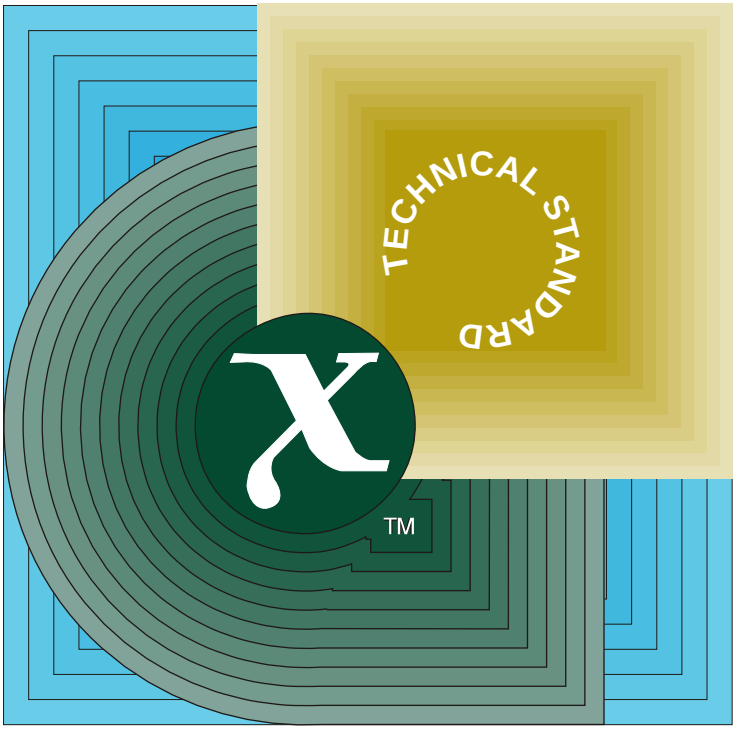
Technical Standard

# Systems Management:
# Distributed Software Administration
# DCE-RPC Interoperability (XDSA-DCE)




THE Open GROUP

[This page intentionally left blank]

*CAE Specification*

# Systems Management: Distributed Software Administration — DCE-RPC Interoperability (XDSA-DCE)

*The Open Group*



CAE Specification

Systems Management: Distributed Software Administration — DCE-RPC Interoperability (XDSA-DCE)

ISBN: 1-85912-137-3
Document Number: C430

Published in the U.K. by The Open Group, February 1997.

Any comments relating to the material contained in this document may be submitted to:

The Open Group
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

OGSpecs@opengroup.org

# Contents

## List of Figures

# Preface

## The Open Group

The Open Group is an international open systems organisation that is leading the way in creating the infrastructure needed for the development of network-centric computing and the information superhighway. Formed in 1996 by the merger of the X/Open Company and the Open Software Foundation, The Open Group is supported by most of the world's largest user organisations, information systems vendors and software suppliers. By combining the strengths of open systems specifications and a proven branding scheme with collaborative technology development and advanced research, The Open Group is well positioned to assist user organisations, vendors and suppliers in the development and implementation of products supporting the adoption and proliferation of open systems.

With more than 300 member companies, The Open Group helps the IT industry to advance technologically while managing the change caused by innovation. It does this by:

- consolidating, prioritising and communicating customer requirements to vendors
- conducting research and development with industry, academia and government agencies to deliver innovation and economy through projects associated with its Research Institute
- managing cost-effective development efforts that accelerate consistent multi-vendor deployment of technology in response to customer requirements
- adopting, integrating and publishing industry standard specifications that provide an essential set of blueprints for building open information systems and integrating new technology as it becomes available
- licensing and promoting the X/Open brand that designates vendor products which conform to X/Open Product Standards
- promoting the benefits of open systems to customers, vendors and the public.

The Open Group operates in all phases of the open systems technology lifecycle including innovation, market adoption, product development and proliferation. Presently, it focuses on seven strategic areas: open systems application platform development, architecture, distributed systems management, interoperability, distributed computing environment, security, and the information superhighway. The Open Group is also responsible for the management of the UNIX trade mark on behalf of the industry.

## The X/Open Process

This description is used to cover the whole Process developed and evolved by X/Open. It includes the identification of requirements for open systems, development of CAE and Preliminary Specifications through an industry consensus review and adoption procedure (in parallel with formal standards work), and the development of tests and conformance criteria.

This leads to the preparation of a Product Standard which is the name used for the documentation that records the conformance requirements (and other information) to which a vendor may register a product. There are currently two forms of Product Standard, namely the Profile Definition and the Component Definition, although these will eventually be merged into one.

The X/Open brand logo is used by vendors to demonstrate that their products conform to the relevant Product Standard. By use of the X/Open brand they guarantee, through the X/Open Trade Mark Licence Agreement (TMLA), to maintain their products in conformance with the Product Standard so that the product works, will continue to work, and that any problems will be fixed by the vendor.

### Open Group Publications

The Open Group publishes a wide range of technical literature, the main part of which is focused on specification development and product documentation, but which also includes Guides, Snapshots, Technical Studies, Branding and Testing documentation, industry surveys and business titles.

There are several types of specification:

- *CAE Specifications*

  CAE (Common Applications Environment) Specifications are the stable specifications that form the basis for our product standards, which are used to develop X/Open branded systems. These specifications are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement a CAE Specification can enjoy the benefits of a single, widely supported industry standard. In addition, they can demonstrate product compliance through the X/Open brand. CAE Specifications are published as soon as they are developed, so enabling vendors to proceed with development of conformant products without delay.

- *Preliminary Specifications*

  Preliminary Specifications usually address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations. They are published for the purpose of validation through implementation of products. A Preliminary Specification is not a draft specification; rather, it is as stable as can be achieved, through applying The Open Group's rigorous development and review procedures.

  Preliminary Specifications are analogous to the *trial-use* standards issued by formal standards organisations, and developers are encouraged to develop products on the basis of them. However, experience through implementation work may result in significant (possibly upwardly incompatible) changes before its progression to becoming a CAE Specification. While the intent is to progress Preliminary Specifications to corresponding CAE Specifications, the ability to do so depends on consensus among Open Group members.

- *Consortium and Technology Specifications*

  The Open Group publishes specifications on behalf of industry consortia. For example, it publishes the NMF SPIRIT procurement specifications on behalf of the Network Management Forum. It also publishes Technology Specifications relating to OSF/1, DCE, OSF/Motif and CDE.

  Technology Specifications (formerly AES Specifications) are often candidates for consensus review, and may be adopted as CAE Specifications, in which case the relevant Technology Specification is superseded by a CAE Specification.

In addition, The Open Group publishes:

- *Product Documentation*

  This includes product documentation — programmer's guides, user manuals, and so on — relating to the Pre-structured Technology Projects (PSTs), such as DCE and CDE. It also includes the Single UNIX Documentation, designed for use as common product documentation for the whole industry.

- *Guides*

  These provide information that is useful in the evaluation, procurement, development or management of open systems, particularly those that relate to the CAE Specifications. The Open Group Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming conformance to a Product Standard.

- *Technical Studies*

  Technical Studies present results of analyses performed on subjects of interest in areas relevant to The Open Group's Technical Programme. They are intended to communicate the findings to the outside world so as to stimulate discussion and activity in other bodies and the industry in general.

- *Snapshots*

  These provide a mechanism to disseminate information on its current direction and thinking, in advance of possible development of a Specification, Guide or Technical Study. The intention is to stimulate industry debate and prototyping, and solicit feedback. A Snapshot represents the interim results of a technical activity.

### Versions and Issues of Specifications

As with all *live* documents, CAE Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards compatible and those which are not:

- A new *Version* indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it *replaces* the previous publication.
- A new *Issue* indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

### Corrigenda

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published on the World-Wide Web at **http://www.opengroup.org/public/pubs**.

### Ordering Information

Full catalogue and ordering information on all Open Group publications is available on the World-Wide Web at **http://www.opengroup.org/public/pubs**.

### This Document

This document defines a mechanism using DCE RPC for providing interoperability for the POSIX 1387.2 standard for Software Administration. The POSIX 1387.2 standard does not provide for interoperability. When the POSIX 1387.2 standard is taken together with this specification's use of the DCE RPC technology to provide interoperability between implementations, a full distributed software administration environment is defined.

### Structure

This specification is organised as follows:

- **Chapter 1** provides an introduction to the XDSA-DCE specification and the POSIX 1387.2 standard. It describes the relationship between this statement and the POSIX 1387.2 standard.
- **Chapter 2** provides an overview of the XDSA-DCE interface. It describes the roles, processes, and features specified in this document.
- **Chapter 3** defines the DCE RPC interfaces.
- **Chapter 4** defines the DCE IDL data types used in this specification.
- **Chapter 5** defines the enumerated and defined type values used in the XDSA-DCE RPC interfaces.
- **Chapter 6** defines extensions to two POSIX 1387.2 standard utilities for registering (*swreg*) and listing (*swlist*) distributions and installed software collections.
- **Chapter 7** defines the *swagentd* command which serves the XDSA-DCE daemon RPC interface for local or remote software management tasks.
- **Chapter 8** describes the XDSA-DCE security model, specifies the *swacl* command which defines distributed management of Access Control Lists, and describes the use of the DCE RDACL interfaces to support this command.

### Typographical Conventions

The following typographical conventions are used throughout this document:

- **Bold** font is used in text for filenames, keywords, type names, data structures and their members.
- *Italic* strings are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote:
  - — variable names, for example, substitutable argument prototypes, parameters and environment variables
  - — options in text
  - — function calls are shown as follows: *name*( )
- Normal font is used for the names of constants and literals.
- OSF IDL code fragments are shown in **helvetica bold** font.

# Trade Marks

Motif®, OSF/1® and UNIX® are registered trade marks and the ‘‘X Device’’™ and The Open Group™ are trade marks of The Open Group.

OMG® and Object Management® are registered trade marks of the Object Management Group, Inc.

POSIX (Portable Operating System Interface) is a registered trademark of the US Institute of Electrical and Electronic Engineers (IEEE).

# Referenced Documents

The following documents are referenced in this Specification:

DCE Directory

CAE Specification, December 1994, X/Open DCE: Directory Services (ISBN: 1-85912-078-4, C312).

DCE RPC

CAE Specification, August 1994, X/Open DCE: Remote Procedure Call (ISBN: 1-85912-041-5, C309).

This specification is now also ISO International Standard ISO/IEC 11578:1996, Information technology — Open Systems Interconnection — Remote Procedure Call (RPC)

DCE Security

Preliminary Specification, April 1996, X/Open DCE: Authentication and Security Services (ISBN: 1-85912-013-X, P315). Chapter 4: OSF Interface Definition Language (IDL) Definition.

POSIX 1387.2

IEEE Std. 1387.2-1995, Information Technology — Portable Operating System Interface (POSIX) System Administration — Part 2: Software Administration.

Chapter 1

# Introduction

## 1.1 Scope and Purpose of XDSA-DCE

This document is the X/Open Distributed Software Administration (XDSA-DCE) specification. XDSA-DCE provides a mechanism for addressing the interoperability needs of the POSIX 1387.2 standard for Software Administration.

The scope of the POSIX 1387.2 standard does not include interoperability. The XDSA-DCE specification defines a way in which the various distributed roles communicate and transfer data. This interface is one means to achieve interoperability between different implementations of the POSIX 1387.2 standard.

XDSA-DCE is implemented using the X/Open Distributed Computing Environment (DCE) remote procedure calls (RPC). The interface has no DCE run-time requirements, although it supports the use of DCE naming and security services when operating in a DCE cell.

XDSA-DCE is designed with the intention that it will support the current and future versions of the POSIX 1387.2 standard by only adding additional legal values and associated semantics to the current RPC parameters, but not modifying the protocol.

## 1.2 Scope of the POSIX 1387.2 Standard

The scope of the POSIX 1387.2 standard for software administration includes the following:

- A standard layout for software.

  This is a common exported structure for software distributions that contain the software organized into manageable objects.

- A definition for information about installed software.

  These are the definitions of the software objects, and the attributes they support.

- A standard set of commands for manipulating software.

  This is the set of command definitions needed to manage the software, as well as the behaviors associated with the software attributes.

## 1.3 The POSIX 1387.2 Standard

The referenced document **POSIX 1387.2** is the IEEE POSIX Standard 1387.2-1995: Information Technology — Portable Operating System Interface (POSIX) System Administration — Part 2: Software Administration.

Throughout this XDSA-DCE specification, in any description of POSIX 1387.2 where a variance with this standard arises, the referenced **POSIX 1387.2** standard is the definitive specification.

## 1.4 POSIX 1387.2 Distributed Roles

While not defining protocols for interoperability, the POSIX 1387.2 standard has defined the distributed model by defining the distributed roles. The three key roles related to interoperability are the **manager, source** and **target** roles shown in Figure 1-1.

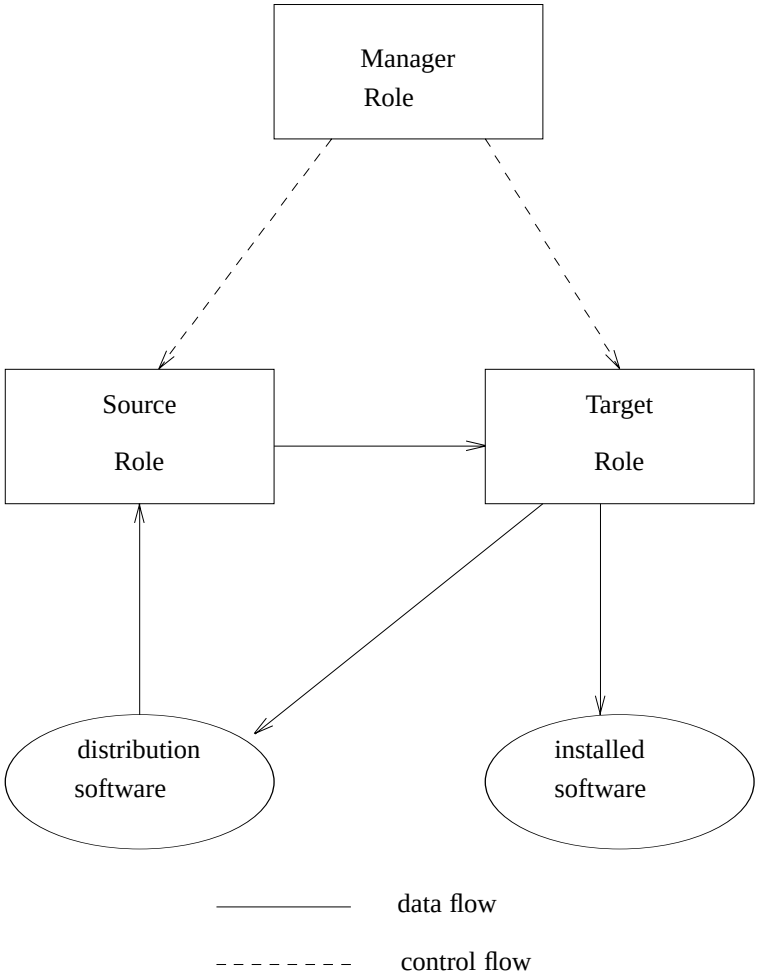



**Figure 1-1** POSIX 1387.2 Distributed Roles

The manager role is the process that interacts with the administrator. It sends requests and receives results from the target role, and sends requests and receives data from the source role.

The target role operates on software collections (performing the administrative task). It requests and receives the actual software files from the source role.

The source role receives requests for data or actual software files and returns the data and software files in the software collection.

Examining the syntax of the POSIX 1387.2 standard commands, there are four main components:

- Target Selections
  This is a list of target objects that the software specifications will be applied to. The target objects are manageable software collections, identified by a host and a path.

- Software Selections
  This is a list of software specifications that identify particular software objects to be applied to the targets.
- Command
  This is the operation to apply to the software in each target. Most operations define an analysis phase to determine if the operation will like succeed, and an execution phase that actually perform the operation.
- Options
  These are used to additionally define the previous operation.

Thus, distributed operation involves sending the software selections, command and options from the manager to each target.

Some software commands also involve taking software from a source and transferring it (applying it) to the target. Therefore, distributed operation also involves the target requesting software files for the software selections it is processing from the source.

## 1.5 Terminology

This specification assumes some familiarity with the terminology used in the POSIX 1387.2 standard.

It also uses some terms which have specific meaning for XDSA-DCE. In particular, XDSA-DCE uses some different terms for the same concepts described in the POSIX 1387.2 standard.

The key POSIX 1387.2 standard and XDSA-DCE terms are defined in the Glossary at the end of this specification.

## 1.6 Conformance

**Mandatory**

An implementation is conformant to this specification if it implements at least the required portions of this specification as defined in Chapters 1 through 6.

**Options**

The following aspects of XDSA-DCE are optional:

XDSA-DCE Daemon
: The XDSA-DCE Daemon option is defined by Chapter 7 of this specification.

XDSA-DCE Security
: The XDSA-DCE Security option is defined by Chapter 8 of this specification.

In order for an implementation to conform to any of these options, it must conform to all of the specification defined for that option.

Chapter 2

# XDSA-DCE RPC Interface Overview

This section presents an overview of the XDSA-DCE RPC interface:

- XDSA-DCE Block Diagram
- XDSA-DCE Roles and Processes
- XDSA-DCE RPC Features.

## 2.1 XDSA-DCE Block Diagram

It is helpful to present a block diagram of an example implementation using the XDSA-DCE, in order to better understand the relationship of the RPC interface to the overall architecture.

