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

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

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

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

ISBN 0-7381-1568-1

Contents

FIGURES

TABLES

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:

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.

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

1.2 Normative References

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

{1} ISO/IEC 646: 1991, *Information technology—ISO 7-bit coded character set for information interchange* (International Reference Version).[2]

{2} ISO/IEC 9945-1: 1996 (ANSI/IEEE Std 1003.1-1996), *Information technology — Portable Operating System Interface (POSIX®) — Part 1: System Application Program Interface (API) [C Language].* [3]

{3} ISO/IEC 9945-2: 1993 (ANSI/IEEE Std 1003.2-1992), *Information technology — Portable Operating System Interface (POSIX®) — Part 2: Shell and Utilities.*

{4} ISO/IEC 10646-1: 1993, *Information technology—Universal Multiple-Octet Coded Character Set (UCS)—Part 1: Architecture and Basic Multilingual Plane.*

1.3 Conformance

1.3.1 Implementation Conformance

1.3.1.1 Conforming POSIX.7.2 Implementation

A Conforming POSIX.7.2 Implementation, also known as a "conforming implementation," shall meet all the following criteria:

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

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

3) IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 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.

1.3.2 Distribution Conformance

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

1.3.2.1 Strictly Conforming POSIX.7.2 Distribution

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

(1) Shall contain only those files and directories defined in the Software Packaging Layout (see Section 5)

(2) Shall contain only those software structures and attributes within those structures described in this part of ISO/IEC 15068

(3) Shall use only the values of software structure attributes defined as valid within described in this part of ISO/IEC 15068

(4) Shall use only the POSIX.2 {3} shell, `sh`, to invoke control files

(5) Shall not use facilities, structures, attributes, or values designated as *obsolescent*

Within this part of ISO/IEC 15068, any restrictions placed upon a Conforming POSIX.7.2 Distribution also shall restrict a Strictly Conforming POSIX.7.2 Distribution.

1.3.2.2 Conforming POSIX.7.2 Distribution

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

(1) Additional attributes in the manner supported by this part of ISO/IEC 15068.

(2) Additional files within the distribution, except where this part of ISO/IEC 15068 prohibits such files.

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

1.3.2.2.1 ISO/IEC Conforming POSIX.7.2 Distribution

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

1.3.2.2.2 <National Body> Conforming POSIX.7.2 Distribution

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

1.3.2.3 Conforming POSIX.7.2 Distribution Using Extensions

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

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

In addition to the documentation required of a Conforming POSIX.7.2 Distribution, such a distribution

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

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

(3) Shall document the extent to which the software contained in the distribution can be managed by the utilities in accordance with this part of ISO/IEC 15068

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

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

1.3.2.4 Documentation

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

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.

1.4 Test Methods

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

Section 2: Terminology and General Requirements

2.1 Conventions

2.1.1 Editorial Conventions

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

Table 2-1 – Typographical Conventions

Reference	**Example**
Attribute of a Structure (object)	*filesets* or *product.filesets*
C-Language Data Type	*long*
C-Language Function	*system*()
Cross Reference: Annex	Annex A
Cross Reference: Clause	2.3
Cross Reference: Other Standard	ISO/IEC 9999-1 {*n*}
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	<*directory pathname*>
Script Types	**`preinstall`**
Special Character	**`<newline>`**
Symbolic Constant, Limit	{_POSIX_VDISABLE}, {LINE_MAX}
Table Reference	Table 6-1
Utility Extended Option	*reinstall*
Utility Extended Option with Value	*reinstall=true*
Utility Name	**`awk`**
Utility Operand	*file_name*
Utility Option	**`-c`**
Utility Option with Option-Argument	**`-w`** *width*
Value of an Attribute	**`true`**

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

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

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

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

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

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

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

(1) Terms defined in 2.2.1, and 2.2.2 are expressed as subclause titles. Alternative forms of the terms appear in [brackets]. At the conclusion of this type of definition, the designation of a standard within [brackets] indicates that the text of the definition was copied from that standard, with only editorial changes, if any.

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

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

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

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

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

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

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

`%`, **`begin`**	When no confusion would result, the character or string is rendered in the Courier font and used directly in the text.

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

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

Parenthetical expressions within normative text also contain normative information. The general typographic hierarchy of parenthetical expressions is:

{ [( )] }

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

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

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

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

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

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

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

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

In some cases certain characters are interpreted as special characters by the 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.

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

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

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

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

— Configure it for conformance with this part of ISO/IEC 15068;

— Select some of the various optional facilities described by this part of ISO/IEC 15068, through customization by local system administrators or operators.

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

NOTE: See 2.2.1.5 and 1.3.1.3.

2.2.1.5 implementation defined: An indication that the implementation provider shall define and document the requirements for correct program constructs and correct data of a value or behavior.

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.

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

2.2.1.6 may: An indication of an optional feature or behavior of the implementation that is not required by this part of ISO/IEC 15068, although there is no prohibition against providing it.

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

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

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

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

2.2.1.9 should:

(1) With respect to implementations, an indication of an implementation 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

Distributions.

2.2.1.10 system documentation: All documentation provided with an implementation, except the conformance document.

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

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

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

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

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

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

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

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

2.2.2 General Terms

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

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

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

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

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

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

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

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

NOTE: See 3.8.

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

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

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

NOTE: See Section 5.

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

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

The configuration of software is performed by this role.

2.2.2.11 command line interface: A means of invoking utilities by issuing commands from within a POSIX.2 {3} shell, implying that neither graphics nor windows are required.

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

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

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

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

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

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

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

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

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

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

Decimal character strings shall consist only of the following characters:

`0 1 2 3 4 5 6 7 8 9`

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

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

NOTE: See 2.2.2.21 and 2.2.2.35.

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

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

A dependency is the object upon which another object depends.

2.2.2.23 dependency_spec: A `software_spec` that describes a dependency.

NOTE: See 4.1.4.1.

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

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

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

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

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

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

2.2.2.28 distribution catalog: The catalog of metadata for a distribution software_collection.

Unlike a catalog for an installed_software object, a distribution catalog is stored in a particular exported catalog structure that is part of the software packaging layout.

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

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

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

This is also referred to as downgrading or reverting.

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

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

2.2.2.32 execution phase: The operations a software administration utility performs that modify the target.

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

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

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

NOTE: See Section 5.

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

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

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

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

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

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

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

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

NOTE: See 3.9.

2.2.2.39 file storage structure: The storage directories in the software packaging layout under which the actual software files for each fileset are located.

NOTE: See Section 5.

2.2.2.40 fully qualified software_spec: A `software_spec` that always identifies a software object unambiguously.

NOTE: See 4.1.4.1.

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

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

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

2.2.2.43 hexadecimal character string: A sequence of characters from the set of hexadecimal digits, preceded by the two characters `0x` (zero followed by a lowercase “x”).

Hexadecimal character strings shall consist only of the following characters:

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

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

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

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

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

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

2.2.2.46 INDEX file: The file within an exported catalog containing the metadata describing the software objects and attributes for all bundles, products, subproducts and filesets.

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

2.2.2.47 INFO file: For each product and fileset, the file within an exported catalog containing the metadata describing the software_file objects and attributes.

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

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

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

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

2.2.2.50 installed_software: A software_collection containing installed software.

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

2.2.2.51 installed_software catalog: The catalog of metadata for an installed_software software_collection.

Unlike a catalog for a distribution object, the storage and format of an installed_software catalog is undefined within this part of ISO/IEC 15068. The 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.

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

2.2.2.53 integer character string: A decimal character string, an octal character string, or a hexadecimal character string.

NOTE: See 2.2.2.19, 2.2.2.63, and 2.2.2.43.

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

2.2.2.55 kernel: The nucleus of the operating system.

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

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

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

2.2.2.58 locatable software: Software that contains locatable filesets.

2.2.2.59 manager role: Where each task is initiated.

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

2.2.2.60 metadata: The information kept about software.

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

2.2.2.61 newline string: A white space string consisting only of the `<newline>` character.

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

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

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

Octal character strings shall consist only of the following characters:

```
0 1 2 3 4 5 6 7
```

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

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

NOTE: See 2.2.2.21.

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

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

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

2.2.2.67 pathname character string: A sequence of characters from the portable filename character, including the `/` (slash) character.

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

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

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

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

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

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

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

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

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

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

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`.

2.2.2.75 rebooting: An implementation-defined procedure generally used to terminate and then restart operations on the target system.

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

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

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

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

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

2.2.2.79 session: An execution of a software administration command from initiation to completion on all applicable roles.

2.2.2.80 shell token string: A sequence of shell tokens.

A shell token string shall be a portable character string.

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

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

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

NOTE: See 2.2.2.90.

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

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

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

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

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

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

NOTE: See 5.2.

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

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

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

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

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

2.2.2.90 software object: An object that inherits attributes of the software common class, meaning a bundle, product, subproduct, or fileset object.

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

NOTE: For a detailed description, see Section 5.

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

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

A software pattern match string shall be portable character string.

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

NOTE: See 4.1.4.1.1.

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

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

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

2.2.2.97 source role: Where the software exists in a form suitable for distribution, forming a context for the establishment of a repository of software from which the manager may choose to distribute to targets.

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

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

NOTE: See 3.10.

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

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

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

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

2.2.2.101 target: The specification of a target distribution object, or installed software object, for a software administration utility. The target host provides a means to locate the target role and the target path is a path accessible to the target host.

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

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

2.2.2.104 target role: Where software is installed, removed, listed, and otherwise operated on by the utilities.

For example, when installing software, the target is where software is installed 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 installed_software objects or the distributions from which software is being removed.

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

This is also referred to as upgrading.

2.2.2.106 vendor: A supplier of packaged software.

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

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

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

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

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

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

2.2.2.110 white space string: A sequence of one or more white space characters (as defined in 2.2.2.191 of POSIX.2 {3}) including `<space>`, `<tab>`, and `<newline>`.

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

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

Such characters are used in software pattern match strings 2.2.2.92.

2.2.3 Abbreviations

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

2.2.3.1 API: Application Programming Interface

2.2.3.2 CLI: Command Line Interface, as defined in 2.2.2.11.

2.2.3.3 C Standard: ISO/IEC 9899: 1990, *Information technology—Programming languages—C* {B13}.

2.2.3.4 CRC: Cyclic Redundancy Check.

2.2.3.5 GUI: Graphical User Interface, as defined in 2.2.2.41.

2.2.3.6 IRV: ISO/IEC 646: 1991, *Information technology—ISO 7-bit coded character set for information interchange*, International Reference Version {1}.

2.2.3.7 newline: A newline string, as defined in 2.2.2.61.

2.2.3.8 OS: Operating System.

2.2.3.9 POSIX.1: ISO/IEC 9945-1: 1990 (ANSI/IEEE Std 1003.1-1990): *Information technology—Portable Operating System Interface (POSIX®)—Part 1: System Application Program Interface (API) [C Language]* {2}.

2.2.3.10 POSIX.1a: IEEE P1003.1a, *Draft Standard for Information technology—Portable Operating System Interface (POSIX®)—Part 1: System Application Program Interface (API) [C Language]* {B21}.

2.2.3.11 POSIX.2: ISO/IEC 9945-2: 1993 (ANSI/IEEE Std 1003.2-1992): *Information technology—Portable Operating System Interface (POSIX®)—Part 2: Shell and Utilities* {3}.

2.2.3.12 POSIX.7.2: This part of ISO/IEC 15068.

2.2.3.13 PSF: product specification file as defined in 2.2.2.71.

2.2.3.14 symlink: symbolic link, as defined in 2.2.2.99.

2.2.3.15 UTF-8: UCS Transformation Format 8, as defined in ISO/IEC 10646-1: 1993, *Information technology—Universal Multiple-Octet Coded Character Set (UCS)—Part 1: Architecture and Basic Multilingual Plane, Amendment 2: UCS Transformation Format 8 (UTF-8), 1996* {4}.

2.2.3.16 white space: A white space string, as defined in 2.2.2.110.

2.3 Dependencies on Other Standards

2.3.1 Features Inherited From POSIX.1

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

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

2.3.1.1 File System

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

2.3.1.2 Environment Variables

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

2.3.1.3 Data Interchange Format

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

2.3.2 Features Inherited From POSIX.2

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

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

Section 3: Software Structures

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

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

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

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

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

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

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

The attribute tables in this section list the following information:

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

Length The maximum permitted length of the value of the attribute.

All attribute values in this part of ISO/IEC 15068 are represented only as strings. The length is the maximum permitted length of the value in bytes or, for attributes whose values are lists, the maximum permitted number of items permitted in the list. Since the means of storing such data for installed software is undefined within this part of ISO/IEC 15068, an implementation may store such values internally in different structures for installed software. See 2.2.2.28, 2.2.2.33, and 2.2.2.51.

Permitted Values

The character sequences permitted as values for this attribute.

Default Value

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

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

The attribute tables are broken into the following three groups:

(1) The top group contains the attributes that are used to identify a particular instance.

(2) The middle group contains the rest of the attributes that describe other information or behaviors for the object.

(3) The bottom group contains the attributes that describe the objects contained within this object. The way in which these lists are represented in software definition files is described in 5.2. The way in which these lists are represented by **swlist** is described in 4.6.3.

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

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

3.1 Software_Collection

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

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

Each software_collection has a catalog associated with it that contains the metadata describing all software objects in that collection.[4)]

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

Table 3-1 – Attributes of the Software_Collection Common Class

Attribute	Length	Permitted Values	Default Value
path	Undefined	Pathname character string	Implementation defined
dfiles	64	Filename character string	`dfiles`
layout_version	64	`1.0`	`1.0`
pfiles	64	Filename character string	`pfiles`
bundles	Undefined	List of `bundle_software_spec`s	Empty list
products	Undefined	List of `product_software_spec`s	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_spec`s.

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_spec`s.

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
uuid	64	Portable character string	Empty string
media	Undefined	List of media *sequence_number* 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**
sequence_number	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
catalog	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
tag	64	Filename character string	Empty string
title	256	Portable character string	Empty string
description	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:

description	A more detailed description of the vendor or information about the vendor.
tag	A short identifying name of the vendor that supplied the product. This attribute is used to to distinguish products and bundles from different vendors, and for resolving software specifications. Each software vendor should attempt to have a unique value for the *tag* attribute.
title	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
tag	64	Filename character string	None
create_time	16	Integer character string	None
description	Undefined	Portable character string	Empty string
mod_time	16	Integer character string	None
size	32	Integer character string	None
title	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:

create_time	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}.
description	A more detailed description of the software object.
mod_time	A value that shall be set by the implementation to be the time that the catalog information for this object was last written.

Time shall be represented as seconds since the Epoch, as defined in 5.6.1.3 of POSIX.1 {2}.

size The sum of the sizes in bytes of all files and control_files contained within the software object.

For objects other than filesets, the value is computed dynamically as required. See 5.2.6, 5.2.7, and 5.2.8.

tag A short name associated with the software object.

It is the one attribute that is always required to identify a software object. For more information on software selections, see 4.1.4.1.

title A longer name associated with the software object, used for display purposes.

3.7 Products

Products can contain filesets, which can be grouped into subproducts. Products are named by their *tag* attributes. A particular product object is uniquely identified within a software_collection by the *tag* attribute and by the version distinguishing attributes. The attributes that uniquely distinguish a particular product version within a software_collection are *revision*, *architecture*, *vendor_tag*, *location*, and *qualifier*.

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

See 4.1.4.1.2 on software compatibility.

3.7.1 Product Attributes

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

all_filesets This is a list of all filesets defined for the product, as opposed to what is currently installed, described by the *filesets* attribute. The *all_filesets* attribute is used to determine completeness of this product when another software object has a dependency on this product. In checking a product prerequisite or corequisite, the existence of a *fileset.tag* in *all_filesets* that is not actually **`installed`** or **`available`** indicates that the dependency is not satisfied.

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

architecture A vendor-defined string used to distinguish variations of a product.

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
architecture	64	Portable character string	Empty string
location	Undefined	Pathname character string	<*product.directory*>
qualifier	64	Portable character string	Empty String
revision	64	Portable character string	Empty string
vendor_tag	64	Filename character string	Empty string
all_filesets	Undefined	List of fileset *tag* values	Empty list
control_directory	Undefined	Filename character string	<*product.tag*>
copyright	Undefined	Portable character string	Empty string
directory	Undefined	Pathname character string	`/`
instance_id	16	Filename character string	`1`
is_locatable	8	One of: `true`, `false`	`true`
postkernel	Undefined	Pathname character string	implementation defined
layout_version	64	`1.0`	`1.0`
machine_type	64	Software pattern matching string	Empty string
number	64	Portable character string	Empty string
os_name	64	Software pattern matching string	Empty string
os_release	64	Software pattern matching string	Empty string
os_version	64	Software pattern matching string	Empty string
control_files	Undefined	List of *control_file.tag* values	Empty list
subproducts	Undefined	List of *subproduct.tag* values	Empty list
filesets	Undefined	List of *fileset.tag* 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 be relocated to the *location* directory during installation by replacing the *product.directory* portion with the *product.location*.

filesets	A list of the values of the *tag* attribute for all the filesets in the product that are currently **installed** (in an installed_software object) or **available** (in a distribution).
instance_id	A single attribute that distinguishes versions of products (and bundles) with the same tag. It is a simple form of the version distinguishing attributes, valid only within the context of an exported catalog.
is_locatable	A Boolean value indicating whether any of the filesets in the product have the *is_locatable* attribute set to **true**.
layout_version	This attribute, and its value, are included for future use.
location	Used for resolving **software_specs** for installed software. A specific product location refers to all filesets of that product that are installed at that location. This is the path beneath which the relocatable files of that product are stored. See 4.5.7.3.1. This attribute is valid only for products in installed_software.
machine_type	A software pattern matching string describing valid machine members of the *uname* structure as defined by 4.4.1 of POSIX.1 {2}. It is used for determining compatibility.
number	The semantics associated with the values of this attribute are undefined. This attribute can be used to store such vendor-defined values as part number, order number or serial number.
os_name	A software pattern matching string describing valid sysname members of the *uname* structure as defined by 4.4.1 of POSIX.1 {2}. It is used for determining compatibility.
os_release	A software pattern matching string describing valid release members of the *uname* structure as defined by 4.4.1 of POSIX.1 {2}. It is used for determining compatibility.
os_version	A software pattern matching string describing valid version members of the *uname* structure as defined by 4.4.1 of POSIX.1 {2}. It is used for determining compatibility.

postkernel	
	The path to the script that is run after the kernel filesets have been installed.
	Any product containing kernel filesets should include this path. If this attribute is supplied, the corresponding script shall be run if it exists relative to the root directory of the installed_software. If this attribute is not supplied, then the implementation-defined path (the default value for the attribute) shall be used if it exists relative to the root directory of the installed_software. Note that the use of an alternate root directory may mean that the default path does not exist relative to the root directory of the installed_software.
qualifier	Specified by a user when installing software and used for identifying a product (or set of product versions) using a logical name.
	Applies only to products in installed_software.
revision	A vendor-defined string describing the revision of the product.
	It is used for presentation purposes and for resolving software specifications. The revision shall be interpreted as a **.** (period) separated string. See 4.1.4.1.
subproducts	A list of the values of the *tag* attribute for all the subproducts in the product.
vendor_tag	A short identifying name of the vendor that supplied the product.
	This attribute may also be used to identify a vendor object containing additional attributes describing the vendor.
	This attribute is used to distinguish software objects, allowing more than one vendor to produce a product with the same value of the other version distinguishing attributes. It is used for presentation purposes and for resolving software specifications.

3.8 Bundles

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

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

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

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
architecture	64	Portable character string	Empty string
location	Undefined	Pathname character string	<*bundle.directory*>
qualifier	64	Portable character string	Empty String
revision	64	Portable character string	Empty string
vendor_tag	64	Filename character string	Empty string
contents	Undefined	List of `software_spec`s	Empty list
copyright	Undefined	Portable character string	Empty string
directory	Undefined	Pathname character string	Empty string
instance_id	16	Filename character string	`1`
is_locatable	8	One of: `true`, `false`	`true`
layout_version	64	`1.0`	`1.0`
machine_type	64	Software pattern matching string	Empty string
number	64	Portable character string	Empty string
os_name	64	Software pattern matching string	Empty string
os_release	64	Software pattern matching string	Empty string
os_version	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_spec`s 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.

It is a simple form of the version distinguishing attributes, valid only within the context of an exported catalog.

is_locatable	A Boolean vaue indicating whether any of the contents in the bundle have the *is_locatable* attribute set to `true`.
layout_version	This attribute, and its value, are included for future use.
location	An attribute whose value is set when installing software and used for resolving `software_spec`s for installed software. When installing a bundle the *bundle.location* is prefixed to the location specification for each `software_spec` in the contents of the bundle, before that `software_spec` is resolved. The *contents* attribute of the bundle is not modified. Applies only to bundles in installed_software.
machine_type	A software pattern matching string describing valid machine members of the *uname* structure as defined in 4.4.1 of POSIX.1 {2}. It is used for determining compatibility.
number	The semantics associated with the values of this attribute are undefined. This attribute can be used to store such vendor-defined values as part number, order number or serial number.
os_name	A software pattern matching string describing valid sysname members of the *uname* structure as defined in 4.4.1 of POSIX.1 {2}. It is used for determining compatibility.
os_release	A software pattern matching string describing valid release members of the *uname* structure as defined in 4.4.1 of POSIX.1 {2}. It is used for determining compatibility.
os_version	A software pattern matching string describing valid version members of the *uname* structure as defined in 4.4.1 of POSIX.1 {2}. It is used for determining compatibility.
qualifier	Specified by a user when installing software, and used for identifying a bundle (or set of bundle versions) using a logical name. Applies only to bundles in installed_software.
revision	A vendor-defined string used to distinguish different revisions of bundles from one another. It is used for presentation purposes and for resolving software specifications.

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

3.9 Filesets

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

The fileset class inherits attributes from the software common class. Filesets contain the actual files and control_files that make up the software product. A particular fileset object is identified within a product by the *tag* attribute.

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

Table 3-9 – Attributes of the Fileset Class

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

3.9.1 Fileset Attributes

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

control_directory
: The name of the fileset control directory below which the control_files for the fileset are stored within an exported catalog. See 5.1.

control_files
: A list of the values of the *tag* attribute for the control_files in the fileset.

corequisites
: A list of **dependency_spec**s for software required to be installed and configured for this fileset to work.

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

: The software specified by the **dependency_spec** shall be complete in order for the dependency to be resolved successfully. See *all_filesets* in 3.7.

exrequisites
: A list of **dependency_specs** for software required not to be installed when this fileset is installed.

: Dependencies shall be considered when installing, configuring, verifying, and removing software. See 4.3.7.2, 4.5.7.2, 4.9.7.2, and 4.10.7.2.

: No part of the software specified by the **dependency_spec** may be installed in order for this dependency to be resolved successfully.

files
: A list of the values of the *path* attribute for the files in the fileset.

is_kernel
: A Boolean value indicating the fileset requires a kernel rebuild.

is_locatable
: A Boolean value indicating if the fileset may be re-located during installation.

is_reboot
: A Boolean value indicating the host on which the fileset is configured should be re-booted.

location
: Specifies the location below which relocatable files are stored.

: This attribute is only valid for filesets in installed software. It differs from the *product.directory* attribute only if relocation was specified during installation. See 4.5.7.3.1.

media_sequence_number
: Identifies the *media.sequence_number* for the medium on which the files for this fileset is found.

: If a single fileset spans multiple media, this attribute identifies a list of *media.sequence_number* 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.

prerequisites A list of **`dependency_spec`**s for software required to be installed prior to the installation of this fileset and configured prior to the configuration of this fileset.

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

The software specified by the **`dependency_spec`** shall be complete in order for the dependency to be resolved successfully. See *all_filesets* in 3.7.

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

revision Defines the revision of the fileset.

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

state An indication of the current status of the fileset.

This attribute may have one of the following values: **`configured`**, **`installed`**, **`corrupt`**, **`removed`**, **`available`**, and **`transient`**.

3.10 Subproducts

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

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

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

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

Table 3-10 – Attributes of the Subproduct Class

Attribute	Length	Permitted Values	Default Value
contents	Undefined	List of *tag* 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
cksum	16	Integer character string	None
compressed_cksum	16	Integer character string	None
compressed_size	16	Integer character string	None
compression_state	16	One of: `uncompressed`, `compressed`, `not_compressible`	`uncompressed`
compression_type	64	Filename character string	Empty string
revision	64	Portable character string	Empty string
size	16	Integer character string	None
source	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.

This attribute can be used for computation of disk space analysis when the file will remain compressed after a copy.

compression_state

Indicates which one of the following conditions is true:

— Uncompressed but permitted to be compressed in a distribution (if this attribute has the value **`uncompressed`** or if no value is supplied for the attribute)

— Already compressed (if this attribute has the value **`compressed`**)

— Uncompressed and not permitted to be compressed in a distribution (if this attribute has the value **`not_compressible`**)

compression_type

Specifies the compression method used to compress the file if the value of the *compression_state* attribute is **`compressed`**.

The values supported for *compression_type* are implementation defined. The way in which an implementation uses this value to implement or execute the compression or uncompression of a file is undefined.

revision Describes a string indicating the revision level of the file.

size Indicates the size of the software file in bytes.

This attribute has the same values and meaning as *st_size* 5.6.1 of POSIX.1 {2}.

source When used in a PSF, this attribute specifies the pathname of the file or control_file to be placed in the distribution by the **`swpackage`** utility.

3.12 Files

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

The file class inherits attributes from the software_file common class.

A particular file object is identified within a fileset by the *path* attribute. When a file is located on a distribution, the *path* attribute indicates the intended installation location of the file. The value of the *path* attribute is also the path below the storage directory for that fileset within file storage structure of the distribution (see 5.1.2). While a file is installed (in an installed_software object), the *path* attribute indicates the actual location of the file. This path is relative to the root directory for that installed_software object.

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

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

Table 3-12 – Attributes of the File Class

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

3.12.1 File Attributes

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

gid The numeric group id of the file.

This attribute has the same values and meaning as *st_gid*, 5.6.1 of POSIX.1 {2}.

group The group name of the file.

This attribute has the same values and meaning as *gr_name*, 9.2.1 of POSIX.1 {2}.

is_volatile Indicates a file whose contents can change, or that can be removed after it has been installed.

link_source The pathname of the target of the link.

This attribute only has meaning if the file type is a hard or symbolic link.

major This attribute only has meaning if the file type is character or block special file.

This attribute has the same values and meaning as the *devmajor* field in the `tar` archive specified in 10.1.1 of POSIX.1 {2}.

minor This attribute only has meaning if the file type is character or 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}.

mode	An octal representation of the permissions bits of the file.
	This attribute has the same values and meaning as *st_mode*, 5.6.1 of POSIX.1 {2}, except that this attribute has no meaning if the file type is a hard or symbolic link.
mtime	The time of the last data modification of the file.
	This attribute has the same values and meaning as *st_mtime*, 5.6.1 of POSIX.1 {2}.
owner	The name of the owner of the file.
	This attribute has the same values and meaning as *pw_name*, 9.2.2 of POSIX.1 {2}.
path	The pathname of the file.
type	Supported file types are those described in 5.6.1.1 of POSIX.1 {2}, plus hard link and symbolic link.
	The permitted values of this attribute are the following: `f` (regular file), `d` (directory), `h` (hard link), `s` (symbolic link), `p` [named pipe (FIFO)], `b` (block special device), and `c` (character special device).
uid	The numeric user id of the file.
	This attribute has the same values and meaning as *st_uid*, 5.6.1, of POSIX.1 {2}.

