example, it
1410 can use single or double quotes and can contain like quotes if escaped
1411 with backslash. It can also support the same level of internationalization
1412 as the POSIX shell.
- 1413 (4) The multiple value convention is consistent with white space separating
1414 tokens in commands (operands) and allows commas to be used in the
1415 **single_value**. This also allows multiple values to be specified without
1416 using quotes (although quotes are still needed for multiple values on the
1417 command line).
- 1418 (5) If the extended option specification contains any white space at all, then
1419 the entire specification shall be quoted if used on a command line. This is
1420 because the -x option, which conforms to POSIX.2 {3}, requires exactly
1421 one value that is then processed using the above syntax.
- 1422 (6) When specified on the command line, multiple option specifications can
1423 be included after a single -x option if included in quotes and separated by
1424 spaces. Multiple -x options may also be used.
- 1425 (7) For option and defaults files, blank lines and all comment text [any
1426 sequence of characters beginning with an unescaped # (pound) and con-
1427 tinuing through the end of that line] are ignored according to the shell
token recognition rules as described in 3.3 of POSIX.2 {3}.

1428 **B.4.1.5.3.1 Precedence for Option Specification**

1429 The first rule defines typical precedence of system defaults, then a user defined
1430 set of defaults, then per task exceptions or specifications. The second rule sup-
1431 ports normal use models of defining multiple “sets” of target_selections and
1432 software_selections, and being able to operate on the union of those sets. Also, the
1433 -f and -t options are simply another form for specifying operands, and are at the
1434 same level of precedence, and are thus combined with other selections. The third
1435 rule is generally an error, and the behavior is undefined (i.e., it may be an error,
1436 or an implementation may chose to implement last- or first-wins). For example on
1437 HP-UX:

```
1438 $ cc -O -g x.c  
1439 $ cc: warning 414: Debug and Optimization are mutually exclusive.  
1440       -g option ignored.
```

```
1441 $ cc -g -O x.c  
1442 $ cc: warning 414: Debug and Optimization are mutually exclusive.  
1443       -O option ignored.
```

1444 It might be convenient to have a mechanism to allow the system administrator to
1445 define a default in the system defaults file that cannot be overridden by a user.
1446 Such a function may be supplied by an implementation as an extension. This may
1447 also be considered as part of a future revision to this part of ISO/IEC 15068.

1448 **B.4.1.5.4 Standard Input**

1449 There is no additional rationale provided for this subclause.

1450 **B.4.1.5.5 Input Files**

1451 There is no additional rationale provided for this subclause.

1452 **B.4.1.5.6 Access and Concurrency Control**

1453 For example, if the *installed_software_catalog* is referenced by a path on the file
1454 system, then the user can create a catalog in their own user work space (creating
1455 their own *installed_software* object), and install and manage software in that
1456 *installed_software* object.

1457 If the catalog is stored in a file, then a corresponding ability to create an
1458 *installed_software* object (and thus, a catalog) is needed.

1459 Access control includes such things as requiring particular authority to operate on
1460 particular software or software_collections. Concurrency control is the prevention
1461 of more than one writer at a time to the catalog or data areas. Restrictions to
1462 prevent multiple concurrent writers were originally part of the draft, but later
1463 determined to be excessively restrictive. It is conceivable that more than one
1464 writer could safely be active at a time if the work involves no common files.
1465 Failure to allow multiple concurrent readers of the catalog, or other data files, is
1466 strongly discouraged.

1467 There are two aspects to access control as follows:

- Those related to file system access and hence determined by the operating system (i.e., the ability to write the files described by the software file

1468 objects)

1469 — Any additional access control on the software objects, including access to
1470 (possibly remote) software collections

1471 For access control to the files themselves, this part of ISO/IEC 15068 defaults to
1472 the file permissions defined by POSIX.1 {2}. File attributes are defined for the
1473 files, hence the implementation will set the POSIX.1 {2} file permissions based on
1474 those values. Deviations from this model are permitted only for implementations
1475 running on file systems that are not POSIX.1 conformant, and then only as long
1476 as the implementation documents the resulting behavior.

1477 Any additional access control to the software objects defined in this part of
1478 ISO/IEC 15068 (e.g., permission to install specific software into specific software
1479 collections on specific hosts), is undefined. An implementation may choose to have
1480 no access control. For example, anyone may install any software to any system as
1481 long as the previous POSIX.1 {2} file permissions are satisfied. An implementation
1482 may also choose to provide both authorization and authentication for access to all
1483 software objects and hosts, as well as a distributed interface for managing the
1484 access control lists.

1485 Like the definition of the model to implement distributed aspects of this standard,
1486 access control beyond that required by the underlying operating system is
1487 undefined. It was determined that both of these rely on technologies that have
1488 not been formally standardized, and may better be addressed in other forums.