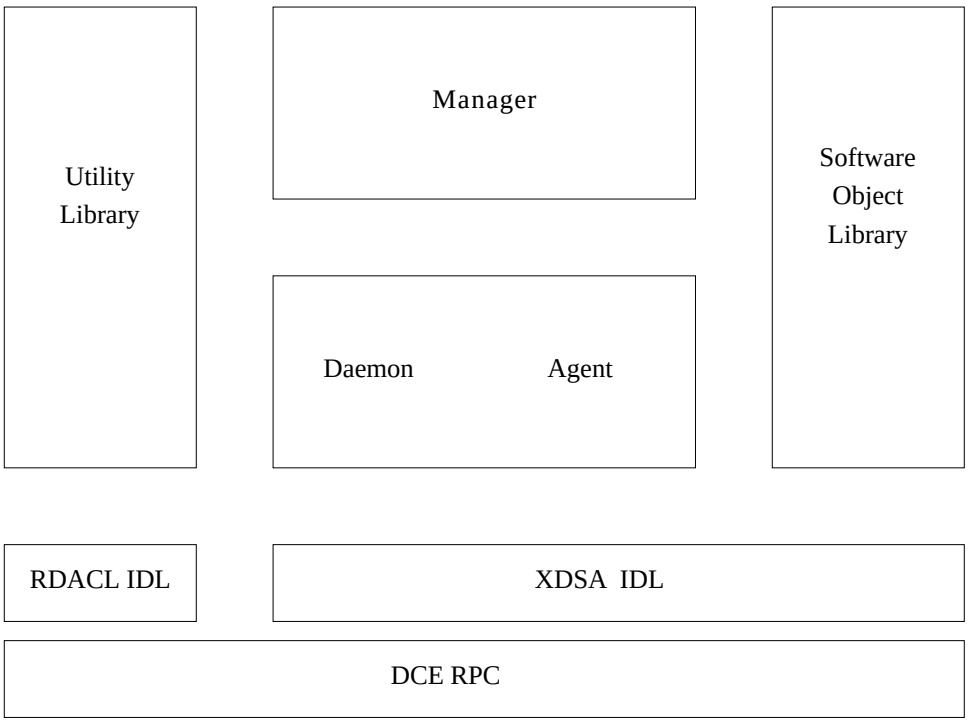



**Figure 2-1** XDSA-DCE Model

Referring to Figure 2-1:

- The management tasks are accessible to the administrator as POSIX 1387.2 standard command line (CLI) or graphical (GUI) user interfaces to the manager process. The manager manages the operations on the source and target software collections through the XDSA-DCE RPC interface.
- Operations on software objects can be implemented using the software object library. This library manipulates the software files, and manages the meta data for the software objects (for example, getting and setting attributes, commiting and deleting objects, loading software files and executing software scripts). These interfaces could eventually integrate into an object manager provided by a management framework.

- The utilities library can provide interfaces to common services such as event handling, logging, security, and access control. These interfaces can also integrate with services provided by a management framework.
- The actual management operations are performed by software daemon and agent processes. These may be the same process or different processes, allowing performance tuning and eventual migration to management frameworks for the daemon (managing the host object).
- The distributed aspects of an implementation are implemented using the XDSA-DCE RPC interface. This interface is defined in terms of DCE Interface Definition Language (IDL). Additionally, an implementation can use the DCE supplied RDACL interfaces, implemented by the daemon process, to manage Access Control Lists (ACLs).

## 2.2 XDSA-DCE Roles and Processes

The XDSA-DCE implements the manager role as one manager process and the source and target roles as a set of daemon and agent processes as shown in Figure 2-2

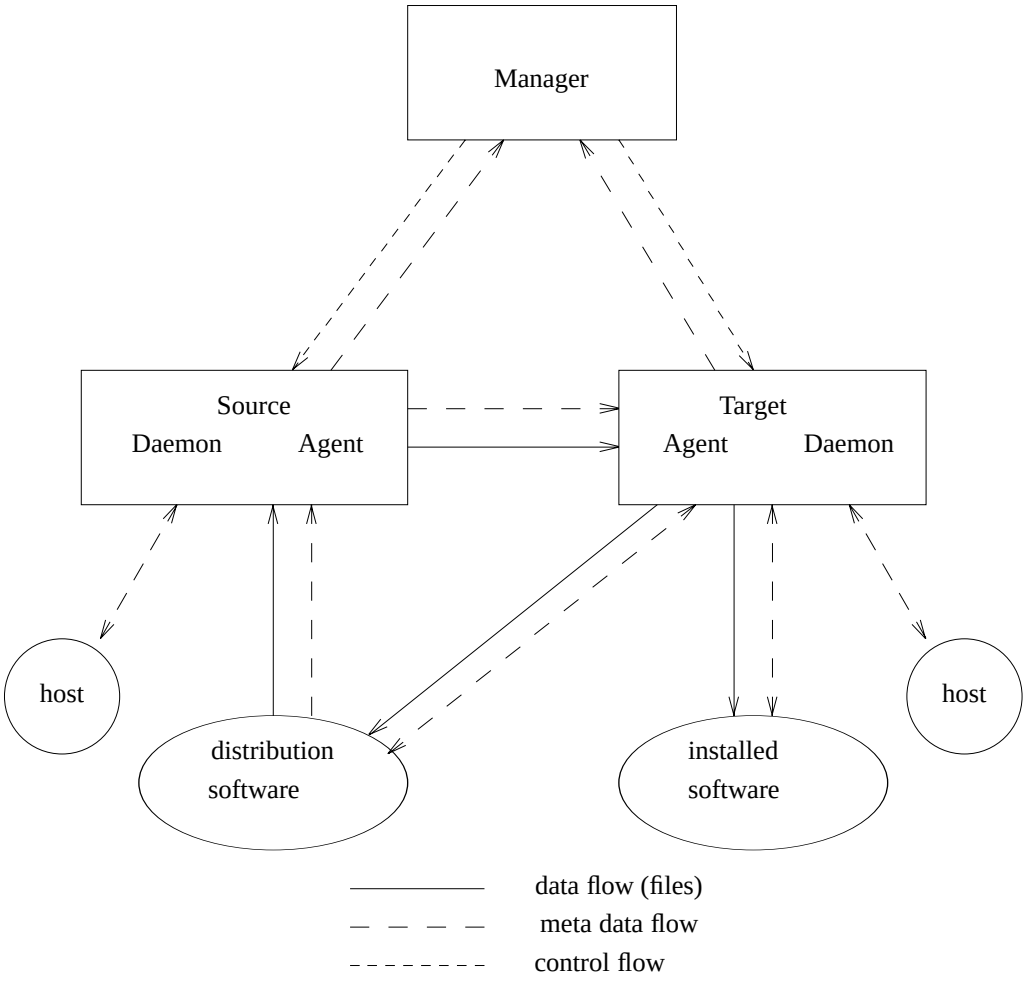



**Figure 2-2** XDSA-DCE Roles

The manager retrieves software information from the source. This can be used by a command line or graphical user interface to present software selections to the user. Then, both the GUI and command line interface send the resolved software selections to the targets.

The target is implemented using a daemon/agent pair of processes. The daemon is always running on each target host, and spawns an agent session for each target. The agent session can be a separate process, or part of the daemon process. The agent session performs the software operation, including retrieving information and files from the source.

The source is implemented as the same daemon/agent pair. The daemon is always running on each source host and spawns an agent session for each source request. The source agent serves information and files from the source software collection for the target agent and manager.

In this way, in relation to the POSIX 1387.2 standard software hierarchy where a host object contains a set of distribution and installed software collections, the daemon essentially manages the host object and each agent manages one software collection.

## 2.3 XDSA-DCE RPC Features

The XDSA-DCE RPC interfaces define parameters following the main components of the POSIX 1387.2 standard commands described above:

- Target

  As described above, an agent session is defined for each target. After initiating an agent for the target, the rest of the task addresses each agent.

- Software Selections

  The software selections are communicated between the various roles as strings using the standard software selection syntax defined by the POSIX 1387.2 standard. Using the standard software identifier syntax is important for interoperability as it insulates the implementations from any implementation specific software object identifiers.

- Command

  The task type is the primary parameter in the RPC interface. Further, there is a separate interface for analysis and execution. This point is important for interactive interfaces, where results of analysis can be viewed, and analysis can be repeated, before committing to execution. It also is necessary to support consistent multiple target success where all targets need to pass analysis before execution is committed to for any target. If the execution phase suspends, then it too can be retried.

- Options

  The options are communicated using the standard syntax defined by the POSIX 1387.2 standard. This syntax is important for both interoperability and extensibility. This same parameter is used to pass any additional options that can be described as **name=value** strings, and is used for features beyond the POSIX 1387.2 standard definition, including additional control options and security information.

The key roles for the source agent is to provide files and data to the manager and target agent.

- Software Files

  The software files to be transferred are identified by the software specification and the file path. An important aspect of this interface is that the target only requests the files that it requires, allowing for target optimizations for files that are already up to date.

- Software Information

  For listing software objects and attributes, applying software selections by the manager or agent, and retrieving control scripts to be executed by the manager or agent, the same file transfer interface is used.

  While natural for control scripts, this is also important to interoperability for software object and attribute information. Using this interface, all information is transferred in the POSIX 1387.2 standard external format (software packaging layout) via INDEX and INFO files. This saves implementations from having to support an additional exported format in addition to the POSIX 1387.2 standard external format and their own internal representation.

  The POSIX 1387.2 standard software packaging layout is also used for transferring information about installed software objects.

  This design is also impacted by the security model. In the XDSA-DCE security model, read access to distribution includes the permission to list all of the products in the distribution. Thus the entire global INDEX may be transferred. However, product files are only accessible for analysis and execution (loading) if the user has read access to the product.

In a model where products may only be listed by those with read access to the product, then only those products with read access would be included in the global INDEX tranferred.

Both the source and target agent define **sessions** that maintain state from one call to the next, as shown in Figure 2-3

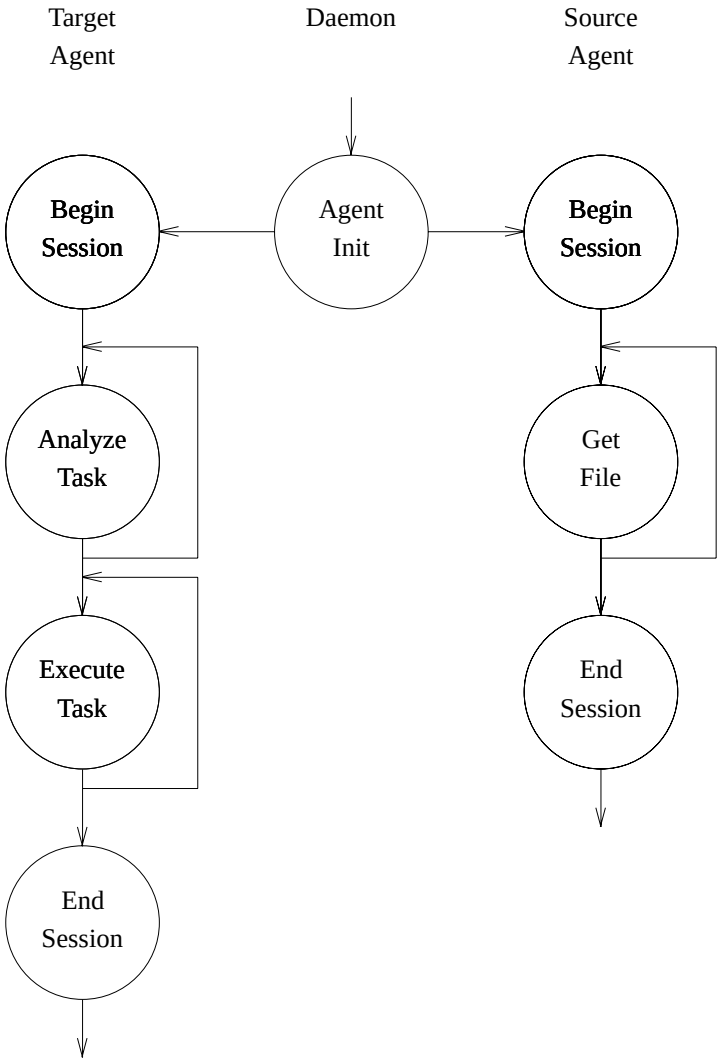



**Figure 2-3** XDSA-DCE Target and Source Sessions

There are a number of points in this design:

- Begin Session

  The concept of a session is important primarily to reserve (read or write lock) the software collection resource for the duration of the operation. This increases reliability of multiple target operations, since the resources are committed before the operation begins.

  For the interactive interface, the user has immediate feedback if the operation will fail due to the target already being unavailable.

  In the case of the source agent, significant overhead (such as security and database access) is avoided for the file retrieval RPC by caching information.

- End Session

  By maintaining the session until an explicit end session call is made, the interactive user can query the agent for results of an operation even after the operation execution has completed.

- Initiate Agent

  Having a separate interface to initiate a source or target agent allows for a number of flexible implementations from an implementation of a separate daemon and agent, to a single daemon process that also performs the agent tasks (the latter simply returns an RPC binding to itself).

  Some implementations may use a separate, smaller, daemon since the daemon must always be resident in memory and memory utilization is critical for smaller systems. Also, on source systems, a single process would require up front memory allocation for the maximum number of source connections.

  The separate daemon interface also logically maps to a different level of managed object than the agent (the host as opposed to the software collection). Its possible to envision that the host object may have a single daemon (part of the management framework) for disparate managed objects, and separate agent programs for each type of object.

The XDSA-DCE RPC interface also reflects a need to not have any **requirements** on management frameworks, while still allowing integration with these frameworks. This is important since installed software collection is needed for initial install of an operating system where the management framework may never exist, and for installed software collection and update of the management framework itself.

- Events

  Each RPC interface has a results array return parameter that contains a list of event types and severity, and additional information that usually contains the number of occurrences of the event.

  The RPC also has return parameters for each software object that was analyzed or executed. This contains a summary of the events that occurred on each object (results of that operation). This information is used by XDSA-DCE to present per software object results to the user.

  While the XDSA-DCE specification is not currently integrated into an event notification service, there is a single interface that could optionally send events to the event service while still building the results array for manager not using that service.

- Logging

  Similarly, an implementation does not require a logging service, but does not preclude the eventual use of one either. Figure 2-4 on page 11 shows the interface for logfile retrieval during the course of an operation (which can be displayed along with the status to the user in a graphical user interface).

- Status

  The status of the task (analysis or execution) is not implemented by events. Rather, the agent keeps track of its progress (bytes loaded, percent complete, time to completion, etc) and returns that to the manager when requested through the same interface as the logfile.

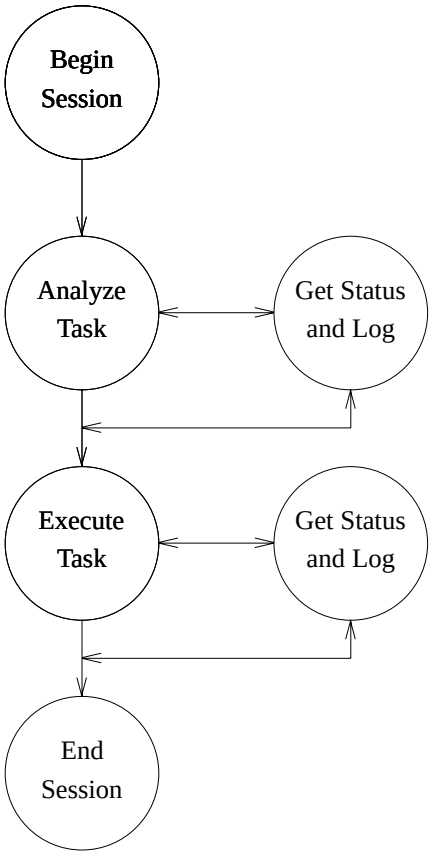



**Figure 2-4** XDSA-DCE Status and Log Retrieval

- Authentication

  The first aspect of the XDSA-DCE security model is authentication. When operating in a DCE cell, the DCE RPC automatically provides the information needed to authenticate the caller. The manager user is authenticated by the target agent and source agent. The target agent is authenticated by the source agent.

  The XDSA-DCE interface also supports a mode of secure operation that does not depend on the run-time DCE needed to maintain a DCE cell, and also supports operations across DCE cells. While not as reliable as DCE authentication, the XDSA-DCE internal method still is reliable (more so than ARPA/Berkeley for example) through its use of an encrypted, configurable, "secret". Both the secret, and the caller information is communicated using the same **options** parameter that is used for the POSIX 1387.2 standard and other XDSA-DCE specific options.

- Delegation

  In the distributed model described above, the target agent contacts the source agent for software files to install or copy. Delegation allows the agent to also notify the source agent not only of itself (the caller), but also of the manager user that is requesting the software (the initiator).

  Since run-time DCE does not support delegation directly, the RPC interface options parameters are used to pass the initiator information in any case, in the same way that the caller options are used for XDSA-DCE internal authentication.

- Authorization

  The second aspect of the XDSA-DCE security model is authorization for access to particular objects. The authorization model is fairly flexible in how it relates to the RPC interface. The interface is responsible for communicating the information necessary to determine authorization to an object (via caller and initiator information). The target and source agent (by examining their ACLs, next) can then refuse a particular request (for example, creation of a new distribution collection object) or part of a request (for example, modification of, or access to a particular product in a list of products).

  Thus, authorization (ACL) information is not communicated via the XDSA-DCE interfaces, just the refusals of authorization in the results of those calls.

- Access Control Lists

  Authorization to objects is determined by the access control lists associated with the objects. XDSA-DCE supports a security ACL scheme that includes a set of permission types (read, write, insert, test, control) for a set of principal (caller) types (object_owner, object_group, user, group, host, other, any_other) on a set of object types (host, distribution, installed software collection, product). Which permissions are needed for which objects for each task type are listed with the descriptions of the task types.