3.13 Control_Files

Control_files can be scripts, data files, or `INFO` files. The product and fileset `INFO` files in the software packaging layout are included as control_files. Control scripts are the vendor-supplied scripts executed at various steps by the software administration utilities.

The control_file class inherits attributes from the software_file common class.

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

3.13.1 Control_File Attributes

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

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

Table 3-13 – Attributes of the Control File Class

Attribute	Length	Permitted Values	Default Value
tag	64	Filename character string	None
interpreter	Undefined	Filename character string	`sh`
path	Undefined	Filename character string	None
result	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`.

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**.

`-c` *catalog*
: Used to specify a file with the software definition file syntax or directory with the exported catalog structure.

This is where software catalog information (metadata) is to be stored to or retrieved from. If this information fits into one file, then the catalog can be a file, otherwise it shall be a directory. See section 5 and 5.2.

Applies to **`swask`**, **`swconfig`**, **`swinstall`**, **`swlist`**, and **`swmodify`**.

`-d` Indicates to the utility that the operation is on a distribution instead of installed_software.

Applies to **`swlist`**, **`swmodify`**, **`swremove`**, and **`swverify`**.

`-f` *file* Reads the list of *software_selections* from *file*.

If this option is specified multiple times, all the software specified by each file shall be included in the operation. All of the software specified by using this option, as well as all the software specified directly as arguments to the utility, shall be included in the operation.

The file shall contain one software selection per line where a software selection uses the syntax for **`software_spec`** defined in 4.1.4.1. Blank lines shall be ignored. Within the file, the # (pound) character shall act as a comment character. On any line containing a # (pound) character, all characters that follow the # (pound) character up to, but excluding, the next **`<newline>`**, shall be ignored.

Applies to all utilities.

`-p` Previews the operation without making any permanent modifications to the target.

An implementation should run any control scripts that are executed as part of the selection or analysis phase of the command being previewed, but shall not run any that are executed in the execution phase.

This option can be used with any or all of the other options to understand the impact of an operation before performing it.

Applies to **`swconfig`**, **`swcopy`**, **`swinstall`**, **`swmodify`**, **`swpackage`**, and **`swremove`**.

`-r` Indicates to the utility that the operation is on an installed_software object located at an alternate root, instead of either a distribution or the installed_software object located at `/`.

Applies to **`swinstall`**, **`swlist`**, **`swmodify`**, **`swremove`**, and **`swverify`**.

-s *source*
Specifies the software source for the operation.

For **swinstall**, **swask**, and **swcopy** a source can be specified using the syntax in 4.1.4.2. For **swpackage**, the source shall be a product specification file.

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

-t *targetfile*
Reads the list of targets from *targetfile*.

If this option is specified multiple times, all the targets specified by each file shall be included in the operation. All of the targets specified by using this option, as well as all the targets specified directly as arguments to the utility, shall be included in the operation.

The file shall contain one target per line, where a target uses the syntax for **software_collection_spec** defined in 4.1.4.2. Blank lines shall be ignored. Within the file, the # (pound) character shall act as a comment character. On any line containing a # (pound) character, all characters that follow the # (pound) character up to, but excluding, the next **<newline>**, shall be ignored.

Applies to **swconfig**, **swcopy**, **swinstall**, **swlist**, **swmodify**, **swpackage**, **swremove**, and **swverify**.

-u This is the option used to specify undo or delete behavior to a utility.

Applies to **swconfig** and **swmodify**.

-x *option=value*
Used to override the value of an extended option in the defaults file.

The extended options supported are described in 4.1.5.2. This option can be specified multiple times. If any extended option is defined more than once, the precedence rules from 4.1.5.3.1 shall be used.

Applies to all utilities.

-X *options_file*
Used to override the defaults specified in the system defaults file.

The options supported are described in 4.1.5.2. This option can be specified multiple times. If any extended option from any file is defined more than once, the precedence rules from 4.1.5.3.1 shall be used.

The file shall have the format defined in 4.1.5.3.

Applies to all utilities.

4.1.3.1 Non-interactive Operation

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

The way in which **swinstall**, **swcopy**, and **swpackage** utilities handle multiple volumes for sources or targets is implementation defined.

4.1.4 Operands

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

4.1.4.1 Software Specification and Logic

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

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

```
%token          FILENAME_CHARACTER_STRING     /* as defined in 2.2.2.37  */
%token          NEWLINE_STRING                /* as defined in 2.2.2.61  */
%token          PORTABLE_CHARACTER_STRING     /* as defined in 2.2.2.68  */
%token          SOFTWARE_PATTERN_MATCH_STRING /* as defined in 2.2.2.92  */
%token          WHITE_SPACE_STRING            /* as defined in 2.2.2.110 */


%start  software_selections
%%

software_selections      : software_selections ws software_spec
                         | software_spec
                         ;

software_spec            : bundle_software_spec
                         | product_software_spec
                         ;

bundle_software_spec     : bundle_qualifier version
                         | bundle_qualifier '.' product_qualifier  version
                         ;

bundle_qualifier         : bundle_qualifier '.' bundle_tag
                         | bundle_tag
                         ;

product_software_spec    : product_qualifier version
                         ;

product_qualifier        : product_tag subproduct_qualifier fileset_qualifier
                         ;

subproduct_qualifier     : /* empty */
                         | subproduct_qualifier '.' subproduct_tag
```

```
                            | '.' subproduct_tag
                            ;

fileset_qualifier           : /* empty */
                            | '.' fileset_tag
                            ;

bundle_tag                  : sw_pattern
                            ;

product_tag                 : sw_pattern
                            ;

fileset_tag                 : sw_pattern
                            ;

subproduct_tag              : sw_pattern
                            ;

version                     : /* empty */
                            | ',*'
                            | version_qualifier
                            ;

version_qualifier           : version_qualifier ver_item
                            | ver_item
                            ;

ver_item                    : ',' ver_id '='
                            | ',' ver_id '=' sw_pattern
                            | ',' 'r' rel_op dotted_string
                            ;

sw_pattern                  : SOFTWARE_PATTERN_MATCH_STRING
                            ;

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

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

dotted_string               : FILENAME_CHARACTER_STRING
                            ;

ws                          : WHITE_SPACE_STRING
                            ;


%start  dependency_spec
%%

dependency_spec             : dependency_spec '|' software_spec
                            | software_spec
                            ;
```

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

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

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

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

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

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

Table 4-1 – Software_spec Version Identifiers

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

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

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

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

software_spec are included in the list of software that match the **software_spec**.

(2) The comparison performed when the operator is **=** shall be a software pattern match as described in 2.2.2.92. If the **ver_id** is specified and the value is an empty string, then the comparison is successful only if the corresponding attribute is not specified. See 4.1.4.1.1.

(3) When **rel_op** is used, the comparison shall be performed on the specified attribute by dividing it into segments separated by the **.** (period) character.

NOTE: As defined in the syntax for **ver_item**, this comparison is required for *revision* only and any other comparison is undefined.

If there is no period in an attribute, it contains one segment. The segments shall be compared with the corresponding segments of the **dotted_string**. If all characters in both segments to be compared are decimal digit characters (0-9), the comparison shall be based on the decimal numeric value of the segments, starting with the leftmost segment.

NOTE: Leading zeros are acceptable in such segments.

If either segment includes any character other than a decimal digit character, a string comparison shall be made to determine the relation. String comparisons shall be made using, as a collation sequence, the order of characters in IRV {1}. If one operand has fewer segments than the other, the unmatched segments shall be compared against the value **0** (zero).

(4) When applied to software in installed software collections, use of either the **l** (location) or **q** (qualifier) **ver_id** shall cause comparison with the value of the *location* or *qualifier* attribute respectively for each product or bundle in the installed_software object.

For distributions, use of either the **l** (location) or **q** (qualifier) **ver_id** shall be ignored for the purpose of comparisons. Although not used for comparisons, the location and qualifier **ver_id**s are used by the **swinstall** utility as the *location* attribute for installing the software, and the *qualifier* attribute for the software, respectively.

(5) When software selections are applied to a source or target, and a **software_spec** resolves to more than one software object, then the **software_spec** shall be considered ambiguous. An ambiguous selection may be elective or incidental. An elective ambiguous selection occurs when a **sw_pattern** in a **software_spec** contains a wildcard character or when the version contains a **rel_op**, or when the **sw_pattern** is missing. In all other cases the selection is an incidental ambiguous selection. An incidental ambiguous selection is only valid for **swlist**, and for other utilities generates an event.
(SW_ERROR: SW_SELECTION_NOT_FOUND_AMBIG)

(6) If the **software_spec** begins with a *bundle.tag* definition, then that 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.

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

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

4.1.4.1.1 Fully Qualified Software_spec

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

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

4.1.4.1.2 Software Compatibility

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

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

4.1.4.2 Source and Target Specification and Logic

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

```
%token          HOST_CHARACTER_STRING     /* as defined in 2.2.2.43  */
%token          PATHNAME_CHARACTER_STRING /* as defined in 2.2.2.64  */
%start  target
%%

target          : software_collection_spec
                ;

software_collection_spec : HOST_CHARACTER_STRING ':' PATHNAME_CHARACTER_STRING
```

```
                               | HOST_CHARACTER_STRING ':'
                               | HOST_CHARACTER_STRING
                               | PATHNAME_CHARACTER_STRING
                               ;

%start  source
%%

source          : software_collection_spec
                ;
```

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

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

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

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

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

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

4.1.5 External Influences

4.1.5.1 Defaults and Options Files

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

4.1.5.2 Extended Options

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

allow_downdate=false

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

If *allow_downdate=false*, do not allow installation of a lower revision of a fileset that is already installed at a higher revision in this location.

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

Applies to **`swinstall`**.

allow_incompatible=false

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

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

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

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

allow_multiple_versions=false

Controls the ability to configure multiple versions of a product.

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

Applies to **`swconfig`**.

ask=false

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

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

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`**.

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

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

autoreboot=false

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

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

Applies to **`swinstall`**.

autorecover=false

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

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

If *autorecover=true* and an error occurs, implementations shall, for any fileset having an install error, execute the **`unpostinstall`** script (if the **`postinstall`** script had been run) and the **`unpreinstall`** script, restore the files within the fileset from a copy saved prior to the failed install, and the restore the value of the *state* attribute. After recovery of the applicable filesets, installation can continue with the rest of the filesets in that product and the rest of the products in the software selections.

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

Applies to **`swinstall`**.

autoselect_dependencies=as_needed

Controls automatic dependency selection.

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

If *autoselect_dependencies=as_needed*, (the default for **`swinstall`** and **`swcopy`**), then autoselected dependencies shall only be operated upon if the dependency is not already met on the target. This value 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 are not already met on the target, an error shall occur when *enforce_dependencies*=*true*.

Applies to **swask**, **swconfig**, **swcopy**, **swinstall**, and **swverify**.

autoselect_dependents=*false*

Controls automatic dependency selection.

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

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

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

check_contents=*true*

Controls verification of file contents.

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

If *check_contents*=*false*, then **swverify** shall not check the attributes.

This applies to both distribution and installed_software files.

Applies to **swverify**.

check_permissions=*true*

Controls verification of file permissions.

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

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

This only applies to installed_software files.

Applies to **swverify**.

check_requisites=*true*

Controls verification of fileset requisites.

If *check_requisites*=*true*, then **swverify** shall check the *prerequisites*, *corequisites*, and *exrequisites* attributes of files.

If *check_requisites*=*false*, then **swverify** shall not check the attributes.

This applies to both distribution and installed_software.

Applies to **swverify**.

check_scripts=true

Controls the running of the **verify** script.

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

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

Applies to **swverify**.

check_volatile=false

Controls check of volatile files.

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

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

Applies to **swverify**.

compress_files=false

Controls whether uncompressed files are to be compressed in the target distribution, as specified by the value of *compression_type*.

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

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

Applies to **swcopy**.

compression_type=implementation_defined_value

Specifies the compression type used to compress the software files.

The values supported for *compression_type* are implementation defined.

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

Applies to **swcopy**.

defer_configure=false

Controls automatic configuration at install.

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

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

Applies to **swinstall**.

distribution_source_directory = implementation_defined_value
Specifies the default distribution directory.

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

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

distribution_target_directory = implementation_defined_value
Specifies the default distribution target.

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

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

distribution_target_serial = implementation_defined_value
Specifies the default distribution target.

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

Applies to **swpackage**.

enforce_dependencies = true
Controls the enforcement of dependency specifications.

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

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

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

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

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

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

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

enforce_locatable=true

Controls the handling of errors when relocating a non-relocatable fileset.

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

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

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

enforce_scripts=true

Controls the handling of errors generated by scripts.

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

If *enforce_scripts=false*, all script errors shall be treated as warnings, and the utility shall attempt to continue operation.

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

files

Lists the pathnames of file objects to be added or deleted.

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

Applies to **swmodify**.

follow_symlinks=false

Controls the following of symbolic links

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

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

Applies to **swpackage**.

installed_software_catalog=implementation_defined_value

Specifies installed software catalog.

This extended option, along with the installed_software *path* attribute, defines the logical installed_software object upon which the utility is operating. This extended option is resolved relative to the **PATHNAME_CHARACTER_STRING** portion of the *targets* operand. See 4.1.4.2.

This option allows an implementation to define where the catalog information is stored. This option also allows multiple logical installed_software objects to share the **PATHNAME_CHARACTER_STRING** where the software is installed.

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

logfile=implementation_defined_value
Specifies the location of the the logfile for the management role.

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

Applies to all utilities except **swlist**.

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

See 4.1.6.5.

Applies to all utilities except **swlist**.

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

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

Applies to **swpackage**.

media_type=directory
The default media type.

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

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

Applies to **swpackage**.

one_liner=implementation_defined_value
Specifies attributes to list.

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

Applies to **swlist**.

reconfigure=false
Controls reconfiguring of software.

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

Applies to **swconfig**.

recopy=false
Controls copying of filesets.

If *recopy=false*, do not copy a fileset that is already available on the target at the same version.

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

Applies to **swcopy**.

reinstall=false
Controls reinstallation of filesets.

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

If *reinstall=true*, then reinstall the fileset even if this version is already installed.

Applies to **swinstall**.

select_local=true
Controls default selection of target.

If *select_local=true*, and no targets are specified, then the local host shall be selected as the target.

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

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

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

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

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

See the *select_local* option.

Applies to all utilities in this part of ISO/IEC 15068 except **swpackage**.

uncompress_files=false
Controls whether compressed files are to be uncompressed in the target distribution, as specified by the value of the *compression_type* attribute of the file.

If *uncompress_files=false*, all files with a *compression_state* attribute value of **compressed** shall remain compressed, and the status of uncompressed files shall be determined by the value of *compress_files*.

If *uncompress_files=true*, all compressed files shall be uncompressed before being written to the target distribution.

Applies to **swcopy**.

verbose=1

Controls the amount of output sent by the utility to stdout and stderr, but not to log files.

For values that are non-negative integers, an increase in *verbose* shall not decrease the information sent stdout and stderr. All implementations shall support the values `0` (zero) and `1` (one). If *verbose=0*, nothing shall be written to either stdout or stderr. The effect of other values of *verbose* is undefined. See also 4.1.6.3 and 4.1.6.4.

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

4.1.5.3 Extended Options Syntax

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

```
%token          FILENAME_CHARACTER_STRING /* as defined in 2.2.2.37  */
%token          PORTABLE_CHARACTER_STRING /* as defined in 2.2.2.68  */
%token          SHELL_TOKEN_STRING        /* as defined in 2.2.2.80  */
%token          WHITE_SPACE_STRING        /* as defined in 2.2.2.110 */


%start  software_option
%%

software_option          : command_qualifier  keyword '=' value
                         ;

command_qualifier        : /* empty */
                         | command '.'
                         ;

value                    : multi_value
                         | single_value
                         ;

multi_value              : value ws single_value
                         | single_value
                         ;

single_value             : SHELL_TOKEN_STRING
                         ;

command                  : FILENAME_CHARACTER_STRING
                         ;

keyword                  : FILENAME_CHARACTER_STRING
                         ;

ws                       : WHITE_SPACE_STRING
                         ;
```

With respect to this syntax, the following apply:

command

A **keyword** prefixed by the **command** name applies to that utility only.

If no prefix exists, then the **keyword** applies to all the utilities that support it.

keyword

Names the option or operand being defined, e.g., *allow_incompatible* for **swinstall**, and *verbose* for all the utilities.

The allowable characters for a **keyword** are as defined in the filename character set, plus the - (hyphen) character.

value

Assigns the value to the **keyword**.

All extended options are single valued except those that contain lists of **software_specs** or **software_collection_specs**. Quoting of strings and escaping of characters shall be handled as specified in 3.2 and 3.3 of POSIX.2 {3}.

When specified on the command line, multiple option specifications can be included after a single -x option if included in quotes and separated by white space. Multiple -x options can also be used.

For option and defaults files, blank lines and all comment text shall be ignored. Comment text is any sequence of characters beginning with a # (pound) character that is neither escaped nor quoted, and continues through the end of that line.

If the white space between single values contains a **<newline>**, either it shall be escaped or the entire value shall be quoted.

The following are examples of this syntax:

```
loglevel=1
allow_incompatible=false
autoselect_dependencies="as_needed"
software="Foo,r=1.2,a=hp-ux bar,a=Aix_3.2"
targets="hosta:/ hostb hostc:"

software="Foo,r=1.2,a=hp-ux
bar,a=Aix_3.2"

targets="hosta:/
         hostb
         hostc:
         "
```

4.1.5.3.1 Precedence for Option Specification

Multiple option or operand specifications have a precedence that defines which specifications are used.

Only the option specifications with the highest level of precedence are used for each option and operand. The precedence is the following, in increasing order:

(1) System defaults file

(2) User defaults file

(3) Options file

(4) Command line options and operands

If there are multiple instances of options at any particular level, then the following rules apply:

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

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

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

4.1.5.4 Standard Input

4.1.5.5 Input Files

The sections describing the **swpackage** and **swmodify** utilities specifies additional input files specific to those utilities.

4.1.5.6 Access and Concurrency Control

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

If the user of a utility does not have the proper authorization to run a utility, access a software_collection, or access software objects within that utility, the target may generate an event.
(SW_ERROR: SW_ACCESS_DENIED)

If the concurrency control mechanism prevents simultaneous operation on a software collection or software object, the target may generate an event.
(SW_ERROR: SW_CONFLICTING_SESSION_IN_PROGRESS)

If the command will proceed anyway, then the target may generate an event.
(SW_WARNING: SW_CONFLICTING_SESSION_IN_PROGRESS)

If the concurrency control mechanism fails for other reasons, the target may generate an event.
(SW_ERROR: SW_SOC_LOCK_FAILURE)

4.1.5.7 Environment Variables

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

LANG	This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6 of POSIX.2 {3}.
LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.
LC_CTYPE	This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in values for vendor-defined attributes).
LC_MESSAGES	This variable shall determine the language in which messages should be written.
LC_TIME	This variable shall determine the format of dates (*create_date* and *mod_date*) when displayed by `swlist`. It should also be used by all utilities when displaying dates and times in stdout, stderr, and logging.
TZ	This variable shall determine the time zone for use when displaying dates and times.

4.1.6 External Effects

4.1.6.1 Control Script Execution and Environment

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

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

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

(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.

During the execution of each such control script, the following environment variables shall be defined for the environment of the control_script:

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

SW_CONTROL_DIRECTORY
The directory where the executing script is located.

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

SW_CONTROL_TAG
The *tag* of the script being executed.

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

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

This is the value of the *location* attribute.

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

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

SW_ROOT_DIRECTORY
The *installed_software.path* attribute of the installed software object within which the software containing this control_file shall be installed.

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

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

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

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

SW_SESSION_OPTIONS is made available to **request** scripts, the *targets* option shall contain a list of **software_collection_spec**s for all targets specified for the command. When the file pointed to by **SW_SESSION_OPTIONS** is made available to other scripts, the *targets* option shall contain the single **software_collection_spec** for the targets on which the script is being executed.

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

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

See 4.1.4.1.1.

4.1.6.1.1 Control Script Stdout and Stderr

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

4.1.6.1.2 Control Script Return Code

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

Table 4-2 – Script Return Codes

Return Code	Effect of Return Code	Status
0	The script executed successfully. The utility will proceed normally.	SW_NOTE
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
2	The script had a warning. The utility will generate a warning event and continue.	SW_WARNING
3	The script is forcing a deselection of this product or fileset. The utility will generate a note and skip this product or fileset during any further processing.	SW_NOTE
4-31	Reserved.	

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

An implementation can define behaviors for additional script return codes. Any 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_spec**s	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 processsing. 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 processsing. Manager info: target, number of filesets. Target info: **software_spec**s	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_spec**s	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_spec**s	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_spec**s	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_spec**s	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 *allow_downdate* option. Manager info: target, number of filesets. Target info: **software_spec**s	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 *allow_multiple_versions* option. Manager info: target, number of products. Target info: **software_spec**s	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 *allow_multiple_versions* option. Manager info: target, number of products. Target info: **software_spec**s	69
SW_DEPENDENCY_NOT_MET	SW_ERROR SW_WARNING	One or more dependencies cannot be met. Whether error or warning is controlled by *enforce_dependencies* option. Manager info: target, number of filesets. Target info: **software_spec**s, **dependency_spec**s	70
SW_NOT_COMPATIBLE	SW_ERROR SW_WARNING	One or more products are incompatible for this target. Whether error or warning is controlled by *allow_incompatible* option. Manager info: target, number of products. Target info: **software_spec**s	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_spec**s	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_spec**s	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 *enforce_dsa* 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 *enforce_dsa* 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 *enforce_dsa* option. Manager info: target, number of filesets. Target info: `software_spec`s	76
SW_SAME_REVISION_INSTALLED	SW_NOTE	One or more filesets have the same revision and are being reinstalled because *reinstall=true* or *recopy=true*. Manager info: target, number of filesets. Target info: `software_spec`s	77
SW_ALREADY_CONFIGURED	SW_NOTE	One or more filesets are already configured Whether they are reconfigured is controlled by *reconfigure* option. Manager info: target, number of filesets. Target info: `software_spec`s	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_spec`s	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_spec`s	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 *enforce_locatable* option. Manager info: target, number of filesets. Target info: `software_spec`s	86
SW_SAME_REVISION_SKIPPED	SW_NOTE	One or more filesets have the same revision and are being skipped because *reinstall=false* or *recopy=false*. Manager info: target, number of filesets. Target info: `software_spec`s	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 SW_WARNING SW_ERROR	The execution phase ends on the target. Manager info: target. Target info: none.	89
SW_SELECTION_SKIPPED_IN_ANALYSIS	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_spec**s	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_spec**s	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 *tag*	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 *tag*	96
SW_FILESET_WARNING	SW_WARNING	One or more filesets had a warning. Manager info: target, number of filesets. Target info: **software_spec**s	97
SW_FILESET_ERROR	SW_ERROR	One or more filesets had an error. Manager info: target, number of filesets. Target info: **software_spec**s	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_spec**s	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_spec**s	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_spec**s, script *tag*	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_spec`s, script *tags*	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_spec`s	106
SW_REQUEST_ERROR	SW_ERROR	One or more `request` scripts failed. Manager info: `software_spec`s	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 *tag*	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)

— If the operation needs to modify the target, and it is on a read-only media or file system, generate an event.
(SW_ERROR: SW_SOC_IS_READ_ONLY)

— An implementation may generate the following events, if they are applicable to that implementation, when validating a target:
(SW_ERROR: SW_SERVICE_NOT_AVAILABLE)
(SW_WARNING: SW_OTHER_SESSIONS_IN_PROGRESS)
(SW_ERROR: SW_CONNECTION_LIMIT_EXCEEDED)
(SW_NOTE: SW_SOC_IS_REMOTE)
(SW_ERROR: SW_FILESYSTEMS_NOT_MOUNTED)
(SW_ERROR: SW_FILESYSTEMS_MORE_MOUNTED)

4.1.7.1.3 Specifying the Source

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

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

A source shall satisfy the following validation checks:

— If the source does not exist or if it cannot be accessed, generate an event.
(SW_ERROR: SW_SOURCE_ACCESS_ERROR)

— If administrative access is denied by the source role for that source, generate an event.
(SW_ERROR: SW_ACCESS_DENIED)

— Obtain information on what software is available from the source. If the information cannot be retrieved, or if an problem occurs while processing it, generate an event.
(SW_ERROR: SW_SOURCE_ACCESS_ERROR)

— If the source is a serial medium, and the *media.sequence_number* is not 1, generate an event.
(SW_ERROR: SW_SOURCE_NOT_FIRST_MEDIA)

NOTE: Only the first medium has the catalog information on it.

4.1.7.1.4 Software Selections

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

4.1.7.2 Analysis Phase

This subclause summarizes the common operations and events in the analysis phase. The analysis phase occurs before the execution phase, and involves executing checks to determine whether or not the execution should be attempted.

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

To begin the analysis phase, the management role (the host on which the utility was invoked) communicates the selection information to each target in the target list. The target role accesses the source (for `swinstall` or `swcopy`) or target (for other utilities) to get the product catalog information for the software selections. The product catalog information includes control script information in the *control_files* attribute of filesets within each product.

— An implementation may generate any of the following events, if they are applicable to that implementation, when attempting the analysis or execution phase.
(SW_ERROR: SW_AGENT_INITIALIZATION_FAILED)
(SW_ERROR: SW_ILLEGAL_STATE_TRANSITION)
(SW_ERROR: SW_BAD_SESSION_CONTEXT)

— If the target role cannot access the source, generate an event.
(SW_ERROR: SW_SOURCE_ACCESS_ERROR)

— If an implementation supports access control for particular operations for particular software objects, and if access is denied for any software object, generate an event.
(SW_ERROR: SW_ACCESS_DENIED)

— If a fileset has an error for which there is not a more specific event defined, generate the generic event.
(SW_ERROR: SW_FILESET_ERROR)

— If a fileset has a warning for which there is not a more specific event defined, generate the generic event.
(SW_WARNING: SW_FILESET_WARNING)

— An implementation may generate the following events as applicable to the error handling procedures for that implementation:
(SW_NOTE: SW_SKIPPED_PRODUCT_ERROR)
(SW_NOTE: SW_SKIPPED_GLOBAL_ERROR)
(SW_WARNING: SW_FILE_IS_READ_ONLY)

See each utility section for the analysis operations specific to each utility. When the analysis phase ends, generate an event.
(SW_NOTE: SW_ANALYSIS_ENDS)

4.1.7.3 Execution Phase

This subclause summarizes the common operations and events in the execution phase.