1489 **B.4.1.6 External Effects**

1490 **B.4.1.6.1 Control Script Execution and Environment**

1491 The provision for interpreters other than **sh** was requested by users among the
1492 developers of this part of ISO/IEC 15068, as well as producers representing sys-
1493 tems that might lack a POSIX.2 {3} or even POSIX.1 {2} operating system. By mak-
1494 ing this provision, many felt that a greater degree of acceptance and usefulness
1495 could be gained.

1496 The restrictions placed on the option syntax are such that each of the options can
1497 be easily parsed and hence set by a control script by simply sourcing the file. The
1498 term “sourcing” as used here implies the use of the “.” command in the POSIX.2 {3}
1499 shell. For example, the following are formats for the **SW_SESSION_OPTIONS**
1500 file that can be sourced:

```
1501     loglevel=1
1502     enforce_dependencies=false
1503     software="A B C"
```

```
1504     loglevel=1
1505     enforce_dependencies=false
1506     software="
1507     A
1508     B
1509     C"
```

It is possible for the scripts to determine the *loglevel* for the command from the file pointed to by **SW_SESSION_OPTIONS**, and use that to affect the amount of

1510 stdout generated.

1511 An implementation may have an implementation-defined user controllable
1512 behavior that invokes error handling procedures in the case of warnings returned
1513 from script execution.

1514 The purpose of the environment variables is to pass vital information to the
1515 scripts so that they may operate appropriately under different circumstances. For
1516 example, they may want to take very different actions when
1517 **SW_ROOT_DIRECTORY** is some value other than /. It has also been discussed
1518 that there may need to be some way to pass other information to these scripts
1519 such as option values specified in the defaults and options file that control policy.
1520 This can be achieved with the **SW_SESSION_OPTIONS** variable, which points
1521 to a file containing all the options passed to the command, including options, selec-
1522 tions, and targets.

1523 One reason that this part of ISO/IEC 15068 differentiates install and remove
1524 scripts from **configure** scripts is to separate installing software from
1525 configuring software for actual use. This supports installing software to alternate
1526 root directories on servers for use by clients that configure that software.

1527 The developers of this part of ISO/IEC 15068 also discussed, but did not include,
1528 the use of several of these variables for setting the value of specific utility options
1529 when the utilities are called from control scripts. These variables are as follows:

1530 **SW_ROOT_DIRECTORY**

1531 could be used to specify the *directory* portion of all *target* operands

1532 **SW_LOCATION**

1533 could be used to specify the *product.location* portion of all *software_-*
1534 *selection* operands

1535 **SW_CATALOG**

1536 could be used to specify the value of the *installed_software_catalog*
1537 option

1538 The scripts need to be aware of the environment under which they are operating.
1539 The environments that these scripts run under are as follows:

1540 — All scripts

1541 Each script shall be passed its script *tag*, the root directory to which instal-
1542 ling, the product directory where the product is located, and the control
1543 directory where the script is being executed from, as the environment vari-
1544 ables **SW_CONTROL_TAG**, **SW_ROOT_DIRECTORY**, **SW_LOCATION**,
1545 and **SW_CONTROL_DIRECTORY**, respectively.

1546 — **preinstall**, **postinstall**, **preremove**, **postremove**, **unprein-**
1547 **stall**, **unpostinstall**

1548 The **install** and **remove** scripts are run when loading or removing the
1549 software, or when recovering from a failed install. These may be executed
1550 by **swinstall** or **swremove** running on a host with a different architec-
1551 ture from the software. So, only the set of POSIX.2 {3} utilities are
1552 guaranteed to be available on the server. Since the architecture of the file
1553 server is not necessarily known, the path to these commands is passed to
the scripts via the environment variable **SW_PATH**.

1554 Additionally, these scripts need to know the alternate root directory so that
1555 operations are within the context of that root, not the root of the file server.
1556 (This directory is supplied to the scripts via an environment variable
1557 **SW_ROOT_DIRECTORY**.) It is critical that **SW_ROOT_DIRECTORY**
1558 is honored by these scripts.

1559 — **checkinstall, checkremove, verify**

1560 It is expected, but not assumed, that these scripts mostly check the state of
1561 a system that will actually run the software. For the install check (**chec-**
1562 **kinstall**) script, again only a minimum set of commands is guaranteed to
1563 be available. This is because all check scripts are executed before any new
1564 software is installed. For the remove check and verify scripts
1565 (**checkremove** and **verify**), files from this software and its prerequisites
1566 are guaranteed to be available.

1567 Because of alternate root installation, these scripts also need to be aware of
1568 the **SW_ROOT_DIRECTORY**.

1569 — **configure, unconfigure**

1570 These scripts configure the system for the software. Therefore they can run
1571 architecture specific commands, including files that are part of the software
1572 that defines the scripts. They are only run within the context of the system
1573 that will actually use the software. The **SW_ROOT_DIRECTORY** will
1574 always be **/**.