  Product level ACLs are defined for products in distributions. A distribution can contain products for a large variety of consumers and eventual installed software collections. Product level ACLs are not defined for products in installed software collections. Installed products share the same filesystem, the POSIX 1387.2 standard supports different installed software objects sharing the same filesystem installed software collection for the purpose of different management (including security) domains, and the semantics of update and multiple versions would be significantly complicated by product level security in installed software collections.

- Distributed ACL Management

  XDSA-DCE also provides a distributed ACL management command, swacl. This command supports a similar syntax to the other POSIX 1387.2 standard commands (that is, options, software selections and target selections).

  The RPC interface used to implement this command are the standard RDACL interface definitions supplied with DCE, and are not documented here. In an XDSA-DCE implementation, the server side of RDACL interface is implemented in the daemon. The manager then uses standard client DCE security interfaces to manage ACLs.

- Naming and Registration

  Naming is used in XDSA-DCE for locating sources and targets. As with other management framework services, XDSA-DCE is designed to be operated both with and without run-time DCE Naming Services. Using DCE Naming Services, lists of target hosts and source hosts, along with the RPC protocols for contacting those host's daemons, can be retrieved.

  Like some aspects of security, the use of naming is somewhat independent of the RPC interface. For example in XDSA-DCE, the manager can resolve the location of a distribution using naming, then pass the source information to the agent the same way it would if not using a naming service.

  Independent of run-time DCE, there is an XDSA-DCE interface to query hosts for available (registered) distributions and installed software collections on that host. XDSA-DCE also provides a distributed command to register distributions in the host object, **swreg**. (Distributions and installed software collections are also automatically registered and

unregistered as part of the POSIX 1387.2 standard **swcopy** command). There are four XDSA-DCE RPC interfaces for managing registration of distribution and installed software collections: see *sw_rpc_register_depot*( ) on page 33, *sw_rpc_unregister_depot*( ) on page 34 and *sw_rpc_is_registered_depot*( ) on page 32.

- Multiple and Alternate Sources

  Related to naming are other features that abstract source location. There are two main aspects to this feature: supporting access to software from more than one source in a session, and supporting the use different sources by different agents in one task.

  The latter is supported by XDSA-DCE implementation where different targets can be configured for different sources. This can be used to support two step install (copying to the target before installing from a local copy), hierarchical distribution, and distributed installed software collection optimized for source proximity and performance.

  The impact on the XDSA-DCE RPC is an additional option to communicate to the agent to choose its own source. Additionally, even though XDSA-DCE does not currently support an agent installing different products from different sources, the RPC interface does support this since each software specification can have a different source associated with it (allowing the manager to still perform selection).

- Disk Space Analysis

  There are two additional XDSA-DCE RPC interfaces defined which support transfer of detailed disk space analysis from the target agent to the manager. This can be used by a manager GUI to show the impact of each software item (product or fileset) on each target filesystem, or can be used for verbose output in a CLI manager.

Chapter 3

# XDSA-DCE RPC Interface Specification

This chapter describes the XDSA-DCE RPC interface used for communication between the manager, target and source roles. Target information and tasks are communicated between the manager and targets. Source information is communicated between the source and manager. Source information and source files are communicated between the source and target.

The XDSA-DCE RPC interface is defined in three DCE RPC interface definitions:

- the daemon interface
- the agent interface
- the type definitions interface.

Each interface definition contains a UUID line identifying the interface and version of the interface, and interface body that contains the procedure and type definitions.

The XDSA-DCE daemon RPC interface is defined as follows:

```
[uuid(92F278D6-1723-11CC-9A0F-08000935358F),  version(1.0)]

interface sdu_rpc_daemon
{

        import "sdu_rpc_defs.idl";

        /*
        * The XDSA-DCE daemon interface defines the following procedures:
        *   sw_rpc_get_depots
        *   sw_rpc_register_depot
        *   sw_rpc_unregister_depot
        *   sw_rpc_is_registered_depot
        *   sw_rpc_agent_init
        */

        /* procedure definitions */

} /* sdu_rpc_daemon */
```

The XDSA-DCE agent RPC interface is defined as follows:

```
[uuid(DD1933A0-B026-11CB-B03C-080009199BEB), version(1.0)]

interface sdu_rpc_agent
{

        import "sdu_rpc_defs.idl";

        /*
        * The XDSA-DCE agent interface defines the following procedures:
        *   sw_rpc_begin_session
        *   sw_rpc_end_session
        *   sw_rpc_analyze_task
        *   sw_rpc_execute_task
        *   sw_rpc_abort_task
        *   sw_rpc_get_task_status_and_log
        *   sw_rpc_get_dsa_impact_data
        *   sw_rpc_get_dsa_volume_list
        *   sw_rpc_get_soc_file
        */

        /* procedure definitions */

} /* sdu_rpc_agent */
```

The XDSA-DCE type definitions interface is defined in Chapter 4.

**NAME**

sw_rpc_abort_task - abort task

**SYNOPSIS**

```
void sw_rpc_abort_task
(
        [in]        handle_t                agent_binding,
        [in]        sw_rpc_context_t        session_context,
        [in]        sw_rpc_options_t        *control_options,
        [out]       sw_rpc_session_phase_t  *session_state,
        [in, out]   sw_rpc_results_t        *results
);
```

**DESCRIPTION**

This function aborts the execution of a task that has entered a suspended state. Task execution that has suspended may be aborted using this function, or resumed via the *sw_rpc_execute_task*( ) function.

The results of the function are returned via the *results* parameter, which is a structure of type *sw_rpc_results_t*. This structure contains a *summary_status* field which indicates the overall success of the function, and a *results_list* array which lists all error, warning, and informational events that occurred during execution of the function.

The value returned in the *summary_status* is a mask of the following possible values:

SW_NOTE
: The function was successful, but one or more informational events occurred.

SW_WARNING
: The function was successful, but one or more warning events occurred.

SW_ERROR
: The function was not successful, due to the occurrence of one or more error events.

The *results_list* field is a variable, conformant array of items which describe each event's result status (that is, error, warning, or note), its result code, and additional result information indicating number of occurrences, etc.

**RETURNS**

No return value (void).

**ERRORS**

Error, warning, and informational events are indicated via the *results* parameter. Refer to the *Result Codes* section for a description of the possible events.

**NAME**

sw_rpc_agent_init - initialise agent

**SYNOPSIS**

```
void sw_rpc_agent_init
(
        [in]        handle_t                daemon_binding,
        [in]        sw_rpc_options_t        *control_options,
        [in]        sw_rpc_string_ref_t     soc_path,
        [in]        sw_rpc_task_t           task_type,
        [out]       sw_rpc_string_full_t    *agent_string_binding,
        [in, out]   sw_rpc_results_t        *results
);

} /* sdu_rpc_daemon */
```

**DESCRIPTION**

Tell the XDSA daemon to schedule the XDSA agent, in preparation for performing a management task on a distribution or installed software collection. A string binding is returned in *agent_string_binding*, which is then used to bind to the newly-scheduled agent in the *sw_rpc_begin_session*( ) function.

This call exists to support a process model in which the daemon exists as a separate process and schedules agents to do the actual tasks. For implementations that have the daemon and agent as the same process, this call is still required to initiate the agent sesssion within the daemon process.

An RPC binding handle is supplied in *daemon_binding*, which is used to establish the connection with the daemon.

The location of the software collection is supplied in *soc_path*. This is supplied as an absolute path with respect to the containing host's filesystem.

The task that is to be performed on the software collection is indicated by *task_type*.

The results of the function are returned via the *results* parameter, which is a structure of type *sw_rpc_results_t*. This structure contains a *summary_status* field which indicates the overall success of the function, and a *results_list* array which lists all error, warning, and informational events that occurred during execution of the function.

The value returned in the *summary_status* is a mask of the following possible values:

SW_NOTE
: The function was successful, but one or more informational events occurred.

SW_WARNING
: The function was successful, but one or more warning events occurred.

SW_ERROR
: The function was not successful, due to the occurrence of one or more error events.

The *results_list* field is a variable, conformant array of items which describe each event's result status (that is, error, warning, or note), its result code, and additional result information indicating number of occurrences, etc.

**RETURNS**

No return value (void).

**ERRORS**

Error, warning, and informational events are indicated via the *results* parameter. Refer to the *Result Codes* section for a description of the possible events.

**NAME**

sw_rpc_analyze_task - analyse task

**SYNOPSIS**

```
void sw_rpc_analyze_task
(
        [in]            handle_t                agent_binding,
        [in]            sw_rpc_context_t        session_context,
        [in]            sw_rpc_options_t        *control_options,
        [in, out]       sw_rpc_selections_t     *software_selections,
        [out]           sw_rpc_session_phase_t  *session_state,
        [in, out]       sw_rpc_results_t        *results
);
```

**DESCRIPTION**

This function initiates the checks that are performed prior to the actual execution of a management task on a distribution or installed software collection. The caller must have previously established a management session for the software collection, which is identified by *session_context*. The nature of the analysis is determined by the given task, which was specified when the session was established.

Software specifications may be specified at any level (bundles, products, subproducts, or filesets), and are passed in the *software_selections* parameter. In order to receive the most detailed result information when the call returns, the *software_selections* should be specified at the fileset level. Selection information is specified using a conformant array of software selection item structures, each of which contains the software specification string, the UUID, and the install or copy source of the selected software object. Additionally, each software selection item structure contains fields which are modified by the agent to indicate the expected action, error, and warning information that was collected during task analysis.

The expected actions determined during analysis for a given selected software object are listed in the section *Analysis States.*

The *session_state* output parameter controls whether the task can proceed to execution phase. If the value is SW_ANALYSIS_COMPLETED_PHASE, then one or more software selections passed analysis without errors, and they can be executed on. If no software passed analysis, then the state returned is SW_SELECTION_PHASE. In either case, the task can also repeat the analysis or end the session. See Section 5.5 on page 51 for all possible states.

The results of the function are returned via the *results* parameter, which is a structure of type *sw_rpc_results_t*. This structure contains a *summary_status* field which indicates the overall success of the function, and a *results_list* array which lists all error, warning, and informational events that occurred during execution of the function.

The value returned in the *summary_status* is a mask of the following possible values:

SW_NOTE
: The function was successful, but one or more informational events occurred.

SW_WARNING
: The function was successful, but one or more warning events occurred.

SW_ERROR
: The function was not successful for one or more *software_selections*, due to an occurrence of one or more error events.

The *results_list* field is a variable, conformant array of items which describe each event's result status (that is, error, warning, or note), its result code, and additional result information indicating number of occurrences, etc.

**RETURNS**

No return value (void).

**ERRORS**

Error, warning, and informational events are indicated via the *results* parameter. Refer to the *Result Codes* section for a description of the possible events.

**NAME**

sw_rpc_begin_session - begin session

**SYNOPSIS**

```
void sw_rpc_begin_session
(
        [in]        handle_t                agent_binding,
        [in]        sw_rpc_options_t        *control_options,
        [in]        sw_rpc_string_ref_t     soc_path,
        [in]        sw_rpc_task_t           task_type,
        [out]       sw_rpc_uname_attrs_t    *uname_attrs,
        [out]       sw_rpc_context_t        *session_context,
        [out]       sw_rpc_session_phase_t  *session_state,
        [in, out]   sw_rpc_results_t        *results
);
```

**DESCRIPTION**

This function establishes a session context with the XDSA agent for the management of a software distribution or installed software collection.

The location of the software collection is supplied in *soc_path.* This is supplied as an absolute path with respect to the containing host's filesystem.

The task that is to be performed on the software collection is indicated by *task_type.*

The session may be refused if the requesting manager does not have adequate authorization to manage the specified software collection, or if another conflicting management operation is already in progress on the software collection. If a session context is successfully established, a session handle is returned in *session_context*.

The session context is implemented using the DCE RPC context handle mechanism, enabling the XDSA agent to maintain the session's state across multiple RPCs. The context handle mechanism is also used by the agent to detect communication failure and/or the unexpected death of an XDSA manager.

The results of the function are returned via the *results* parameter, which is a structure of type *sw_rpc_results_t.* This structure contains a *summary_status* field which indicates the overall success of the function, and a *results_list* array which lists all error, warning, and informational events that occurred during execution of the function.

The value returned in the *summary_status* is a mask of the following possible values:

SW_NOTE
: The function was successful, but one or more informational events occurred.

SW_WARNING
: The function was successful, but one or more warning events occurred.

SW_ERROR
: The function was not successful, due to the occurrence of one or more error events.

The *results_list* field is a variable, conformant array of items which describe each event's result status (that is, error, warning, or note), its result code, and additional result information indicating number of occurrences, etc.

**RETURNS**

No return value (void).

**ERRORS**

Error, warning, and informational events are indicated via the *results* parameter. Refer to the *Result Codes* section for a description of the possible events.

**NAME**

sw_rpc_end_session - end session

**SYNOPSIS**

```
void sw_rpc_end_session
(
        [in]            handle_t            agent_binding,
        [in, out]       sw_rpc_context_t    *session_context,
        [in]            sw_rpc_options_t    *control_options,
        [in, out]       sw_rpc_results_t    *results
);
```

**DESCRIPTION**

This function terminates a previously-established session for a distribution or installed software collection. The session to be terminated is indicated in *session_context*. When a session is terminated, the agent will release whatever resources and state were being maintained for it.

The results of the function are returned via the *results* parameter, which is a structure of type *sw_rpc_results_t*. This structure contains a *summary_status* field which indicates the overall success of the function, and a *results_list* array which lists all error, warning, and informational events that occurred during execution of the function.

The value returned in the *summary_status* is a mask of the following possible values:

SW_NOTE
: The function was successful, but one or more informational events occurred.

SW_WARNING
: The function was successful, but one or more warning events occurred.

SW_ERROR
: The function was not successful, due to the occurrence of one or more error events.

The *results_list* field is a variable, conformant array of items which describe each event's result status (that is, error, warning, or note), its result code, and additional result information indicating number of occurrences, etc.

**RETURNS**

No return value (void).

**ERRORS**

Error, warning, and informational events are indicated via the *results* parameter. Refer to the *Result Codes* section for a description of the possible events.

**NAME**

sw_rpc_execute_task - execute task

**SYNOPSIS**

```
void sw_rpc_execute_task
(
        [in]        handle_t                agent_binding,
        [in]        sw_rpc_context_t        session_context,
        [in]        sw_rpc_options_t        *control_options,
        [in, out]   sw_rpc_selections_t     *software_selections,
        [out]       sw_rpc_session_phase_t  *session_state,
        [in, out]   sw_rpc_results_t        *results
);
```

**DESCRIPTION**

This function initiates the actual execution of a management task on a distribution or installed software collection. This function is also used to resume the execution of a task that has entered a suspended state.

The caller must have previously established a management session for the software collection, which is identified by *session_context*. The task that is to be performed was specified when the session was established.

Software specifications may be specified at any level (bundles, products, subproducts, or filesets), and are passed in the *software_selections* parameter. In order to receive the most detailed result information when the call returns, the *software_selections* should be specified at the fileset level. Selection information is specified using a conformant array of software selection item structures, each of which contains the software specification string, the UUID, and the install or copy source of the selected software object. Additionally, each software selection item structure contains fields which are modified by the agent to indicate the resulting action, error, and warning information that was collected during task execution.

The resultant actions taken during execution for a given selected software object are listed in the section *Execution States*.

The *session_state* output parameter controls whether the task can continue a suspended execution phase. If the value is SW_SUSPENDED_PHASE, then *sw_rpc_execute_task* can be called again to resume the execution. See Section 5.5 on page 51 for all possible states.

The results of the function are returned via the *results* parameter, which is a structure of type *sw_rpc_results_t*. This structure contains a *summary_status* field which indicates the overall success of the function, and a *results_list* array which lists all error, warning, and informational events that occurred during execution of the function.

The value returned in the *summary_status* is a mask of the following possible values:

SW_NOTE
: The function was successful, but one or more informational events occurred.

SW_WARNING
: The function was successful, but one or more warning events occurred.

SW_ERROR
: The function was not successful for one or more *software_selections*, due to an occurrence of one or more error events.

SW_SUSPEND
: Task execution was suspended due to the occurrence of a suspend event.

The *results_list* field is a variable, conformant array of items which describe each event's result status (that is, error, warning, or note), its result code, and additional result information indicating number of occurrences, etc.

**RETURNS**

No return value (void).

**ERRORS**

Error, warning, and informational events are indicated via the *results* parameter. Refer to the *Result Codes* section for a description of the possible events.

**NAME**

sw_rpc_get_depots - get list of existing distributions or installed software collections

**SYNOPSIS**

```
void sw_rpc_get_depots
(
        [in]            handle_t                daemon_binding,
        [in]            sw_rpc_options_t        *control_option
        [in, out]       sw_rpc_depot_list_t     *depot_list,
        [in, out]       sw_rpc_results_t        *results
);
```

**DESCRIPTION**

The *sw_rpc_get_depots*( ) function returns a list of software distributions or installed software collections that are being served by a particular XDSA daemon. Whether the list contains distributions or installed software collections is determined by the value of the *level* control option. The list is returned via the *depot_list* parameter.

The results of the function are returned via the *results* parameter, which is a structure of type *sw_rpc_results_t*. This structure contains a *summary_status* field which indicates the overall success of the function, and a *results_list* array which lists all error, warning, and informational events that occurred during execution of the function.

The value returned in the *summary_status* is a mask of the following possible values:

SW_NOTE
: The function was successful, but one or more informational events occurred.

SW_WARNING
: The function was successful, but one or more warning events occurred.

SW_ERROR
: The function was not successful, due to the occurrence of one or more error events.