When the execution phase begins, generate an event.
(SW_NOTE: SW_EXECUTION_BEGINS)

The execution phase proceeds through the steps defined for each utility. The following events are common to all utilities:

— When the execution phase executes a script, generate an event.
(SW_NOTE: SW_CONTROL_SCRIPT_BEGINS)

— When the execution phase begins the key operation on a fileset (such as loading or removing), generate an event.
(SW_NOTE: SW_FILESET_BEGINS)

— When the execution phase begins the key operation on a file (such as loading or removing), generate an event.
(SW_NOTE: SW_FILE_BEGINS)

— If at any time there is an error rebuilding any catalog files, generate an event.
(SW_ERROR: SW_DATABASE_UPDATE_ERROR)

— If a fileset has an error for which there is not a more specific event defined, generate the generic event.
(SW_ERROR: SW_FILESET_ERROR)

— If a fileset has a warning for which there is not a more specific event defined, generate the generic event.
(SW_WARNING: SW_FILESET_WARNING)

— For **swinstall** and **swcopy** from a distribution that spans multiple media, an implementation may generate the following events to convey needed media change information. An implementation may, but need not, provide such support for other utilities.
(SW_ERROR: SW_WRONG_MEDIA_SET)
(SW_NOTE: SW_NEED_MEDIA_CHANGE)
(SW_NOTE: SW_CURRENT_MEDIA)

— An implementation may generate the following events for software that will not be executed due to analysis results for that software.
(SW_NOTE: SW_SELECTION_SKIPPED_IN_ANALYSIS)
(SW_ERROR: SW_SELECTION_NOT_ANALYZED)

See each utility section for the execution operations specific to each utility. When the execution phase completes, generate an event.
(SW_NOTE: SW_EXECUTION_ENDS)

4.1.7.3.1 Fileset State Transitions

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

configured	Indicates the fileset in an installed_software object has been configured. This state applies to filesets in installed_software objects.
installed	Indicates that the specified fileset has been installed successfully.
corrupt	Indicates that the most recent attempt to handle the fileset was not successful and any recovery actions that were attempted were similarly unsuccessful. Software can transition from this state via the **swinstall**, **swcopy**, or **swremove** utilities. Other implementation-defined methods may also exist for transitioning from this state.

This state applies to filesets in distributions and installed_software objects.

removed This state indicates that the files for the fileset has been removed but the information remains.

The default behavior when removing software with **swremove** is to also remove its information (metadata) from the catalog. An implementation may define a means for removing software, while maintaining the catalog information. The catalog information can be separately removed with the **swmodify** utility.

This state applies to filesets in distributions and installed_software objects.

available Indicates the fileset is present in the distribution and may be operated on (copied, installed, etc.) using the appropriate utilities.

This state applies to filesets in a distribution.

transient Indicates that the fileset is currently being acted upon by one of the utilities that modify software files, thus the state of the software is not well defined.

This state should be replaced by another before the utility completes. The presence of this state in the software_collection when no utility is running indicates that a utility was previously interrupted (power failure, kill, etc.) and was not able to record a final state indication into the software_collection catalog. In such case, the implementation-defined recovery procedures can be used to restore the product to a another state.

This state applies to filesets in distributions and installed_software objects.

4.1.8 Exit Status

The utility shall return one of the following exit codes:

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

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

4.1.9 Consequences of Errors

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

4.1.9.1 Error Conditions

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

4.2 `swask` — Ask for user responses

4.2.1 Synopsis

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

4.2.2 Description

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

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

4.2.3 Options

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

`-c` *catalog*

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

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

`-f` *file*

`-s` *source*

`-t` *targetfile*

`-x` *option=value*

`-X` *options_file*

4.2.4 Operands

The `swask` utility supports the *software_selections* operand described in 4.1.

Specifying values for the *targets* operand does not imply operations on the target role. The values are simply written as a list of targets for the *targets* option in the file made available to the `request` script via the **SW_SESSION_OPTIONS** environment variable. See 4.1.6.1.

4.2.5 External Influences

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

4.2.5.1 Extended Options

The `swask` utility supports the following extended options. The description in 4.1 applies.

autoselect_dependencies =true

distribution_source_directory =implementation_defined_value

distribution_source_serial =implementation_defined_value

logfile=implementation_defined_value

loglevel=1

ask=true

software

verbose=1

4.2.6 External Effects

See 4.1.

4.2.7 Extended Description

See 4.1 for general information. There are two phases in the `swask` utility, as follows:

(1) Selection phase

(2) Execution phase

4.2.7.1 Selection Phase

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

— If the `-s` option is specified and the selection is not available from the source, generate an event. If the `-s` option is not specified and the selection is not available from the target, generate an event.
(SW_ERROR: SW_SELECTION_NOT_FOUND)

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

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

4.2.7.2 Execution Phase

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

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

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

— Set the value of the **SW_CONTROL_DIRECTORY** environment variable to the directory below which the **`request`** script writes the **`response`** file. It may be set to the control directory where the **`response`** file will eventually be held, or it may be set to another, temporary, directory.

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

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

(2) If `-s` was not specified, any **`response`** file from the catalog of the target or targets specified

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

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

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

— If a **request** script returns an error and *enforce_scripts* = *true*, generate an event and invoke the implementation-defined error handling procedures.
(SW_ERROR: SW_REQUEST_ERROR)

— If a **request** script returns an error and *enforce_scripts* = *false*, generate an event.
(SW_WARNING: SW_REQUEST_ERROR)

— If a **request** script returns a warning, generate an event.
(SW_WARNING: SW_REQUEST_WARNING)

4.2.8 Exit Status

See 4.1.

4.2.9 Consequences of Errors

See 4.1.

4.3 swconfig — Configure software

4.3.1 Synopsis

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

4.3.2 Description

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

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

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

4.3.3 Options

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

`-c` *catalog*

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

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

`-f` *file*

`-p`

`-t` *targetfile*

`-u` Undo configuration

This option tells the `swconfig` utility to unconfigure the software, instead of configuring it.

`-x` *option=value*

`-X` *options_file*

4.3.4 Operands

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

4.3.5 External Influences

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

4.3.5.1 Extended Options

The `swconfig` utility supports the following extended options. The description in 4.1 applies.

allow_incompatible=false

allow_multiple_versions=false

ask=false

autoselect_dependencies =true

autoselect_dependents =false

enforce_dependencies =true

installed_software_catalog=implementation_defined_value

logfile=implementation_defined_value

loglevel=1

reconfigure=false

select_local = true

software

targets

verbose = 1

4.3.6 External Effects

See 4.1.

4.3.7 Extended Description

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

(1) Selection phase

(2) Analysis phase

(3) Execution phase

4.3.7.1 Selection Phase

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

— If the selection is not found, generate an event.
(SW_WARNING: SW_SELECTION_NOT_FOUND)

— If the selection is not found at that product location but it does exist in another location, generate an event.
(SW_WARNING: SW_SELECTION_NOT_FOUND_RELATED)

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

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

4.3.7.2 Analysis Phase

The following checks are made:

— If configuring, check for compatibility of selections and, if the software is not compatible, generate an event as follows. See 4.1.4.1.2.

— If *allow_incompatible=true*, generate an event.
(SW_WARNING: SW_NOT_COMPATIBLE)

— If *allow_incompatible=false*, generate an event.
(SW_ERROR: SW_NOT_COMPATIBLE)

— If configuring and the state is **`corrupt`** or **`transient`**, generate an event.
(SW_ERROR: SW_SELECTION_IS_CORRUPT)

— If the state is already configured when configuring, or not configured when unconfiguring, generate an event. This event is independent of whether the software will be reconfigured or not, which in turn is controlled by the *reconfigure* option.
(SW_NOTE: SW_ALREADY_CONFIGURED)

— If configuring this software will create a new configured version of a fileset (see 2.2.2.109,) then generate an event as follows.

  — If *allow_multiple_versions*=*true*, generate an event.
  (SW_WARNING: SW_NEW_MULTIPLE_VERSION)

  — If *allow_multiple_versions*=*false*, generate an event.
  (SW_ERROR: SW_NEW_MULTIPLE_VERSION)

— If the dependencies are not in the **configured** state and have not been autoselected to be configured, generate an event as follows.

  — If *enforce_dependencies* =*false*, generate an event.
  (SW_WARNING: SW_DEPENDENCY_NOT_MET)

  — If *enforce_dependencies* =*true*, generate an event.
  (SW_ERROR: SW_DEPENDENCY_NOT_MET)

4.3.7.3 Execution Phase

The sequential relationship between the configure operations is shown in the following list. Products are ordered by prerequisite dependencies if any. Fileset operations are also ordered by any prerequisites.

(1) Configure each product.

  (a) Run **configure** script for the product.

  (b) Configure each fileset in the product.

    [1] Run the **configure** script for the fileset.

    [2] Update the result of the script (control_file). Update the state of the fileset to **configured** in the database for the installed_software object.

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

4.3.7.3.1 Unconfigure Operations

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

(1) Unconfigure each product.

  (a) Unconfigure each fileset in the product.

    [1] Run the **unconfigure** script for the fileset.

    [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.

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

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

4.3.7.3.2 Executing `configure` Scripts

In this operation, **swconfig** executes vendor-supplied **configure** or **unconfigure** scripts.

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

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

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

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

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

4.3.8 Exit Status

See 4.1.

4.3.9 Consequences of Errors

See 4.1.

4.4 swcopy — Copy distribution

4.4.1 Synopsis

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

4.4.2 Description

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

4.4.3 Options

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

`-f` *file*

`-p`

`-s` *source*

`-t` *targetfile*

`-x` *option=value*

`-X` *options_file*

4.4.4 Operands

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

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

4.4.5 External Influences

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

4.4.5.1 Extended Options

The `swcopy` utility supports the following extended options. The description in 4.1 applies.

autoselect_dependencies =as_needed

compress_files=false

compression_type=implementation_defined_value

distribution_source_directory =implementation_defined_value

distribution_target_directory =implementation_defined_value

enforce_dependencies=true

enforce_dsa=true

logfile=implementation_defined_value

loglevel=1

recopy=false

select_local=true

software

targets

uncompress_files=false

verbose=1

4.4.6 External Effects

See 4.1.

4.4.7 Extended Description

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

(1) Selection phase

(2) Analysis phase

(3) Execution phase

4.4.7.1 Selection Phase

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

The **swcopy** utility supports corequisite and prerequisite dependencies for autoselection, but treats them equally (no ordering considerations like **swinstall**). The **swcopy** utility ignores exrequisites.

For **swcopy** 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 shall not be added to the list.

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

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

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

4.4.7.2 Analysis Phase

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

The target role checks the following requirements:

— If the target distribution does not exist on a host, create the path to the target with default attributes and generate an event.
(SW_NOTE: SW_SOC_CREATED)

— Check that selected filesets are not the same version as already available. If they are the same version, generate an event as follows.

  — If *recopy*=*false*, note that these filesets will be skipped by generating an event.
  (SW_NOTE: SW_SAME_REVISION_SKIPPED)

  — If *recopy*=*true*, note that they will be recopied by generating an event.
  (SW_NOTE: SW_SAME_REVISION_INSTALLED)

— Verify that the needed dependencies of the filesets are met. If the needed dependencies are not met, generate an event as follows.

  — If *enforce_dependencies*=*true*, generate an event.
  (SW_ERROR: SW_DEPENDENCY_NOT_MET)

  — If *enforce_dependencies*=*false*, generate an event.
  (SW_WARNING: SW_DEPENDENCY_NOT_MET)

— Check that there is enough free disk space on the target file system to copy the selected products.

  — If there is not enough disk space and *enforce_dsa*=*true*, generate an event.
  (SW_ERROR: SW_DSA_OVER_LIMIT)

  — If there is not enough disk space and *enforce_dsa*=*false*, generate an event.
  (SW_WARNING: SW_DSA_OVER_LIMIT)

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

4.4.7.3 Execution Phase

As the result of a **swcopy**, products and bundles, if specified, are copied to the target, which is a distribution software object.

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

(1) Copy each product.

(a) Create the distribution catalog information for the product and its contained subproducts.

(b) Copy each fileset in the product.

[1] Create the distribution catalog information for the fileset, setting the state to **`transient`**.

[2] Load the files for the fileset.

[3] Update the state of the fileset to **`available`**.

(2) Copy each bundle.

(a) Create the distribution catalog information for the bundle.

4.4.7.3.1 File Loading

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

If a file load fails for any other reason such as a lost connection to the remote source or tape eject, then the fileset load fails. The following are the errors that can occur during the file loading step:

— If an error or warning occurs while loading a file onto a target, an event is generated and, for errors, the implementation-defined error handling procedures invoked.

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

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

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

— If the error was a source access problem, the user may attempt to correct the problem and start over with the fileset that had the failure.
(SW_ERROR: SW_SOURCE_ACCESS_ERROR)

4.4.7.3.2 Compression

If *compress_files=true* is specified, then the files shall be compressed as follows in copying to the target distribution:

— **`INDEX`** and **`INFO`** files shall not be compressed.

— All files that have the *compression_state* attribute undefined or its value set to **`uncompressed`** shall be compressed. If the file cannot be compressed, generate an event.
(SW_ERROR: SW_COMPRESSION_FAILURE)

— All files that have the value of the *compression_state* attribute set to **`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.

— If a source file is already compressed, and the value of its *compression_type* attribute is different than the value of the *compression_type* extended option, the behavior is undefined. Unless the implementation can successfully uncompress the file and then compress it with the correct type, generate an event.
(SW_ERROR: SW_COMPRESSION_FAILURE)

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

— All files with a *compression_state* attribute value of `compressed` shall be uncompressed as part of the copy. If the file cannot be uncompressed, generate an event.
(SW_ERROR: SW_COMPRESSION_FAILURE)

— All other files shall be copied as is.

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

4.4.7.3.3 Copying Into Existing Products

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

4.4.8 Exit Status

See 4.1.

4.4.9 Consequences of Errors

See 4.1.

4.5 `swinstall` — Install software

4.5.1 Synopsis

`swinstall` **[**`-p`**]** **[**`-r`**]** **[**`-c` *catalog***]** **[**`-f` *file***]** **[**`-s` *source***]** **[**`-t` *targetfile***]**
[`-x` *option*=*value***]** **[**`-X` *options_files***]** **[***software_selections***]** **[**@ *targets***]**

4.5.2 Description

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

4.5.3 Options

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

`-c` *catalog*

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

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

`-f` *file*

`-p`

`-r`

`-s` *source*

`-t` *targetfile*

`-x` *option=value*

`-X` *options_file*

4.5.4 Operands

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

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

4.5.5 External Influences

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

4.5.5.1 Extended Options

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

allow_downdate=false

allow_incompatible=false

ask=false

autoreboot=false

autorecover=false

autoselect_dependencies=as_needed

defer_configure=false

distribution_source_directory=implementation_defined_value

enforce_dependencies=true

enforce_locatable=true

enforce_scripts=true

enforce_dsa=true

installed_software_catalog=implementation_defined_value

logfile=implementation_defined_value

loglevel=1

reinstall=false

select_local=true

software

targets

verbose=1

4.5.6 External Effects

See 4.1.

4.5.7 Extended Description

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

(1) Selection phase

(2) Analysis phase

(3) Execution phase

4.5.7.1 Selection Phase

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

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

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

If a fileset has an exrequisite on another software object, and that other software object is part of the specified software selection, either explicitly or as part of another selection, then the fileset is excluded. If two filesets have exrequisites on each other, then the behavior is implementation defined.

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

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

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

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

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

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

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

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

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

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

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

4.5.7.2 Analysis Phase

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

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

The target role checks the following requirements:

— If the target installed_software object does not exist on a host, create the path to the target with default attributes, and generate an event.
(SW_NOTE: SW_SOC_CREATED)

— Check that selected filesets are not the same version as already installed. If any of the selected filesets have already been installed with the same version, generate an event as follows.

  — If *reinstall*=*false*, note that they will be skipped by generating an event.
  (SW_NOTE: SW_SAME_REVISION_SKIPPED)

  — If *reinstall*=*true*, note that they will be reinstalled by generating an event.
  (SW_NOTE: SW_SAME_REVISION_INSTALLED)

— Check that selected filesets are not lower versions of the fileset already installed on the host. If the any of the selected filesets are already installed with a higher version, generate an event as follows.

  — If *allow_downdate*=*false*, generate an event.
  (SW_ERROR: SW_HIGHER_REVISION_INSTALLED)

  — If *allow_downdate*=*true*, generate an event.
  (SW_WARNING: SW_HIGHER_REVISION_INSTALLED)

— Execute vendor-supplied **checkinstall** scripts to perform product-specific checks of the target.

  — If a **checkinstall** script returns an error, and *enforce_scripts*=*true*, generate an event and invoke the implementation-defined error handling procedure.
  (SW_ERROR: SW_CHECK_SCRIPT_ERROR)

  — If a **checkinstall** script returns an error and *enforce_scripts*=*false*, generate an event.
  (SW_WARNING: SW_CHECK_SCRIPT_ERROR)

  — If a **checkinstall** script returns a warning, generate an event.
  (SW_WARNING: SW_CHECK_SCRIPT_WARNING)

  - — If a **checkinstall** script has a return code of 3, generate an event and unselect the fileset (or all filesets in the product for a product level script).
    (SW_NOTE: SW_CHECK_SCRIPT_EXCLUDE)

- — Verify that the needed dependencies of the filesets are met. If any of the dependencies are not met, generate an event as follows.

  - — If *enforce_dependencies*=*true*, generate an event.
    (SW_ERROR: SW_DEPENDENCY_NOT_MET)

  - — If *enforce_dependencies*=*false*, generate an event.
    (SW_WARNING: SW_DEPENDENCY_NOT_MET)

- — Check that there is enough free disk space on the target file system to install the selected products.

  - — If there is not enough disk space and *enforce_dsa*=*true*, generate an event.
    (SW_ERROR: SW_DSA_OVER_LIMIT)

  - — If there is not enough disk space and *enforce_dsa*=*false*, generate an event.
    (SW_WARNING: SW_DSA_OVER_LIMIT)

- — An implementation may generate an event, as follows, if disk space analysis encountered any problems that prevented the analysis.

  - — If *enforce_dsa*=*true*, generate an event.
    (SW_ERROR: SW_DSA_FAILED_TO_RUN)

  - — If *enforce_dsa*=*false*, generate an event.
    (SW_WARNING: SW_DSA_FAILED_TO_RUN)

How disk space analysis is implemented is undefined. However an implementation shall account at least for the sizes of the files and control_files being installed, the additional sizes from the vendor-supplied **space** file described in 5.2, and any additional space required by the implementation-defined recovery process for saving files when *autorecover*=*true*.

4.5.7.3 Execution Phase

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

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

(1) Install each product.

  (a) Create the installed_software catalog information for the product and its contained subproducts.

(b) Run the **preinstall** script for the product.

(c) Install each fileset in the product.

[1] Create the installed_software catalog information for the fileset, setting the state to **transient**. Also update the state of any existing fileset that is being updated or downdated to **transient**.

[2] Run the **preinstall** script for the fileset.

[3] Load the files for the fileset.

[4] Run the **postinstall** script for the fileset.

[5] Update the results of the scripts. Update the state of the fileset to **installed**. Also set the state of any existing fileset that is being updated or downdated to **removed** or remove the catalog information for that fileset.

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

(2) Install each bundle.

(a) Create the installed_software catalog information for the bundle.

(3) Configure each product in accordance with 4.3.7.3.2.

Configuration shall be done at this point by the **swinstall** utility only if *defer_configure=false*, the target directory is /, and no filesets with the *is_reboot* attribute equal to true have been installed.

(a) Run the **configure** script for the product.

(b) Configure each fileset in the product.

[1] Run the **configure** script for the fileset.

[2] Update the result of the script. Update the state of the fileset to **configured** in the catalog for the installed_software object.

Configuration shall not be executed by **swinstall** if the software creates a multiple version, the target directory is not /, or if the software is incompatible and *allow_incompatible=false* (see 4.1.4.1.2). In these cases, **swconfig** may be used.

If events with a status of SW_ERROR are detected during the execution phase, the **swinstall** utility generates the appropriate event, any log entries, and invokes the implementation-defined error handling procedures. For each fileset that failed, the installed_software catalog is updated to the state **corrupt**.

4.5.7.3.1 File Location

If an alternate root directory was specified (a value for *installed_software.path* other than /), then the alternate root directory is used as a prefix to the *file.path* attribute to determine the file location in the file system. See 4.1.4.2.

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

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

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

4.5.7.3.2 Preinstall Scripts

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

— If a `preinstall` script returns an error and *enforce_scripts* = *true*, generate an event and invoke the implementation-defined error handling procedures.
(SW_ERROR: SW_PRE_SCRIPT_ERROR)

— If a `preinstall` script returns an error and *enforce_scripts* = *false*, generate an event.
(SW_WARNING: SW_PRE_SCRIPT_ERROR)

— If a `preinstall` script returns a warning, generate an event.
(SW_WARNING: SW_PRE_SCRIPT_WARNING)

Control scripts shall adhere to the specifications in 4.1.6.1.

4.5.7.3.3 File Loading

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

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

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

— If there are too many levels of symbolic links, then the file is skipped and an event is generated.
(SW_WARNING: SW_FILE_WARNING)

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

The file owner and group names are set to the values specified for the *file.owner* 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 process. See 5.2.13.3.

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

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

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

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

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

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

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

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

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

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

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

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

— If the source becomes inaccessible for any reason during the process of loading files, generate an event.
(SW_ERROR: SW_SOURCE_ACCESS_ERROR)

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

4.5.7.3.4 Compression

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

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

4.5.7.3.5 Postinstall Scripts

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

— If a **postinstall** script returns an error and *enforce_scripts*=*true*, generate an event and invoke the implementation-defined error handling procedures.
(SW_ERROR: SW_POST_SCRIPT_ERROR)

— If a **postinstall** script returns an error and *enforce_scripts*=*false*, generate an event.
(SW_WARNING: SW_POST_SCRIPT_ERROR)

— If a **postinstall** script returns a warning, generate an event.
(SW_WARNING: SW_POST_SCRIPT_WARNING)

4.5.7.3.6 Kernel Scripts

Special customization and install steps are executed when processing kernel filesets. Kernel filesets are those for which the value of the *is_kernel* attribute is **true**, causing **swinstall** to modify the fileset load order and to invoke the *postkernel* script. Apart from this, **swinstall** has no special functionality for installing kernels.

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

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

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

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

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

(1) Install the kernel filesets for each product.

  (a) Create the installed_software catalog information for the product.

  (b) Run the **preinstall** script for the product.

  (c) Install each kernel fileset in the product.

    [1] Create the installed_software catalog information for the fileset, setting the state to **transient**.

    [2] Run the **preinstall** script for the kernel fileset.

    [3] Load the files for the kernel fileset.

    [4] Run the **postinstall** script for the kernel fileset.

    [5] Update the results of the scripts. Update the state of the fileset to **installed**.

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

(2) Perform steps after installing kernel filesets by calling the zero or more scripts defined by the *product.postkernel* attributes of each product with a kernel fileset and the implementation-defined default *postkernel* script if a product does not define a *product.postkernel* attribute.

(3) Install the rest of the filesets for each product as described in 4.5.7.3, omitting the kernel filesets already installed.

(4) After all filesets have been installed, the implementation-defined reboot procedure is executed on the target host if a fileset with the is_reboot attribute set to true has been installed and the **SW_ROOT_DIRECTORY** is **/**, and *autoreboot*=*true*. If rebooting, the software is not configured. The products will be configured after the reboot in an implementation-defined manner using the **swconfig** utility.

(5) If not rebooting, then configure each product as described in 4.5.7.3, (including both kernel and non-kernel filesets).

4.5.7.3.7 Rebooting the System

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

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

4.5.7.3.8 Recovery

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

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

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

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

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

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

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

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

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

(3) Install each fileset in the product.

(a) Create the installed_software catalog information for the fileset, setting the state of it and the fileset being updated to **transient**.

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

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

(c) Load the files for the fileset.

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

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

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

(e) Update the results of the scripts. Update the state of the fileset to `installed`.

(4) Run the product `postinstall` script.

If the product `postinstall` script fails, run the product `unpostinstall` script, and perform each of the previously described recovery steps for each fileset.

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

4.5.8 Exit Status

See 4.1.

4.5.9 Consequences of Errors

See 4.1.

4.6 swlist — List software catalog

4.6.1 Synopsis

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

4.6.2 Description

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

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

4.6.3 Options

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

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

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

create_date

If this value is specified, **`swlist`** shall return a sequence of characters representing the date associated with the *create_time* attribute.

The format of this sequence of characters in the POSIX locale shall be equivalent to the default **`date`** format described in 4.15.6.1 of POSIX.2 {3}.

```
date "+%a %b %e %H:%M:%S %Z %Y"
```

The format for other locales is undefined.

mod_date

If this value is specified, **`swlist`** shall return a sequence of characters representing the date associated with the *mod_time* attribute.

The format of this sequence of characters in the POSIX locale shall be equivalent to the default **`date`** format described in 4.15.6.1 of POSIX.2 {3}:

```
date "+%a %b %e %H:%M:%S %Z %Y"
```

The format for other locales is undefined.

software_spec

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

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

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

`-c` *catalog*

Provides a means to list the full catalog structure.

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

The `-c` option specifies a directory below which the catalog information for the specified objects and attributes are stored. The exported catalog structure is used both for distributions and installed_software catalog information. See 5.1.1.

`-d`

`-f` *file*

`-l` *level*

Specifies level at which to list the objects below the specified software.

Level may have values from the enumerated list **`bundle`**, **`product`**, **`subproduct`**, **`fileset`**, **`control_file`**, **`file`**. If the `-l` *level* option is not included, then only the object at the level directly below 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

If no level is specified for bundle and subproduct specifications, all the available or currently installed product and fileset objects, resolved recursively, are listed.

Multiple `-l` *level* options may be used to explicitly control what objects are included.

`-r`

`-t` *targetfile*

`-v` List all the attribute value pairs of the objects specified.

The `-v` option specifies that the format of the output shall be in the **`INDEX`** file format, as defined in 5.2. Which attributes and objects are included is controlled by other options and operands. If the `-a` option is defined, then only those attributes are listed, otherwise all attributes are listed. If there is no `-v` option, then the listing format is undefined (see *one_liner* extended option).

`-x` *option=value*

`-X` *options_file*

4.6.4 Operands

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

4.6.5 External Influences

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

4.6.5.1 Extended Options

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

distribution_target_directory=implementation_defined_value

installed_software_catalog=implementation_defined_value

one_liner=implementation_defined_value

select_local=true

software

targets

4.6.6 External Effects

See 4.1 for general information.

4.6.7 Extended Description

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

(1) Selection phase

(2) Execution phase

4.6.7.1 Selection Phase

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

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

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

4.6.7.2 Execution Phase

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

4.6.8 Exit Status

See 4.1.

4.6.9 Consequences of Errors

See 4.1.

4.7 `swmodify` — Modify software catalog

4.7.1 Synopsis

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

4.7.2 Description

The **`swmodify`** utility provides an object and attribute update, create, and delete interface to the distribution and installed_software catalog information independent of the other utilities. When combined with **`swlist`**, there is a complete read/write interface to the installed_software and distribution catalog information.

4.7.3 Options

The **`swmodify`** utility supports the following options. Where there is no description, the description in 4.1 applies.

`-a` *attribute=value*

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

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

`-c` *catalog*

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

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

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

`-d`

`-f` *file*

`-p`

`-r`

`-t` *targetfile*

`-u` Deletes the objects or attributes specified.

`-x` *option=value*

`-X` *option_file*

4.7.4 Operands

The **swmodify** utility supports the *software_selections* and *targets* operands described in 4.1.

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

4.7.5 External Influences

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

4.7.5.1 Extended Options

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

distribution_target_directory=implementation_defined_value

installed_software_catalog=implementation_defined_value

files

logfile=implementation_defined_value

loglevel=1

select_local=true

software

targets

verbose=1

4.7.5.2 Standard Input

4.7.5.3 Input Files

The source input files may be in one of the following:

— software definition file
  Described in 5.2.

— exported catalog structure
Described in 5.2.

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

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

4.7.6 External Effects

See 4.1.

4.7.7 Extended Description

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

(1) Selection Phase

(2) Analysis Phase

(3) Execution Phase

4.7.7.1 Selection Phase

4.7.7.1.1 Specifying the Source

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

— If the file parsing discovers syntax errors or missing but required attributes, generate an event.
(SW_ERROR: SW_SOURCE_ACCESS_ERROR)

4.7.7.1.2 Software Selections

If there are no software selections specified, then all software from the catalog is processed. 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.