1575 **B.4.1.6.1.1 Control Script Behavior**

1576 Control scripts allow vendors to perform tasks and operations, in addition to those
1577 that the tasks perform. The **swinstall**, **swverify**, and **swremove** utilities
1578 may each execute one or more vendor-supplied scripts. The presence of these
1579 scripts in the distribution is optional. Vendors of software to be installed need
1580 only provide those scripts that meet a particular need of the software. The follow-
1581 ing summarizes the standard scripts:

1582 **request** (Request script)

1583 This is the only script that may be interactive. This script may be
1584 run by **swask**, **swinstall**, or **swconfig** after selection, and before
1585 the “analysis” phase in order to request information from the
1586 administrator that will be needed for the **configure** script when
1587 that script is run later.

1588 This script is executed on the manager role and it is the responsibil-
1589 ity of the script to write all information into the **response** file in the
1590 directory where the script is being executed (the
1591 **SW_CONTROL_DIRECTORY**). The utilities will then copy this
1592 file to the **SW_CONTROL_DIRECTORY** on the target role where
1593 the **configure** and other scripts are executed from. This
1594 **response** file may be used by any other scripts, but particularly the
1595 **configure** script.

1596 A recommended syntax for the **response** file is a set of attribute-
value pairs that can be easily sourced by the **configure** script. For
example, a **request** script might look like the following:

```

1597         echo "Enter path to locate the database"
1598         read $path
1599         echo db_path=$path >> $SW_CONTROL_DIR/response

```

1600 And the using **configure** script:

```

1601         $db_path=$default_db_path
1602         if [ -f $SW_CONTROL_DIR/response ]
1603         then
1604             . $SW_CONTROL_DIR/response
1605         fi
1606         create_db ( $db_path )

```

1607 **response** (Response file)

1608 The **response** file is generated by the **request** script and located
 1609 in the same directory as that and the other scripts. The format and
 1610 content of the **response** file is implementation defined. For exam-
 1611 ple, it may be a list of environment variable definitions so that the
 1612 consumer script can just source this file.

1613 **checkinstall** (Install check script)

1614 This script is run by **swinstall** during the “analysis” phase in an
 1615 attempt to ensure that the installation (and configuration) succeeds.
 1616 For example, the OS run state, running processes, or other prere-
 1617 requisite conditions beyond dependencies may be checked. Running
 1618 this script shall be free of side-effects, e.g., processes may not be
 1619 killed, files may not be moved or removed, etc.

1620 **preinstall** (Install preload script)

1621 This script is run by **swinstall** prior to loading the software files.
 1622 For example, this script may remove obsolete files, or move an exist-
 1623 ing file aside during an update.

1624 This script and the next script are part of the “load” phase of the
 1625 software installation process. Within each product, all **preinstall**
 1626 scripts are run (order is dictated by any prerequisites), all filesets are
 1627 loaded, then all **postinstall** scripts are run.

1628 **postinstall** (Install postload script)

1629 This script is run by **swinstall** after loading the software files. For
 1630 example, this script may move a default file into place.

1631 **unpreinstall** (Recovery preload script)

1632 This script is run by **swinstall** after restoring the software files
 1633 when recovering filesets. For example, this script may undo the
 1634 actions of the **preinstall** script.

1635 **unpostinstall** (Recovery postload script)

1636 This script is run by **swinstall** before restoring the software files if
 1637 the **postinstall** script has been run. It can be used to undo the
 1638 actions of the **postinstall** script.

verify (Verify script)

1639 This script is run by the **swverify** command any time after the
1640 software has been installed or configured. Like other scripts it is
1641 intended to verify anything that the commands do not verify by
1642 default. For example, this script may check that the software is
1643 configured properly and has a proper license to use it.

1644 **fix** (Fix script) This script is run by the **swverify** command when the
1645 **-F** option is used. Its purpose is to correct any problems reported by
1646 the **verify** script.

1647 **checkremove** (Remove check script)

1648 The remove check script is run by **swremove** during the remove
1649 “analysis” phase to allow any vendor-defined checks before the
1650 software is permanently removed. For example, the script may
1651 check whether anyone is currently using the software.

1652 **preremove** (Preremove script)

1653 This script is executed just before removing files. It may be destruc-
1654 tive to the software being removed, as removal of files is the next
1655 step. It is the companion script to the install postload script (**pos-
1656 tinstall**). For example, it may remove files that the install post-
1657 load script created.

1658 This script and the next script are part of the “remove” phase of the
1659 software remove process. Within each product, all remove
1660 **preremove** scripts are run (in the reverse order dictated by any
1661 prerequisites), all files are removed, then all remove **postremove**
1662 scripts are run.

1663 **postremove** (Postremove script)

1664 This script is executed just after removing files. It is the companion
1665 script to the the install preload script (**preinstall**). For example,
1666 if this was a patch files set, then the install preload script may move
1667 the original file aside, and this remove postload script may move the
1668 original file back if the patch was removed.