The *results_list* field is a variable, conformant array of items which describe each event's result status (that is, error, warning, or note), its result code, and additional result information indicating number of occurrences, etc.

**RETURNS**

No return value (void).

**ERRORS**

Error, warning, and informational events are indicated via the *results* parameter. Refer to the *Result Codes* section for a description of the possible events.

**NAME**

sw_rpc_get_dsa_impact_data - get disk space analysis impact data

**SYNOPSIS**

```
void sw_rpc_get_dsa_impact_data
(
        [in]            handle_t                agent_binding,
        [in]            sw_rpc_context_t        session_context,
        [in]            sw_rpc_options_t        *control_options,
        [in]            sw_rpc_string_ref_t     filesystem,
        [in, out]       sw_rpc_fsvol_list_t     *fsvol,
        [in, out]       sw_rpc_results_t        *results
);
```

**DESCRIPTION**

The second Disk Space Analysis step involves requesting data showing the impact that installing or removing the specified objects has on a given volume.

This operation is task-specific, since to do the actual calculations, the agent needs to know if the software objects are being installed or removed, and whether or not the operation is being performed on a distribution or installation software collection.

The results of the function are returned via the *results* parameter, which is a structure of type *sw_rpc_results_t*. This structure contains a *summary_status* field which indicates the overall success of the function, and a *results_list* array which lists all error, warning, and informational events that occurred during execution of the function.

The value returned in the *summary_status* is a mask of the following possible values:

SW_NOTE
: The function was successful, but one or more informational events occurred.

SW_WARNING
: The function was successful, but one or more warning events occurred.

SW_ERROR
: The function was not successful, due to the occurrence of one or more error events.

The *results_list* field is a variable, conformant array of items which describe each event's result status (that is, error, warning, or note), its result code, and additional result information indicating number of occurrences, etc.

**RETURNS**

No return value (void).

**ERRORS**

Error, warning, and informational events are indicated via the *results* parameter. Refer to the *Result Codes* section for a description of the possible events.

**NAME**

sw_rpc_get_dsa_volume_list - get disk space analysis volume list

**SYNOPSIS**

```
void sw_rpc_get_dsa_volume_list
(
        [in]            handle_t                agent_binding,
        [in]            sw_rpc_context_t        session_context,
        [in]            sw_rpc_options_t        *control_options,
        [in, out]       sw_rpc_volumes_list_t   *volumes,
        [in, out]       sw_rpc_results_t        *results
);
```

**DESCRIPTION**

The impact of an install or remove task is presented on a per-volume (that is, mounted filesystem) basis. Retrieving Disk Space Analysis information is therefore a two-step process. This call performs the first step, which is to retrieve the list of volumes that comprise the distribution or installation software collection (ideally, but would probably be easier to just list all of the volumes on the destination host).

The results of the function are returned via the *results* parameter, which is a structure of type *sw_rpc_results_t*. This structure contains a *summary_status* field which indicates the overall success of the function, and a *results_list* array which lists all error, warning, and informational events that occurred during execution of the function.

The value returned in the *summary_status* is a mask of the following possible values:

SW_NOTE
: The function was successful, but one or more informational events occurred.

SW_WARNING
: The function was successful, but one or more warning events occurred.

SW_ERROR
: The function was not successful, due to the occurrence of one or more error events.

The *results_list* field is a variable, conformant array of items which describe each event's result status (that is, error, warning, or note), its result code, and additional result information indicating number of occurrences, etc.

**RETURNS**

No return value (void).

**ERRORS**

Error, warning, and informational events are indicated via the *results* parameter. Refer to the *Result Codes* section for a description of the possible events.

**NAME**

sw_rpc_get_soc_file - get software collection file

**SYNOPSIS**

```
void sw_rpc_get_soc_file
(
        [in]            handle_t                agent_binding,
        [in]            sw_rpc_context_t        session_context,
        [in]            sw_rpc_options_t        *control_options,
        [in]            uuid_t                  swobj_uuid,
        [in]            sw_rpc_string_ref_t     swobj_spec,
        [in]            sw_rpc_string_ref_t     file_path,
        [out]           sw_rpc_bytepipe_t       file_contents,
        [in, out]       sw_rpc_results_t        *results
);

} /* interface sdu_rpc_agent */
```

**DESCRIPTION**

The *sw_rpc_get_soc_file*( ) function returns the contents of a logical file contained in a distribution or installed software collection. The *session_context* parameter identifies the session in which the software collection is being managed. The *swobj_uuid* and *swobj_spec* parameters identify the software object within the software collection that contains the requested file. The *file_path* parameter is the relative pathname of the requested logical file.

A logical file can be catalog information, a control file, or a product file that would exist in a software collection conforming to this standard. The *file_path* parameter is the path to the file relative to the directory where catalog, control, or product files for the software object indicated by *swobj_uuid* and *swobj_spec* would exist in a software collection conforming to this standard.

The results of the function are returned via the *results* parameter, which is a structure of type *sw_rpc_results_t*. This structure contains a *summary_status* field which indicates the overall success of the function, and a *results_list* array which lists all error, warning, and informational events that occurred during execution of the function.

The value returned in the *summary_status* is a mask of the following possible values:

SW_NOTE
: The function was successful, but one or more informational events occurred.

SW_WARNING
: The function was successful, but one or more warning events occurred.

SW_ERROR
: The function was not successful, due to the occurrence of one or more error events.

The *results_list* field is a variable, conformant array of items which describe each event's result status (that is, error, warning, or note), its result code, and additional result information indicating number of occurrences, etc.

**RETURNS**

No return value (void).

**ERRORS**

Error, warning, and informational events are indicated via the *results* parameter. Refer to the *Result Codes* section for a description of the possible events.

**NAME**

sw_rpc_get_task_status_and_log - get task status and log

**SYNOPSIS**

```
void sw_rpc_get_task_status_and_log
(
        [in]        handle_t                agent_binding,
        [in]        sw_rpc_context_t        session_context,
        [in]        sw_rpc_options_t        *control_options,
        [out]       sw_rpc_interim_status_t *interim_status,
        [out]       sw_rpc_bytepipe_t       log_data,
        [out]       sw_rpc_session_phase_t  *session_state,
        [in, out]   sw_rpc_results_t        *results
);
```

**DESCRIPTION**

This function retrieves interim status and log entries during task analysis or execution, enabling the caller to gauge progress as these long-duration operations are performed.

The session for which status is retrieved is indicated by *session_context*.

The status data is returned in *interim_status*, which is a data structure containing fields which indicate:

- the current sub-task being performed
- the estimated time to completion
- the total number of kbytes that will be loaded/removed in the operation
- the current number of kbytes loaded/removed so far
- the current software object being operated on.

The results of the function are returned via the *results* parameter, which is a structure of type *sw_rpc_results_t*. This structure contains a *summary_status* field which indicates the overall success of the function, and a *results_list* array which lists all error, warning, and informational events that occurred during execution of the function.

The value returned in the *summary_status* is a mask of the following possible values:

SW_NOTE
: The function was successful, but one or more informational events occurred.

SW_WARNING
: The function was successful, but one or more warning events occurred.

SW_ERROR
: The function was not successful, due to the occurrence of one or more error events.

The *results_list* field is a variable, conformant array of items which describe each event's result status (that is, error, warning, or note), its result code, and additional result information indicating number of occurrences, etc.

**RETURNS**

No return value (void).

**ERRORS**

Error, warning, and informational events are indicated via the *results* parameter. Refer to the *Result Codes* section for a description of the possible events.

**NAME**

sw_rpc_is_registered_depot - check distribution or installed software collection root is registered

**SYNOPSIS**

```
void sw_rpc_is_registered_depot
(
        [in]            handle_t                daemon_binding,
        [in]            sw_rpc_options_t        *control_options,
        [in]            sw_rpc_string_ref_t     depot_path,
        [in, out]       sw_rpc_results_t        *results
);
```

**DESCRIPTION**

The *sw_rpc_is_registered_depot*( ) function checks whether a software distribution or installed software collection is registered on the host (possibly under an alias). Whether the path to check is a depot or root is determined by the value of the *level* control option.

The results of the function are returned via the *results* parameter, which is a structure of type *sw_rpc_results_t*. This structure contains a *summary_status* field which indicates the overall success of the function, and a *results_list* array which lists all error, warning, and informational events that occurred during execution of the function.

The value returned in the *summary_status* is a mask of the following possible values:

SW_NOTE
: The function was successful, but one or more informational events occurred.

SW_WARNING
: The function was successful, but one or more warning events occurred.

SW_ERROR
: The function was not successful, due to the occurrence of one or more error events.

The *results_list* field is a variable, conformant array of items which describe each event's result status (that is, error, warning, or note), its result code, and additional result information indicating number of occurrences, etc.

**RETURNS**

No return value (void).

**ERRORS**

Error, warning, and informational events are indicated via the *results* parameter. Refer to the *Result Codes* section for a description of the possible events.

**NAME**

sw_rpc_register_depot - register distribution or installed software collection

**SYNOPSIS**

```
void sw_rpc_register_depot
(
        [in]            handle_t                daemon_binding,
        [in]            sw_rpc_options_t        *control_options,
        [in]            sw_rpc_string_ref_t     depot_path,
        [in, out]       sw_rpc_results_t        *results
);
```

**DESCRIPTION**

The *sw_rpc_register_depot*( ) function registers the software distributions or installed software collections located in the callee's file system at the path indicated by the *depot_path* parameter. Whether the path to register is a distribution or installed software collection is determined by the value of the *level* control option.

The results of the function are returned via the *results* parameter, which is a structure of type *sw_rpc_results_t*. This structure contains a *summary_status* field which indicates the overall success of the function, and a *results_list* array which lists all error, warning, and informational events that occurred during execution of the function.

The value returned in the *summary_status* is a mask of the following possible values:

SW_NOTE
: The function was successful, but one or more informational events occurred.

SW_WARNING
: The function was successful, but one or more warning events occurred.

SW_ERROR
: The function was not successful, due to the occurrence of one or more error events.

The *results_list* field is a variable, conformant array of items which describe each event's result status (that is, error, warning, or note), its result code, and additional result information indicating number of occurrences, etc.

**RETURNS**

No return value (void).

**ERRORS**

Error, warning, and informational events are indicated via the *results* parameter. Refer to the *Result Codes* section for a description of the possible events.

**NAME**

sw_rpc_unregister_depot - unregister distribution or installed software collection

**SYNOPSIS**

```
void sw_rpc_unregister_depot
(
        [in]            handle_t                daemon_binding,
        [in]            sw_rpc_options_t        *control_options,
        [in]            sw_rpc_string_ref_t     depot_path,
        [in, out]       sw_rpc_results_t        *results
);
```

**DESCRIPTION**

The *sw_rpc_unregister_depot*( ) function deletes the software distribution or installed software collection located in the callee's file system at the path indicated by the *depot_path* parameter from the list of served distributions on the callee's system. Whether the path to unregister is a distribution or installed software collection is determined by the value of the *level* control option.

The results of the function are returned via the *results* parameter, which is a structure of type *sw_rpc_results_t*. This structure contains a *summary_status* field which indicates the overall success of the function, and a *results_list* array which lists all error, warning, and informational events that occurred during execution of the function.

The value returned in the *summary_status* is a mask of the following possible values:

SW_NOTE
: The function was successful, but one or more informational events occurred.

SW_WARNING
: The function was successful, but one or more warning events occurred.

SW_ERROR
: The function was not successful, due to the occurrence of one or more error events.

The *results_list* field is a variable, conformant array of items which describe each event's result status (that is, error, warning, or note), its result code, and additional result information indicating number of occurrences, etc.

**RETURNS**

No return value (void).

**ERRORS**

Error, warning, and informational events are indicated via the *results* parameter. Refer to the *Result Codes* section for a description of the possible events.

Chapter 4

# XDSA-DCE RPC Type Definitions

This chapter describes the DCE IDL for data types used in the XDSA-DCE RPC interface specification. These include types for:

- Strings
- Session Context Handles
- Source and Target Specification
- Host Information
- Task Types
- Control Options
- Result Status
- Result Codes
- Function Results
- Software State
- Session State
- Selections
- Interim Status
- File Transfer
- Disk Space Analysis.

## 4.1 Type Definition Interface

The XDSA-DCE type definition interface is defined as follows:

```
[uuid(59646C2C-B027-11CB-9302-080009199BEB), version(1.0)]
interface sdu_rpc_defs
{

        /* type definitions */

} /* sdu_rpc_defs */
```

## 4.2 Strings

Strings are passed either as RPC *reference* pointers or as RPC *full* pointers. Strings that are declared with the *ptr* attribute (that is, *full*) may be passed with a NULL value, but incur more overhead.

```
/*
 * String data type.  The full pointer version is used in cases
 * where the pointer may be NULL.
 */

typedef    [ref, string]   char    *sw_rpc_string_ref_t;
typedef    [ptr, string]   char    *sw_rpc_string_full_t;
```

## 4.3 Session Context Handles

A session handle is implemented as an RPC context handle, which enables the agent to associate RPCs with a given session. The context mechanism is also used by the agent to detect when the manager dies unexpectedly or communication with the manager is lost.

```
/*
 * The session context type.  This is implemented as an RPC context
 * handle so the agent can detect the death of the RPC client.
 */

typedef    [context_handle]   void    *sw_rpc_context_t;
```

## 4.4 Source and Target Specification

A target or source software collection is identified by the string binding to its host, combined with its absolute path in the host's filesystem. This string binding is the string form of the *daemon_binding* parameter used to initialize an agent via the *sw_rpc_agent_init*( ) function, not a copy of the *agent_string_binding* parameter returned from that function.

```
/*
 * Constructed type for software collection descriptor type.
 */

typedef struct
{

        sw_rpc_string_full_t  soc_binding; /* string binding to daemon */
        sw_rpc_string_full_t  soc_path;    /* absolute path on host */

} sw_rpc_soc_desc_t;
```

## 4.5 Host Information

Host information consists of *uname*( ) style attributes that are used to determine software compatibility.

```
/*
 * Constructed type for passing "uname" attributes.
 */

typedef struct
{
        sw_rpc_string_full_t    sysname;        /* uname sysname */
        sw_rpc_string_full_t    release;        /* uname release */
        sw_rpc_string_full_t    version;        /* uname version */
        sw_rpc_string_full_t    machine;        /* uname machine */
} sw_rpc_uname_attrs_t;
```

The host information also includes information listing the available distributions or installed software collections on the host.

```
/*
 * Constructed type for list of distributions or installed software collections.
 */

typedef struct
{

        unsigned32                      max_depots;
        unsigned32                      num_depots;
        [size_is(max_depots),
           length_is(num_depots)]
                sw_rpc_string_full_t    depot_paths[];

} sw_rpc_depot_list_t;
```

## 4.6 Task Types

The *task type* determines what task the agent will perform on the software object collection during the course of the session (corresponds to the command that was invoked). This is also referred to as the *session type*.

```
/*
 * Type for the set of defined task types.
 */

typedef  unsigned32      sw_rpc_task_t;
```

## 4.7 Control Options

Control options affect a function's behavior, influencing the semantics of the task performed, the way certain events are interpreted, and the data that the function returns.

```
/*
 * Constructed type for passing control options.
 * Options are passed in "keyword=value" strings.
 */

typedef struct
{

        unsigned32                      num_options;
        [size_is(num_options)]
                sw_rpc_string_ref_t     options_list[]; /* options */

} sw_rpc_options_t;
```

## 4.8 Result Status

The result status type is used in three instances:

1. to indicate the overall result of a function upon its return
2. to indicate the effect of each individual result that occurred during a function's execution
3. to indicate the cumulative effect of the results that occurred during task analysis or execution for each individual selected software object.

Values for this type are composed by masking together the individual NOTE, WARNING, ERROR and SUSPEND effect values.

```
/*
 * Mask of the possible NOTE, WARNING, ERROR or SUSPEND
 * effects of one or more results.
 */

typedef         unsigned32      sw_rpc_result_status_t;
```

## 4.9 Result Codes

The result code type is used to identify a result that occurred during a function's execution.

```
/*
 * Identifies a specific result.
 */

typedef         unsigned32      sw_rpc_result_code_t;
```

## 4.10 Function Results

The results of a function are returned as a structure containing an overall summary result status field, and an array of items indicating the code, effect, and auxiliary information identifying each specific result which occurred during the function's execution. The auxiliary information for each result is implemented as an integer whose meaning is specific to a the code, such as a count on the number of times the event occurred, or a media number needed in a tape change.

```
#define SW_MAX_RESULTS      256

/*
 * Constructed types for passing function results.  The value
 * assigned to max_results (by the caller) should be at least
 * SW_MAX_RESULTS.
 */

typedef struct
{

        sw_rpc_result_code_t    result_code;    /* which result */
        sw_rpc_result_status_t  result_status;  /* effect */
        unsigned32              result_info;    /* auxiliary info */

} sw_rpc_result_item_t;

typedef struct
{

        sw_rpc_result_status_t  summary_status;         /* summary of results */
        unsigned32              max_results;            /* max size of array */
        unsigned32              num_results;            /* cur size of array */
        [size_is(max_results),
           length_is(num_results)]
                sw_rpc_result_item_t results_list[];    /* list of results */

} sw_rpc_results_t;
```

## 4.11 Software State

This type is used to indicate the action that the agent will take (as a result of task analysis), or did take (as a result of task execution), for each selected software object.

It is passed back to the manager in the selections array when an analyze, execute or abort task function returns.

```
/*
 * Type for the possible predicted state, or resultant
 * state for each software object.
 */

typedef         unsigned32      sw_rpc_swobj_state_t;
```

## 4.12 Session Phase

This type is used to indicate the current phase of the session. A session's phase is modified (that is. ''transitioned'') via the analyze, execute, and abort task functions. Session phase also determines which functions may be called at a particular point in time.