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

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

4.7.7.2 Analysis Phase

See 4.1.

4.7.7.3 Execution Phase

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

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

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

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

4.7.8 Exit Status

See 4.1.

4.7.9 Consequences of Errors

See 4.1.

4.8 swpackage — Package distribution

4.8.1 Synopsis

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

4.8.2 Description

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

4.8.3 Options

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

`-f` *file*

`-p`

`-s` *psf*
This option specifies the pathname of the PSF, which describes the details of the packages that `swpackage` operates on.

`-x` *option=value*

`-X` *options_file*

4.8.4 Operands

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

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

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

4.8.5 External Influences

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

4.8.5.1 Extended Options

The `swpackage` utility supports the following extended options. The description in 4.1 applies.

distribution_target_directory=implementation_defined_value

distribution_target_serial=implementation_defined_value

enforce_dsa=true

follow_symlinks=false

logfile=implementation_defined_value

loglevel=1

media_capacity=0

media_type=directory

psf_source_file=psf

software

verbose=1

4.8.5.2 Product Specification File

See Section 5.

4.8.6 External Effects

See 4.1.

4.8.7 Extended Description

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

(1) Selection Phase

(2) Analysis Phase

(3) Execution Phase

4.8.7.1 Selection Phase

4.8.7.1.1 Specifying Targets

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

4.8.7.1.2 Specifying the Source

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

The selection phase reads (and parses) the PSF as follows to obtain the information from the source PSF:

— The product, subproduct, and fileset structure

— The files contained in each fileset

— The attributes associated with these objects

If the file parsing discovers syntax errors, or missing but required attributes, generate an event.
(SW_ERROR: SW_SOURCE_ACCESS_ERROR).

4.8.7.1.3 Software Selections

If there are no software selections specified, then all software from the PSF is processed. 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.

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

4.8.7.2 Analysis Phase

The analysis phase consists of the following steps:

- Check the dependency specifications for irregularities (such as circular prerequisites or missing dependencies). If there are irregularties, generate an event.
(SW_WARNING: SW_DEPENDENCY_NOT_MET)
- Before a new storage directory is created, check to see if this product version has the same identifying attributes as an existing product version, namely the same tag, revision, architecture, and vendor_tag. If all the identifying attributes match, then the user is repackaging (modifying) an existing version. If a fileset within that product is being repackaged, generate an event.
(SW_NOTE: SW_SAME_REVISION_INSTALLED)
- Check the existence and attributes of the control_files and files that the PSF defines. If any are missing, generate an event.
(SW_ERROR: SW_FILE_NOT_FOUND)
- Check that there is enough free disk space on the target file system to package the selected products.
    - If there is not enough space and *enforce_dsa*=*true*, generate an event.
    (SW_ERROR: SW_DSA_OVER_LIMIT)
    - If there is not enough space and *enforce_dsa*=*false*, generate an event.
    (SW_WARNING: SW_DSA_OVER_LIMIT)

4.8.7.3 Execution Phase

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

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

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

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

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

Certain errors can occur as follows when packaging the files:

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

4.8.8 Exit Status

The `swpackage` utility returns as follows:

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

4.8.9 Consequences of Errors

See 4.1.

4.9 `swremove` — Remove software

4.9.1 Synopsis

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

4.9.2 Description

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

The `swremove` utility removes software installed at the local host or at the targets specified on the command line. It also removes software from local or remote distributions.

4.9.3 Options

The `swremove` utility supports the following options. Where there is no description, the description in 4.1 applies.

`-d`

`-f` *file*

`-p`

`-r`

`-t` *targetfile*

`-x` *option=value*

`-X` *options_file*

4.9.4 Operands

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

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

4.9.5 External Influences

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

4.9.5.1 Extended Options

The `swremove` utility supports the following extended options. The description in 4.1 applies.

autoselect_dependents=false

distribution_target_directory=implementation_defined_value

enforce_dependencies=true

enforce_scripts=true

installed_software_catalog=implementation_defined_value

logfile=implementation_defined_value

loglevel=1

select_local=true

software

targets

verbose=1

4.9.6 External Effects

See 4.1.

4.9.7 Extended Description

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

(1) Selection phase

(2) Analysis phase

(3) Execution phase

4.9.7.1 Selection Phase

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

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

— If the selection is not found, generate an event.
(SW_WARNING: SW_SELECTION_NOT_FOUND)

— If the selection is not found at that product directory, generate an event.
(SW_WARNING: SW_SELECTION_NOT_FOUND_RELATED)

— If a single version of the software is not uniquely identified from the product and product attributes specified, generate an event.
(SW_ERROR: SW_SELECTION_NOT_FOUND_AMBIG)

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

4.9.7.2 Analysis Phase

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

The target role makes the following checks:

— When removing installed software, execute vendor-supplied **checkremove** scripts to perform product-specific checks of the target.

  — If a **checkremove** script returns an error and *enforce_scripts* = *true*, an event is generated and the implementation-defined error handling procedure is invoked.
  (SW_ERROR: SW_CHECK_SCRIPT_ERROR)

  — If a **checkremove** script returns an error and *enforce_scripts* = *false*, generate an event.
  (SW_WARNING: SW_CHECK_SCRIPT_ERROR)

- — If a **checkremove** script returns a warning, generate an event.
  (SW_WARNING: SW_CHECK_SCRIPT_WARNING)

- — If a **checkremove** script has a return code of 3, generate an event and unselect the fileset (or all filesets in the product for a product level script).
  (SW_NOTE: SW_CHECK_SCRIPT_EXCLUDE)

— Verify that the dependencies are met. The **swremove** utility does not remove a fileset if it is required by other filesets that have not been selected for removal or cannot be removed. If a non-selected fileset depends on a selected fileset, generate an event as follows.

- — If *enforce_dependencies* = *true*, invoke the implementation-defined error handling procedure and generate an event.
  (SW_ERROR: SW_DEPENDENCY_NOT_MET)

- — If *enforce_dependencies* = *false*, generate an event.
  (SW_WARNING: SW_DEPENDENCY_NOT_MET)

If a software object has been specified for removal and there is a bundle referring to that object that has not also been specified for removal, the behavior is implementation defined. Likewise, if a fileset or subproduct object has been specified for removal, and there is a subproduct referring to that object that has not also been specified for removal, the behavior is implementation defined.

4.9.7.3 Execution Phase

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

(1) Unconfigure each product.

If the fileset has been configured more than once, the **unconfigure** script shall unconfigure each instance.

(a) Unconfigure each fileset in the product.

[1] Run the **unconfigure** script for the fileset.

[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.

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

(2) Remove each product.

(a) Run the **preremove** script for the product.

(b) Remove each fileset in the product.

[1] Update the state of the fileset to **transient** in the catalog for the installed_software object.

[2] Run the **preremove** script for the fileset.

[3] Remove the files for the fileset.

[4] Run the **postremove** script for the fileset.

[5] Update the results of the scripts. Update the state of the fileset to **removed** in the catalog for the installed_software object or remove the catalog information for the fileset.

(c) Run the **postremove** script for the product.

(d) If the catalog information has been removed for all filesets in the product, an implementation can also remove the catalog information for the product and its contained subproducts.

(3) Remove each bundle.

(a) Remove the installed_software catalog information for the bundle.

(4) If the catalog information has been removed for all products and bundles in the installed_software object, an implementation can also remove the catalog information for the installed_software object.

For each fileset that failed to be removed, the installed_software catalog information is updated to the state **corrupt**.

4.9.7.3.1 Executing preremove Scripts

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

— If a **preremove** script returns an error and *enforce_scripts*=*true*, generate an event and invoke the implementation-defined error handling procedures.
(SW_ERROR: SW_PRE_SCRIPT_ERROR)

— If a **preremove** script returns an error and *enforce_scripts*=*false*, generate an event.
(SW_WARNING: SW_PRE_SCRIPT_ERROR)

— If a **preremove** script returns an warning, generate an event.
(SW_WARNING: SW_PRE_SCRIPT_WARNING)

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

4.9.7.3.2 File Removing

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

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

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

— If a file was recorded in the catalog as a symbolic link to another file, and it is still a symbolic link on the file system, remove the symbolic link, but do not remove the target file.

— If a file was recorded in the catalog as a file, but exists as a symbolic link on the file system, remove the symbolic link, but do not remove the target file.

— If the pathname to the file includes a symbolic link, this path is followed and the correct file is removed.

All files that are targets of symbolic links are removed when the fileset to which they belong is removed.

4.9.7.3.3 Executing `postremove` Scripts

In this step of the execution phase, **`swremove`** executes software **`postremove`** scripts.

— If a **`postremove`** script returns an error and *enforce_scripts*=*true*, generate an event and invoke the implementation-defined error handling procedures.
(SW_ERROR: SW_POST_SCRIPT_ERROR)

— If a **`postremove`** script returns an error and *enforce_scripts*=*false*, generate an event.
(SW_WARNING: SW_POST_SCRIPT_ERROR)

— If a **`postremove`** script returns an warning, generate an event.
(SW_WARNING: SW_POST_SCRIPT_WARNING)

4.9.7.3.4 Kernel Reconfiguration

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

4.9.7.3.5 Removing From a Distribution

The list of operations is simpler for removing filesets from a distribution than for installed_software.

(1) Remove each product.

(a) Remove each fileset in the product.

[1] Update the state of the fileset to **`transient`** in the catalog for the distribution.

[2] Remove the files for the fileset.

[3] Update the state of the fileset to **`removed`** in the catalog for the distribution or remove the catalog information for the fileset.

(b) If the catalog information has been removed for all filesets in the 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`**.

(2) Remove each bundle.

(a) Remove the distribution catalog information for the bundle.

(3) If the catalog information has been removed for all products and bundles in the distribution object, an implementation can also remove the catalog information for the distribution object.

4.9.8 Exit Status

See 4.1.

4.9.9 Consequences of Errors

See 4.1.

4.10 `swverify` — Verify software

4.10.1 Synopsis

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

4.10.2 Description

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

4.10.3 Options

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

-d

-f *file*

-F Correct problems as well as report them.

If *check_permissions*=*true*, correct the corresponding problems reported.

If *check_scripts*=*true*, correct the corresponding problems reported.

The -F option only applies to installed software.

`-r`

`-t` *targetfile*

`-x` *option=value*

`-X` *options_file*

4.10.4 Operands

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

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

4.10.5 External Influences

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

4.10.5.1 Extended Options

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

allow_incompatible=false

autoselect_dependencies=true

check_contents=true

check_permissions=true

check_requisites=true

check_scripts=true

check_volatile=false

distribution_target_directory=implementation_defined_value

enforce_dependencies=true

enforce_locatable=true

installed_software_catalog=implementation_defined_value

logfile=implementation_defined_value

loglevel=1

select_local=true

software

targets

verbose=1

4.10.6 External Effects

See 4.1.

4.10.7 Extended Description

See 4.1. for general information. The key phases in the **swverify** utility are the following:

(1) Selection phase

(2) Analysis phase

(3) Execution phase

4.10.7.1 Selection Phase

Like **swremove**, software selections apply to the software installed (or available in the case of a distribution).

Each specified selection is added to the selection list after it passes the following checks:

— If the selection is not found, generate an event.
(SW_WARNING: SW_SELECTION_NOT_FOUND)

— If the selection is not found at that product location, but that product exists at another location, generate an event.
(SW_WARNING: SW_SELECTION_NOT_FOUND_RELATED)

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

4.10.7.2 Analysis Phase

This subclause details the analysis phase for **swverify**. No aspect of the target host environment is modified unless the -F option is specified. The target role accesses its software_collection catalog to get the information for the selected software.

The target role makes the following checks:

— For each product that is incompatible with the uname attributes of the target host, generate an event as follows. See 4.1.4.1.2.

  — If *allow_incompatible=false*, generate an event.
  (SW_ERROR: SW_NOT_COMPATIBLE)

  — If *allow_incompatible=true*, generate an event.
  (SW_WARNING: SW_NOT_COMPATIBLE)

  Applies to installed software.

— For each fileset whose state is other than **installed**, **configured**, **available**, or **removed**, generate an event.
(SW_WARNING: SW_SELECTION_IS_CORRUPT)

  Applies to distributions and installed software.

— If a dependency cannot be met, generate an event as follows.

  — If *enforce_dependencies*=*true*, generate an event. (SW_ERROR: SW_DEPENDENCY_NOT_MET)

  — If *enforce_dependencies*=*false*, generate an event. (SW_WARNING: SW_DEPENDENCY_NOT_MET)

  Applies to distributions and installed software.

— Executes vendor-supplied **`verify`** scripts.

  — If a **`verify`** script returns an error, generate an event. (SW_ERROR: SW_CHECK_SCRIPT_ERROR)

  — If a **`verify`** script returns an warning, generate an event. (SW_WARNING: SW_CHECK_SCRIPT_WARNING)

  Applies to installed software.

— The following file level checks are made:

  — Check for missing files and directories. For installed software, if *check_volatile*=*false*, then this check shall not be made for files with *file.is_volatile* equal `true`.

    Applies to distributions and installed software.

  — Check for files that have been modified.

    — For distributions, check *size*, *cksum*, and *mtime*.

    — For installed software, check *mode*, *owner*, *group*, *size*, *cksum*, *mtime*, *revision*, *major*, and *minor*, if defined for that file object.

      If *check_volatile*=*false*, then these checks shall not be made for files with *file.is_volatile* equal to `true`.

  — If a file is compressed, then the *compressed_size* and *compressed_cksum* attributes of the file should be checked instead of the *size* and *cksum* attributes.

    Applies to distributions.

  — Check symbolic links for correct values.

    Applies to distributions and installed software.

  If any of these checks fail for any file, generate an event. (SW_ERROR: SW_FILE_ERROR)

4.10.7.3 Execution Phase

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

4.10.7.3.1 Executing `fix` Scripts

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

— If a **`fix`** script returns an error, generate an event. (SW_ERROR: SW_PRE_SCRIPT_ERROR)

— If a **`fix`** script returns a warning, generate an event. (SW_WARNING: SW_PRE_SCRIPT_WARNING)

Control scripts shall adhere to the specifications in 4.1.6.1.

4.10.7.3.2 File Level Fix

The following file level fixes are made:

— Missing directories are created (except volatile unless the *check_volatile* option is true)

— Files that have been modified (except volatile unless the *check_volatile* option is true) are fixed for *mode*, *owner*, *group*, *major*, and *minor*, as applicable

— Symbolic links are recreated to correct their values

If any of these fixes fail for any file, generate an event. (SW_ERROR: SW_FILE_ERROR)

4.10.8 Exit Status

See 4.1.

4.10.9 Consequences of Errors

See 4.1.

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:

— A direct access file system as described in this clause

— A serial access extended **cpio** or extended **tar** archive as described in 5.3 of POSIX.1 {2}

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

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

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

Directory or File	Purpose
<*path*>/catalog/	Contains all information about the distribution
<*path*>/catalog/INDEX	Global index of distribution and its contents
<*path*>/catalog/dfiles/	Contains distribution attributes stored in files
<*path*>/catalog/dfiles/...	
<*path*>/catalog/<*product1*>/	Storage for information on the first product
<*path*>/catalog/<*product1*>/pfiles	Contains all product attributes stored in files
<*path*>/catalog/<*product1*>/pfiles/INFO	Control_file information for this product
<*path*>/catalog/<*product1*>/pfiles/<*script1*>	First vendor-defined control_file
<*path*>/catalog/<*product1*>/pfiles/...	Additional vendor-defined control_files
<*path*>/catalog/<*product1*>/<*fileset1*>	Storage for information and scripts on this fileset
<*path*>/catalog/<*product1*>/<*fileset1*>/INFO	File and control_file information for this fileset
<*path*>/catalog/<*product1*>/<*fileset1*>/<*script1*>	First vendor-defined control_file
<*path*>/catalog/<*product1*>/<*fileset1*>/...	Additional vendor-defined control_files
<*path*>/catalog/<*product1*>/<*fileset2*>/	Storage for information and scripts on the next fileset
<*path*>/catalog/<*product1*>/<*fileset2*>/...	
<*path*>/catalog/<*product2*>/	Storage for information on the next product
<*path*>/catalog/<*product2*>/...	
<*path*>/<*product1*>/	Storage for this product's filesets
<*path*>/<*product1*>/<*fileset1*>/	Storage for this fileset's files
<*path*>/<*product1*>/<*fileset1*>/<*file1*>	Path to first file
<*path*>/<*product1*>/<*fileset1*>/...	Path to additional files
<*path*>/<*product1*>/<*fileset2*>/	Storage for next fileset's files
<*path*>/<*product1*>/<*fileset2*>/*files*	Actual directory structure of files
<*path*>/<*product1*>/<*fileset2*>/...	
<*path*>/<*product2*>/	Storage for next product's filesets
<*path*>/<*product2*>/...	

Figure 5-1 – Example of Software Packaging Layout

The directory structure for the software packaging layout is divided into the following two areas:

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

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

5.1.1 Exported Catalog Structure

The catalog structure describes the software contained in the distribution. It is organized by product, and each product is organized by fileset. The specific contents are described in the following subclauses.

5.1.1.1 INDEX File

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

— **`catalog/INDEX`**

This **`INDEX`** file contains the definition of all software objects in the distribution.

5.1.1.2 Distribution Files

The **`catalog/dfiles/`** directory contains files used to store certain attributes of the distribution object. The distribution information stored can include the following:

— *<attribute>*

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

5.1.1.3 Product Catalog

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

5.1.1.4 Product Control Files

The **`catalog/`**<*product_control_directory*>//**`pfiles/`** directory contains the control_files for the product object. The product control_files include the following:

— *<attribute>*

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

— **`INFO`**

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

— **`checkinstall`**
`preinstall`
...

`postremove`

The vendor-supplied control scripts for the product.

— <*control_file*>

All other vendor-defined control_files for this product.

5.1.1.5 Fileset Control Files

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

— <*attribute*>

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

— **INFO**
Contains the definitions for the control_file and file objects contained within the fileset.

— `checkinstall`
`preinstall`
...
`postremove`

The vendor-supplied control scripts for the fileset.

— <*control_file*>

All other vendor-defined control_files for this fileset.

5.1.2 File Storage Structure

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

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

Each regular file (ones for which *file.type* is `f`) is stored in a location defined by appending the *file.path* attribute to the path of the fileset file storage directory. This may require the creation of additional directories. Other file types (directories, except as needed to store files; hard links and symbolic links) are not required to exist in the distribution. The POSIX.1 {2} file permissions for files in the file storage area are undefined.

5.1.2.1 Control Directory Names

In the simplest case, the value of the *product.tag* attribute is the name of the 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:

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

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

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

These conditions are met by defining a *control_directory* attribute for each product and fileset that is unique within the distribution. The attribute uses the following syntax:

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

%start  control_directory
%%

control_directory        : tag_part
                         | tag_part "."  instance_id_part
                         ;

tag_part                 : FILENAME_CHARACTER_STRING
                         ;

instance_id_part         : FILENAME_CHARACTER_STRING
                         ;
```

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

The **instance_id_part** is a string that, when added after the "." (period), defines a **control_directory** that, for products, is unique within the distribution and, for filesets, is unique within the product. For products, this **instance_id_part** may be the *instance_id* of the product if that *instance_id* was generated considering other products **tag_part**s in addition to *tag* attributes.

5.2 Software Definition File Format

The software definition files contain the software structure and the detailed attributes for distributions, installed_software, bundles, products, subproducts, filesets, files, and control_files. While information on installed software is represented in this form as input to, or output from, the various software administration utilities, the actual storage of this metadata for installed software is undefined. This subclause describes the format of the software definition files as follows:

— The **INDEX** file contains the definition of distribution or installed_software objects as well as the software objects contained within those software_collections. The information in this file is primarily used in selection 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 phases of the utilities.

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

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

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

5.2.1 Software Definition File Syntax

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

The software specification file syntax is as follows. See 2.1.2 for the grammar conventions for this syntax and Annex C for examples of syntax usage.

```
%token          FILENAME_CHARACTER_STRING /* as defined in 2.2.2.37  */
%token          NEWLINE_STRING            /* as defined in 2.2.2.61  */
%token          PATHNAME_CHARACTER_STRING /* as defined in 2.2.2.67  */
%token          SHELL_TOKEN_STRING        /* as defined in 2.2.2.80  */
%token          WHITE_SPACE_STRING        /* as defined in 2.2.2.110 */


%start  software_definition_file
%%

software_definition_file : INDEX
                         | INFO
                         | PSF
                         ;

INDEX                    : soc_definition
                           soc_contents
                         ;

INFO                     : info_contents
                         ;

PSF                      : distribution_definition
                           soc_contents
                         ;

media                    : /* empty */
                         | media_definition
                         ;
```

```
vendors                     : /* empty */
                            | vendors NEWLINE_STRING vendor_definition
                            | vendor_definition
                            ;

bundles                     : /* empty */
                            | bundles NEWLINE_STRING bundle_definition
                            | bundle_definition
                            ;

products                    : /* empty */
                            | products NEWLINE_STRING product_specification
                            | product_specification
                            ;

product_specification       : product_definition
                              product_contents
                            ;

subproducts                 : /* empty */
                            | subproducts NEWLINE_STRING subproduct_definition
                            | subproduct_definition
                            ;

filesets                    : filesets NEWLINE_STRING fileset_specification
                            | fileset_specification
                            ;

fileset_specification       : fileset_definition
                              fileset_contents
                              /* fileset contents not valid in INDEX files */
                            ;

control_files               : /* empty */
                            | control_files NEWLINE_STRING control_file_definition
                            | control_file_definition
                            ;

files                       : /* empty */
                            | files NEWLINE_STRING file_definition
                            | file_definition
                            ;

fileset_contents            : fileset_contents NEWLINE_STRING fileset_content_items
                            | fileset_content_items
                            ;

fileset_content_items       : control_files
                            | files
                            ;

info_contents               : info_contents NEWLINE_STRING info_content_items
                            | info_content_items
                            ;

info_content_items          : control_files
                            | files
                            ;

product_contents            : product_contents NEWLINE_STRING product_content_items
                            | product_content_items
```

```
                             ;

product_content_items        : control_files
                               /* control_files not valid in INDEX files */
                             | subproducts
                             | filesets
                             ;

soc_contents                 : soc_contents NEWLINE_STRING soc_content_items
                             | soc_content_items
                             ;

soc_content_items            : vendors
                             | bundles
                             | products
                             ;

soc_definition               : distribution_definition
                             | installed_software_definition
                             ;

distribution_definition      : software_definition
                               media
                             ;

media_definition             : software_definition
                             ;

installed_software_definition : software_definition
                                  ;

vendor_definition            : software_definition
                             ;

bundle_definition            : software_definition
                             ;

product_definition           : software_definition
                             ;

subproduct_definition        : software_definition
                             ;

fileset_definition           : software_definition
                             ;

control_file_definition      : software_definition
                             | extended_definition
                               /* extended_definition only valid in PSF files */
                             ;

file_definition              : software_definition
                             | extended_definition
                               /* extended_definition only valid in PSF files */
                             ;

software_definition          : object_keyword NEWLINE_STRING
                               attribute_value_list
                             ;

attribute_value_list         : /* empty */
```

```
                         | attribute_value_list attribute_definition NEWLINE_STRING
                         | attribute_definition NEWLINE_STRING
                         ;

attribute_definition     : attribute_keyword WHITE_SPACE_STRING attribute_value
                         ;

object_keyword           : FILENAME_CHARACTER_STRING
                         ;

attribute_keyword        : FILENAME_CHARACTER_STRING
                         ;

extended_definition      : extended_keyword WHITE_SPACE_STRING attribute_value
                         ;

extended_keyword         : FILENAME_CHARACTER_STRING
                         ;

attribute_value          : attribute_value WHITE_SPACE_STRING single_value
                         | single_value
                         | '<' WHITE_SPACE_STRING PATHNAME_CHARACTER_STRING
                         | '<' PATHNAME_CHARACTER_STRING
                         ;