1669 **configure** (Configure script)

1670 This script is executed by **swinstall** after all software has been
1671 installed (including loading files and running **postinstall** scripts)
1672 if software is being installed at /. It is also may run by **swconfig**,
1673 even if the software has already been configured (allowing the
1674 administrator to reconfigure software).

1675 **unconfigure** (Unconfigure script)

1676 This script is executed by **swremove** before any software is removed
1677 (including removing files and running **preremove** scripts), if remov-
1678 ing from software installed at /. It is may also be run by **swconfig**.

1679 Other scripts

1680 The vendor may include other control scripts, such as a script that is
1681 sourced by the above scripts, or scripts not defined in this part of
ISO/IEC 15068. The location of the control scripts is passed to all
scripts via an environment variable **SW_CONTROL_DIRECTORY**.

1682 **B.4.1.6.1.2 Control Script Return Code**

1683 This part of ISO/IEC 15068 only specifies return codes from scripts that affect
1684 operation of the utilities. If the script writer wants to convey additional informa-
1685 tion to the user, such information should be written to stderr or stdout, which gets
1686 recorded by the target role logging. See 4.1.6.5.

1687 A implementation that supports additional behaviors may initiate those behaviors
1688 based on implementation-defined return values from control scripts.

1689 **B.4.1.6.2 Asynchronous Events**

1690 Control script execution and file operations only generate note events at the begin-
1691 ning of the step, not at both the beginning and the end. This is because there is
1692 an event immediately after each script or file completes (e.g., the next file begin-
1693 ning, or the end of fileset execution). Additionally, if there is any error with the
1694 script or file, there is a warning or error event generated after the file completes.

1695 **B.4.1.6.3 Stdout**

1696 There is no additional rationale provided for this subclause.

1697 **B.4.1.6.4 Stderr**

1698 There is a variety of warnings and errors that could occur at the target role. It is
1699 desirable that this information be communicated back to the management role
1700 and displayed on stderr of the management role. Of course the detail of the infor-
1701 mation is determined by the *verbose* level. However, since this part of ISO/IEC
1702 15068 does not specify the communication mechanism between the management
1703 and target roles, it should not place unnecessary requirements on this communi-
1704 cation mechanism, especially since this could be occurring in a distributed
1705 environment with the management role communicating with multiple target
1706 roles. Thus the only requirement here is that the target role be able to communi-
1707 cate a binary fail/success back to the management role.

1708 **B.4.1.6.5 Logging**

1709 For the management role, the information placed in the logfile is equivalent to the
1710 information sent to stderr and stdout if the *verbose* level was the same value as
1711 the *loglevel*.

1712 The stdout, stderr, and logfile output should contain the severity of the event as
1713 part of the output message. For example

1714 **NOTE:** The analysis phase succeeded on target "zook:/".

1715 **WARNING:** Target "zook:/": 2 configure scripts had warnings.

1716 **B.4.1.7 Extended Description**

1717 **B.4.1.7.1 Selection Phase**

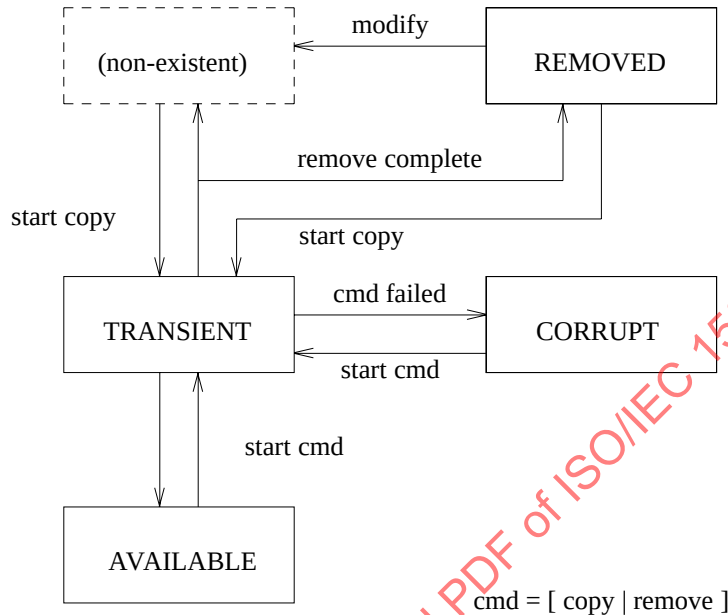
1718 An implementation may define other valid source specifications, such as "well-
known" sources that may be available via a directory service or an object request
broker.