During analysis and execution, the session phase indicates the current sub-phase being performed, which can be retrieved via the get status and log function.

```
/*
 * Type describing the current state of a session.
 */

typedef         unsigned32      sw_rpc_session_phase_t;
```

## 4.13 Selections

These types are used to indicate the software object selections that are passed to an analyze, execute, or abort task function, and to return result data for each software object. A software object is identified by its software specification, uuid, and the source where the software object may be found.

Software object selections are differentiated as either explicit, or due to a dependency relationship.

Result data consists of the predicted (analysis) or resultant (execution) state, and an indicator of the cumulative effect of the results that occurred during task analysis or execution, for each individual selected software object.

```
/*
 * Constructed types for passing software selections.  The
 * swobj_state and result_status are output fields set by
 * the agent after task analysis or execution has completed.
 */

typedef struct
{

        uuid_t                  swobj_uuid;     /* uuid (not used) */
        sw_rpc_string_ref_t     swobj_spec;     /* software spec */
        sw_rpc_soc_desc_t       soc_desc;       /* source path */
        unsigned32              selection_type; /* explicit vs dependency */
        sw_rpc_swobj_state_t    swobj_state;    /* swobj state */
        sw_rpc_result_status_t  result_status;  /* cumulative status */

} sw_rpc_selection_item_t;

typedef struct
{

        unsigned32                              num_selections;
        [size_is(num_selections)]
                sw_rpc_selection_item_t         selections_list[ ];     /* selections */

} sw_rpc_selections_t;
```

## 4.14 Interim Status

This type is used to indicate the intermediate status of a task's execution.

Interim status is returned by the *sw_rpc_get_task_status_and_log*( ) function.

```
/*
 * Constructed type for retrieving interim status data.
 */

typedef struct
{

        sw_rpc_string_full_t    current_swobj;          /* current swobj */
        unsigned32              time_elapsed;           /* time elapsed thus far */
        unsigned32              time_to_completion;     /* estimated time */
        unsigned32              kbytes_done;            /* kbytes done so far */
        unsigned32              kbytes_total;           /* total kbytes estimated */
        unsigned32              files_done;             /* files done so far */
        unsigned32              files_total;            /* total files estimated */
        unsigned32              percent_done;           /* percentage to completion */

} sw_rpc_interim_status_t;
```

## 4.15 File Transfer

This type is used for the transfer of file contents, such as the files that comprise the software catalog, a software object's actual files, and the log file data for a session.

```
/*
 * General "pipe of bytes" typedef, used in transferring logfile
 * entries and file contents.
 */

typedef         pipe    byte    sw_rpc_bytepipe_t;
```

## 4.16 Disk Space Analysis

These types are used to transfer the data generated by performing disk space analysis. Data is gathered on a per-volume as well as a per-fileset basis.

```
/*
 * This is used for RPC transport of disk space analysis results
 * from the agent to the manager.
 */

typedef struct
{

        /* Volume attributes. */

        sw_rpc_string_full_t    filesystem;     /* filesystem */
        unsigned32              type;           /* type of file system */||
        long                    minfree;        /* percent for this volume */
        long                    blocks;         /* total 1K available */
        long                    free_blocks;    /* 1K before operation */

        /*
        * Volume status.
        * All block sizes are normalized to 1024 bytes/block
        */

        unsigned32      state;          /* file system use */
        unsigned32      num_files;      /* number of files to load on this volume  */
        long            current_blocks; /* current for all selected filesets */
        long            needed_blocks;  /* net for all selected filesets */
        long            till_full;      /* blocks available to absolute limit */
        long            till_warn;      /* blocks available to minfree threshold */

} sw_rpc_volume_t;

typedef struct
{

        unsigned32                      max_volumes;    /* max size of array */
        unsigned32                      num_volumes;    /* cur size of array */
        [size_is(max_volumes),
           length_is(num_volumes)]
                sw_rpc_volume_t         volumes[];      /* list of volumes */

} sw_rpc_volumes_list_t;

/*
 * This is the per volume / per fileset information
 */

typedef struct
{

        sw_rpc_string_full_t    swobj_spec;             /* software spec */
        uuid_t                  swobj_uuid;             /* uuid (not used) */
        unsigned32              fileset_index;          /* index (not used) */
        unsigned32              num_files;              /* Effect of fileset */
        unsigned32              current_blocks;
        unsigned32              needed_blocks;
```

```
} sw_rpc_fsvol_t;

typedef struct
{

        unsigned32              max_fsvol;      /* max size of array */
        unsigned32              num_fsvol;      /* cur size of array */
        [size_is(max_fsvol),
           length_is(num_fsvol)]
                sw_rpc_fsvol_t  fsvol[];        /* list of fsvol entries */

} sw_rpc_fsvol_list_t;
```

*Chapter 5*

# *XDSA-DCE RPC Type Values*

This chapter summarizes the values that each of the enumerated or defined types used in the XDSA-DCE RPC interfaces can take. These types include:

- Task Types
- Selection Types
- Volume Types
- Result Status
- Fileset State
- Session Phase
- Result Codes.

## 5.1 Task Types

The task type determines what task the agent will perform on the software collection during the course of the session (corresponds to the POSIX 1387.2 standard command that was invoked). This is also referred to as the *session type*. It applies to the *task_type* paramenter of *sw_rpc_agent_init*( ) and *sw_rpc_begin_session*( ).

SW_UNDEFINED_TASK[0]
: This is an undefined task (used internally for initialization).

SW_READ_TASK[1]
: The task involves reading files from a source distribution. Note that this task type does not map directly onto an command, but rather the source role.

SW_LIST_TASK[2]
: The task involves reading the database information from an installed software collection.

SW_DLIST_TASK[3]
: The task involves reading the database information from a distribution.

SW_INSTALL_TASK[4]
: The task involves installing software objects to an installed software collection.

SW_COPY_TASK[5]
: The task involves copying software objects to a distribution.

SW_REMOVE_TASK[6]
: The task involves removing software objects from an installed software collection.

SW_DREMOVE_TASK[7]
: The task involves removing software objects from a distribution.

SW_VERIFY_TASK[8]
: The task involves verifying software objects in an installed software collection.

SW_DVERIFY_TASK[9]
: The task involves verifying software objects in a distribution.

SW_CONFIGURE_TASK[10]
: The task involves configuring software objects in an installed software collection.

SW_UNCONFIGURE_TASK[11]
: The task involves unconfiguring software objects in an installed software collection.

SW_MODIFY_TASK[12]
: The task involves modifying the catalog information in a installed software collection.

SW_DMODIFY_TASK[13]
: The task involves modifying the catalog information in a distribution.

## 5.2 Selection Types

This designates whether the software selection passed from the manager to the agent was explicitly selected by the user, or is a dependency automatically included when the *autoselect_dependencies* option is set to TRUE.

It applies to the *selection_type* member of the *software_selections* parameter for *sw_rpc_analyze_task*( ) and *sw_rpc_execute_task*( ).

SW_SELECTED[0]
: The software selection was explicitly selected.

SW_DEPENDENCY_SELECTED[1]
: The software selection was automatically selected to resolve dependencies.

## 5.3 Volume Types and States

The *volume type* designates whether the affected volume is a local file system, remote file system, or read-only file system. This applies to the *type* member of the *volumes* parameter for *sw_rpc_dsa_volume_list*( ).

SW_VOL_NORMAL[0]
The file system is a local writable file system.

SW_VOL_NFS[1]
The file system is a remote writable file system.

SW_VOL_READONLY[2]
The file system is a local or remote read only file system.

The *volume state* designates the summary of the disk space impact on a volume. This applies to the *state* member of the *volumes* parameter for *sw_rpc_dsa_volume_list*( ):

SW_VOL_NOT_USED[0]
Disk space analysis determines the file system is not affected.

SW_VOL_USED[1]
Disk space analysis determines the file system will be affected, but will still have space available to non- superusers.

SW_VOL_WARN[2]
Disk space analysis determines the file system will be affected, will still have free space, but will not have space available to non-superusers.

SW_VOL_FULL[3]
Disk space analysis determines the file system would be affected beyond its capacity.

## 5.4 Software State

For task analysis, the software state indicates the action that the agent will take on this software selection. For task execution, the software state indicates the action that the agent did take on this software selection by describing the state the selection ended up in.

It applies to the *swobj_state* member of the *software_selections* paramenter of *sw_rpc_analyze_task*( ) and *sw_rpc_execute_task*( ).

### 5.4.1 Analysis States

SW_NOT_YET_STATE[8]
The software object has not been analyzed yet.

SW_INSTALLED_STATE[4]
The software object will be a new install.

SW_AVAILABLE_STATE[3]
The software object will be a new copy.

SW_REINSTALLED_STATE[9]
The software object will be a reinstall or recopy.

SW_UPDATED_STATE[10]
The software object will be an update.

SW_DOWNDATED_STATE[11]
The software object will be a downdate.

SW_NEW_VERSION_STATE[12]
The software object will create a new (multiple) version.

SW_REMOVED_STATE[6]
The software object will be removed.

SW_CONFIGURED_STATE[5]
The software object will be configured.

SW_RECONFIGURED_STATE[13]
The software object will be reconfigured.

SW_UNCONFIGURED_STATE[14]
The software object will be unconfigured.

SW_TRANSIENT_STATE[1]
The software object is in the transient state (for verify).

SW_CORRUPT_STATE[2]
The software object is in the corrupt state (for verify).

SW_NON_EXISTENT_STATE[7]
The software object does not exist.

SW_SKIPPED_IN_ANALYSIS_STATE[16]
The software object will be skipped as per defined behavior (for example, reinstall=false) or due to an error (for example, checkinstall error).

### 5.4.2 Execute States

The execute states are either the current state of the software object on the distribution or installed software collection, or are a state that reflects its failure.

SW_NOT_YET_STATE[0]
: The software object has not been processed yet.

SW_INSTALLED_STATE[4]
: The software object resulted in an INSTALLED state.

SW_AVAILABLE_STATE[3]
: The software object resulted in an AVAILABLE state.

SW_REMOVED_STATE[6]
: The software object resulted in a REMOVED state.

SW_CONFIGURED_STATE[5]
: The software object resulted in a CONFIGURED state.

SW_TRANSIENT_STATE[1]
: The software object resulted in a TRANSIENT state.

SW_CORRUPT_STATE[2]
: The software object resulted in a CORRUPT state.

SW_NON_EXISTENT_STATE[7]
: The software object does not exist.

SW_SKIPPED_IN_ANALYSIS_STATE[16]
: The software object was determined to be skipped in analysis.

SW_SKIPPED_IN_EXECUTION_STATE[15]
: The software object has been skipped before processing this software object, possibly due to an error in related software.

## 5.5 Session Phase

This is the current phase or state of the task, which is used to determine which RPC functions can be called, as well as what phase the agent is in. These are all target role phases or states, with the exception of the SW_SOURCE_PHASE (source role only).

It applies to the *session_state* parameter of *sw_rpc_analyze_task*( ), *sw_rpc_execute_task*( ), *sw_rpc_abort_task*( ) and *sw_rpc_get_task_status_and_log*( ).

### 5.5.1 Static Phases

These are the states that the target session can be in between RPC calls.

SW_INITIAL_PHASE[0]
: This is the state the agent is in before a source or target session has begun. The function *sw_rpc_begin_session*( ) is allowed from this state.

SW_SOURCE_PHASE[18]
: The agent is initialized to this state if the *sw_rpc_begin_session*( ) call for an SW_READ_TASK type was successful. The function *sw_rpc_get_soc_file*( ) is allowed from this state.

SW_SELECTION_PHASE[1]
: The agent is initialized to this state if the *sw_rpc_begin_session*( ) call for a task type besides SW_READ_TASK was successful. If a global error occurs during task analysis, then the session will return from this state. The following functions:
  *sw_rpc_analyze_task*( )
  *sw_rpc_get_soc_file*( )
  *sw_rpc_get_status_and_log*( )
  *sw_rpc_end_session*( )
  are allowed from this state.

SW_ANALYSIS_COMPLETED_PHASE[2]
: Task analysis has completed without a global error, and task execution is now possible. The following functions:
  *sw_rpc_execute_task*( )
  *sw_rpc_get_dsa_volume_list*( )
  *sw_rpc_get_dsa_impact_data*( )
  *sw_rpc_get_status_and_log*( )
  *sw_rpc_end_session*( )
  are allowed from this state.

SW_SUSPENDED_PHASE[3]
: The task execution has been suspended. The functions *sw_rpc_get_status_and_log*( ) , *sw_rpc_execute_task*( ) and *sw_rpc_abort_task*( ) are allowed from this state.

SW_READY_FOR_REBOOT_PHASE[4]
: The agent has completed execution of an SW_INSTALL_TASK and the agent is waiting for a confirmation to reboot. The functions *sw_rpc_get_status_and_log*( ) and *sw_rpc_end_session*( ) are allowed from this state.

SW_ABORTED_PHASE[5]
: The agent entered the suspended state and was instructed to abort the task. The functions *sw_rpc_get_status_and_log*( ) and *sw_rpc_end_session*( ) are allowed from this state.

SW_COMPLETED_PHASE[6]
: The task execution has completed. The functions *sw_rpc_get_status_and_log*( ) and *sw_rpc_end_session*( ) are allowed from this state.

### 5.5.2 Analyzing Phases

These are the phases that the target session can be in during an analyze task call. They are only used for display purposes, and can be retrieved via *sw_rpc_get_status_and_log*( ) function.

SW_ANALYZING_PHASE[7]
Task analysis is in progress.

SW_ANALYZING_TARGET_PHASE[8]
The target checks are being done.

SW_ANALYZING_SOFTWARE_PHASE[9]
The software selections are being checked.

SW_ANALYZING_SCRIPTS_PHASE[10]
The check or verify scripts are being executed.

SW_ANALYZING_FILES_PHASE[11]
The files are being checked.

### 5.5.3 Executing Phases

These are the states that the target session can be in during an analyze task call. They are only used for display purposes, and can be retrieved via *sw_rpc_get_status_and_log*( ) function.

SW_EXECUTING_PHASE[12]
Task execution is in progress.

SW_EXECUTING_PRESCRIPT_PHASE[13]
A preinstall or preremove script is being executed.

SW_EXECUTING_FILES_PHASE[14]
The files are being loaded or removed.

SW_EXECUTING_POSTSCRIPT_PHASE[15]
A postinstall or postremove script is being executed.

SW_EXECUTING_CONFIGURE_PHASE[16]
A configure or unconfigure script is being executed.

SW_EXECUTING_KERNEL_BUILD_PHASE[17]
A kernel build command is being executed.

## 5.6 Result Status

Result status is used in three instances:

1. to indicate the overall result of a function upon its return
2. to indicate the ‘‘effect’’ of each individual result code (event) that occurred during a function’s execution
3. to indicate the cumulative effect of the results that occurred during task analysis or execution for each individual selected software object.

It applies to the *result_status* and *summary_status* members of the *results* parameter for all calls. It also applies to the *result_status* member of the *selections* parameter for *sw_rpc_analyze_task*( ) and *sw_rpc_execute_task*( ).

SW_NOTE[0001]
This event was interpreted as informational only.

SW_WARNING[0002]
This event was interpreted as a warning.

SW_ERROR[0004]
This event was interpreted as an error.

SW_SUSPEND[0008]
This event caused a suspend of the task.

The result status (severity of a result code) can change based on the control options for the task (for example, a ‘‘dependency not met’’ result code can be an ERROR or a WARNING depending on the *enforce_dependencies* control options).

## 5.7 Result Codes

Result codes exist for each type of event that can occur during an agent task. The results array returned from each RPC function is a list of (result code, result status, result info) tuples. Thus, the results array summarizes the events that occurred during the RPC function.

It applies to the *result_code* member of the *results* paramenter for all calls.

Listed below are the result codes that can be returned for each function, their associated result status values, and the meaning of the auxiliary info (if other than number of occurrences of this event).

### 5.7.1 Generic RPC Result Codes

SW_ILLEGAL_STATE_TRANSITION[1]
The agent is in the wrong state to accept this call.
ERROR: always.
The *result_info* contains the current state.

SW_BAD_SESSION_CONTEXT[2]
The session context sent does not identify a valid session.
ERROR: always.

SW_ILLEGAL_OPTION[3] An illegal or unrecognized option was sent.
ERROR: always.
The *result_info* contains the number of options.

SW_ACCESS_DENIED[4]
The user has insufficient privilege to perform the requested operation.
ERROR: always.

SW_MEMORY_ERROR[5]
The agent had a memory allocation error (for example, out of swap).
ERROR: always.

SW_RESOURCE_ERROR[6]
The agent had a resource allocation error such a maximum number of processes, number of files open, etc.
ERROR: always.

SW_INTERNAL_ERROR[7]
The agent had an internal implementation error.
ERROR: always.

### 5.7.2 Get Distributions Result Codes

SW_MORE_DATA[20]
More distributions are registered than fit in the structure passed to this call.
WARNING: always.
The *result_info* contains the number of registered distributions.

### 5.7.3 Register Distribution Result Codes

SW_ALREADY_REGISTERED[22]
Distribution to register is already registered (possibly under a different filesystem pathname).
NOTE: always.

SW_SOC_DOES_NOT_EXIST[31]
The requested target or source (distribution or installed software collection) does not exist.
ERROR: always.

### 5.7.4 Unregister Distribution Result Codes

SW_NOT_REGISTERED[23]
Distribution to unregister is not registered.
ERROR: always.

### 5.7.5 Is Distribution Registered Result Codes

SW_NOT_REGISTERED[23]
The distribution or installed software collection specified is not registered.
ERROR: always.