single_value             : SHELL_TOKEN_STRING
                         ;
```

The following syntax rules are applicable to software definition files:

(1) All keywords and values are represented as character strings.

(2) Each keyword is located on a separate line. Keywords can be preceded by white space (tab, space). White space separates the keyword from the value.

(3) Comments can be placed on a line by themselves or after the keyword-value syntax. They are designated by preceding them with the # (pound) character. The way in which comments are used in **INDEX** and **INFO** is undefined.

(4) All object keywords have no values. All attribute keywords have one or more values.

(5) An attribute value ends on the same line as the keyword with one exception. Attribute values can span lines if and only if the value is prefixed and suffixed with the " (double quote) character.

(6) When an attribute value begins with < (less than), the remainder of the string value shall be interpreted as a filename whose contents will be used as a quoted string value for the attribute. For **INDEX** files, the filename shall be a path relative to the control directory for that distribution, product, or fileset. For PSF files, the filename shall be a path to a file on the host that contains the file.

(7) The use of " (double quote) is not required when defining a single line string value that contains embedded white space. Trailing white space shall be removed; embedded white space shall be used. The quotes can be used.

(8) The **"** (double quote), **#** (pound), and \ (backslash) characters can be included in multi-line string values by "escaping" them with \ (backslash).

(9) The order of attributes is not significant, except that the *layout_version* shall be the first attribute defined in an **INDEX** file for a distribution or installed_software object.

5.2.1.1 Keyword and Attribute Semantics

The keywords and attribute types have the following semantics:

(1) The object keywords **distribution**, **installed_software**, **bundle**, **product**, **subproduct**, **fileset**, **control_file**, and **file** each define a new object of that type. The keywords **distribution**, **installed_software**, **product**, and **fileset** also define nested blocks that contain the objects describing the software hierarchy.

(2) If an attribute is not supplied, then its default value shall be used, unless no default value is permitted.

(3) Attributes that have Boolean permitted values shall be described by the strings **true** and **false**.

(4) Attributes that have an enumerated set of permitted values shall be described by one of the enumerated values. Enumerated values shall not contain spaces and shall be case sensitive. In addition, abbreviations of the string shall not be allowed. For example, **conf** is not equivalent to **configured**.

(5) For attributes whose values are integer character strings, the default value shall be used if the attribute is not supplied. If the first two characters of an integer character string are **0x** (zero followed by a lowercase "x"), then the value shall be interpreted as hexadecimal. Otherwise, if the first character of an integer character string is **0** (zero), then the value shall be interpreted as octal. Attribute values denoting time shall be integer character strings that signify seconds since the Epoch.

(6) Attributes whose permitted values are lists of *tag*s or **software_spec**s can be described either by one or more repeating keywords, each listing one or more *tag*s or **software_spec**s separated by white space (e.g., for *subproduct.contents* or *fileset.prerequisites*), or by blocks of object fragments (e.g., product, fileset, and file definitions).

The former is used when the hierarchy is defined by reference, and the latter is used when the hierarchy is defined by containment. For example, subproducts and filesets are contained within products, but filesets are referenced by subproducts.

(7) Attributes that have permitted values of **software_pattern_matching_string** are software pattern matching strings as described in 2.2.2.92. For all product attributes related to the *uname* structure (as defined in 4.4.1 of POSIX.1 {2}), an empty string 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	*layout_version*
path	*path*
dfiles	*dfiles*
pfiles	*pfiles*
uuid	*uuid*

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	*sequence_number*

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.

5.2.4 Installed Software Definition

`installed_software`

`layout_version`	*layout_version*
`path`	*path*
`dfiles`	*dfiles*
`pfiles`	*pfiles*
`catalog`	*catalog*

The storage of catalogs for installed software is undefined. With the use of the **`swlist`** utility, the contents of such catalogs may be manifested in exported catalog form. The rules contained within this subclause shall apply when the contents of an installed software catalog is manifested in exported catalog form.

`INDEX` files can contain installed_software definitions. This describes the attributes for installed_software objects when listed in exported catalog structure using **`swlist`**.

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

An **`INDEX`** file does not contain the *path* or *catalog* attributes; they are generated dynamically by **`swlist`**.

An **`INDEX`** file shall contain the *layout_version* attribute as the first attribute defined in the file.

5.2.5 Vendor Definition

`vendor`

`tag`	*tag*
`title`	*title*
`description`	*description*

`INDEX` and PSF files can contain vendor definitions. The *tag* attribute is required for all vendor objects.

5.2.6 Bundle Definition

`bundle`

`tag`	*tag*
`architecture`	*architecture*
`location`	*location*
`qualifier`	*qualifier*
`revision`	*revision*
`vendor_tag`	*vendor_tag*
`contents`	*contents*
`copyright`	*copyright*
`create_time`	*create_time*
`description`	*description*
`directory`	*directory*
`instance_id`	*instance_id*
`is_locatable`	*is_locatable*
`machine_type`	*machine_type*
`mod_time`	*mod_time*
`number`	*number*
`os_name`	*os_name*
`os_release`	*os_release*

os_version	*os_version*
size	*size*
title	*title*

INDEX and PSF files can contain bundle definitions. The *tag* and *contents* attributes are required for all bundles.

Neither file contains the *size* attribute. The value of the *size* attribute is generated dynamically based on the sizes of the filesets currently contained within the bundle.

An INDEX file shall also contain an *instance_id* attribute. The value of the *instance_id* attribute is generated dynamically by **swpackage** or **swcopy**. An INDEX file for installed software shall contain a *create_time* attribute and a *mod_time* attribute for each bundle.

Only bundle definitions for installed software may contain either the *location* or *qualifier* attributes; bundle definitions for distributions shall not contain either the *location* or *qualifier* attributes.

A PSF should not contain either the *location* or *qualifier* attributes; they shall be ignored when parsing the file.

5.2.7 Product Definition

product

tag	*tag*
architecture	*architecture*
qualifier	*qualifier*
revision	*revision*
vendor_tag	*vendor_tag*
all_filesets	*all_filesets*
control_directory	*control_directory*
copyright	*copyright*
create_time	*create_time*
directory	*directory*
description	*description*
instance_id	*instance_id*
is_locatable	*is_locatable*
postkernel	*postkernel*
location	*location*
machine_type	*machine_type*
mod_time	*mod_time*
number	*number*
os_name	*os_name*
os_release	*os_release*
os_version	*os_version*
size	*size*
title	*title*

INDEX and PSF files can contain product definitions. The *tag* and *control_directory* attributes are required for all products.

Neither file contains the *size* attribute. The value of the *size* attribute is generated dynamically based on the sizes of the filesets currently contained within the product.

An INDEX file for installed software shall contain a *create_time* attribute and a *mod_time* attribute for each product.

The *control_files*, *filesets*, and *subproducts* attributes are not stored as attributes, but rather as **`control_file`**, **`fileset`**, and **`subproduct`** definitions. These attributes are not included in **`swlist -v`** output. Rather, they are generated only by **`swlist -a`** *attribute*.

An **`INDEX`** file shall contain an *instance_id* attribute. The value of the *instance_id* attribute is generated by **`swpackage`** or **`swcopy`**. An **`INDEX`** file shall also contain an *all_filesets* attribute in addition to the fileset definitions. This attribute is generated by **`swpackage`** and represents all filesets defined for the product, as opposed to those that are currently contained within the product. The value of the *filesets* and *all_filesets* attributes may differ, since some originally defined filesets might not be copied or installed. Only product definitions for installed software may contain either the *location* or *qualifier* attributes; product definitions for distributions shall not contain either the *location* or *qualifier* attributes.

A PSF should not contain either the *location* or *qualifier* attributes; they shall be ignored when parsing the file.

5.2.8 Subproduct Definition

`subproduct`	
`tag`	*tag*
`contents`	*subproducts*
`create_time`	*create_time*
`description`	*description*
`mod_time`	*mod_time*
`size`	*size*
`title`	*title*

`INDEX` and PSF files can contain subproduct definitions. The *tag* and *contents* attributes are required for all subproducts.

Neither file contains the *size* attribute. The value of the *size* attribute is generated dynamically based on the sizes of the filesets currently contained within the subproduct.

An **`INDEX`** file for installed software shall contain a *create_time* attribute and a *mod_time* attribute for each subproduct.

5.2.9 Fileset Definition

`fileset`	
`tag`	*tag*
`control_directory`	*control_directory*
`corequisites`	*corequisites*
`create_time`	*create_time*
`description`	*description*
`exrequisites`	*exrequisites*
`is_reboot`	*is_reboot*
`is_kernel`	*is_kernel*
`is_locatable`	*is_locatable*
`location`	*location*
`media_sequence_number`	*media_sequence_number*
`mod_time`	*mod_time*
`prerequisites`	*prerequisites*
`revision`	*revision*

`size`	*size*
`state`	*state*
`title`	*title*

INDEX and PSF files can contain fileset definitions. The *tag* and *control_directory* attributes are required for all filesets.

The *control_files* and *files* attributes are not stored as attributes, but rather as **control_file** and **file** definitions. These attributes are not included in **swlist -v** output. Rather, they are generated only by **swlist -a** *attribute*.

An **INDEX** file shall contain a *size* attribute for each defined fileset. Fileset definitions for distributions that span multiple media shall contain the *media_sequence_number* attribute. An **INDEX** file for installed_software shall contain a *create_time* and a *mod_time* attribute for each fileset.

A PSF should not contain the *location*, *media_sequence_number*, *size*, or *state* attributes; they shall be ignored when parsing the file. The value of the *size* attribute is generated dynamically by **swpackage** based on the sizes of the files and control_files.

5.2.10 Control_File Definition

`control_file`

`tag`	*tag*
`cksum`	*cksum*
`compressed_cksum`	*compressed_cksum*
`compressed_size`	*compressed_size*
`compression_state`	*compression_state*
`compression_type`	*compression_type*
`interpreter`	*interpreter*
`path`	*path*
`revision`	*revision*
`size`	*size*
`source`	*source*
`result`	*result*

INFO and PSF files can contain control_file definitions.

A PSF should not contain the *cksum*, *compressed_cksum*, *compressed_size*, *compression_state*, *compression_type*, *size*, or *result* attributes; they shall be ignored when parsing the file. The values of the *size* and *cksum* attributes are generated dynamically by **swpackage** based on the source file. A PSF shall contain a *source* attribute. A PSF can contain a *path* attribute. If it does not, **swpackage** shall use the basename obtained from the value of the *source* attribute as the value of the *path* attribute. A PSF can contain a *tag* attribute. If it does not, **swpackage** shall use the basename obtained from the value of the *path* attribute as the value of the *tag* attribute. The **swpackage** utility shall resolve the value of the *tag* attribute after it resolves the value of the *path* attribute.

An **INDEX** file shall contain the *tag*, *path*, *cksum*, and *size* attributes. Control_file definitions for installed software shall also contain the *result* attribute. **INDEX** files should not contain the *source* attribute; it shall be ignored when parsing the file.

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* attribute for the **INFO** control_file itself is not defined. An implementation can choose to store certain software object attributes, such as *copyright*, as control_files.

5.2.11 File Definition

`file`	
`path`	*path*
`cksum`	*cksum*
`compressed_cksum`	*compressed_cksum*
`compressed_size`	*compressed_size*
`compression_state`	*compression_state*
`compression_type`	*compression_type*
`gid`	*gid*
`group`	*group*
`is_volatile`	*is_volatile*
`link_source`	*link_source*
`major`	*major*
`minor`	*minor*
`mode`	*mode*
`mtime`	*mtime*
`owner`	*owner*
`revision`	*revision*
`size`	*size*
`source`	*source*
`type`	*type*
`uid`	*uid*

`INFO` and PSF can contain file definitions.

A PSF shall contain a *source* attribute. A PSF should not contain the *cksum* *compressed_cksum* *compressed_size*, *compression_state*, *compression_type*, or *size* attributes; they shall be ignored when parsing the file. Device files (including the *major* and *minor* attributes) should not be defined in a PSF (but can be added via **`swmodify`** after being created by a **`configure`** script); they shall be ignored when parsing the file. The values of the *size* and *cksum* attributes are generated dynamically by **`swpackage`** based on the source file. A PSF can contain a *path* attribute, otherwise the source is used to defined the path by **`swpackage`**. A PSF can contain *gid*, *group*, *link_source*, *mode*, *mtime*, *owner*, *type*, and *uid* attributes, otherwise they are retrieved from the source file by **`swpackage`**. A PSF can contain *is_volatile* and *revision* attributes. Automatic determination of the file revision is implementation defined.

An **`INFO`** file should not contain the *source* attribute; it shall be ignored when parsing the file. Table 5-1 shows the required, optional, and non-applicable attributes for each of the file types in an **`INFO`** file. The file types are described in 3.13. Within a fileset, no more than one copy of a file shall be stored with the same path.

Within a PSF file, if the same file is defined more than once, the attributes from the last definition shall be used and shall redefine the attributes previously defined. This action shall not cause additional copies of the file to be stored in the 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.

5.2.13 Extended File Definitions

A PSF can contain extended file definitions. The **swpackage** utility supports these extended file definition mechanisms:

directory mapping
: A PSF can point the **swpackage** utility at a source directory containing the files for the fileset. In addition, a PSF can map this source directory to the appropriate (destination) directory containing this subset of the filesets files.

recursive (implicit) file definition
: If a directory mapping is active, a PSF can direct the **swpackage** utility to include all files (recursively) from within the directory in the fileset.

explicit file definition
: For all or some of the files in the fileset, a PSF can name each source file and destination path with a one line per file syntax.

default permission definition
: For all or some of the files in the fileset, a PSF can define a default set of permissions.

excluding files
: Files that otherwise would be included can be explicitly excluded.

including files
: File definitions may be included from a separate file.

These mechanisms can all be used in combination with the others.

5.2.13.1 Directory Mapping

directory *source* [*path*]

This syntax defines a *source* directory under which subsequently listed files are located. In addition, the user can map the *source* directory to a *destination* directory under which the packaged files will be located.

The destination directory shall be an absolute pathname, and shall be used as a prefix to the *path* attribute for each of the files.

The source directory can be either an absolute pathname, or a relative pathname. If relative, the **swpackage** utility interprets it relative to the current working directory in which the utility was invoked.

If the *source* directory does not exist, the **swpackage** utility shall generate an error.

5.2.13.2 Recursive File Definition

file *

This syntax directs the **swpackage** utility to include every file (and directory) 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.

The **directory** keyword shall be specified before the **file** * specification is used. After finishing the recursive processing of the source directory, the **swpackage** utility processes further specifications with respect to the original directory.

All other attributes for the destination file object are taken from the source file, unless a **file_permissions** keyword is active. This keyword is described below.

The user can specify multiple **directory** and **file** * pairs to gather all files from different source directories into a single fileset.

5.2.13.3 Explicit File Definition

file **[-t** *type***]** **[-m** *mode***]** **[-o** *owner***[,***uid***]]** **[-g** *group***[,***gid***]** **[-n]** **[-v]** *source* **[***path***]**

Instead of, or in addition to, the recursive file specification, the user can explicitly specify the files to be packaged into a fileset.

This syntax may be used to redefine an attribute of a previously defined file. All attributes not specifically listed remain the same.

The **directory** keyword can be used to define a source (and destination) for explicitly specified files. If no **directory** keyword is active, then the full source path and the absolute destination path (the *path* attribute) shall be specified for each file.

The meaning of each of these fields is as follows:

file This keyword specifies an existing file or directory, perhaps within the currently active source directory, to include in the fileset. It can also specify a directory, hard link, or symbolic link that does not exist as a source file, but is created when the fileset is installed.

source When specifying an existing source file, this value defines the path to it.

If this is a relative path, the **swpackage** utility searches for it relative to the source directory set by the **directory** keyword. If no source directory is active, the **swpackage** utility searches for it relative to the current working directory in which the utility was invoked.

All attributes for the destination file object are taken from the source file, unless a **file_permissions** keyword is active, or the **-m**, **-o**, or **-g** options are also included in the file specification.

When specifying a new directory to be created upon installation where there is no destination *path* specified, the *source* defines the path of the installed directory. When specifying a new hard link or symbolic link to be created upon installation, the *source* defines the pathname of the installed file to use as the source for the new file.

path

When specifying a new or existing source file, this value defines the destination path at which the file will be created or installed. If *path* is a relative path, the active destination directory set by the **directory** keyword shall be prefixed to it. If the path is relative, and no destination directory is active, the **swpackage** utility shall generate an error. If the path is not specified, then the *source* is used as the *path* with the appropriate mapping done with the active destination directory (if any).

-t *type* When creating a new directory, hard link or symbolic link (a file in the fileset that does not exist in the source), this option shall be specified to define the file type. The following file types can be created:

d Create a directory. If only the *source* is specified, it is used as the *path*. Otherwise, the *source* is used to retrieve the attributes for the directory created at *path*. If the path is not specified, or any attributes, then default values of the attributes shall be implementation defined.

h Create a hard link. Both the *source* and *path* shall be specified. The *source* is the pathname of the installed file object to be used as the source for the new hard link (the *link_source* attribute).

s Create a symbolic link. Both the *source* and *path* shall be specified. The *source* is the pathname of the installed file object to be used as the source for the new symbolic link (the *link_source* attribute).

Files with the types **c** (character special), **b** (block special), and **p** (named pipe | FIFO) are not supported by **swpackage** and **swinstall** and can be created via a **configure** control script. In general, device files and pipes are created during system configuration on the system actually running the software. Also, there can be files of other types that the **swpackage** utility does not recognize and that shall therefore cause an error.

-m *mode* This option defines the (octal) mode for a file or directory.

-o [*owner*][,*uid*]
This option defines the name or uid, or both, of the owner of the destination file. If only the *owner* is specified, then the *owner* and *uid* attributes are set for the destination file object based on the database of the packaging host. If only the *uid* is specified, it is set as the *uid* attribute for the destination object and no owner name is assigned. If both are specified, each sets the corresponding attribute for the file object. If neither are specified, then the owner and uid of the file shall be used as found in the file system of the packaging host. See 4.5.7.3.3.

During an installation, the *owner* attribute is used to set the owner name and uid unless the owner name is not defined in the target system user database. In this case, the value of the *uid* attribute is used to set the uid.

-g [*group*][,*gid*]
This option defines the name or gid, or both, of the group of the destination file. If only the *group* is specified, then the *group* and *gid* attributes are set for the destination file object based on the database

of the packaging host. If only the *group* is specified, and it contains digits only, it is interpreted as the gid, and is set as the *gid* attribute for the destination object; no group name is assigned to the object. If both are specified, each sets the corresponding attribute for the file object. If neither are specified, then the group and gid of the file shall be used as found in the file system of the packaging host. See 4.5.7.3.3.

During an installation, the *group* attribute is used to set the group name and gid, unless the group name is not defined in the target system group database. In this case, the *gid* attribute is used to set the gid.

`-n` This option indicates that the file is not compressible.

`-v` The use of `-v` on a source line is used to specify that the file is volatile (contents, attributes or existence can change after installation).

5.2.13.4 Default Permission Definition

`file_permissions [-m` *mode* `| -u` *umask*`][-o [`*owner*`[,]][`*uid*`]] [-g [`*group*`[,]][`*gid*`]]`

A destination file object inherits the mode, owner, and group of the source file. The `file_permissions` keyword can be specified as follows to set a default permission mask, owner, and group for all the files being packaged into the fileset:

`file_permissions`
This keyword only applies to the fileset it is defined in. Multiple `file_permissions` can be specified, and subsequent definitions simply replace previous definitions.

`-m` *mode* This option defines a default (octal) mode for all file objects.

`-u` *umask*
Instead of specifying an octal mode as the default, the user can specify an octal *umask*() value that gets "subtracted" from the mode of an existing source file, or applied for each nonexistent file, to generate the mode of the destination file.

By specifying a *umask*() value the user can set a default mode for executable files, non-executable files, and directories. A specific mode can be set for any file, as described above.

`-o` [*owner*[,]][*uid*]
This option defines the name or uid, or both, of the owner of the destination file. See the discussion of the `-o` option in 5.2.13.3.

`-g` [*group*[,]][*gid*]
This option defines the name or gid, or both, of the group of the destination file. See the discussion of the `-g` option in 5.2.13.3.

5.2.13.5 Excluding Files

`exclude` *source*

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.

If the source specifies a directory, then all files below that directory are excluded.

5.2.13.6 Including Files

`file <` *include_file*

The **file** keyword can be used to include definitions for files from a separate include_file by specifying a < (less than) character followed by the include_file.

5.2.14 Space Control_file

path `[+|-]`*size*

For each path listed in the **space** file, the **swinstall** utility shall add the size, in bytes, to the disk space requirements. The size can be positive (reflecting the maximum transient or permanent disk space required for the install), or negative (reflecting space freed by one of the scripts executed by the **swinstall** command). An implementation shall consider positive records and may consider negative records.

5.3 Serial Format and Multiple Media

A distribution in the serial format of the software packaging layout is a bit-stream representation, implemented as a set of POSIX.1 {2} extended **cpio** or extended **tar** archives which contain files in the directory structure of the software packaging layout defined in Section 5.

A serial distribution can be stored on any serial medium. A serial distribution can also be stored in any file, within the file system, which supports the storing of POSIX.1 {2} extended **cpio** or extended **tar** archives. How a system reads or writes to the different media devices is outside the scope of this part of ISO/IEC 15068.

Implementations shall support serial distributions if the underlying operating system supports the **pax** utility, as defined in POSIX.2 {3}, or otherwise supports reading and writing of the extended **tar** and extended **cpio** archives defined in POSIX.1 {2}. If serial distributions are supported, the serial distribution formats supported shall include extended **tar** and extended **cpio**.

The distribution is implemented as a set of one or more POSIX.1 {2} extended **cpio** or extended **tar** archives. The archives reside on a set of one or more serial media, or in a file. Each media in a serial distribution shall contain one and only one archive.

A distribution may span multiple media in a hierarchical structure. In this case, the set of files on any particular media, including the attributes defined in any software definition files, should be similar to that for a serial archive. In other words, the decision for which files are put on which media should be the same whether the distribution is serial or hierarchical. Space considerations on media may cause some differences.

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.

(1) The catalog files (which contain all the information describing the software contained in the distribution), as well as the control scripts, in this relative order.

   (a) The global **INDEX** file, as described in 5.1.1.1

   (b) The distribution files, as described in 5.1.1.2

   (c) The product catalog files, product by product, as described in 5.1.1.3

      [1] The product control files, as described in 5.1.1.4

      [2] The fileset control files, fileset by fileset, as described in 5.1.1.5

(2) The actual software files, fileset by fileset, as described in 5.1.2

   (a) Prerequisites of filesets before the filesets that depend on them

   (b) Kernel filesets before non-kernel filesets (except where kernel filesets have prerequisites on non-kernel)

(3) Each medium shall have (as its first file, if a serial medium)

   (a) A global **INDEX** file, **catalog/INDEX** that shall at least contain the *distribution.uuid* and *media.sequence_number* attributes (used to identify a particular media within a particular distribution)

(4) Each archive shall start at the beginning of the medium. Multiple archives on one medium are not allowed.

Additionally, in order to increase the usability of multiple media serial distributions, the following guidelines should be used and in decreasing importance:

— Each medium should contain complete files wherever possible. If a file is larger than the the capacity defined by the *media_capacity* option, then the behavior is implementation defined.

— Each medium should contain complete filesets whenever possible or practical. The *fileset.media_sequence_number* attribute is the number of the medium where the fileset begins. If a fileset is larger then the medium size, then the *fileset.media_sequence_number* attribute shall contain the list of *media.sequence_number*s describing the media that contain this fileset.

— Each medium should contain complete products whenever possible or practical.

— Each medium should contain needed dependencies whenever possible or practical.

Thus, a conforming implementation shall be able to

— Read the **INDEX** off of the first medium for the Selection Phase

— Scan the first medium (and those following as needed) for the necessary catalog files for the Analysis Phase

— Request the next needed medium for the next needed fileset based on *media_sequence_number* during the Execution Phase

— Request the next medium when the fileset spans media

Note that in all respects, a serial distribution shall conform to the specifications of the extended **`cpio`** or extended **`tar`** archives. See 10.1.1 and 10.1.2 of POSIX.1 {2}. This includes, but is not limited to, the following:

— Recording format

— Character sets

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 Varembé, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse.

3) IEC documents can be obtained from the IEC office, 3 rue de Varembé, Case Postale 131, CH-1211, Genève 20, Switzerland/Suisse.

{B14} ISO/IEC 10164-18: 1997, *Information technology — Open Systems Interconnection — Systems Management — Part 18: Software Management Function.*

{B15} ISO/IEC 10646-1: 1993, *Information technology—Universal Multiple-Octet Coded Character Set (UCS)—Part 1: Architecture and Basic Multilingual Plane.*

{B16} ISO/IEC TR 10000-1: 1992, *Information technology—Framework and taxonomy of International Standardized Profiles—Part 1: General principles and documentation framework.*

{B17} ISO/IEC JTC 1 N1335, *Final Report of ISO/IEC JTC 1 TSG-1 on Standards necessary to define Interfaces for Application Portability (IAP).*

{B18} International Organization for Standardization/Association Française de Normalisation. *Dictionary of Computer Science/Dictionnaire de L'Informatique.* Geneva/Paris: ISO/AFNOR, 1989.

{B19} IEEE Std 100-1992, *IEEE Standard Dictionary of Electrical and Electronics Terms.*[4)]

{B20} IEEE Std 1003.0-1995, *IEEE Guide to the POSIX® Open Systems Environment (OSE).*

{B21} IEEE P1003.1a/D12, *Draft Revision to Information technology—Portable Operating System Interface (POSIX®) Part 1: System Application Program Interface (API) [C Language]*[5)]

{B22} IEEE P2003/D7, *Standard for Information Technology—Test Methods for Measuring Conformance to POSIX®.*

{B23} IEEE P2003.2/D11, *Standard for Information Technology—Test Methods for Measuring Conformance to POSIX®—Part 2: Shell and Utilities.*

{B24} RFC 819, Su, Z. and Postel, J. B. *Domain naming convention for Internet user applications.*[6)]

{B25} RFC 822, Crocker, D. *Standard for the format of ARPA Internet text messages.*

{B26} RFC 920, Postel, J. B. and Reynolds, J. K. *Domain requirements.*

{B27} RFC 921, Postel, J. B. *Domain name system implementation schedule — revised.*

{B28} RFC 1123, Braden, R. T. *Requirements for Internet hosts — application and support.*

4) IEEE publications can be obtained from IEEE Publications, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331. Telephone: 1 (800) 678-IEEE or +1 (908) 981-1393 (outside US).

5) Numbers preceeded by P are IEEE authorized standards projects that were not approved by the IEEE Standards Board at the time this publication went to press. For information about obtaining drafts, contact the IEEE.

6) Internet Requests for Comments (RFC) are available from the DDN Network Information Center, SRI International, Menlo Park, CA, USA 94025.

{B29} RFC 1514, Grillo, P. and Waldbusser, S. *Host Resources MIB*

{B30} American Telephone and Telegraph Company. *System V Interface Definition (SVID), Issues 2 and 3.* Morristown, NJ: UNIX Press, 1986, 1989.[7)]

{B31} University of California at Berkeley—Computer Science Research Group. *4.3 Berkeley Software Distribution, Virtual VAX-11 Version.* Berkeley, CA: The Regents of the University of California, April 1986.

{B32} X/Open Company, Ltd. *X/Open Preliminary Specification, Systems Management: Distributed Software Administration.* Reading, UK: X/Open Company, 1995.

{B33} X/Open Company, Ltd. *X/Open Guide, Systems Management: Identification of Management Services.* Reading, UK: X/Open Company, 1993.

{B34} X/Open Company, Ltd. *X/Open Guide, Systems Management: Managed Object Guide.* Reading, UK: X/Open Company, 1993.

{B35} X/Open Company, Ltd. *X/Open CAE Specification, Systems Management: Management Protocols API.* Reading, UK: X/Open Company, 1994.

{B36} X/Open Company, Ltd. *X/Open Guide, Systems Management: Reference Model.* Reading, UK: X/Open Company, 1993.

{B37} X/Open Company, Ltd. *X/Open Preliminary Specification, The Common Object Request Broker: Architecture and Specification.* Reading, UK: X/Open Company, 1993.

{B38} X/Open Company, Ltd. *X/Open Portability Guide, Issue 2.* Amsterdam: Elsevier Science Publishers, 1987.

{B39} X/Open Company, Ltd. *X/Open Specification, X/Open Portability Guide, Issue 3.* Reading, UK: X/Open Company, 1992.

{B40} X/Open Company, Ltd. *X/Open Specification, XPG4.* Reading, UK: X/Open Company, 1992.

7) This is one of several documents that represent an industry specification in a related area. The creators of such documents may be able to identify newer versions that may be interesting.

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.

B.1.3.2 Distribution Conformance

B.1.3.2.1 Strictly Conforming POSIX.7.2 Distribution

The Strictly Conforming Distribution class is intended to provide the highest degree of portability for a distribution. Conformance to this class guarantees that any conforming implementation can install this software properly.

B.1.3.2.2 Conforming POSIX.7.2 Distribution

The Conforming Distribution class is intended to guarantee that any conforming implementation can copy or install this software properly. This class also allows for additional functionality, which may come either from implementations that can take advantage of additional attributes, or from software being able to store and retrieve that information from any Conforming Implementation.

B.1.3.2.3 Conforming POSIX.7.2 Distribution Using Extensions

This class is intended to allow evolution of this standard, but in an open, consistent and well-documented manner. Examples of this are compressed media or bootable serial media. Both of these are features were recognized as important, but upon which consensus was not reached.

This class also provides flexibility for distributions needing to conform to other constraints related to the support of POSIX.1 {2} and POSIX.2 {3}. Users among the developers of this part of ISO/IEC 15068 strongly voiced the desire to support interpreters other than `sh`. Support for other interpreters also permits the use of such distributions on systems, such as DOS, which are not conformant with POSIX.1 {2} or POSIX.2 {3}.

B.1.3.2.4 Documentation

There is no additional rationale provided for this subclause.

B.1.4 Test Methods

There is no additional rationale provided for this clause.

B.2 Terminology and General Requirements

B.2.1 Conventions

B.2.1.1 Editorial Conventions

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.

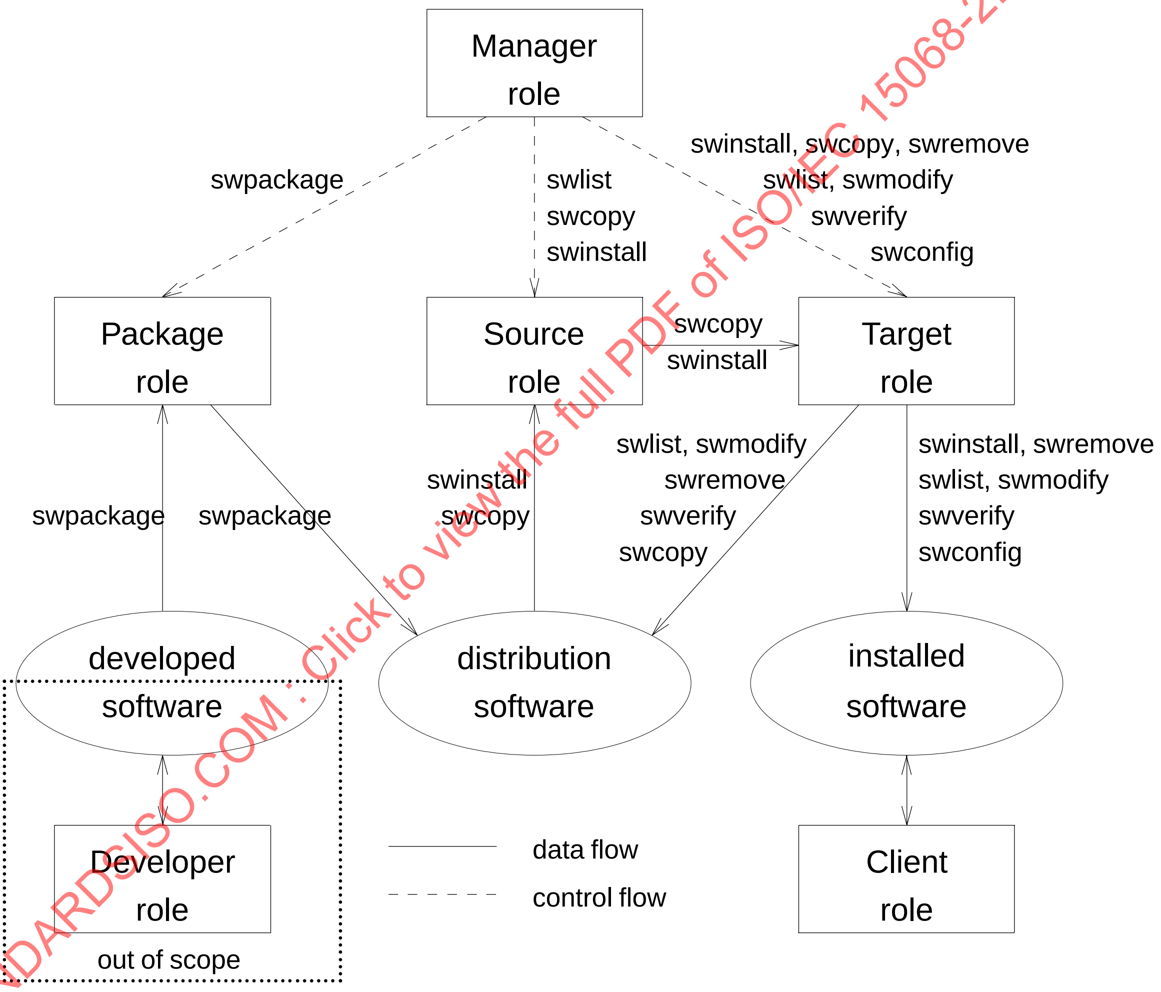

Figure B-1 – Roles in Software Administration

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 neutral concept of role is introduced. This stems from a need to refer to things that occur logically, if not physically, on what might be thought of as a client or server. In the context of this part of ISO/IEC 15068, roles are simply a convenient way of referring to where function is apparent, with no implication for how this is actually implemented.

It may be helpful to think of roles as separate processes, one per role, but that is only one possible implementation. Roles may operate on separate systems, or hosts, although all roles may operate on the same host. For example, the packager role creates an initial distribution. When copying this distribution, the source role provides read access to the distribution files, while the target role writes the new copy. This new copy may then be read by another source role for another install or copy.

For any implementation, a role consists of the entire set of tasks that may occur within the role. A task is a set of well-defined behaviors and state changes in the managed objects. Tasks are initiated by the system administrator using a specific command in the command line interface (CLI). Tasks are defined by this part of ISO/IEC 15068 in terms of the state changes on the software objects on target hosts.

As each task proceeds, different roles are involved. These roles may be realized on a single machine or could involve a different machine for each role.

Developer Role	Where the software is developed, tested, and maintained. This role is outside the scope of this part of ISO/IEC 15068. In Figure B-1, software is developed by the development role in some environment that results in it being in the developed state.
Manager Role	Where each task is initiated and is concerned with taking appropriate action at the completion or failure of a task. Manager control is understood as a more common need than target control, so at least that should be supported. For this reason, the manager role sets the options for a task, and each of the target hosts implements those options. So, any extension involves a set of ways to define selected control over particular policies. A design for this has not been pursued beyond recognizing the complexity of the problem. The manager role provides the means of controlling the way software is created, transferred, and installed. In particular it provides an administrative interface to the other roles, enabling their activities to be controlled in a coordinated manner.
Packager Role	Where software that has been developed is organized in a form suitable for distribution. The packager role transforms a product from the format produced by the developer role to the format specified by

this part of ISO/IEC 15068 for use by the next stage of the process, the source, and manager roles. The packager role defines the requirements for this transformation to be successful — the input (the product specification file, and the files it describes), the command line interface to initiate the transformation (`swpackage`) utility, and the output of the packaging task (packaging layout).

Two distinct, but related, formats for packaged software are supported by this part of ISO/IEC 15068 — a structured format residing within a POSIX.1 {2} hierarchical file system (such as disk, CD, etc.), and a bit stream representation residing on any serial device or file (such as tape, `tar` archive file, etc.).

Source Role

Where the software exists in a form suitable for distribution and hence forms a context for the establishment of a repository of software from which the manager may choose to distribute to the target.

Software exists in the source until it is removed by a task initiated by the manager.

The source role provides a repository where software may be stored, and provides access for those roles that require the software.

Target Role

The target of a task.

For example, when installing software, the target is where software is installed after having been delivered from a source. As another example, the target for a copy task command refers to the distribution where products are added. For management tasks like removing software, the target refers to either the installed_software objects or the distributions from which software is being removed.

Part of the distributed model involves the target role granting permission to the manager role to perform various software administration tasks. Authority for certain classes of tasks may be individually controllable, for example, modifying vs. listing installed products. While it is entirely conceivable that the target may want to restrict the way authorized tasks are performed, it is beyond the scope of this part of ISO/IEC 15068.

Client Role

Where the software is actually executed or used, which may be different from where it is actually installed.

Software is configured for use on the client.

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 sharing. A manager role initiates the install task with the source role serving the software from the distribution and the target role installing the software on a fileserver. After the installation is complete, then a client role on each client sharing this software performs configuration for the shared software and the client host.

It is important to understand the difference between the target and client roles. The client role is where the software is actually used and where configuration of the software takes place, while the target role is where the software is installed. Although in many cases these are the same machines, in some cases they are different and the separation of configuration from installation is important. Each target from the *targets* operand of an install or configure task may identify a target role (if installing but not configuring), a client role (if just configuring), or both (if installing and configuring).

Two examples of when these roles refer to different machines are:

— Proxy install (installing on one system for use by another system) where configuration of the software is done separately, because the target may not have the necessary capabilities or information, or both, for configuring the client.

— The target is a file server, and there are multiple clients that access the software installed on the file server. Each client may require separate configuration as targets of **swconfig**.

Figure B-1 shows a split between the manager role and the other roles. The administrative interface to software administration is provided in the manager role, from which the individual tasks that take place in the other roles are controlled.

This part of ISO/IEC 15068 defines a set of utilities that is such an administrative interface. These utilities provide basic facilities for controlling the individual tasks. Other management applications may be built that provide much more comprehensive software administration facilities. This part of ISO/IEC 15068 defines facilities that enable management applications to control software administration across any number of systems with conforming implementations.

One item of note among the general terms is the definition of symbolic link (see 2.2.2.99). While not yet standardized (see POSIX.1a {B21}), symbolic links are an entrenched part of existing practice. This part of ISO/IEC 15068 makes no attempt to independently define symbolic links. Rather, the functional characteristics 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.

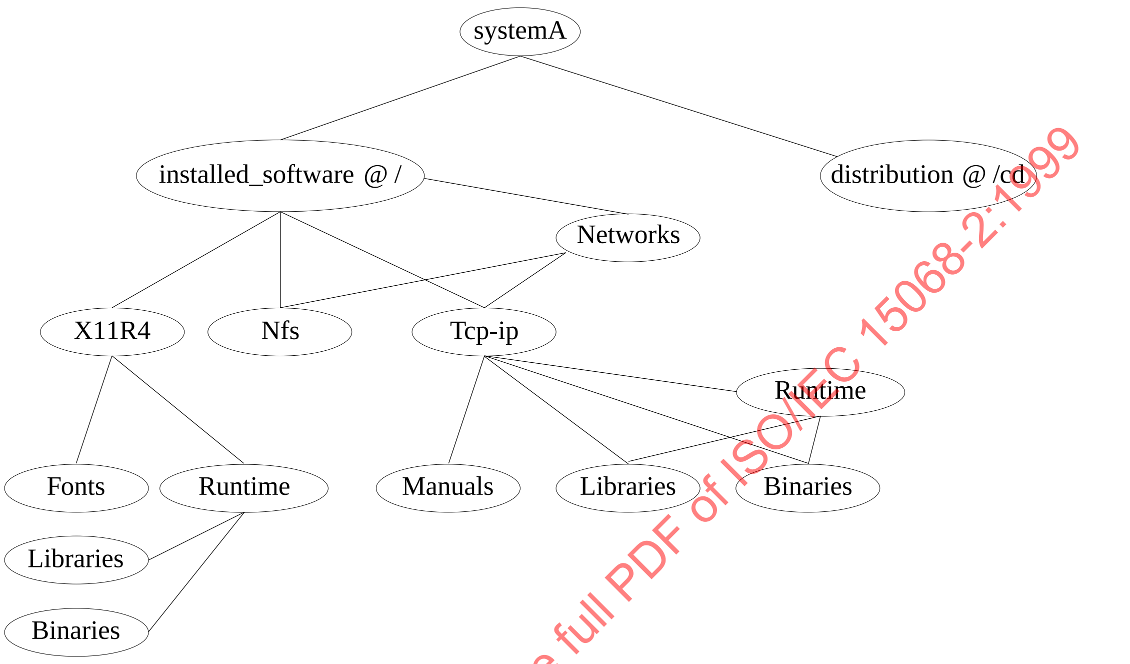

Figure B-2 – Example of Software Structure

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

	combination of the two.
bundle	A bundle is a grouping of other software objects and is a convenient way to reference a set of software.
fileset	A fileset consists of the actual files plus control scripts. Filesets are generally the lowest level of software object that can be operated on by the utilities.
subproduct	Subproducts are a grouping of other subproducts, or of filesets, or of some combination, that resolve to a group of filesets. Subproducts are a convenient way to aggregate filesets.

The software_files define the files and control_files that are contained in the software objects that are operated on during a software administration utility. There are two classes of software_files as described in the following:

control_file	Control_files consist of control scripts and other files that are used in various ways by the utilities. Control scripts are executed by the utilities at various points in a task. Control scripts provide a way to perform steps, in addition to those executed by the utilities, at various points in the task such as preinstall checking, postinstall customization, configuration, and verification. Either a single script with multiple entry points, or multiple scripts can be defined.
	Most control scripts are run on the target, which may be a different architecture than the client on which the software operates. They should, therefore, use POSIX.2 {3} utilities, except where they can determine that they are running on the client.
	In addition to scripts, other control_files provide input to the control scripts, or to the utilities directly (e.g., the **response** and **space** control_files).
file	Files are the lowest level of object defined by this part of ISO/IEC 15068. Files contain the attributes describing the file including the contents of the file and its installed location.

The distributions and installed_software objects are the sources or targets of a software administration command. The software objects (products, filesets, bundles, and subproducts) are the objects that are being applied to those targets.

This part of ISO/IEC 15068 describes the structure and the attributes for software_collections, software objects, and software_files. It also describes the behaviors for the utilities that operate on these objects. However, these structure definitions are not managed object classes in the ISO sense because the behaviors are not described in terms of methods within object classes.

NOTE: Object classes are templates for the creation of object instances. They are analogous to the 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 data manipulation, including creation, modification, and listing, so that the actual storage of the data is appropriately hidden from the application using the objects.

Figure B-3 shows the components of the software object hierarchy. The containment arrows designate objects that are defined within the context of their containing objects. An object can only exist within one containing object. The identifier of an object (for example, the *tag* attribute of a fileset) only needs to be unique within the scope of the containing object.

The reference arrows designate objects that are included when this object is operated on. An object may be referred to by more than one object. Bundles need not refer to entire products, but can refer to individual filesets or subproducts. Fileset and subproduct objects can be referenced directly by bundles by also identifying the product of which the fileset or subproduct is a part.

Figure B-4 shows the software administration common classes and the software objects that inherit attributes from these common classes.

Interoperability between implementations of this part of ISO/IEC 15068 may be achieved through the definition of methods for the first two of these common classes, software_collections and software. The software_collections are the source and target objects for software administration, while the software objects are the objects that are operated on within the context of the software_collections. Operations on individual software_files independent of operations on software objects is undefined.

This part of ISO/IEC 15068 also does not define how remote file systems are managed. In the simplest case, each file system is "local" to a single host, and all installations may be directed to the file system through an agent process on that host. Thus, files on a file system are contained within one of the installed software collections contained below that host. On the other hand, an implementation may also choose to allow installation to a remote file system over a remote file system protocol. That is, the target process is running on a host that is different from the one that contains the file system. In this case, the files on that file system may be contained within the same software collection as before, or may be contained within a local software collection. In another implementation, all software collections may be stored within a global naming service instead of below any particular host.

An implementation may choose to define a software host object, or manage software as part of a more general host object. The attributes of a host object that are of interest to this part of ISO/IEC 15068 are shown in the following table:

The following are the attributes of the hosts that contain software_collections managed by this part of ISO/IEC 15068:

distributions	The list of *distribution.path* attributes for distributions in the software host object. These describe the **PATHNAME** portion of a software_collection source or target.
host	Identifier used to specify the host portion of a software source or target. Identification of a remote host system is dependent on the networking services implementation and thus the syntax and

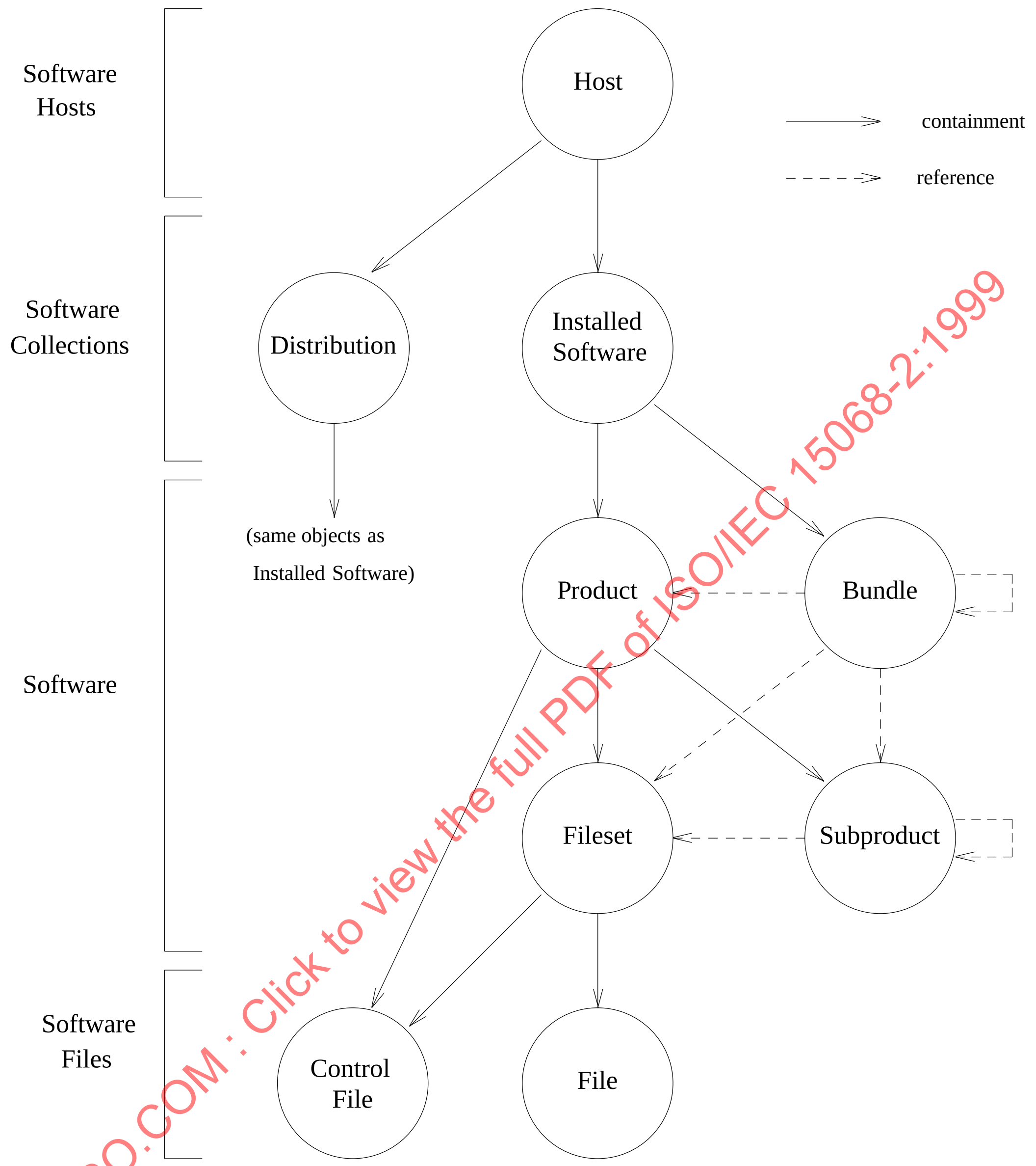

Figure B-3 – Software Object Containment

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

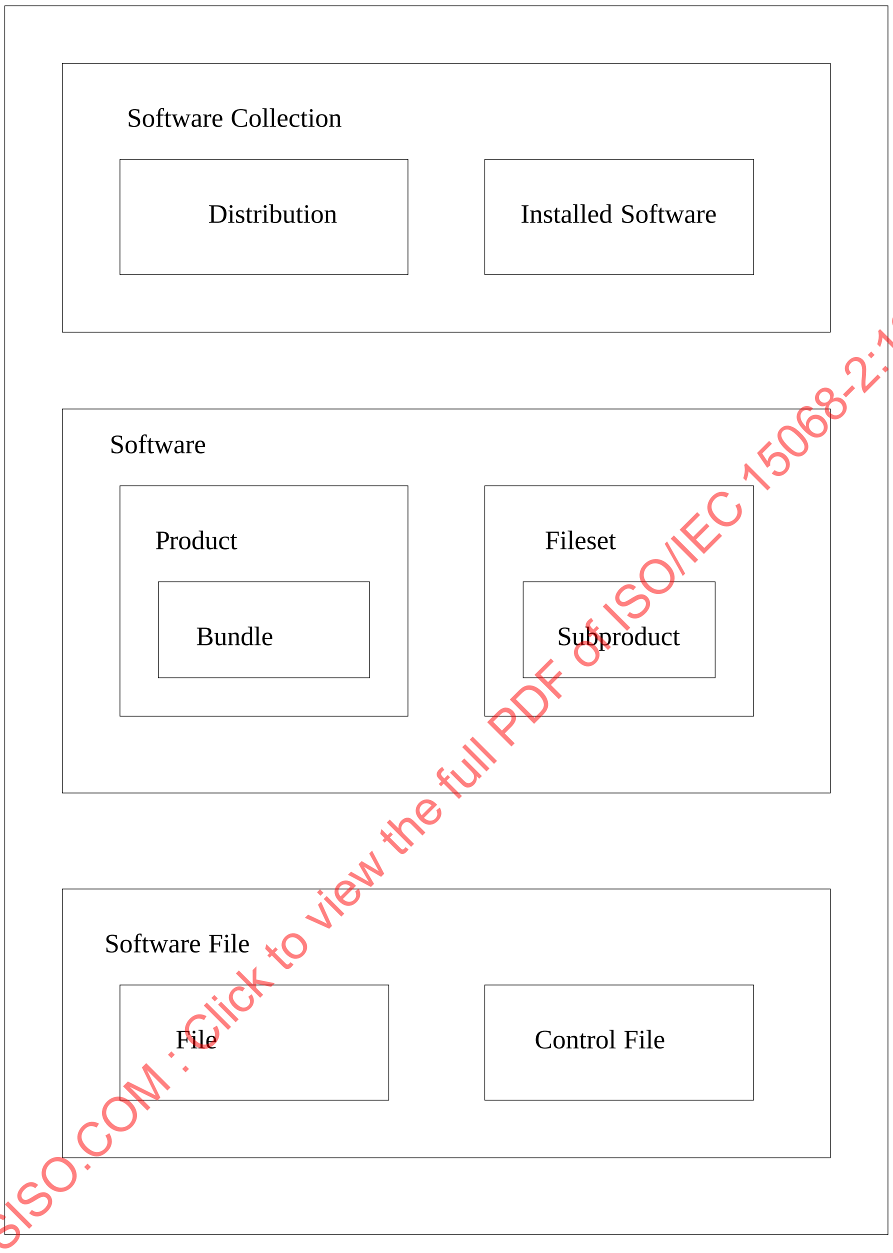

Figure B-4 – Software Object Inheritance

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
host	Undefined	Portable character set	None
os_name	32	Portable character set	None
os_release	32	Portable character set	None
os_version	32	Portable character set	None
machine_type	32	Portable character set	None
distributions	Undefined	List of distribution directories	Empty list
installed_software	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 path that may be contained within a supported archive.

The need for the *media_sequence_number* attribute is to number the tapes (or disks or whatever) if a distribution is on more than one of them. If there is only one, then its number is 1.

The following attributes at one point were listed as distribution attributes. However, it was determined that the only time it could be guaranteed that these attributes were accurate was for an initial distribution definition. As soon as a **swcopy** or **swremove** operation occurred on a distribution, the attributes could be invalid because it would be impossible to modify these attributes in any logical manner based on the operation. It is recognized that these attributes are valuable and many vendors may choose to put them in as vendor extensions.

tag	A short name associated with the distribution, used for selecting the distribution from the command line
title	A longer name used for display purposes.
description	A more detailed description of the contents of the distribution.
revision	A revision associated with the distribution.
media_type	Describes the type of media being used (e.g., CD-ROM, 8 mm, etc.)
copyright	The copyright notice for the distribution.
create_time	The date, in seconds since the Epoch, when the distribution was made.
number	The vendor part number for the distribution.
architecture	A sequence of characters used by a vendor to describe the machine or product. This is presumably more "user friendly" than the values returned by the **uname** utility.

Usually distributions will be created upon creation of the first product with **swpackage** or **swcopy**. Usually distributions will be removed as a part of removing the last product with **swremove**. An implementation may choose to provide more explicit control for creation and deletion of empty distributions. The **swcopy** and **swremove** utilities should be used for this purpose. The **swmodify** utility may also be used.

B.3.3 Media

There is no additional rationale provided for this clause.

B.3.4 Installed_Software

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.

The installed_software catalog may be located by something as simple as a pathname where the catalog is stored as a file, or it could be located in a more complicated fashion such as with a key from a directory service used to identify all or part of a database.

B.3.5 Vendor

The *vendor.tag* attribute is intended to distinguish software objects from different vendors that happen to have the same *product.tag*. A vendor should attempt to choose a *vendor.tag* that is unique among all vendors.

B.3.6 Software

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.

This standard has defined four related software objects — products, filesets, bundles, and subproducts. See Figure B-4. Implementations are encouraged to present these to the user as hierarchy of similar ‘‘software’’ objects, and to actually implement these so that they differ only as needed. That is to say, an implementation should use inheritance from a common class as much as possible. The rationale for the four differently named software objects is as follows:

— Products and filesets are concepts firmly entrenched in existing practice. All of the many practices that have contributed to this standard have included these two levels. Manageable software objects necessarily includes some files to manage. This is the basis of a software product. Additionally, most application software has both required and optional pieces, so often only a subset of the product may be installed. Thus, a fileset is chosen as a ‘‘set of files’’ and a product is a collection of filesets that have a number of shared attributes, and are distributed in a single distribution (usually from a single vendor).

— It was agreed that a ‘‘recursive notational convenience’’ was very desirable. Additionally, many (but not all) existing practices had realized the need for various overlapping groupings of software into new ‘‘configurations.’’ Bundles and subproducts are merely ‘‘macro’’ or ‘‘recursive’’ products and filesets, respectively. Just as products and filesets are a bit different, the use of bundles and subproducts are a bit different. Bundles provide a way to make products out of existing products or parts of products. Subproducts provide a way to provide selectable units that may overlap in fileset contents. For example, a fileset may be part of ‘‘runtime’’ support as well as ‘‘development’’ environment subproducts. Finally, bundles and subproducts are recursive in that they may contain other bundles and subproducts, respectively.

— The containment of filesets and subproducts within products allows for derived naming of components of a product — that is, a simple *tag* for a 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*.

Tying DMTF Verify to **swverify** (and **swlist**) execution on the software object seems closest to the intention of DMTF. DMTF defines the following values:

```
Value       Meaning
-----       --------
0           an error occurred; check status code
1           component does not exist
2           verify not supported
4           component exists, functionality untested
5           component exists, functionality unknown
6           component exists, functionality no good
7           component exists, functionality good
```

The **swverify** command can generate return codes for DMTF value 6 (return code 1) and value 7 (return code 0). The **swlist** command can be used to test for existence, DMTF 1. A lack of **verify** scripts could be related to DMTF 2 or DMTF 7.

B.3.7 Products

The value of the *revision* attribute is interpreted as a **.** (period) separated string, as defined in 3.7 and further in 4.1.4.1. This definition permits the use of such a string, but does not require it. The string can be constructed entirely without the use of periods. An example of the comparison is

```
    A1.003.01 < A.004.00 < B.000.00
    A1_003_01 < A_004_00 < B_000_00
    First < Second < Third
    First < Fourth < Second
```

Historically, some implementations computed the value of *instance_id* sequentially, while other implementations have used an algorithm based on the product *tag*, *vendor_tag*, and the various machine type attributes. No implementation is specified, other than to guarantee that the *tag* and *instance_id* uniquely identify the product within the distribution or installed_software object. This is to make it easier to specify a particular product when there are other products sharing the same *tag* as would be the case when there are different product instances in a distribution for several machine types or multiple concurrent versions on a host.

The *vendor_tag* attribute is intended to be universally unique to distinguish product and bundle software objects that otherwise would be treated as the same object if the *tag*, *revision*, and *architecture* attributes were the same. Guaranteeing universal uniqueness is difficult at best, and no need was seen at present to cause the value of *vendor_tag* to be either some sort of machine-generated universally unique value or officially registered.

Multiple versions of the "same" product or bundle (ones with the same value for the *tag* attribute) is supported by each version possessing values of the version distinguishing attributes unique within that installed software catalog.

The *architecture* attribute should include information related to four *uname*() structure members. The *architecture* attribute is needed for **software_spec**s since the patterns used for determining compatibility in the attributes related to *uname*() can be somewhat complex and contain patterns, while **software_spec**s themselves can contain patterns.

It is recommended that a set of guidelines be used for the architecture attributes to maintain a consistent "syntax" for related architectures. This increases the usability of this field for users selecting software. An example guideline is to order any information contained in the value of the attribute in a consistent way, separated by a consistent delimiter. For example

```
    architecture sunos_4.1_sun4
```

for a product with the attributes

```
    os_name sunos
    os_rev 4.1.*
    os_ver *
    machine_type sun4*
```

Another example is

```
    architecture hp-ux_9_pa-risc
```

for a product with the attributes

```
    os_name hp-ux
    os_rev 9.*|10.*
    os_ver [a..e]
    machine_type 9000/[6..8]???
```

Product machine attributes describe the target systems on which this product may be installed. Each of these keywords are related to a POSIX.1 {2} *uname*() member and may be defined as a simple string, or a software pattern matching notation. How compatible software is determined depends on whether the products are being installed on the system that will be using them, or whether the installation will be used by other systems with perhaps different attributes.

If a uname attribute is undefined, the behavior is essentially the same as if it were defined to be * (meaning compatible with all systems).

The product directory for an application should be the directory that is part of all paths in the product. Thus, if an application has three filesets that contain files below `/appl/console`, `/appl/agent`, and `/appl/data` respectively, the *product.directory* attribute should be set to `/appl`. If a user relocates the product with a command like

```
    swinstall appl,r=1.0,l=/disk2/appl
```

then all three filesets have the same location attribute. If the user relocates the product to three different locations

```
    swinstall appl.console,r=1.0,l=/disk1/appl
    swinstall appl.agent,r=1.0,l=/disk2/appl
    swinstall appl.data,r=1.0,l=/disk3/appl
```

then each fileset will have a different location attribute. There will be three product instances containing the three filesets (since products versions are distinguished by location), but the user can still identify all three filesets as one with the specification

```
    swverify appl,r=1.0,l=*
```

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
    swinstall appl.agent,r=1.0,l=/disk2/appl,q=current
    swinstall appl.data,r=1.0,l=/disk3/appl,q=current
```

And subsequently identify all pieces with

```
    swverify appl,q=current
```

The *postkernel* attribute supports the ability to install one operating system in proxy (to an alternate root) by another implementation that does not understand that operating system. All products that contain kernel filesets that will be installed into the same installed_software object should have the same path defined. There should be one core OS kernel fileset that includes this path in its set of files so that it has been installed by the time the *postkernel* script is executed.