1719 **B.4.1.7.2 Analysis Phase**

1720 There is no additional rationale provided for this subclause.

1721 **B.4.1.7.3 Execution Phase**

1722



1723

1724

1725 **Figure B-5 – Fileset State Transitions (Within Distributions)**

1726 It is clear that some vendors may want additional states. But allowing other
 1727 values would make it problematic for any implementation trying to make deci-
 1728 sions on how to deal with filesets with unrecognized states. An implementation
 1729 may create an additional fileset attribute that would further modify the meaning
 1730 of the attribute. For example, they may create an attribute called *state_info* and
 1731 this attribute may have the value of **files_missing** when the *state* attribute is
 1732 set to **corrupt**. There could be several valid values of this new attribute to
 1733 describe various possibilities of a **corrupt** state. Of course, since this would be
 1734 an implementation-specific extension, other implementations would not need to
 1735 recognize this attribute or its semantics.

1736 Figure B-5 and Figure B-6 show the state transition diagrams for Installed
 1737 Software and Distributions.

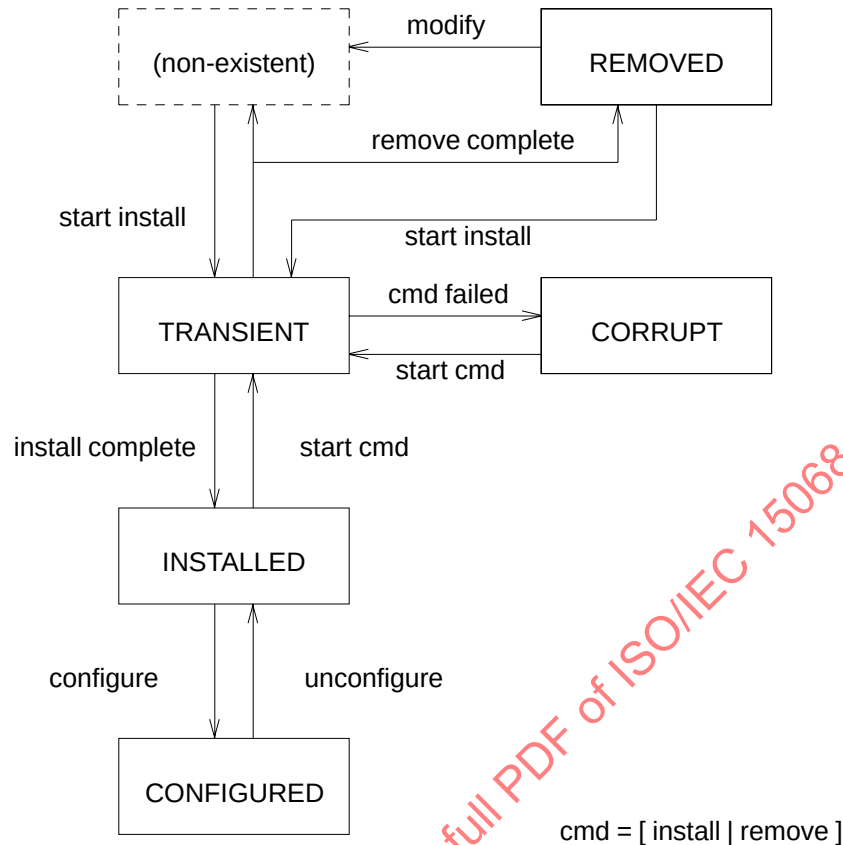
1738 **B.4.1.8 Exit Status**

1739 There is no additional rationale provided for this subclause.

1740 **B.4.1.9 Consequences of Errors**

1741 Since the utilities in this part of ISO/IEC 15068 operate on multiple software
 objects for multiple targets, the handling of error conditions (which is basically a
 policy decision) is complex. For example, should success be only all or nothing; or
 only, if all operations succeed for a specific host; or only, if all operations succeed

1742



1743

1744

1745 **Figure B-6 – Fileset State Transitions (Within Installed Software)**

1746 for a specific product on all hosts? The type of error or errors causing such
 1747 definitions of success or failure is implementation defined. Whether or not the
 1748 user may specify policies regarding the way in which errors are handled is also
 1749 implementation defined.

1750 **B.4.2 swask — Ask for user responses**

1751 The purpose of this utility is to provide support for interactive requirements for
 1752 software. By being able to execute these interactive scripts independently of
 1753 **swinstall** and **swconfig**, it allows those utilities to still be scheduled for non-
 1754 interactive execution.

1755 The **request** scripts can be used to ask the administrator questions, or requests,
 1756 where responses are needed by the software before installation or configuration.

1757 **B.4.2.1 Synopsis**

1758 There is no additional rationale provided for this subclause.

1759 **B.4.2.2 Description**

1760 The utility may be used to perform the following task:

1761 — Answer the requests of software that has interactive customization needs

1762 The **swask** utility and **request** scripts are for software specific questions only.
1763 It does not provide any mechanism for implementation specific questions,
1764 although an implementation can choose to support implementation, site, or
1765 system-specific enhancements as normal implementation extensions.