### 5.7.6 Initialize Agent / Begin Session Result Codes

SW_AGENT_INITIALIZATION_FAILED[10]
Failed to initialize an agent.
ERROR: always.

SW_SERVICE_NOT_AVAILABLE[11]
The daemon is in the process of an orderly shutdown and not accepting requests for new sessions.
ERROR: always.

SW_OTHER_SESSIONS_IN_PROGRESS[12]
There are other sessions in progress that may affect the results of this task.
WARNING: always.

SW_CONNECTION_LIMIT_EXCEEDED[30]
The limit of existing connections to this source software collection has already been reached.
ERROR: always.

SW_SOC_DOES_NOT_EXIST[31]
The requested target or source (distribution or installed software collection) does not exist.
ERROR: unless install or copy task.

SW_SOC_IS_CORRUPT[32]
The software collection exists, but the information is corrupt. It needs to be fixed or removed.
ERROR: always.

SW_SOC_CREATED[34]
The target software collection did not previously exist and was created.
NOTE: always.

SW_CONFLICTING_SESSION_IN_PROGRESS[35]
The software collection is already locked by another session. A read or write lock on the software collection was denied.

WARNING: if a list task read lock was attempted.
ERROR: otherwise.

SW_SOC_LOCK_FAILURE[36]
Cannot lock onto the target due to an external error (for example, lock file not creatable).
ERROR: always.

SW_SOC_IS_READ_ONLY[37]
The software collection is located on a read-only filesystem.
ERROR: install, copy and remove tasks.
NOTE: otherwise.

SW_SOC_IS_REMOTE[38]
The software collection is located on a remote filesystem (for example, NFS).
ERROR: if *allow_remote_filesystem_writes* option is false for install, copy and remove.
NOTE: otherwise.

SW_SOC_INCORRECT_MEDIA_TYPE[39]
The software collection is not the correct media type for the requested task, for example, the software collection is tape and the task is a distribution copy or remove.
ERROR: always.

SW_SOC_IS_SERIAL[40]
The distribution software collection is in serial format.
NOTE: always.

SW_SOC_INCORRECT_MEDIA_TYPE[41]
The software collection is not the correct type (distribution or installed software collection) for the task. For example, it is an installed software collection and the task was for a distribution.
ERROR: always.

SW_CANNOT_OPEN_LOGFILE[42]
Cannot open the software collection logfile.
ERROR: always.

SW_SOC_AMBIGUOUS_TYPE[49]
The software collection is of the wrong type (distribution or installed software) for the operation.
ERROR: always.

### 5.7.7 End Session Result Codes

SW_TERMINATION_DELAYED[50]
The agent is currently analyzing or executing a task and will terminate the session once completed.
NOTE: always.
The *result_info* contains the current state.

SW_CANNOT_INITIATE_REBOOT[51]
The agent failed to initiate the reboot operation of an install task. It must be manually executed.
WARNING: always.

### 5.7.8 Analyze Task Result Codes

SW_EXREQUISITE_EXCLUDE[56]
One or more filesets were excluded automatically as software identified as exrequisites was also specified to be selected.
NOTE: always.

SW_CHECK_SCRIPT_EXCLUDE[57]
One or more checkinstall or checkremove scripts have caused the software to be unselected and excluded from further processing.
NOTE: always.

SW_CONFIGURE_EXCLUDE[58]
One or more configure or unconfigure scripts have caused the software to be unselected and excluded from further processing.
NOTE: always.

SW_SELECTION_IS_CORRUPT[59]
The software selection was found, but its state was *corrupt* or *transient*.
ERROR: always.

SW_SOURCE_ACCESS_ERROR[60]
Generic failure contacting or retrieving information from the source.
ERROR: always.

SW_SOURCE_NOT_FIRST_MEDIA[61]
The source does not have a media number of 1 (needed for retrieval of the INDEX).
ERROR: always.

SW_SELECTION_NOT_FOUND[62]
One or more software selections can not be found.
ERROR: install or copy.
NOTE: otherwise.
The *result_info* contains the number of filesets.

SW_SELECTION_NOT_FOUND_RELATED[63]
One or more software selections can not be found as specified, but the same product tag exists.
ERROR: install or copy.
NOTE: otherwise.
The *result_info* contains the number of filesets.

SW_SELECTION_NOT_FOUND_AMBIG[64]
One or more software selections can not be unambiguously determined.
ERROR: install or copy.
NOTE: otherwise.
The *result_info* contains the number of filesets.

SW_FILESYSTEMS_NOT_MOUNTED[65]
One or more filesystems in the filesystem table are not mounted.
ERROR: If *mount_all_filesystems* option is true.
WARNING: If *mount_all_filesystems* option is false.
The *result_info* contains the number of unmounted filesystems.

SW_FILESYSTEMS_MORE_MOUNTED[66]
One or more filesystems mounted are not in filesystem table.
WARNING: always.
The *result_info* contains the number of extra filesystems mounted.

SW_HIGHER_REVISION_INSTALLED[67]
One or more filesets have a higher revision already installed.
ERROR: If *allow_downdate* option is false.
WARNING: If *allow_downdate* option is true.
The *result_info* contains the number of filesets.

SW_NEW_MULTIPLE_VERSION[68]
One or more products would create a new version in an installed software collection.
ERROR: If *allow_multiple_versions* option is false.
NOTE: If *allow_multiple_versions* option is true.
The *result_info* contains the number of products.

SW_EXISTING_MULTIPLE_VERSION[69]
The task is operating on an existing multiple version of one or more products.
ERROR: If trying to install two versions into one location at the same time.
WARNING: If *allow_multiple_versions* option is false.
NOTE: If *allow_multiple_versions* option is true.
The *result_info* contains the number of products.

SW_DEPENDENCY_NOT_MET[70]
One or more dependencies can not be met.
ERROR: If *enforce_dependencies* option is true.
WARNING: If *enforce_dependencies* option is false.
The *result_info* contains the number of filesets.

SW_NOT_COMPATIBLE[71]
One or more products are incompatible for this target.
ERROR: If *allow_incompatible* option is false.
WARNING: If *allow_incompatible* option is true.
The *result_info* contains the number of products.

SW_CHECK_SCRIPT_WARNING[72]
One or more checkinstall, checkremove or verify scripts had a warning.
WARNING: always.
The *result_info* contains the number of filesets.

SW_CHECK_SCRIPT_ERROR[73]
One or more checkinstall, checkremove or verify scripts failed.
ERROR: always.
The *result_info* contains the number of filesets.

SW_DSA_INTO_MINFREE[74]
DSA shows that the space requirements encroach into minfree, but not over the absolute limit on one or more volumes.
ERROR: If *enforce_dsa* option is true.
WARNING: If *enforce_dsa* option is false.
The *result_info* contains the number of volumes that are beyond minfree.

SW_DSA_OVER_LIMIT[75]
DSA shows that the space requirements are over the absolute limit on one or more volumes.
ERROR: If *enforce_dsa* option is true.
WARNING: If *enforce_dsa* option is false.
The *result_info* contains the number of volumes that are beyond minfree.

SW_DSA_FAILED_TO_RUN[76]
DSA had an internal error and failed to run.
ERROR: If *enforce_dsa* option is true.

WARNING: If enforce_dsa option is false.

SW_SAME_REVISION_INSTALLED[77]
One or more filesets have the same revision and are being reinstalled or recopied because the *reinstall* or *recopy* option is true.
NOTE: always.
The *result_info* contains the number of filesets.

SW_ALREADY_CONFIGURED[78]
One or more filesets are already configured.
NOTE: If *reconfigure* option is true.
NOTE: If *reconfigure* option is false.
The *result_info* contains the number of filesets.

SW_SKIPPED_PRODUCT_ERROR[79]
One or more filesets will be skipped because of another error within their product.
NOTE: always.
The *result_info* contains the number of filesets.

SW_SKIPPED_GLOBAL_ERROR[80]
One or more filesets will be skipped because of a global error (such as DSA failure) within the analyze phase.
NOTE: always.
The *result_info* contains the number of filesets.

SW_FILE_IS_REMOTE[81]
One or more files would be created or removed on a remote filesystem.
WARNING: if *write_remote_files* option is false.
NOTE: if *write_remote_files* option is true.
The *result_info* contains the number of files.

SW_FILE_IS_READ_ONLY[82]
One or more files will not be attempted to be created or removed over a read only filesystem mount.
WARNING: always.
The *result_info* contains the number of files.

SW_FILE_NOT_REMOVABLE[83]
One or more files will not be able to be removed (text busy, or non empty directories).
WARNING: always.
The *result_info* contains the number of files.

SW_FILE_WARNING[84]
One or more files had warnings (for example, for verify).
WARNING: always.
The *result_info* contains the number of files.

SW_FILE_ERROR[85]
One or more files had errors (for example, for verify).
ERROR: always.
The *result_info* contains the number of files.

SW_NOT_LOCATABLE[86]
One or more filesets are not locatable and have been specified with an alternate location.
ERROR: if the *enforce_locatable* option is true.
WARNING: if the *enforce_locatable* option is false.

SW_SAME_REVISION_SKIPPED[87]
One or more filesets have the same revision and are not being reinstalled or recopied because the *reinstall* or *recopy* option is false.
NOTE: always.

### 5.7.9 Execution Result Codes

SW_SELECTION_NOT_FOUND[62]
One or more software selections can not be found.
ERROR: install or copy.
NOTE: otherwise.
The *result_info* contains the number of filesets.

SW_FILE_IS_REMOTE[81]
One or more files were skipped (or written if *write_remote_files* option is false) over a remote filesystem.
WARNING: If *write_remote_files* option is false for install, copy and remove.
NOTE: If *allow_remote_filesystem_writes* option is true.
The *result_info* contains the number of files.

SW_FILE_IS_READ_ONLY[82]
One or more files were skipped over a read only filesystem mount.
WARNING: for install, copy and remove.
The *result_info* contains the number of files.

SW_FILE_NOT_REMOVABLE[83]
One or more files that were not be able to be removed (for remove or update).
WARNING: always.
The *result_info* contains the number of files.

SW_FILE_WARNING[84]
One or more files had other load or remove warnings.
WARNING: always.
The *result_info* contains the number of files.

SW_FILE_ERROR[85]
One or more files that had other load or remove errors.
ERROR: always.
The *result_info* contains the number of files.

SW_SELECTION_SKIPPED_IN_ANALYSIS[90]
One or more selections have already been determined to be skipped in analysis.
NOTE: always.
The *result_info* contains the number of filesets.

SW_SELECTION_NOT_ANALYZED[91]
One or more software selection were found, but were not analyzed.
ERROR: always.
The *result_info* contains the number of filesets.

SW_WRONG_MEDIA_SET[92]
The agent failed to reopen access to physical source media, new media's software collection uuid does not match original media's software collection uuid, or a new or old software collection uuid is invalid format or invalid value.
ERROR: always.

SW_NEED_MEDIA_CHANGE[93]
The agent has suspended for a media change.
NOTE: always.
SUSPEND: always.
The *result_info* contains the number of the needed media.

SW_CURRENT_MEDIA[94]
The agent currently is accessing this media number (for example, in the drive).
NOTE: always.
The *result_info* contains the number of the current media.

SW_PRE_SCRIPT_WARNING[95]
One or more preinstall or preremove scripts had a warning.
WARNING: always.
The *result_info* contains the number of filesets.

SW_PRE_SCRIPT_ERROR[96]
One or more preinstall or preremove scripts failed.
ERROR: always.
SUSPEND: if a kernel preinstall script.
The *result_info* contains the number of filesets.

SW_FILESET_WARNING[97]
One or more filesets had warnings.
WARNING: always.
The *result_info* contains the number of filesets.

SW_FILESET_ERROR[98]
One or more filesets had an error.
ERROR: always.
SUSPEND: if a kernel fileset had an error.
The *result_info* contains the number of filesets.

SW_POST_SCRIPT_WARNING[99]
One or more postinstall or postremove scripts had a warning.
WARNING: always.
The *result_info* contains the number of filesets.

SW_POST_SCRIPT_ERROR[100]
One or more postinstall or postremove scripts failed.
ERROR: always.
SUSPEND: if a kernel postinstall script.
The *result_info* contains the number of filesets.

SW_POSTKERNEL_WARNING[101]
The kernel build script had a warning.
WARNING: always.

SW_POSTKERNEL_ERROR[102]
The kernel build script failed.
ERROR: always
SUSPEND: always.

SW_CONFIGURE_WARNING[103]
One or more configure or unconfigure scripts had a warning.
WARNING: always.
The *result_info* contains the number of filesets.

SW_CONFIGURE_ERROR[104]
One or more configure or unconfigure scripts failed.
ERROR: always.
The *result_info* contains the number of filesets.

SW_DATABASE_UPDATE_ERROR[105]
An update to the installed software collection catalog or distribution catalog failed.
ERROR: always.

### 5.7.10 Abort Task Result Codes

There are no additional result codes for this function.

### 5.7.11 Get Status and Log Result Codes

There are no additional result codes for this function.

### 5.7.12 Get DSA Volumes Result Codes

SW_MORE_VOLUMES_AVAILABLE[110]
There are more volumes than were allocated to return.
WARNING: always.
The *result_info* contains the total number of volumes available.

### 5.7.13 Get DSA Impact Data Result Codes

SW_FILESET_NOT_COMPUTED[111]
One or more filesets requested for disk space has not be computed.
WARNING: always.
The *result_info* contains the number of filesets.

### 5.7.14 Get Software Collection File Result Codes

SW_ACCESS_DENIED[4]
The user has insufficient privilege to perform the requested operation.
ERROR: always.

SW_SELECTION_NOT_FOUND[62]
The software selection can not be found.
ERROR: always.

SW_IO_ERROR[8]
Failed getting the file due to I/O error.
ERROR: always.

SW_SOURCE_ACCESS_ERROR[60]
Generic source access failure.
ERROR: always.

SW_CANNOT_COMPRESS[112]
Request for compressed file is refused due to no support or lack of cpu resources. Re-request the file without compression.
ERROR: always.

SW_FILE_NOT_FOUND[113]
File is missing from remote software collection.
ERROR: always.

## 5.8 Options

Options are passed as *name=value* strings in the *control_options* parameter of each RPC call. This section describes the POSIX 1387.2 standard as well as the XDSA-DCE options that the XDSA-DCE implementation currently recognizes.

The RPC interface is designed to be extensible for implementation specific extensions. If a server (daemon or agent) gets an option it does not recognize however, the call fails (see result codes). The caller can retry the call with a different set of options, for example only using the options defined in the standard.

### 5.8.1 Register Options

These are passed in the *sw_rpc_get_depots*( ), *sw_register_depot*( ), *sw_unregister_depot*( ) and *sw_rpc_is_depot_registered*( ) functions.

The POSIX 1387.2 standard options that are passed via the RPC include he following (with the default values when used with these functions):

*installed_software_catalog=*

*level=depot*

### 5.8.2 Analyze and Execute Task Options

These are passed in the *sw_rpc_analyze_task*( ) function for the appropriate commands (the commands to which they apply). Each call to analyze task resets these options (so that different analysis behavior can be effected. Only the loglevel option affects execute task.

The POSIX 1387.2 standard options that are passed via the RPC include the following (with their default values):

*allow_downdate=false*

*allow_incompatible=false*

*allow_multiple_versions=false*

*autoreboot=false*

*autorecover=false*

*autoselect_dependencies=true*

*autoselect_dependents=false*

*check_contents=true*

*check_permissions=true*

*check_requisites=true*

*check_scripts=true*

*check_volatile=false*

*compress_files=false*

*compression_type=implementation_defined_value*

*defer_configure=false*

*enforce_dependencies=true*

*enforce_locatable=true*

*enforce_scripts=true*

*enforce_dsa=true*

*files=*

*installed_software_catalog=implementation_defined_value*

*loglevel=1*

*reconfigure=false*

*recopy=false*

*reinstall=false*

*uncompress_files=false*

The rest of the POSIX 1387.2 standard options are resolved in the context of the manager (for example for source and target selection).

XDSA-DCE supports additional options to control the behavior of the analyze and execute operations.

*attributes=*
Defines the list of attributes supplied via the POSIX 1387.2 standard *-a* option. The value is a space separated list of *attribute=value* pairs. Since the list is space separated, any value containing whitespace needs to be quoted.

Applies to **swmodify**.

*catalog=*
Defines the path to the catalog file or directory supplied via the POSIX 1387.2 standard *-c* option.

Applies to **swmodify**, **swinstall** and **swconfig**.

*use_alternate_source=false*
Empowers each target agent to use its own configured alternate source, instead of the one specified by the user. If false, each target agent will use the same source, namely the source specified by the user and validated by the command.

Applies to **swcopy** and **swinstall**.

*register_new_depot=true*
Register a newly created distribution with the local host object. This action allows other XDSA-DCE commands to automatically ‘‘see’’ this distribution. If set to false, a new distribution will not be automatically registered. (It can be registered later with the **swreg** command.)

Applies to **swcopy**.

*register_new_root=true*
Register a newly created installed software collection with the local host object. This action allows other XDSA-DCE commands to automatically ‘‘see’’ this installed software collection. If set to false, a new installed software collection will not be automatically registered. (It can be automatically registered later with the **swreg** command).

Applies to **swinstall**.

*mount_all_filesystems=true*
Attempt to automatically mount all filesystems in the filesystem table at the beginning of the analysis phase, to ensure that all listed filesystems are mounted before proceeding. This policy helps to ensure that files which may be on unmounted filesystems are available. If set to false, the mount operation is not attempted, and no check of the current mounts is performed.