In general, a product with no `preinstall` or `postinstall` scripts is recoverable. However, if there are `preinstall` or `postinstall` scripts, then `unpreinstall` and `unpostinstall` scripts shall be provided if any steps need to be undone to support autorecovery.

There was an issue whether dependencies should be an attribute of a product. The following types of dependencies have been discussed:

— Fileset to fileset within a product

— Fileset to (some other) product

— Fileset in one product to fileset in another product

— Product to product

— Product to fileset in some other product

— Product to fileset in that product (essentially mandatory fileset)

The last three dependency types are not necessary if the first three types exist (which they do), since those dependencies can be specified in terms of the others. For example, if an entire product depends on a second product, then the second product can be defined as a dependency for all filesets in the first product.

The developers of this part of ISO/IEC 15068 recognized that numerous additional dependency requirements are possible, particularly for software updates. These may be handled via `checkinstall` scripts, and can be considered for future revisions of this part of ISO/IEC 15068.

The intention behind the inclusion of the *layout_version* attribute within a product is that it be required if its value is different than that for its associated software_collection.

B.3.8 Bundles

Bundles serve two purposes — they allow the software supplier to group different subsets of products into new configurations or products, and they allow the software administrator to build useful groups of software (configurations) from already defined bundles and products.

The bundle class does not have *location* or *directory* attributes. This is because `software_spec`s 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_spec`s in the contents will result in the same `software_spec`s 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.

B.3.9 Filesets

The *media_sequence_number* is used for serial distributions to describe which media the archive containing the fileset starts on. There is generally one archive per media, unless a fileset is larger than a media. Each media has a unique sequence number whether it begins an archive or continues a previous one.

At one point a fileset *class* attribute existed that could contain the value of recommended, mandatory, or optional. The attribute was removed because it was felt that this part of ISO/IEC 15068 could not specify any behavior for the attribute. It would be possible to make a specific fileset mandatory by having all other filesets in that product specify it as either a prerequisite or corequisite.

Another way to handle recommended, mandatory, or optional filesets would be to create subproducts with *tag*s of the appropriate names. Although this part of ISO/IEC 15068 does not specify any behavior based on the name of *subproduct.tag*, a specific implementation could define behavior as an extension.

B.3.10 Subproducts

Unlike bundles, subproduct definitions (that are internal to a product) are copied or installed when any fileset specified in the *contents* attribute of the subproduct is copied or installed. Products are meant to be sets of related software and are usually created and managed by one person or organization. Additionally, subproducts are normally used to specify useful subsets of filesets within the product, which in turn are useful for dependencies. With subproducts, the "parts make up the whole."

B.3.11 Software_Files

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.

The *compression_type* attribute allows compressing and uncompressing of individual files during `swcopy`, and uncompressing during `swinstall`. The way in which an implementation uses this attribute is undefined, although the general thought was that this would normally be the name of a compression/uncompression routine with a simple interface. An implementation should be flexible in locating routines specified by compression_type, utilizing any or all of the following:

— Built-in knowledge of the *compression_type* format for compressing and uncompressing

— The product control directory for a program named in *compression_type*

— **PATH** on the target system for a program named in *compression_type*

No particular compression method is specified in the standard largely because the developers of this part of ISO/IEC 15068 saw no standard for file compression and 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 the attributes *FAT_Hidden* and *FAT_Readonly*.

There was some debate whether the *major* and *minor* attributes are appropriate or not since there is no standard that specifies how these files are created. In addition, this part of ISO/IEC 15068 specifies that the serial distribution be in POSIX.1 {2} **cpio** or **tar** format; however these attributes are biased towards **tar** format as opposed to **cpio** format.

Other SVR4 supported file types are **f**, **d**, **l**, **s** (like SDU RF, DR, HL, SL); **b**, **c**, **p** (special files and pipes); and **e**, **v**, **x** (editable and volatile files; and exclusive directories).

Considered was a size file type (**z**) that was removed in favor of the **space** control_file similar to SVR4. An implementation may choose to internally implement a size type or a separate size_file object to represent the data from this file.

The developers of this part of ISO/IEC 15068 considered an *is_exclusive* (directory) attribute that was removed due to objections that the utilities would remove files that they did not have recorded in their database. Also, this was not a common need, and can be implemented either as vendor extension or by having software **fix** script implement similar functionality.

There has been much discussion about compression being handled within the scope of this part of ISO/IEC 15068. Currently there are ways that both implementations and individual software products can handle compression. Compression can be handled through cooperation of the source and target roles, if they are from the same implementation. Software vendors can choose to ship their files compressed and uncompress them as part of the **postinstall** script. They can add a **space** control_file to account for the extra space required.

A similar need would apply to other post processing, such as for ANDF files that are processed as part of postinstall or configuration.

Though it may be adequate for protecting against accidental damage, the existing POSIX.2 {3} **cksum** is considered inadequate for virus protection. Implementations may wish to create additional vendor-defined attributes and utility behaviors for this purpose.

Each of the prerequisite or corequisite **dependency_spec**s in the list is required to resolve successfully in order for dependencies to be met. Also, a **dependency_spec** can contain alternate **software_spec**s separated by the **|** (vertical line) character (see 4.1.4.1). So, if a fileset has a corequisite dependency on software, expressed with a Boolean equation **(A|B|C)&(D|E)**, this can be specified in a PSF as

```
corequisite A|B|C
corequisite D|E
```

There are files (particularly for OS software such as **/etc/rc** for SVR4 and **autoexec.bat** for DOS) that are modified between software update times. These may be termed modifiable files. Although OS modifiable files are slowly being replaced by mechanisms where applications can simply add their own requirements as separate read-only files in a particular directory, there currently 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.

B.3.13 Control Files

Using *tag*s 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 *tag*s. A concern on PC or DOS systems is that requiring more than one control script for all *tag*s 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.

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.

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 software can be combined with the install software task.

Reinstall software (**swinstall**)

This task is simply installing the exact same software that was previously already installed.

Configure software (**swconfig**)

This task takes place on the client role that will be using the installed software. Configuration makes that software ready to use. Configured software can also be reconfigured as required or can be unconfigured (to deactivate a particular version or prepare it for removal).

Update software (**swinstall**)

This task updates the target file system by installing a newer revision of software than is already installed. This is also referred to as upgrading.

The new revision of software can be installed in the same location as the current revision. In this case, the software **configure** scripts executed by the configure task need to handle saving or updating the necessary configuration data.

The new revision of software can alternatively be installed in a location different than the current revision. In this case, the old revision may be unconfigured by the **unconfigure** script executed as part of the unconfigure task, and the new revision is configured by the **configure** scripts executed as part of the configure task.

Downdate software (**swinstall**)

This task "downdates" the target file system by installing an older revision of software than is already installed. This is also referred to as "downgrading" or "reverting."

The older revision of software can be installed in the same location as the current revision. In this case, the configuration process of the older version handles the necessary changes in configuration.

The older revision of software can alternatively be installed in a location different from that of the current revision. In this case, the new revision can be unconfigured via the unconfigure task, and the older revision can be configured either independently, or as part of install.

Recover software (**swinstall**)

This task restores the previous version of software (if it exists) in the case where an update, downdate, or reinstall of software fails. This part of ISO/IEC 15068 defines the minimum required support for automatic recovery process in the install task.

Apply software patch (**swinstall**)

This task replaces part of a software fileset with a new set of files by installing a fileset with those new files in the same location as the fileset being patched. This is also referred to as fixing software.

This part of ISO/IEC 15068 currently does not provide any special functionality for patching software. Filesets related through naming conventions and prerequisites can be used. Backup of patched files can be achieved via install control scripts.

Consensus was not reached as to how patches should be managed and what level of functionality is required. Issues include whether patches are cumulative or complete, whether they involve partial products or filesets (or even files), how volatile files should be managed, naming or versioning schemes, and the level of rollback support required (number of patch removes that are supported for cumulative patches). It was concluded that this could not be standardized at this point, but that this part of ISO/IEC 15068 does provide a sufficient base for implementing patch functionality.

Remove installed software (**swremove**)

This task removes software from an installed_software object where it previously was installed. Parts of software products (subproducts or filesets) can be removed at different times.

If the system where the software is installed was also using the software, unconfiguring the software can be combined with the remove software task.

Remove software patch (**swremove**)

This task removes a patch fileset. This is also referred to as rejecting software.

This part of ISO/IEC 15068 does not provide any special functionality for patching software. Filesets related through naming conventions and prerequisites can be used. Restoring patch files when removing the patched can be achieved via remove control scripts.

Verify the installed software (**swverify**)

This task checks that software previously installed still exists and is intact. If operating on a system that was configured to use the software, it can also check that the software is configured properly.

List installed software information (**swlist**)

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 the amount of information provided.

Fix installed software information (**swmodify**, **swverify**)
: This task modifies information about software that has been installed on a target. Options are available to specify which software packages, and what information about those packages, are modified.

Package software (**swpackage**)
: This task takes place in the packager role and transforms developed software into the software packaging layout suitable for distribution. The metadata that defines the software objects to be packaged is contained in the product specification file (PSF).

Copy distribution software (**swcopy**)
: This task copies distribution software between a source and a target, for subsequent use of that target as a source. Copying software can be used to merge distributions, to distribute products to the installation targets, and then install from that local copy, or to copy part of a distribution to a removable media for physical distribution (as opposed to electronic distribution).

Remove distribution software (**swremove**)
: This task removes products from a target distribution.

Check/verify distribution software (**swverify**)
: This task checks that a target distribution exists and is intact.

List distribution information (**swlist**)
: This task lists source or target distribution information. Options are available to specify which objects in a distribution are to be listed, and to control the amount of information provided.

Fix distribution information (**swmodify**, **swverify**)
: This task modifies information that describes, and is contained within, target distributions. Options are available to specify which objects in a distribution are modified.

License installed software (undefined)
: How software licenses are managed is undefined within this part of ISO/IEC 15068.

The task definitions were based on study of existing practice for software administration. This included presentations on existing practice by many different system vendors and system administrators. From these, a functionally adequate base was selected upon which all parties could build. While it was recognized that this did not address every concern, it was felt that that the utility descriptions (including detailed behavior), software structure definitions, and media layout, provided 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.

Table B-3 below compares and contrasts the SVR4, HP-UX, SCO Unix, and AIX software management utilities. While this table is neither complete nor current, 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 (customize, etc.)	customize scripts and check scripts are not interactive	each package may have a configuration script	scripts called by installp may not interactive
check installation	pkgchk	not applicable	yes	lppchk
display software package info	pkginfo	swlist	custom+	lslpp
make a package	pkgmk	swpackage	distmaster	yes
generate preliminary input to packager	pkgproto	not applicable (swpackage may do some of this)	distmaster	not applicable
remove a package	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 a reboot (the clients of the software need to be rebooted). An alternate root containing operating system software can be thought of as a root to which one could *chroot*(). From a usability standpoint, it is important that alternate roots are understood to be different than relocating a software product, or specifying an alternate catalog for the same root.

Related to the `-s` option, an implementation could define an additional source syntax to use well-known sources whose existence is available through some sort of directory service.

The `-s` option could be extended to supported multiple source specifications. There are several possible ways to interpret multiple sources, including searching sources sequentially, ignoring all specifications after the first one, using the last specification, or choosing the ‘‘best’’ source based on criteria such as performance or ability to reduce network load. It may even be desirable for multiple source specifications to be interpreted differently for different commands.

An implementation may implement the `-p` option (preview) by simply executing the command through the analysis phase. Alternatively, an implementation may emulate the execution phase, listing the operations that would occur, including listing control scripts that would be run, but not actually performing those operations. As preview is undefined, other alternatives are possible.

B.4.1.3.1 Non-interactive Operation

It is recognized that there may need to be some sort of interaction with the user in order to handle multiple volumes (e.g., tapes) for sources and targets.

B.4.1.4 Operands

Discussion pointed to the fact that the `@` character does not have any applicable precedence as a separator of operands. It was concluded that the use of `@` in mail addresses and BSD commands is a bit different. Another point was that having two lists of operands was not desirable in any case.

On the other hand, the two types of operands are the two key objects upon which the utilities operate. The syntax is valid according to the utility guidelines from 2.10.2 of POSIX.2 {3}. Distributed utilities extend the problem space that POSIX.2 {3} has already addressed, thus the need for precedence might be less. Thus, it was decided that the `@` was acceptable, and perhaps desirable over the alternatives.

One alternative was to move one or both operands to options (such as `-S` for software and `-T` for targets). But, it was felt that this was not necessary because there are already `-f` and `-t` options for files containing lists of operands. Another point was that listing target operands on the command line was not critical in any case, as an administrator of many systems would not use either the @ *targets* or `-T` *target* syntax.

B.4.1.4.1 Software Specification and Logic

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:

```
software_spec   : bundle_tags [ product_tags ] [ version ]
                | product_tags [ version ]
                | '*' [ version ]
                ;
bundle_tags     : bundle { '.' bundle }
                ;
product_tags    : product
                  [ '.' subproduct { '.' subproduct } ]
                  [ '.' fileset ]
                ;
version         : { ',r' rel_op revision }
                  [ ',a=' architecture ]
                  [ ',v=' vendor_tag ]
                  [ ',l=' location ]
                  [ ',q=' qualifier ]
                | ',*'
                ;
rel_op          : '==' | '!=' | '>=' | '<=' | '<' | '>'
                ;
```

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_spec**s are

```
*
Networks
Networks.X11
Networks.X11.Runtime,a=*80?86*
X11
X11.Runtime
X11.Runtime,r=4,v=CloneInc
X11.Runtime,r>=4.0,r<5.0
X11,r=4.03.07,l=/usr/X11R4
X11,r=5.00,l=/usr/X11R5,q=latest
X11,*
*,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 having this part of ISO/IEC 15068 limit the flexibility slightly in this area was deemed as acceptable.

This part of ISO/IEC 15068 permits the use of values other than those defined in 4.1.4.1 for **ver_id**. This part of ISO/IEC 15068 also permits the use of **ver_id** in conjunction with attributes and objects other than the first listed in a **software_spec**. This allows additional flexibility for identifying software objects.

A possible syntax for these vendor extensions include, but are not limited to

ver_id	attribute	object
br	revision	bundle
ba	architecture	bundle
bv	vendor_tag	bundle
bl	location	bundle
pr	revision	product
pa	architecture	product
pv	vendor_tag	product
pl	location	product
fr	revision	fileset
fl	location	fileset

The **ver_id fr** is seen as most useful since it can identify a particular fileset object within a product where the product may not have a revision, but the fileset does. Note however, that any object can still be identified with only the attributes defined in this part of ISO/IEC 15068. For example, if a bundle includes two partial products with the same *tag* value but different revisions or locations, these partial products could be identified with the standard syntax by excluding the bundle portion of the **software_spec**. For example, **bundle.product,pr=1.3** could also be identified by a **software_spec** of **product,r=1.3**.

In another example, the fileset that could be identified by its revision (**product.fileset,fr=1.3**) could also be identified by a **software_spec** (e.g., **product,r=revision**), where revision refers to the product revision, including possibly the empty string.

Relocation occurs by replacing the *product.directory* part of each file path as it occurs in the distribution, with the *location* specified and using the resulting path for installation. This is still relative to the installed_software directory described below. See 4.5.7.3.1 for more information.

Using a **sw_pattern** in a **software_spec** is a way for the user to indicate that all software objects that match the **software_spec** are to be included. For example, applying the **software_spec "*"** to **swcopy** means to copy all software in the distribution. Applying the **software_spec "Foo,*"** to **swremove** means to remove all versions of Foo.

The behavior for **swlist** is different (by default including all software if none is specified) because this is the command that is used to find all versions of software, and because listing cannot negatively affect the state of the software_collection.

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.

This specification provides the means to select products and specify dependencies using a single syntax. The use of the shell-type pattern match specified in 3.13 of POSIX.2 {3} allows for reasonable specification of sets of values that share such patterns. Thus, for example, a specification of `"a=HP-UX*"` may be used to select packages for any of a set of architectures. The specification using the relational operators provides support for testing the type of release/version specifications that are frequently used by vendors. In particular, it provides support for testing when a numeric test is needed (e.g., comparing 2.9 to 2.10 as version levels of a product). Additional operators such as `>>` were considered. The specification of the `>>` operator allows the user to specify the selection of the most recent (highest version number) of a set of otherwise identical packages. This exposes to the interface the mechanism used by `swinstall` to select such a package.

The range of attributes that may be specified allows for selection of packages that may be needed to support code serving to alternate architectures, or other operating environments. In addition, it provides the needed support to specify installed software that may only be distinguished by the location of installation.

Examples of fully qualified `software_spec`s are

```
    Foo,r=3.0,a=,v=XT
    BundleA.Foo,r=1.0,a=,v=XT
    Dow.Bar,r=2.0,a=SunOS,v=,l=/opt/foo.2
```

It is possible for bundles to contain `software_spec`s that are not fully qualified. This is not recommended for bundle definitions provided by software vendors because the results of operations on this bundle may be undesirable for an administrator. However, there is some flexibility provided by ambiguous software specs that administrators may want to use.

For example, a bundle with contents `"*.Man"` could be used to manipulate all `"Man"` filesets or subproducts in all products.

If a vendor includes any wildcards in a `software_spec` in a bundle definition, then the *vendor_tag* attribute should be included and its value should have no wildcards, thus limiting the scope of the pattern matching.

The difference between `"FOO,v="` and `"FOO"` is that the first will only match a product or bundle `"FOO"` where vendor is not defined, while the second will match a product or bundle `"FOO"` with any vendor definition.

B.4.1.4.2 Source and Target Specification and Logic

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_collection_spec` syntax:

```
software_collection_spec   : [ host ] [ ':' ] [ path ]
```

Examples of distribution `software_collection_spec`s are

```
/var/spool/sw
hostA
hostA.cloneinc.com
```

```
hostA:/var/spool/sw
15.1.94.296
15.1.94.296:/depots/applications
```

Examples of installed_software **software_collection_spec**s are

```
/
hostA
hostA.cloneinc.com
hostA:/
15.1.94.296
15.1.94.296:/exports/applications
```

Target distributions in the serial format need not be supported for **swverify**, **swremove**, and **swmodify** as this requires the implementation to unload the entire distribution, merge in the changes, then reload it. The user can accomplish this (and an implementation can implement this) by first copying the distribution into a directory format, implementing the changes, then copying the distribution back to the serial media. This operation also could require significant temporary disk space.

A similar rationale applies to **swcopy**, and **swpackage**, which by default, overwrite the existing distribution instead of merging in the specified software.

B.4.1.5 External Influences

B.4.1.5.1 Defaults and Options Files

For SVR4 or similar file system layout, the defaults file may be located in **/var/adm/sw/defaults**. The use of this location is strongly encouraged.

The difference between "system-level" defaults and "site-level" defaults was discussed.

The former is provided by the implementation of the utilities and the latter is constructed by the administrator. The intent here is for the implementation to respect any customizations to the system level defaults file, so it can be used for site policies. It is recommended that implementations "hard code" the defaults as opposed to relying on the system file containing all definitions, and provide a means to support new options in future releases without changing the site specific values in the system defaults file.

B.4.1.5.2 Extended Options

For SVR4 or similar file system layout, *distribution_source_directory* may be set to **/var/spool/sw**. The use of this location is encouraged.

For SVR4 or similar file system layout, *distribution_target_directory* may be set to **/var/spool/sw**. The use of this location is encouraged.

For SVR4 or similar file system layout, *installed_software_catalog* may be set to **/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 catalog 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.

The location of the storage for the catalog itself is implementation defined.

The **swinstall** utility, and other utilities that operate on installed_software, modify the catalog information based on the outcome of the utility. Information contained within the catalog is resolved in the context of each target.

Originally, it was thought that a catalog would be kept as a flat file in a directory that could be specified using this option. In the interest of generality, so that implementors might be allowed to use databases, the *catalog* attribute is now described as a key. This allows an implementor to either use a flat file or a database or some other form of persistent storage for the information, yet still be able to separate the address space as desired. The motivation for permitting the separate address space stems from the following two cases. First, it seems desirable to allow ordinary (non-root) users to be able to use **swinstall** to store software in their own private space. Likely the only real restriction is a potential lack of write authority to the central storage for the catalog, hence the ability to create a separate catalog. This also allows a user to manage personal software with utilities such as **swremove** or other utilities. Second, installations may wish to deploy stable versions of their software in the normal location, and a test version installed in a second location where access may be more tightly controlled. There may even be other versions installed that are under development. Since this software may have identical attributes, it is desirable to allow such separate space for management. Both of these examples show the need for separate domains of software management.

Two values of *autoselect_dependencies* (*autoselect_dependencies*=*true* and *autoselect_dependencies*=*as_needed*) support different possible policies by the user. Having *autoselect_dependencies*=*true* ensures that all targets are kept in sync, while having *autoselect_dependencies*=*as_needed* prevents the possibility of updating dependency software to a higher revision unnecessarily.

Autoselection of a dependency across products is possible if a compatible product version with the highest revision that meets the dependency is unique. In other words, the same rules apply for dependency selection as for normal selection as described in 4.1.4.1.

For the *ask* option, the **checkinstall** and **configure** scripts are required to detect needed **response** files when they are necessary, and return with the appropriate warning or error.

For the *installed_software_catalog* option, the catalog and the directory together form a key to identify one installed_software object. For example, this would allow the files on the file system to be split up into different management domains. For example, OS software, networking software, and application software could be in three different logical installed_software objects, although they are all installed under the root file system.

B.4.1.5.3 Extended Options Syntax

In the interest of having a single common extended option syntax for all the POSIX system administration standards, the following syntax was agreed upon. As of this writing, the syntax is a superset of that used by this part of ISO/IEC 15068, IEEE P1387.3, and IEEE P1387.4.

```
%token          FILENAME_CHARACTER_STRING /* as defined in 2.2.2.37  */
%token          NEWLINE_STRING            /* as defined in 2.2.2.61  */
%token          PORTABLE_CHARACTER_STRING /* as defined in 2.2.2.68  */
%token          SHELL_TOKEN_STRING        /* as defined in 2.2.2.80  */
%token          WHITE_SPACE_STRING        /* as defined in 2.2.2.110 */

%start  sysadmin_option
%%

sysadmin_option          : qualifier option operator_value
                         ;

qualifier                : compulsory_qualifier command_qualifier
                         ;

compulsory_qualifier     : /* empty */
                         | '-' | '='
                         ;

command_qualifier        : /* empty */
                         | command '.'
                         ;

option                   : keyword op_ws
                         ;

operator_value           : '=='
                         | value_qualifier '=' value
                         ;

value_qualifier          : /* empty */
                         | '+' | '-'
                         ;

value                    : op_ws value ws single_value
                         | op_ws single_value
                         ;

single_value             : value_structure
                         | SHELL_TOKEN_STRING
                         ;

value_structure          : '{'  op_ws value_list op_ws '}'
                         ;

value_list               : /* empty */
                         | value_list ws single_value
                         | single_value
                         ;

command                  : FILENAME_CHARACTER_STRING
                         ;

keyword                  : SHELL_TOKEN_STRING
                         ;

op_ws                    : /* empty */
                         | ws
                         ;
```

```
ws                        : WHITE_SPACE_STRING
                          ;

%start command_line_options
%%
command_line_options      : command_line_options ws sysadmin_option
                          | sysadmin_option
                          ;

%start options_file
%%
options_file              : options_file NEWLINE_STRING option_file_line
                          | option_file_line
                          ;

option_file_line          : op_ws op_comment
                          | op_ws sysadmin_option op_ws op_comment
                          ;

op_comment                : /* empty */
                          | '#' PORTABLE_CHARACTER_STRING
                          ;
```

(1) A - (hyphen) qualifier indicates a compulsory behavior while = (equal) indicates a non-compulsory behavior.

(2) For options that support multiple values, values can be added to the existing list of values by using the += (plus equal) operator. Similarly, values can be removed by using the -= (hyphen equal) operator. Any option can be set to the default value by using the == (equal equal) operator and value combination.

(3) A shell token can be an unquoted or quoted string according to the rules of token recognition rules described in 3.3 of POSIX.2 {3}. For example, it can use single or double quotes and can contain like quotes if escaped with backslash. It can also support the same level of internationalization as the POSIX shell.

(4) The multiple value convention is consistent with white space separating tokens in commands (operands) and allows commas to be used in the `single_value`. This also allows multiple values to be specified without using quotes (although quotes are still needed for multiple values on the command line).

(5) If the extended option specification contains any white space at all, then the entire specification shall be quoted if used on a command line. This is because the `-x` option, which conforms to POSIX.2 {3}, requires exactly one value that is then processed using the above syntax.

(6) When specified on the command line, multiple option specifications can be included after a single `-x` option if included in quotes and separated by spaces. Multiple `-x` options may also be used.

(7) For option and defaults files, blank lines and all comment text [any sequence of characters beginning with an unescaped # (pound) and continuing through the end of that line] are ignored according to the shell token recognition rules as described in 3.3 of POSIX.2 {3}.

B.4.1.5.3.1 Precedence for Option Specification

The first rule defines typical precedence of system defaults, then a user defined set of defaults, then per task exceptions or specifications. The second rule supports normal use models of defining multiple "sets" of target_selections and software_selections, and being able to operate on the union of those sets. Also, the `-f` and `-t` options are simply another form for specifying operands, and are at the same level of precedence, and are thus combined with other selections. The third rule is generally an error, and the behavior is undefined (i.e., it may be an error, or an implementation may chose to implement last- or first-wins). For example on HP-UX:

```
$ cc -O -g x.c
$ cc: warning 414: Debug and Optimization are mutually exclusive.
    -g option ignored.