1766 **B.4.2.3 Options**

1767 There is no additional rationale provided for this subclause.

1768 **B.4.2.4 Operands**

1769 There is no additional rationale provided for this subclause.

1770 **B.4.2.5 External Influences**

1771 There is no additional rationale provided for this subclause.

1772 **B.4.2.6 External Effects**

1773 There is no additional rationale provided for this subclause.

1774 **B.4.2.7 Extended Description**

1775 This part of ISO/IEC 15068 has not included a naming convention or structure for
1776 storing per client response information. If a **request** script requires per client
1777 information, then it needs to store that information for all clients in the
1778 **response** file, and then locate the appropriate information during configuration.

1779 This part of ISO/IEC 15068 provides a means for the **request** script to determine
1780 which clients are being requested, and for the **configure** script to determine
1781 that client. See 4.1.6.1.

1782 For example, a **request** script that requests per client keys is as follows:

```
1783 . $SW_SESSION_OPTIONS
1784 set - $targets
1785 echo 'keys="' > response
1786 for i in $targets
1787 do
1788     echo enter key for $i
1789     read j
1790     echo $i $j >> response
1791 done
1792 echo '"' >> response
```

1793 And the corresponding **configure** script that looks up the correct key from the
1794 **response** file:

```
. $SW_SESSION_OPTIONS
```



```

1795 . response
1796 set - $keys
1797 while [ -n $1 ]
1798 do
1799     if [ "$1" = "$targets" ]
1800     then
1801         echo key is $2
1802         break
1803     fi
1804     shift; shift;
1805 done

```

1806 An implementation needs to ensure that any **response** files that already exist in
 1807 the source or the catalog are copied to the **SW_CONTROL_DIRECTORY** before
 1808 the **request** script is executed. The order of checks for **response** files allows
 1809 for the following precedence:

- 1810 — User input (if *ask=true*)
- 1811 — Pre-existing response file
- 1812 — Pre-existing client configuration
- 1813 — Model response file (from source)

1814 There are numerous ways to implement where the **request** scripts are executed
 1815 and what **SW_CONTROL_DIRECTORY** is set to, for the command

```

1816 swask -s source -c catalog Foo.Bar

```

1817 Example implementation A:

```

1818 mkdir catalog/Foo/Bar
1819 copy request script for Foo.Bar to catalog/Foo/Bar
1820 copy any necessary response file to catalog/Foo/Bar
1821 if catalog/Foo/Bar/response exists, no action
1822 else if source/catalog/Foo/Bar/response exists, copy it
1823 set SW_CONTROL_DIRECTORY=catalog/Foo/Bar
1824 execute catalog/Foo/Bar/request
1825 (remove catalog/Foo/Bar/request)

```

1826 Example implementation B:

```

1827 mkdir catalog/Foo/Bar /usr/tmp/aaaa43542/Foo/Bar
1828 copy request script for Foo.Bar to /usr/tmp/aaaa43542/Foo/Bar
1829 copy any necessary response file to catalog/Foo/Bar
1830 if catalog/Foo/Bar/response exists, no action
1831 else if source/catalog/Foo/Bar/response exists, copy it
1832 set SW_CONTROL_DIRECTORY=catalog/Foo/Bar
1833 execute /usr/tmp/aaaa43542/Foo/Bar/request
1834 (remove /usr/tmp/aaaa43542/Foo/Bar)

```

1835 Example implementation C:

```

1836 mkdir /usr/tmp/aaaa43542
1837 copy any necessary response file to /usr/tmp/aaaa43542
1838 if catalog/Foo/Bar/response exists, copy it
1839 else if source/catalog/Foo/Bar/response exists, copy it
1840 set SW_CONTROL_DIRECTORY=/usr/tmp/aaaa43542
1841 execute source/catalog/Foo/Bar/request
1842 mkdir catalog/Foo/Bar
1843 cp /usr/tmp/aaaa43542/response catalog/Foo/Bar
1844 (remove /usr/tmp/aaaa43542)

```

1842 **B.4.2.8 Exit Status**

1843 There is no additional rationale provided for this subclause.

1844 **B.4.2.9 Consequences of Errors**

1845 There is no additional rationale provided for this subclause.

1846 **B.4.3 swconfig — Configure software**

1847 The purpose of configuration is to configure the host for the software, and
1848 configure the product for host-specific information. For example, software may
1849 need to modify the `/etc/rc` setup file, or the default environment set in
1850 `/etc/profile`. It may need to ensure that proper codewords are in place for
1851 that host, or do some compilations. Unconfiguration undoes these steps.