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

*reinstall_files=false*
Causes all the files in a fileset to always be re-installed, even when the file already exists at the target and is identical to the new file. If set to false, files that have the same checksum (see next option), size and timestamp will not be re-installed. This check enhances performance on slow networks or slow discs.

Applies to **swcopy** and **swinstall**.

*reinstall_files_use_cksum=true*
When the *reinstall_files* option is set to false, this option causes the checksums of the new and old file to be compared to determine if the new file should replace the old one. If set to false, the checksums are not computed, and files are (not) reinstalled based only on their size and timestamp.

Applies to **swcopy** and **swinstall**.

### 5.8.3 Get Status and Log Options

These are passed in the *sw_rpc_get_task_status_and_log*( ) function.

*get_interim_status=true*
Used to designate getting interim status.

*get_log_data=true*
Used to designate getting log data.

*reset_log_offset=true*
Used to get the log data from the beginning, as opposed to only from the offset last retrieved.

### 5.8.4 Get Software Collection File Options

These options are passed in the *sw_rpc_get_soc_file*( ) function:

*soc_control_file=true*
Used to designate whether the file requested is a regular file or a control file.

The POSIX 1387.2 standard options that are passed via the RPC include the following:

*compress_files=false*
When set to true, the file is compressed before transfer.

*compression_type=implementation_defined_value*
Identifies the compression type, if compression is used.

### 5.8.5 Miscellaneous RPC Options

XDSA-DCE defines other options related to the RPC listed here:

*rpc_binding_info=ncadg_ip_udp:[2121]*
Defines the protocol sequences and endpoints which should be used to contact the daemon when DCE naming is not being used. This is not passed through an RPC but used to contact the remote daemon.

Applies to all commands.

*rpc_timeout=5*
Relative length of the communications timeout. Higher values mean longer times; you may need a higher value for a slow or busy network. Also passed through the RPC and used by the target when contacting the source.

Applies to all commands.

*polling_interval=2*
Defines the polling interval used by interactive (GUI) sessions. It specifies how often each target agent will be polled to obtain status information about the task being performed. When operating across wide-area networks, the polling interval can be increased to reduce network overhead.

Applies to all commands.

### 5.8.6 DCE Naming Service Options

These options are used to define the interface to DCE naming. They are not passed via the RPC, but used to find daemons when DCE naming is used.

*use_dce_directory_service=false*
When set to false, there is no requirement for DCE naming. When set to true, naming features for the commands are supported. These features include retrieving target and source daemons, including the rpc binding to those daemons.

*dce_server_group=/.:/subsys/XDSA_DCE/server_group*
The default group name in the DCE namespace where the daemon registers its server name. This allows all target servers to be found easily. This option only applies when DCE naming services are available.

*dce_depot_group=/.:/subsys/XDSA_DCE/depot_group*
The default group name in the DCE namespace where the daemon registers its server name if it is serving one or more distributions. This allows all source servers to be found easily. This option only applies when DCE naming services are available.

*dce_server_register=/.:/hosts/%s/XDSA_DCE*
The default location in the DCE namespace for the daemon to register itself. The ‘‘%s’’ is replaced by the local hostname. This option only applies when DCE naming services are available.

### 5.8.7 DCE Security Service Options

These options are used to define the interface to DCE security. The authentication_service and protection_level are not passed via the RPC, and are configurable by the user. The others are automatically generated and are communicated to the target and source daemons and agents.

*authentication_service=internal*
: When set to *internal*, there is no requirement for DCE security service. The internal security mechanism is used. When set to *dce_secret*, DCE security service is used.

*protection_level=none*
: This is the protection level for DCE data protection ( *none*, *connect*, *call*, *pkt*, *pkt_integ*, *pkt_privacy*). The manager specifies the protection level for any particular session. The daemon and agent can be configured for what minimum level of protection they will accept.

*internal_authn_user=*
: This is how the user part of the caller information (principals) is passed across the RPC when XDSA-DCE internal security is used.

*internal_authn_group=*
: This is how the group part of the caller information (principals) is passed across the RPC when XDSA-DCE internal security is used.

*internal_authn_realm=*
: This is how the realm part of the caller information (principals) is passed across the RPC when XDSA-DCE internal security is used.

*internal_authn_secret=*
: This is the encrypted secret used for XDSA-DCE internal security authentication. For the daemon or agent to accept a request, they must have previously been configured for this secret.

*initiator_user=*
: These are used to support delegation for both internal and DCE security modes. This is how the user part of the initiator information is passed from the target agent to the source agent (the initiator being the manager user).

*initiator_group=*
: These are used to support delegation for both internal and DCE security modes. This is how the group part of the initiator information is passed from the target agent to the source agent (the initiator being the manager user).

*initiator_realm=*
: These are used to support delegation for both internal and DCE security modes. This is how the realm part of the initiator information is passed from the target agent to the source agent (the initiator being the manager user).

*Chapter 6*

# *XDSA-DCE Utilities*

As XDSA-DCE is an extension to the POSIX 1387.2 standard, This XDSA-DCE specification assumes the definition for all utilities defined in that standard. Additionally, XDSA-DCE requires one new utility (**swreg**), and extensions to one existing POSIX 1387.2 standard utility (**swlist**):

- **swreg** — register or unregister distributions and installed software collections
- **swlist** — list registered distributions and installed software collections.

This chapter describes these new and extended utilities.

If DCE Naming Services are used, these utilities are also used to register and list registered hosts serving distributions and installed software collections.

## NAME

swreg — register or unregister distributions and installed software collections

## SYNOPSIS

**swreg [-u] -l level [-f object_file] [-t target_file] [-x option=value] [-X option_file] [objects_to_register] [ @ target_selections]**

## DESCRIPTION

The **swreg** utility controls the visibility of distributions and installed software collections to users who are performing software management tasks.

By default, the **swcopy** utility registers newly created distributions and the **swinstall** utility registers newly created alternate installed software collections. By default, the **swpackage** utility does not register newly created distributions. The **swremove** utility unregisters a distribution or installed software collection if the distribution or installed software collection is empty.

The user invokes **swreg** to explicitly register or unregister a distribution or installed software collection when the automatic behaviors of **swcopy**, **swinstall**, **swpackage** and **swremove** do not suffice. For example:

- making a CD-ROM or other removable media available as a registered distribution
- registering a distribution created directly by **swpackage**
- unregistering a distribution without removing it with **swremove**.

When using DCE Naming Services (see referenced document **DCE Directory**), **swreg** also registers the name of the host serving the distribution or installed software collection in the appropriate XDSA-DCE group if not already done previously for other distributions or installed software collections.

## OPTIONS

The **swreg** utility supports the following options:

-f *object_file*
: Read the list of distribution or installed software collection objects to register or unregister from *object_file* instead of (or in addition to) the command line.

-l *level*
: Specify the level of the object to register or unregister. Exactly one level must be specified. The supported levels are:

  — *installed software collection*
  : The object to be registered is a root.

  — *distribution*
  : The object to be registered is a depot.

-t *target_file*
: Read the list of target hosts on which to register the distribution or installed software collection objects from *target_file* instead of (or in addition to) the command line.

-u
: Causes **swreg** to unregister the specified objects instead of registering them.

-x *option=value*
: Set the *session* option to *value* and override the default value (or a value in an alternate *option_file* specified with the *-X* option). Multiple *-x* options can be specified.

-X *option_file*
Read the session options and behaviors from *option_file*.

**OPERANDS**

The **swreg** utility supports the following syntax for each *object_to_register*. Each path specifies a distribution or installed software collection to be registered or unregistered:

**path**

The **swreg** utility supports the following syntax for each target_selection:

**host**

**EXTERNAL INFLUENCES**

The **swreg** utility supports the following extended options:

*distribution_target_directory=implementation_defined_value*
Defines the location of the distribution object to register if no objects are specified and the level is *distribution*.

*installed_software_catalog=implementation_defined_value*
Specifies the installed software collection catalog that is associated with each installed software collection path to be registered, if different than the implementation default.

*level=*
Defines the default level of objects to register. The valid levels are *root* and *depot*.

*logfile=implementation_defined_value*
This is the default command log file for the **swreg** command.

*loglevel=1*
Controls the amount of output sent by the utility to log files.

*objects_to_register=*
Defines the default objects to register or unregister. There is no supplied default (see *distribution_target_directory* above). If there is more than one object, they must be separated by spaces.

*rpc_binding_info=ncadg_ip_udp:[2121]*
Defines the protocol sequence and endpoint which should be used to contact the daemon.

*rpc_timeout=5*
Relative length of the communications timeout.

*select_local=true*
If no *target_selections* are specified, select the local host as the target of the command.

*targets=*
Defines the default target hosts on which to register or unregister the specified distributions or installed software collections.

There is no supplied default (see *select_local* above). If there is more than target selection, they must be separated by spaces.

*verbose=1*
Controls the amount of output sent by the utility to **stdout** and **stderr**, but not to **logfiles**.

**EXTERNAL EFFECTS**

stdout
The **swreg** utility writes events with a status of SW_NOTE to **stdout** if permitted by the value of the *verbose* option.

stderr

The **swreg** utility writes events with a status of SW_WARNING or SW_ERROR to **stderr**.

logging

The **swreg** utility logs summary events at the host where the command was invoked. It logs events about each **register** or **unregister** operation to the daemon logfile associated with each *target_selection*.

**EXTENDED DESCRIPTION**

The **swreg** utility contacts the daemon on the host targets specified, providing the distribution or installed software collection paths to register or unregister to each daemon. The daemon performs the following checks for each distribution or installed software collection:

- If attempting to register a distribution or installed software collection does not exist, generate an event:
  (SW_ERROR: SW_SOC_DOES_NOT_EXIST)
- If attempting to register a distribution or installed software collection that is already registered, issue an event:
  (SW_NOTE: SW_ALREADY_REGISTERED)
- If attempting to unregister a distribution or installed software collection that is not registered, issue an event:
  (SW_ERROR: SW_NOT_REGISTERED)

If these checks succeed, the daemon then registers the distribution or installed software collection in a local ‘‘host catalog’’. The format of the host catalog is undefined.

If the *use_dce_directory_service* option is set to TRUE, then the daemon also registers its server name in the appropriate XDSA-DCE group if not already done previously for other distributions or installed software collections. If a installed software collection was registered, then the daemon uses the group name defined by the *dce_server_group* option to register its server name. If a distribution was registered, then the daemon uses the group name defined by the *dce_depot_group* option.

**EXIT VALUES**

The **swreg** utility returns one of the following exit codes:

0 The *objects_to_register* were successfully (un)registered.

1 The register or unregister operation failed on all *target_selections*.

2 The register or unregister operation failed on some *target_selections*.

**CONSEQUENCES OF ERRORS**

If there are any errors, then the **register** or **unregister** action fails.

**NAME**

swlist — list registered distributions and installed software collections

**SYNOPSIS**

**swlist -l level [ @ target_selections]**

**DESCRIPTION**

XDSA-DCE extends the POSIX 1387.2 standard **swlist** utility to also list registered distributions and installed software collections on local or remote hosts.

If using DCE Naming Services (see referenced document **DCE Directory**), **swlist** can also be used to list hosts with installed software collections or hosts with distributions that have been registered in the DCE namespace.

**OPTIONS**

XDSA-DCE extends the **swlist** utility to support the following options:

-l *level*

Specify the level of the object to list. In addition to the levels supported in the POSIX 1387.2 standard, the following levels are supported:

*root*
list registered installed software collections on a host

*depot*
list registered distributions on a host.

If DCE Naming Service is being used, the following additional levels are supported:

*host*
list registered hosts containing installed software collections

*host_with_depots*
list registered hosts containing distributions.

**OPERANDS**

The list utility supports the following syntax for each *target_selection* when listing registered installed software collections or distributions:

**[host][:][path]**

**EXTERNAL INFLUENCES**

XDSA-DCE extends the **swlist** utility supports the following extended options:

*level=*

Defines the default level of objects to list. Additional valid levels are *installed software collection*, *distribution*, *host* and *host_with_depots*.

**EXTERNAL EFFECTS**

stdout

The **swreg** utility displays a list of registered installed software collection or distribution paths for all target selections.

The output has one distribution or installed software collection path per line. Blank lines may exist and white space may precede and/of follow the path. On any line containing a # character, all characters that follow the # character up to but excluding the next <newline> character are ignored.

In the cases where the *installed_software_catalog* is also listed, it is listed on the same line as the path, separated by white space.

**EXTENDED DESCRIPTION**

If the level is *distribution* or *installed software collection*, the **swreg** utility contacts the daemon on the host from each target selection. The daemon returns a list of registered distributions or installed software collections for each host target selection.

If the target selection includes a path, then that only that path is listed and only if it is registered.

When listing registered installed software collections, the value of the *installed_software_catalog* for each installed software collection is listed on the same line as the path for the installed software collection, separated by white space, if that value is different than the value of the default *installed_software_catalog* option provided to **swlist**.

If the level is *host* or *host_with_depot*, the **swreg** utility contacts the DCE Naming Service and lists the hosts registered in the group name specified by the *dce_server_group* or *dce_depot_group* option respectively.

**EXIT VALUES**

0 The list operation succeeded for all targets

1 The list operation failed on all *target_selections*

2 The list operation failed on some *target_selections*.

**CONSEQUENCES OF ERRORS**

None.

*Chapter 7*

# *XDSA-DCE Daemon*

This chapter describes the XDSA-DCE daemon and agent processes. This is performed using the **swagentd** command. It is specified in reference manual page (man-page) format.

**NAME**

swagentd — serve local or remote software management tasks

**SYNOPSIS**

**swagentd [-k] [-n] [-r] [-x option=value] [-X option_file]**

**DESCRIPTION**

The roles of target and source systems require one or two processes known as the daemon **swagentd** and agent **swagent**. The **swagentd** serves the daemon XDSA-DCE RPC interface while the **swagent** serves the agent XDSA-DCE interface.

Whether the daemon and agent are separate processes is implementation defined, and relates to whether the operating system supports multi-tasking (in which case **swagentd** and **swagent** may be separate processes) or not (in which case a **swagent** session may be part of the **swagentd** process).

For most purposes, the distinction between the daemon and agent is invisible to the user and they can be viewed as a single process. Each distributed POSIX 1387.2 standard command interacts with the **swagentd** daemon and possibly a **swagent** session to perform its requested tasks.

The *swagentd*( ) daemon must be running before a system is available as a target or source system. This can be done either manually or in a system start-up script. A **swagent** session is initiated by **swagentd** to perform specific software management tasks. If the **swagent** session is a separate process, it is never invoked by the user, only by **swagentd**.

**OPTIONS**

The **swagentd** daemon supports the following options to control its behavior:

-k The *kill* option stops the currently running daemon. Stopping the daemon will not stop any **swagent** sessions currently performing management tasks (such as installing or removing software), but will cause any subsequent management requests to this host to be refused.

-n The *no fork* option runs the daemon as a synchronous process rather than the default behavior of forking to run it asynchronously. This is intended for running the daemon from other utilities that schedule processes, such as **init.**

-r The *restart* option stops any currently running daemon, and then restarts a new daemon. This operation is required whenever modifying default options that apply to the daemon, since defaults are only processed on startup.

-x *option=value*

Set the option to value and override the default value (or a value in an *option_file* specified with the *-X* option). Multiple *-x* options can be specified.

-X *option_file*

Read the session options and behaviors from *option_file*.

**OPERANDS**

The **swagentd** daemon does not support any operands.

## EXTERNAL INFLUENCES

### Extended Options

The daemon supports the following extended options:

*agent=implementation_defined_value*
The location of the **swagent** program invoked by the **swagentd** daemon if the implementation requires a separate agent program.

*logfile=implementation_defined_value*
This is the default log file for the **swagentd** daemon.

*max_agents=-1*
The maximum number of agent sessions that are permitted to run simultaneously. The value of -1 means that there is no limit.

*rpc_binding_info=ncadg_ip_udp:[2121] ncacn_ip_tcp:[2121]*
This defines the protocol sequences and endpoints which may be used to contact the **swagentd** process (the protocols and endpoints that the **swagentd** process is listening on). This value should be consistent among all XDSA-DCE hosts that work together.

The **swagent** session supports the following options. These options apply only to the agent. If a command prefix is used with these options, that command prefix must be ‘‘swagent’’.

*alternate_source=*
If the **swinstall** or **swcopy** manager has set *use_alternate_source=true*, the target agent consults and uses the configured value of its own *alternate_source* option to determine the source that it will use in the **install** or **copy** operation.

The agent’s value for *alternate_source* is specified using the **host:path** syntax. If the host portion is not specified, then the local host is used. If the path portion is not specified, then the path sent by the command is used. If there is no configured value at all for *alternate_source*, the agent will apply the manager-supplied path to its own local host.

*compress_cmd=implementation_defined_value*
Defines the command called by the source agent to compress files before transmission.

*postkernel_cmd=implementation_defined_value*
Defines the default script called by the agent for kernel building after kernel filesets have been loaded in the case where the product containing those filesets has not defined a value for its postkernel attribute.

*mount_cmd=implementation_defined_value*
Defines the command called by the agent to mount all filesystems if the *mount_all_filesystems* option is set to TRUE.

*reboot_cmd=implementation_defined_value*
Defines the command called by the agent to reboot the system after all filesets have been loaded, if any of the filesets were filesets requiring reboot.

*rpc_binding_info=ncadg_ip_udp:[2121]*
This defines the protocol sequence and endpoint used when the agent attempts to contact an alternate source distribution (as specified by the *alternate_source* option).

*uncompress_cmd=implementation_defined_value*
Defines the command called by the target agent to uncompress files after transmission. This command processes files which were stored on the media in a compressed format.