$ cc -g -O x.c
$ cc: warning 414: Debug and Optimization are mutually exclusive.
    -O option ignored.
```

It might be convenient to have a mechanism to allow the system administrator to define a default in the system defaults file that cannot be overridden by a user. Such a function may be supplied by an implementation as an extension. This may also be considered as part of a future revision to this part of ISO/IEC 15068.

B.4.1.5.4 Standard Input

There is no additional rationale provided for this subclause.

B.4.1.5.5 Input Files

There is no additional rationale provided for this subclause.

B.4.1.5.6 Access and Concurrency Control

For example, if the *installed_software_catalog* is referenced by a path on the file system, then the user can create a catalog in their own user work space (creating their own installed_software object), and install and manage software in that installed_software object.

If the catalog is stored in a file, then a corresponding ability to create an installed_software object (and thus, a catalog) is needed.

Access control includes such things as requiring particular authority to operate on particular software or software_collections. Concurrency control is the prevention of more than one writer at a time to the catalog or data areas. Restrictions to prevent multiple concurrent writers were originally part of the draft, but later determined to be excessively restrictive. It is conceivable that more than one writer could safely be active at a time if the work involves no common files. Failure to allow multiple concurrent readers of the catalog, or other data files, is strongly discouraged.

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 sofware file objects)

— Any additional access control on the software objects, including access to (possibly remote) software collections

For access control to the files themselves, this part of ISO/IEC 15068 defaults to the file permissions defined by POSIX.1 {2}. File attributes are defined for the files, hence the implementation will set the POSIX.1 {2} file permissions based on those values. Deviations from this model are permitted only for implementations running on file systems that are not POSIX.1 conformant, and then only as long as the implementation documents the resulting behavior.

Any additional access control to the software objects defined in this part of ISO/IEC 15068 (e.g., permission to install specific software into specific software collections on specific hosts), is undefined. An implementation may choose to have no access control. For example, anyone may install any software to any system as long as the previous POSIX.1 {2} file permissions are satisfied. An implementation may also choose to provide both authorization and authentication for access to all software objects and hosts, as well as a distributed interface for managing the access control lists.

Like the definition of the model to implement distributed aspects of this standard, access control beyond that required by the underlying operating system is undefined. It was determined that both of these rely on technologies that have not been formally standardized, and may better be addressed in other forums.

B.4.1.6 External Effects

B.4.1.6.1 Control Script Execution and Environment

The provision for interpreters other than `sh` was requested by users among the developers of this part of ISO/IEC 15068, as well as producers representing systems that might lack a POSIX.2 {3} or even POSIX.1 {2} operating system. By making this provision, many felt that a greater degree of acceptance and usefulness could be gained.

The restrictions placed on the option syntax are such that each of the options can be easily parsed and hence set by a control script by simply sourcing the file. The term "sourcing" as used here implies the use of the "." command in the POSIX.2 {3} shell. For example, the following are formats for the SW_SESSION_OPTIONS file that can be sourced:

```
loglevel=1
enforce_dependencies=false
software="A B C"

loglevel=1
enforce_dependencies=false
software="
A
B
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 stdout generated.

An implementation may have an implementation-defined user controllable behavior that invokes error handling procedures in the case of warnings returned from script execution.

The purpose of the environment variables is to pass vital information to the scripts so that they may operate appropriately under different circumstances. For example, they may want to take very different actions when **SW_ROOT_DIRECTORY** is some value other than `/`. It has also been discussed that there may need to be some way to pass other information to these scripts such as option values specified in the defaults and options file that control policy. This can be achieved with the **SW_SESSION_OPTIONS** variable, which points to a file containing all the options passed to the command, including options, selections, and targets.

One reason that this part of ISO/IEC 15068 differentiates install and remove scripts from **`configure`** scripts is to separate installing software from configuring software for actual use. This supports installing software to alternate root directories on servers for use by clients that configure that software.

The developers of this part of ISO/IEC 15068 also discussed, but did not include, the use of several of these variables for setting the value of specific utility options when the utilities are called from control scripts. These variables are as follows:

SW_ROOT_DIRECTORY
could be used to specify the *directory* portion of all *target* operands

SW_LOCATION
could be used to specify *product.location* portion of all *software_-selection* operands

SW_CATALOG
could be used to specify the value of the *installed_software_catalog* option

The scripts need to be aware of the environment under which they are operating. The environments that these scripts run under are as follows:

— All scripts

Each script shall be passed its script *tag*, the root directory to which installing, the product directory where the product is located, and the control directory where the script is being executed from, as the environment variables **SW_CONTROL_TAG**, **SW_ROOT_DIRECTORY**, **SW_LOCATION**, and **SW_CONTROL_DIRECTORY**, respectively.

— **`preinstall`**, **`postinstall`**, **`preremove`**, **`postremove`**, **`unpreinstall`**, **`unpostinstall`**

The **`install`** and **`remove`** scripts are run when loading or removing the software, or when recovering from a failed install. These may be executed by **`swinstall`** or **`swremove`** running on a host with a different architecture from the software. So, only the set of POSIX.2 {3} utilities are guaranteed to be available on the server. Since the architecture of the file server is not necessarily known, the path to these commands is passed to the scripts via the environment variable **SW_PATH**.

Additionally, these scripts need to know the alternate root directory so that operations are within the context of that root, not the root of the file server. (This directory is supplied to the scripts via an environment variable **SW_ROOT_DIRECTORY**.) It is critical that **SW_ROOT_DIRECTORY** is honored by these scripts.

— `checkinstall`, `checkremove`, `verify`

It is expected, but not assumed, that these scripts mostly check the state of a system that will actually run the software. For the install check (`checkinstall`) script, again only a minimum set of commands is guaranteed to be available. This is because all check scripts are executed before any new software is installed. For the remove check and verify scripts (`checkremove` and `verify`), files from this software and its prerequisites are guaranteed to be available.

Because of alternate root installation, these scripts also need to be aware of the **SW_ROOT_DIRECTORY**.

— `configure`, `unconfigure`

These scripts configure the system for the software. Therefore they can run architecture specific commands, including files that are part of the software that defines the scripts. They are only run within the context of the system that will actually use the software. The **SW_ROOT_DIRECTORY** will always be `/`.

B.4.1.6.1.1 Control Script Behavior

Control scripts allow vendors to perform tasks and operations, in addition to those that the tasks perform. The `swinstall`, `swverify`, and `swremove` utilities may each execute one or more vendor-supplied scripts. The presence of these scripts in the distribution is optional. Vendors of software to be installed need only provide those scripts that meet a particular need of the software. The following summarizes the standard scripts:

`request` (Request script)

This is the only script that may be interactive. This script may be run by `swask`, `swinstall`, or `swconfig` after selection, and before the "analysis" phase in order to request information from the administrator that will be needed for the `configure` script when that script is run later.

This script is executed on the manager role and it is the responsibility of the script to write all information into the `response` file in the directory where the script is being executed (the **SW_CONTROL_DIRECTORY**). The utilities will then copy this file to the **SW_CONTROL_DIRECTORY** on the target role where the `configure` and other scripts are executed from. This `response` file may be used by any other scripts, but particularly the `configure` script.

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:

```
echo "Enter path to locate the database"
read $path
echo db_path=$path >> $SW_CONTROL_DIR/response
```

And the using **configure** script:

```
$db_path=$default_db_path
if [ -f $SW_CONTROL_DIR/response ]
then
    . $SW_CONTROL_DIR/response
fi
create_db ( $db_path )
```

response (Response file)

The **response** file is generated by the **request** script and located in the same directory as that and the other scripts. The format and content of the **response** file is implementation defined. For example, it may be a list of environment variable definitions so that the consumer script can just source this file.

checkinstall (Install check script)

This script is run by **swinstall** during the "analysis" phase in an attempt to ensure that the installation (and configuration) succeeds. For example, the OS run state, running processes, or other prerequisite conditions beyond dependencies may be checked. Running this script shall be free of side-effects, e.g., processes may not be killed, files may not be moved or removed, etc.

preinstall (Install preload script)

This script is run by **swinstall** prior to loading the software files. For example, this script may remove obsolete files, or move an existing file aside during an update.

This script and the next script are part of the "load" phase of the software installation process. Within each product, all **preinstall** scripts are run (order is dictated by any prerequisites), all filesets are loaded, then all **postinstall** scripts are run.

postinstall (Install postload script)

This script is run by **swinstall** after loading the software files. For example, this script may move a default file into place.

unpreinstall (Recovery preload script)

This script is run by **swinstall** after restoring the software files when recovering filesets. For example, this script may undo the actions of the **preinstall** script.

unpostinstall (Recovery postload script)

This script is run by **swinstall** before restoring the software files if the **postinstall** script has been run. It can be used to undo the actions of the **postinstall** script.

verify (Verify script)

This script is run by the **swverify** command any time after the software has been installed or configured. Like other scripts it is intended to verify anything that the commands do not verify by default. For example, this script may check that the software is configured properly and has a proper license to use it.

fix (Fix script) This script is run by the **swverify** command when the **-F** option is used. Its purpose is to correct any problems reported by the **verify** script.

checkremove (Remove check script)

The remove check script is run by **swremove** during the remove "analysis" phase to allow any vendor-defined checks before the software is permanently removed. For example, the script may check whether anyone is currently using the software.

preremove (Preremove script)

This script is executed just before removing files. It may be destructive to the software being removed, as removal of files is the next step. It is the companion script to the install postload script (**postinstall**). For example, it may remove files that the install postload script created.

This script and the next script are part of the "remove" phase of the software remove process. Within each product, all remove **preremove** scripts are run (in the reverse order dictated by any prerequisites), all files are removed, then all remove **postremove** scripts are run.

postremove (Postremove script)

This script is executed just after removing files. It is the companion script to the the install preload script (**preinstall**). For example, if this was a patch fileset, then the install preload script may move the original file aside, and this remove postload script may move the original file back if the patch was removed.

configure (Configure script)

This script is executed by **swinstall** after all software has been installed (including loading files and running **postinstall** scripts) if software is being installed at **/**. It is also may run by **swconfig**, even if the software has already been configured (allowing the administrator to reconfigure software).

unconfigure (Unconfigure script)

This script is executed by **swremove** before any software is removed (including removing files and running **preremove** scripts), if removing from software installed at **/**. It is may also be run by **swconfig**.

Other scripts

The vendor may include other control scripts, such as a script that is 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**.

B.4.1.6.1.2 Control Script Return Code

This part of ISO/IEC 15068 only specifies return codes from scripts that affect operation of the utilities. If the script writer wants to convey additional information to the user, such information should be written to stderr or stdout, which gets recorded by the target role logging. See 4.1.6.5.

A implementation that supports additional behaviors may initiate those behaviors based on implementation-defined return values from control scripts.

B.4.1.6.2 Asynchronous Events

Control script execution and file operations only generate note events at the beginning of the step, not at both the beginning and the end. This is because there is an event immediately after each script or file completes (e.g., the next file beginning, or the end of fileset execution). Additionally, if there is any error with the script or file, there is a warning or error event generated after the file completes.

B.4.1.6.3 Stdout

There is no additional rationale provided for this subclause.

B.4.1.6.4 Stderr

There is a variety of warnings and errors that could occur at the target role. It is desirable that this information be communicated back to the management role and displayed on stderr of the management role. Of course the detail of the information is determined by the *verbose* level. However, since this part of ISO/IEC 15068 does not specify the communication mechanism between the management and target roles, it should not place unnecessary requirements on this communication mechanism, especially since this could be occurring in a distributed environment with the management role communicating with multiple target roles. Thus the only requirement here is that the target role be able to communicate a binary fail/success back to the management role.

B.4.1.6.5 Logging

For the management role, the information placed in the logfile is equivalent to the information sent to stderr and stdout if the *verbose* level was the same value as the *loglevel*.

The stdout, stderr, and logfile output should contain the severity of the event as part of the output message. For example

```
NOTE:    The analysis phase succeeded on target "zook:/".
WARNING: Target "zook:/":  2 configure scripts had warnings.
```

B.4.1.7 Extended Description

B.4.1.7.1 Selection Phase

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.

B.4.1.7.2 Analysis Phase

There is no additional rationale provided for this subclause.

B.4.1.7.3 Execution Phase

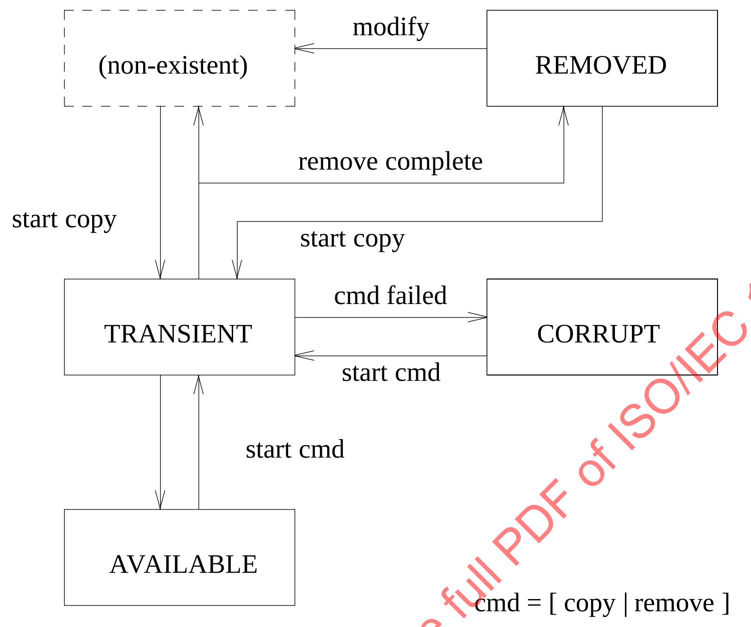

Figure B-5 – Fileset State Transitions (Within Distributions)

It is clear that some vendors may want additional states. But allowing other values would make it problematic for any implementation trying to make decisions on how to deal with filesets with unrecognized states. An implementation may create an additional fileset attribute that would further modify the meaning of the attribute. For example, they may create an attribute called *state_info* and this attribute may have the value of `files_missing` when the *state* attribute is set to `corrupt`. There could be several valid values of this new attribute to describe various possibilities of a `corrupt` state. Of course, since this would be an implementation-specific extension, other implementations would not need to recognize this attribute or its semantics.

Figure B-5 and Figure B-6 show the state transition diagrams for Installed Software and Distributions.

B.4.1.8 Exit Status

There is no additional rationale provided for this subclause.

B.4.1.9 Consequences of Errors

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

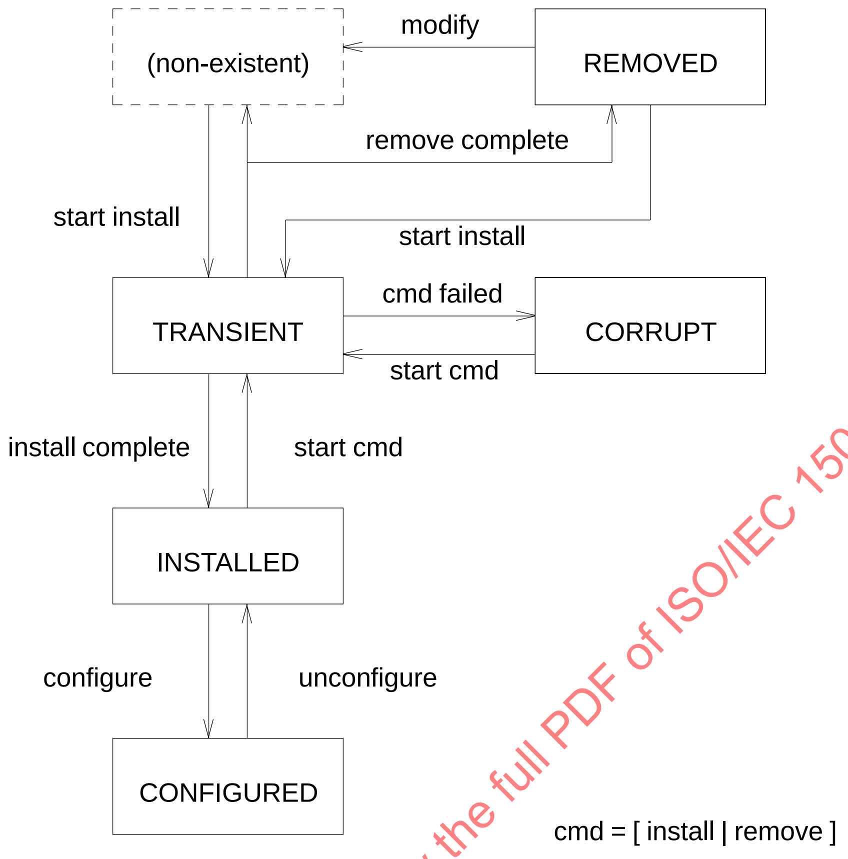

Figure B-6 – Fileset State Transitions (Within Installed Software)

for a specific product on all hosts? The type of error or errors causing such definitions of success or failure is implementation defined. Whether or not the user may specify policies regarding the way in which errors are handled is also implementation defined.

B.4.2 swask — Ask for user responses

The purpose of this utility is to provide support for interactive requirements for software. By being able to execute these interactive scripts independently of **swinstall** and **swconfig**, it allows those utilities to still be scheduled for non-interactive execution.

The **request** scripts can be used to ask the administrator questions, or requests, where responses are needed by the software before installation or configuration.

B.4.2.1 Synopsis

There is no additional rationale provided for this subclause.

B.4.2.2 Description

The utility may be used to perform the following task:

— Answer the requests of software that has interactive customization needs

The **swask** utility and **request** scripts are for software specific questions only. It does not provide any mechanism for implementation specific questions, although an implementation can choose to support implementation, site, or system-specific enhancements as normal implementation extensions.

B.4.2.3 Options

There is no additional rationale provided for this subclause.

B.4.2.4 Operands

There is no additional rationale provided for this subclause.

B.4.2.5 External Influences

There is no additional rationale provided for this subclause.

B.4.2.6 External Effects

There is no additional rationale provided for this subclause.

B.4.2.7 Extended Description

This part of ISO/IEC 15068 has not included a naming convention or structure for storing per client response information. If a **request** script requires per client information, then it needs to store that information for all clients in the **response** file, and then locate the appropriate information during configuration.

This part of ISO/IEC 15068 provides a means for the **request** script to determine which clients are being requested, and for the **configure** script to determine that client. See 4.1.6.1.

For example, a **request** script that requests per client keys is as follows:

```
. $SW_SESSION_OPTIONS
set - $targets
echo 'keys="' > response
for i in $targets
do
   echo enter key for $i
   read j
   echo $i $j >> response
done
echo '"' >> response
```

And the corresponding **configure** script that looks up the correct key from the **response** file:

```
. $SW_SESSION_OPTIONS
```

```
    . response
    set - $keys
    while [ -n $1 ]
    do
       if [ "$1" = "$targets" ]
       then
          echo key is $2
          break
       fi
       shift; shift;
    done
```

An implementation needs to ensure that any **`response`** files that already exist in the source or the catalog are copied to the **SW_CONTROL_DIRECTORY** before the **`request`** script is executed. The order of checks for **`response`** files allows for the following precedence:

— User input (if *ask*=*true*)

— Pre-existing response file

— Pre-existing client configuration

— Model response file (from source)

There are numerous ways to implement where the **`request`** scripts are executed and what **SW_CONTROL_DIRECTORY** is set to, for the command

```
    swask -s source -c catalog Foo.Bar
```

Example implementation A:

```
    mkdir catalog/Foo/Bar
    copy request script for Foo.Bar to catalog/Foo/Bar
    copy any necessary response file to catalog/Foo/Bar
        if catalog/Foo/Bar/response exists, no action
        else if source/catalog/Foo/Bar/response exists, copy it
    set SW_CONTROL_DIRECTORY=catalog/Foo/Bar
    execute catalog/Foo/Bar/request
    (remove catalog/Foo/Bar/request)
```

Example implementation B:

```
    mkdir catalog/Foo/Bar /usr/tmp/aaaa43542/Foo/Bar
    copy request script for Foo.Bar to /usr/tmp/aaaa43542/Foo/Bar
    copy any necessary response file to catalog/Foo/Bar
        if catalog/Foo/Bar/response exists, no action
        else if source/catalog/Foo/Bar/response exists, copy it
    set SW_CONTROL_DIRECTORY=catalog/Foo/Bar
    execute /usr/tmp/aaaa43542/Foo/Bar/request
    (remove /usr/tmp/aaaa43542/Foo/Bar)
```

Example implementation C:

```
    mkdir /usr/tmp/aaaa43542
    copy any necessary response file to /usr/tmp/aaaa43542
         if catalog/Foo/Bar/response exists, copy it
         else if source/catalog/Foo/Bar/response exists, copy it
    set SW_CONTROL_DIRECTORY=/usr/tmp/aaaa43542
    execute source/catalog/Foo/Bar/request
    mkdir catalog/Foo/Bar
    cp /usr/tmp/aaaa43542/response catalog/Foo/Bar
    (remove /usr/tmp/aaaa43542)
```

B.4.2.8 Exit Status

There is no additional rationale provided for this subclause.

B.4.2.9 Consequences of Errors

There is no additional rationale provided for this subclause.

B.4.3 `swconfig` — Configure software

The purpose of configuration is to configure the host for the software, and configure the product for host-specific information. For example, software may need to modify the `/etc/rc` setup file, or the default environment set in `/etc/profile`. It may need to ensure that proper codewords are in place for that host, or do some compilations. Unconfiguration undoes these steps.

B.4.3.1 Synopsis

There is no additional rationale provided for this subclause.

B.4.3.2 Description

This utility may be used to perform the following tasks:

— configuring software on target hosts that will actually be running the software

— configuring independent of the remove and install utilities

— configuring or unconfiguring hosts that share software from another host where the software is actually installed

— reconfiguring when configuration failed, was deferred, or needs to be changed

B.4.3.3 Options

There is no additional rationale provided for this subclause.

B.4.3.4 Operands

There is no additional rationale provided for this subclause.

B.4.3.5 External Influences

There is no additional rationale provided for this subclause.

B.4.3.6 External Effects

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.

One possible file sharing solution involves each client having its own installed software catalog from which the shared software can be configured, and the **`configured`** state can be recorded. This catalog can be built by "link installing" the software; instead of loading files and running **`preinstall`** and **`postinstall`** scripts, link each product's files to the client file system. Then, build the catalog information of this linked software as if it were installed and configure the client.

Another possible solution involves each client recording its configured state in a shared installed software catalog. In order to do this, the installed software could maintain a *configured_instances* attribute to hold a list of configured client names. Each client's **`configure`** and **`unconfigure`** script could add or delete its name from this list.

These scripts could also control whether the installed software *state* attribute was changed from **`installed`** to **`configured`** via **`swconfig`**. If configuring, then the *reconfigure* option would need to be set to *true*. If unconfiguring, then the **`unconfigure`** script could exit with a return code of 3 (exclude) unless the *configured_instances* attribute was empty so that the installed software *state* would remain **`configured`**.

B.4.3.7.1 Examples

B.4.3.8 Exit Status

There is no additional rationale provided for this subclause.

B.4.3.9 Consequences of Errors

There is no additional rationale provided for this subclause.

B.4.4 `swcopy` — Copy distribution

B.4.4.1 Synopsis

There is no additional rationale provided for this subclause.

B.4.4.2 Description

This utility may be used to perform the following tasks:

- — Copy software from one distribution to another
- — Merge software from one distribution into another
- — Copy software to a temporary distribution located to improve **`swinstall`** reliability or performance

B.4.4.3 Options

There is no additional rationale provided for this subclause.

B.4.4.4 Operands

There is no additional rationale provided for this subclause.

B.4.4.5 External Influences

There is no additional rationale provided for this subclause.

B.4.4.6 External Effects

There is no additional rationale provided for this subclause.

B.4.4.7 Extended Description

The `swcopy` utility operates much like the `swinstall` utility. The key distinction between `swcopy` and `swinstall` is the way products are loaded. With `swcopy`, products are not installed for general use below the root directory. Instead, they are placed into a distribution, which can then act as a source for `swinstall`.

B.4.4.7.1 Examples

Copy the *software_selections* listed in `/tmp/load.products` other default values defined in the `/var/adm/sw/defaults` file) as follows:

```
  swcopy -f /tmp/load.products
```

Remove a product Foo from the distribution on the tape device `/dev/rct0` as follows:

```
  swcopy -s /dev/rct0 \* @ /tmp/depot"
  swremove -d Foo @ /tmp/depot"
  swcopy -s /tmp/depot \* @ /dev/rct0"
```

B.4.4.8 Exit Status

There is no additional rationale provided for this subclause.

B.4.4.9 Consequences of Errors

There is no additional rationale provided for this subclause.

B.4.5 swinstall — Install software

The install software task can have different connotations depending upon the life cycle stage of the software package. Figure B-7 illustrates the various state transitions that can occur during an installation of software. These transitions comprise the set of install software tasks supported by this part of ISO/IEC 15068.

The facilities provided by `swinstall` and `swcopy` are basic building blocks on 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.

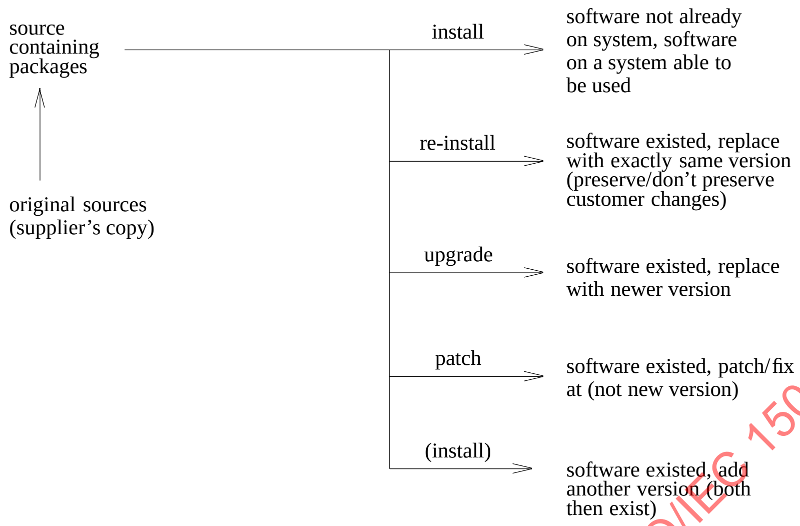
source containing packages
original sources (supplier's copy)
install
software not already on system, software on a system able to be used
re-install
software existed, replace with exactly same version (preserve/don't preserve customer changes)
upgrade
software existed, replace with newer version
patch
software existed, patch/fix at (not new version)
(install)
software existed, add another version (both then exist)

Figure B-7 – Installation State Changes

— Some users might consider **swinstall** to work well for doing remote installation to tens of machines. But with hundreds of machines, perhaps a better strategy would be to use **swcopy** to place a distribution on several servers for use in several parallel **swinstall** invocations. And this example could be cascaded for hierarchical operation with thousands or hundreds of thousands of machines.

— In addition, organizations that use special purpose software distribution programs could choose to use **swcopy** for that purpose, sending a copy of a distribution to some number of machines, pausing until assured that copies have arrived intact at all machines before proceeding, and finally causing **swinstall** to begin work on each machine at the same time.

— An installation process that makes efficient use of network resources might analyze the routing of distributions or files, discover that some traverse a common path before diverging, and cause only one copy to be sent over the common link using the **swcopy** utility.

Table B-4 includes all functionality initially considered for inclusion in the install software task. The second column describes the various operations while the first column lists the degree of support provided by the SVR4, HP-UX, and SCO Unix software management utilities.

B.4.5.1 Synopsis

There is no additional rationale provided for this subclause.