1852 **B.4.3.1 Synopsis**

1853 There is no additional rationale provided for this subclause.

1854 **B.4.3.2 Description**

1855 This utility may be used to perform the following tasks:

- 1856 — configuring software on target hosts that will actually be running the
1857 software
- 1858 — configuring independent of the remove and install utilities
- 1859 — configuring or unconfiguring hosts that share software from another host
1860 where the software is actually installed
- 1861 — reconfiguring when configuration failed, was deferred, or needs to be
1862 changed

1863 **B.4.3.3 Options**

1864 There is no additional rationale provided for this subclause.

1865 **B.4.3.4 Operands**

1866 There is no additional rationale provided for this subclause.

1867 **B.4.3.5 External Influences**

1868 There is no additional rationale provided for this subclause.

1869 **B.4.3.6 External Effects**

1870 There is no additional rationale provided for this subclause.

B.4.3.7 Extended Description

When there is no script, the software is still transitioned to **configured** by **swconfig**. The state of the fileset without any configuration requirements is still changed to denote to the users of the software that the software is ready to use. Having one state for both software that requires configuration, and for software that does not, is easier than checking that all software that requires configuration is in the configured state, and that all software that does not require configuration is in the installed state.

If there are not sufficient (or any for that matter) responses in the **response** file, the **configure** script can log that further interaction is required and exit with a failure. This can prompt the user to execute the **swconfig** utility with **ask=true**.

The **request** script is executed at the manager role at the end of the selection phase, after the user has specified the software, but before analysis or execution begins on the target roles. The developers of this part of ISO/IEC 15068 considered defining a separate phase between the selection and analysis phases for **swconfig** and **swinstall**, but maintained the request steps as part of the selection phase for simplicity.

Reconfiguration may be useful when some system configuration has changed. This may include running with **ask=true** so the user can input different information.

There is the case where the **configure** script is not sufficient for configuring the software. If there is another configuration process that needs to be run, then this process should not change the state of the software to configured. After the other process is run, it can change the state to configured using the **swmodify** utility. There are also situations where there can be multiple configurations of the same installed_software object. This part of ISO/IEC 15068 does not currently address this except by putting the burden on the software to manage the multiple configurations. This part of ISO/IEC 15068 does support the user rerunning the **configure** script each time a new configuration is needed. Using **swconfig -u** can likewise interact with the user to unconfigure one, but not all, of the configurations. For both of these cases, if the script exits with return code 3, the software does not transition to the **installed** (i.e., unconfigured) state.

The **configure** scripts should also adhere to specific guidelines. For example, these scripts are only executed in the context of the host that the software will be running on so they are not as restrictive as customize scripts. However, in a diskless or NFS environment, they need to use file locking on any updates to shared files, as there may be multiple **configure** scripts operating at the same time on these shared files. The **configure** and **unconfigure** scripts need to be noninteractive, but may use the information in the **response** files generated by the **ask** script.

For diskless, cold install (initial OS install), and generally building an OS to a separate disk, **swconfig** can be automatically run after the system reboots to its real host to configure all unconfigured filesets.

This part of ISO/IEC 15068 does not define how file sharing, including diskless machines, should be implemented. However, separable configuration and installation steps provide the basic building blocks.

1916 One possible file sharing solution involves each client having its own installed
1917 software catalog from which the shared software can be configured, and the **con-**
1918 **figured** state can be recorded. This catalog can be built by “link installing” the
1919 software; instead of loading files and running **preinstall** and **postinstall**
1920 scripts, link each product’s files to the client file system. Then, build the catalog
1921 information of this linked software as if it were installed and configure the client.

1922 Another possible solution involves each client recording its configured state in a
1923 shared installed software catalog. In order to do this, the installed software could
1924 maintain a *configured_instances* attribute to hold a list of configured client names.
1925 Each client’s **configure** and **unconfigure** script could add or delete its name
1926 from this list.

1927 These scripts could also control whether the installed software *state* attribute was
1928 changed from **installed** to **configured** via **swconfig**. If configuring, then
1929 the *reconfigure* option would need to be set to *true*. If unconfiguring, then the
1930 **unconfigure** script could exit with a return code of 3 (exclude) unless the
1931 *configured_instances* attribute was empty so that the installed software *state*
1932 would remain **configured**.

1933 **B.4.3.7.1 Examples**

1934 **B.4.3.8 Exit Status**

1935 There is no additional rationale provided for this subclause.

1936 **B.4.3.9 Consequences of Errors**

1937 There is no additional rationale provided for this subclause.

1938 **B.4.4 swcopy — Copy distribution**

1939 **B.4.4.1 Synopsis**

1940 There is no additional rationale provided for this subclause.

1941 **B.4.4.2 Description**

1942 This utility may be used to perform the following tasks:

- 1943 — Copy software from one distribution to another
- 1944 — Merge software from one distribution into another
- 1945 — Copy software to a temporary distribution located to improve **swinstall**
1946 reliability or performance

1947 **B.4.4.3 Options**

1948 There is no additional rationale provided for this subclause.

1949 B.4.4.4 Operands

1950 There is no additional rationale provided for this subclause.

1951 B.4.4.5 External Influences

1952 There is no additional rationale provided for this subclause.

1953 B.4.4.6 External Effects

1954 There is no additional rationale provided for this subclause.

1955 B.4.4.7 Extended Description

1956 The **swcopy** utility operates much like the **swinstall** utility. The key distinc-
1957 tion between **swcopy** and **swinstall** is the way products are loaded. With
1958 **swcopy**, products are not installed for general use below the root directory.
1959 Instead, they are placed into a distribution, which can then act as a source for
1960 **swinstall**.

1961 B.4.4.7.1 Examples

1962 Copy the *software_selections* listed in **/tmp/load.products** other default
1963 values defined in the **/var/adm/sw/defaults** file) as follows:

1964 **swcopy -f /tmp/load.products**

1965 Remove a product Foo from the distribution on the tape device **/dev/rct0** as fol-
1966 lows:

1967 **swcopy -s /dev/rct0 * @ /tmp/depot"**

1968 **swremove -d Foo @ /tmp/depot"**

1969 **swcopy -s /tmp/depot * @ /dev/rct0"**

1970 B.4.4.8 Exit Status

1971 There is no additional rationale provided for this subclause.

1972 B.4.4.9 Consequences of Errors

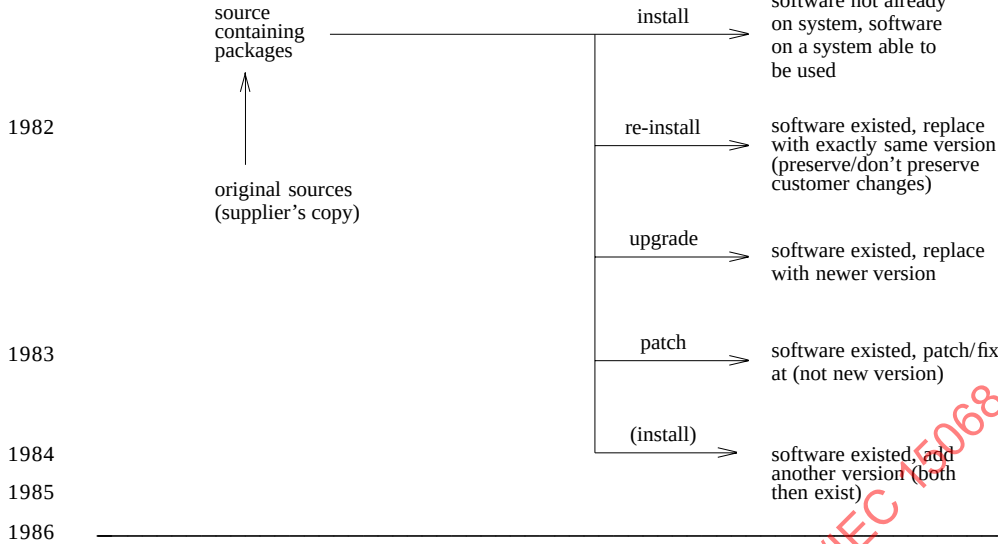
1973 There is no additional rationale provided for this subclause.

1974 B.4.5 swinstall — Install software

1975 The install software task can have different connotations depending upon the life
1976 cycle stage of the software package. Figure B-7 illustrates the various state tran-
1977 sitions that can occur during an installation of software. These transitions
1978 comprise the set of install software tasks supported by this part of ISO/IEC 15068.

1979 The facilities provided by **swinstall** and **swcopy** are basic building blocks on
1980 which other function may be built. A few examples are shown below simply for
the purpose of illustration. In the course of the illustration, various network sizes
are given, but they are fictitious and supplied solely for illustration.

1981



1986

Figure B-7 – Installation State Changes

1987

1988 — Some users might consider **swinstall** to work well for doing remote installation to tens of machines. But with hundreds of machines, perhaps a
1989 better strategy would be to use **swcopy** to place a distribution on several
1990 servers for use in several parallel **swinstall** invocations. And this example
1991 could be cascaded for hierarchical operation with thousands or hundreds
1992 of thousands of machines.

1994 — In addition, organizations that use special purpose software distribution
1995 programs could choose to use **swcopy** for that purpose, sending a copy of a
1996 distribution to some number of machines, pausing until assured that copies
1997 have arrived intact at all machines before proceeding, and finally causing
1998 **swinstall** to begin work on each machine at the same time.

1999 — An installation process that makes efficient use of network resources might
2000 analyze the routing of distributions or files, discover that some traverse a
2001 common path before diverging, and cause only one copy to be sent over the
2002 common link using the **swcopy** utility.

2003 Table B-4 includes all functionality initially considered for inclusion in the install
2004 software task. The second column describes the various operations while the first
2005 column lists the degree of support provided by the SVR4, HP-UX, and SCO Unix
2006 software management utilities.

2007 **B.4.5.1 Synopsis**

2008 There is no additional rationale provided for this subclause.