The **swagent** (target software collection) log files cannot be relocated. They always exist relative to the installed software collection or distribution target path (for example, **/var/adm/sw/swagent.log** for the installed software collection ‘‘/’’, and **/var/spool/sw/swagent.log** for the distribution **/var/spool/sw**).

### Environment Variables

The POSIX 1387.2 standard environment variables affecting the operation of **swagentd** process (and **swagent** session) include:

*LANG*
*LC_ALL*
*LC_TYPE*
*LC_MESSAGES*
*LC_TIME*
*TZ*

The **swagent** session sets the POSIX 1387.2 standard environment variables for use by the control scripts being executed on behalf of the SD commands:

*SW_CATALOG*
*SW_CONTROL_DIRECTORY*
*SW_CONTROL_TAG*
*SW_LOCATION*
*SW_PATH*
*SW_ROOT_DIRECTORY*
*SW_SESSION_OPTIONS*
*SW_SOFTWARE_SPEC*

## EXTERNAL EFFECTS

stdout
: The **swagentd** daemon does not write to **stdout**.

stderr
: The **swreg** utility writes events with a status of SW_WARNING or SW_ERROR to **stderr** for any errors or warnings during startup. After startup, all errors and warnings only are written to the daemon and agent log files.

logging
: The **swagentd** daemon logs all events from the daemon XDSA-DCE RPC interfaces in the logfile location defined by the *logfile* option.

  When operating on installed software collections or distributions, the **swagent** target session logs messages to a logfile relative to the installed software collection or distribution path. The **swagent** source session logs messages to the same logfile location, except when the source is a serial media or a read-only media. In this case, the location of the source logfile is implementation defined.

**EXTENDED DESCRIPTION**

After initialization, the **swagentd** daemon listens for daemon RPC requests and processes them as it receives them.

An agent session is started when the daemon gets the RPC *sw_rpc_agent_init*( ) (which spawns an **swagent** process if needed), followed by the agent RPC *sw_rpc_begin_session*( ). That agent session listens for requests until it gets the RPC *sw_rpc_end_session*( ), which ends the session (and stops the **swagent** process if needed).

**EXIT VALUES**

When the *-n* option is not specified, the **swagentd** returns:

0 The daemon is successfully initialized and is now running in the background.

1 Initialization failed and the daemon terminated.

When the *-n* option is specified, the **swagentd** returns:

0 The daemon successfully initialized and then successfully shutdown.

1 Initialization failed or the daemon unsuccessfully terminated.

**CONSEQUENCES OF ERRORS**

If there is an initialization error, the daemon terminates.

Chapter 8

# XDSA-DCE Security

This chapter presents an overview of Access Control Lists (ACLs) for XDSA-DCE objects, the definition of the distributed utility for managing ACLs, and associated the RPC interface definitions used to implement the utility. It contains the following sections:

- an overview of the XDSA-DCE security model
- the **swacl** utility
- the DCE Security Service RPCs used to manage the XDSA-DCE ACLs.

## 8.1 XDSA-DCE Security Model Overview

Along with the traditional POSIX standard file access permissions, certain XDSA-DCE software objects are protected by ACLs. When any management task attempts to create, read or write these objects, the principal (user) executing that task is checked against the ACL protecting that object.

In the distributed model defined by XDSA-DCE, the daemon and agent processes perform these checks using the principal, group, and realm information passed with each RPC. It is possible for each implementation to have its own implementation of access control (enforced by the daemon and agent). However, by supporting XDSA-DCE ACLs, interoperable distributed management of ACLs is possible.

### 8.1.1 Object Types

ACLs are supported on hosts, installed software collections, distributions and products in distributions.

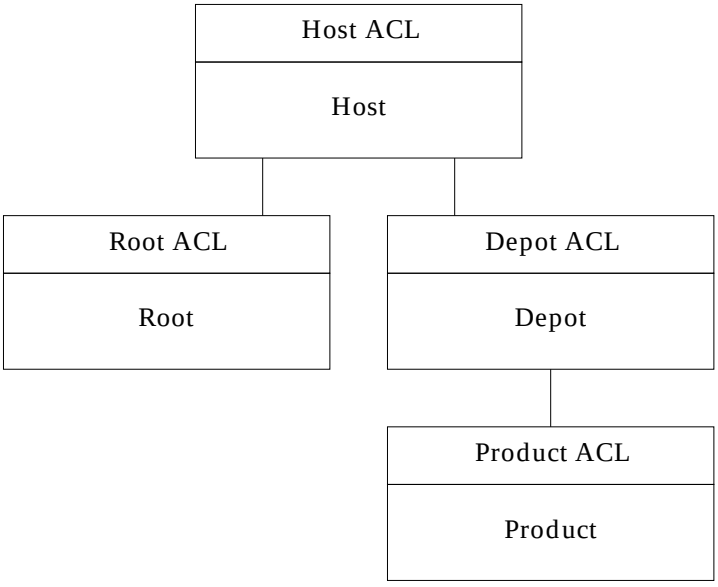



**Figure 8-1** ACL Object Types

The *host* ACL protects the host object, primarily for registering and listing registered distributions and installed software collections.

The *root* ACL protects the installed software collection objects, for installing, listing and otherwise managing installed software.

The *depot* ACL protects the distribution objects, for creating new products in the distribution, as well as listing the products in the distribution.

The *product* ACL protects individual products in the distribution, for serving those products to target agent copy and install tasks, as well as managing those products within the distribution.

There are also *template* ACLs used for the initial ACLs when creating new distributions, products within distributions and installed software collections

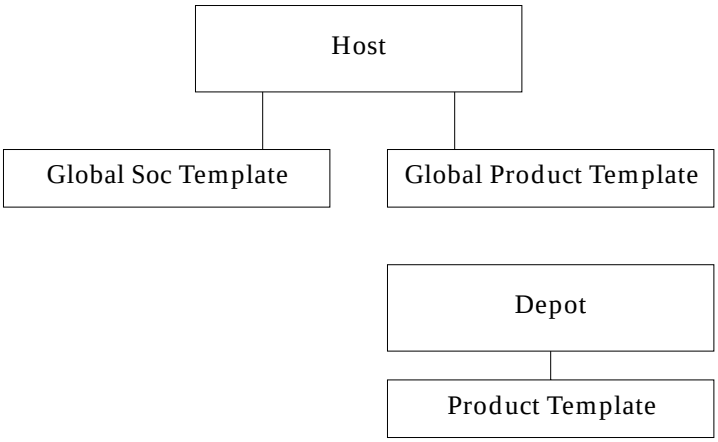



**Figure 8-2** Template ACLs

The *global_soc_template* ACL is used as the initial *root* or *depot* ACL when a new installed software collection or distribution is created.

The *global_product_template* ACL is used for the initial *product_template* ACL when a new distribution is created.

The *product_template* ACL is used for the initial product ACL when a new product in a distribution is created.

### 8.1.2 ACL Entries

Each entry in an ACL has the following form:

**entry_type[:key]:permissions**

For example:

**user:steve@newdist:crwit**

An ACL can contain multiple entries. The output of a list operation for an ACL (using **swacl**) is in the following format:

```
#
# swacl <object_type> Access Control List
#
# For <host|depot>: [<host>][:][<directory>]
#
# Date: <date_string>
#
# Object Ownership:  User=  <user_name>
#                    Group= <group_name>
#                    Realm= <host_name>
#
# default_realm=<host_name>
<entry_type>:[<key>:]<permissions>
<entry_type>:[<key>:]<permissions>
<entry_type>:[<key>:]<permissions>
```

This output can be saved into a file, modified, and then used as input to redefine an ACL using the **swacl** *-F* **option.**

### 8.1.3 Object Ownership

An owner is also associated with every object, as defined by the user name, group and hostname. The owner is the user who created the object. When using **swacl** to list an ACL, the owner is printed as a comment in the header.

### 8.1.4 Default Realm

An ACL defines a default realm for an object. The realm is currently defined as the name of the host system on which the object resides. When using **swacl** to view an ACL, the default realm is printed as a comment in the header.

When DCE Security Services are used (set by the *authentication_service* option), the realm is the DCE cell.

### 8.1.5 Entry Types

The following entry_types are supported:

object_owner
: Permissions for the object's owner, who's identity is listed in the comment header.

  Example:

  ```
  object_owner:crwit.
  ```

object_group
: Permissions for members of the object's group, who's identity is listed in the comment header.

  Example:

  ```
  object_group:crwit.
  ```

user
: Permissions for a named user. This type of ACL entry must include a key that identifies that user. The format for user can be:

  ```
  user:user_name:permissions
  ```

  or:

**user:user_name@hostname:permissions**

Example:

**user:rml:crwit.**

group
Permissions for a named group. This type of ACL entry must include a key that identifies that group. The format for group can be:

**group:group_name:permissions**

or:

**group:group_name@hostname:permissions.**

Example:

**group:adm:crwit.**

host
Permissions for a target agent from the specified host system. Agents require product level read access via either a host, other, or *any_other* entry type in order to copy or install products from distributions. This type of ACL entry must include a key containing a hostname of a system.

Example:

**host:newdist:-r--t.**

other
Permissions for others who are not otherwise named by a more specific entry type. The format for other can be:

**other:permissions**

for others on the local host (only one such entry allowed) or:

**other:@hostname:permissions**

for others at remote hosts (only one such entry per remote host allowed).

Example:

**other:@newdist:-r--t.**

any_other
Permissions for all other users and hosts that do not match a more specific entry in the ACL.

Example:

**any_other:-r--t.**

The order of the above entry types is significant. Except for group entry types, permissions to an object for a user or host are determined by a match to a single ACL.

The user or host is checked against entries of each type in the order shown in the above list until the match is found. If a match is found with an entry type of *group*, then a union of permissions for all group entries that match the users primary and secondary groups are included.

### 8.1.6 Keys

A key is required for user, group and host entry types. A key is optional for other entry types, and specifies the hostname to which the entry applies. Only one other entry type may exist without a key, and this entry applies to users at the default realm (host) of the ACL.

A hostname in a key will be listed in its Internet address format (dot notation) if **swacl** cannot resolve the address using the local lookup mechanism (for example DNS, NIS, or /etc/hosts). A hostname within an ACL entry must be resolvable when used with the **swacl** -*M* and -*D* options. Unresolvable hostname values are accepted in files provided with the **swacl** -*F* option.

### 8.1.7 Permissions

Permissions are represented as the single character abbreviations indicated below. Some permissions either apply only to, or have different meaning for, certain types of objects, as detailed below.

The following permissions may be granted:

c(ontrol)
: Grants permission to modify the ACL using **swacl**.

r(ead)
: Grants permission to read this object.

  On host, distribution, or installed software collection objects, read permission allows **swlist** operations. On products within distributions, read permission allows product files to be read for **swinstall**, **swcopy** and **swlist** operations.

w(rite)
: Grants permission to modify the object.

  On a installed software collection object (for example, installed installed software collection filesystem), this grants permission to modify the products and product files installed into the installed software collection object. On a distribution object, this grants permission to remove an empty distribution. It does not grant permission to modify the products contained within it; write permission is required on each product object in the distribution. On a host object, write permission grants permission to unregister distributions.

i(nsert)
: Grants permission to insert objects into this object.

  On a host object, grants permission to create (insert) a new software distribution or installed software collection object, and to register these objects. On a distribution object, grants permission to create (insert) a new product object.

t(est)
: Grants permission to list the ACL.

a(ll)
: A wildcard which grants all of the above permissions.

  It is expanded by **swacl** to **crwit**.

### 8.1.8 Depot Registration and Access Control

Because the distribution can be stored on a file system, anyone could build something which included a ''Trojan Horse'' (that is, something unrecognized as a threat but which could inflict damage), thereby exposing any system that installed products from it to a security breach. To protect systems from such a situation, a distribution must be registered (either through **swcopy** or **swreg**) before software may be installed or copied from it.

### 8.1.9 Secrets File

If the *authentication_service* option is set to *internal* (DCE Security Services are not being used), an encrypted secret password is sent with each RPC call as a proof of trustworthyness in place of authentication.

**Note:** This proof does not provide the level of DCE Security Service authentication, but does provide an additional level of trust.

Each manager and daemon host maintains a secrets file that contains a set of realm/secret pairs. The realm can be the NIS domain name, internet name, or network address of a manager host, and the secret is the unencrypted password associated with that realm.

The manager searches the secrets file for its realm, and encrypts the secret associated with that realm using **crypt**(3), prepending the salt used to the encrypted secret. It communicates that encrypted secret to the daemon or agent in the *internal_authn_secret* option for XDSA-DCE RPC calls. The daemon or agent serving the RPC call then searches its secrets file for an entry for the manager realm communicated via the *internal_authn_realm* option, encrypts the secret associated with that realm. If the encrypted secrets do not match, the call is not authenticated and it fails.

Each host can define a ''default'' secret that is used if there is not an entry specific to the realm requested, as shown in the following example implementation secrets file:

```
default              quicksilver
argo.finesoft.com    zztop!
15.25.34.122         soundgarden
```

### 8.1.10 Access Control Checks by RPC Call

Access to software objects is protected as part of various RPC calls. These checks are as follows:

- *sw_rpc_get_depots*( )
  The user must have read permission on the host.
- *sw_rpc_register_depot*( )
  The user must have insert permission on the host.
- *sw_rpc_unregister_depot*( )
  The user must have write permission on the host.
- *sw_rpc_is_registered_depot*( )
  The user must have read permission on the host.
- *sw_rpc_begin_session*( )
  If the task is SW_READ_TASK, the distribution must be registered and the user must have read permission on the distribution. If the SW_READ_TASK task is being made by a target agent instead of a manager, then the user (manager initiator of the task) is passed via delegation, and the target host also must have *read* permission on the distribution.

  If the task is SW_LIST_TASK or SW_DLIST_TASK, the user must have *read* permission on the installed software collection or distribution respectively.

If the task is SW_INSTALL_TASK or SW_COPY_TASK and the installed software collection or distribution does not exist, the user must have *insert* permission on the host.

If the task is SW_INSTALL_TASK, SW_REMOVE_TASK, SW_VERIFY_TASK, SW_CONFIGURE_TASK, or SW_MODIFY_TASK, the user must have *write* permission on the installed software collection.

These checks may alternatively be done by *sw_rpc_agent_init*( ) before initiating an agent session.

- *sw_rpc_analyze_task*( )
  If the task is SW_COPY_TASK, and the product does not exist, the user must have *insert* permission on the distribution. If the product does exist, the user must have *write* permission on the product.

  If the task is SW_DREMOVE_TASK or SW_DMODIFY_TASK, the user must have *write* permission on the product.

  If the task is SW_DVERIFY_TASK, the user must have *read* permission on the product.

- *sw_rpc_get_soc_file*( )
  For any task, if the file requested is the catalog INDEX file, the user must have read permission on the distribution or installed software collection. If the request is being made by a target agent instead of a manager, then the user is passed via delegation, and the target host also must have *read* permission.

  For any task, for files other than the catalog INDEX, the user must have *read* permission on the product containing the file. If the request is being made by a target agent instead of a manager, then the user is passed via delegation, and the target host also must have *read* permission.

As a special case, the local *super-user* is granted access to any local object and is granted access to modify any local ACL.

**NAME**

swacl - view or modify software Access Control Lists (ACLs)

**SYNOPSIS**

**swacl -l level [-D acl_entry | -F acl_entry | -M acl_file] [-f software_file] [-t target_file] [-x option=value] [-X option_file] [software_selections] [@ target_selections]**

**DESCRIPTION**

The swacl command displays or modifies the Access Control Lists (ACLs) which:

- protect the specified *target_selections* (software distributions or installed software collections)
- protect the specified *software_selections* on each of the specified *target_selections* (software distributions only).

All installed software collections, software distributions, and products in software distributions are protected by ACLs. The commands permit or prevent specific operations based on whether the ACLs on these objects permit the operation. The **swacl** command is used to view, edit, and manage these ACLs. The ACL must exist and the user must have the appropriate permission (granted by the ACL itself) in order to modify it.

ACLs offer a greater degree of selectivity than standard file permissions. ACLs allow an object's owner (that is, the user who created the object) or the local superuser to define specific read, write, or modify permissions to a specific list of users, groups, or combinations thereof.

**OPTIONS**

When none of the *-M, -D,* or *-F* options are specified, **swacl** prints the requested ACLs to the standard output.

The **swacl** command supports the following options:

-D *acl_entry*
: Deletes an existing entry from the ACL associated with the specified objects. (For this option, the permission field of the ACL entry is not required.) Multiple *-D* options can be specified.

-f *software_file*
: Read the list of *software_selections* from *software_file* instead of (or in addition to) the command line.

-F *acl_file*
: Assigns the ACL contained in *acl_file* to the object. All existing entries are removed and replaced by the entries in the file.Only the ACL's entries are replaced; none of the information contained in the comment portion (lines with the prefix ''#'') of an ACL listing is modified with this option. The *acl_file* is usually the edited output of a **swacl** list operation.

  If the replacement ACL contains no syntax errors and the user has control permission on the ACL (or is the local super user), the replacement succeeds.

-l *level*
: Defines which level of ACLs to view / modify. The supported levels are:

  *host*
  : View / modify the ACL protecting the host systems identified by the *target_selections*.

  *depot*
  : View / modify the ACL protecting the software distributions identified by the *target_selections*.

*root*
View/modify the ACL protecting the installed software collection filesystems identified by the *target_selections*.

*product*
View/modify the ACL protecting the software product identified by he software_selection. Applies only to products in distributions, not installed products in installed software collections.

*product_template*
View/modify the template ACL used to initialize the ACLs of future products added to the software distributions identified by the *target_selections*.

*global_soc_template*
View/modify the template ACL used to initialize the ACLs of future software distributions or installed software collections added to the hosts identified by the *target_selections*.

*global_product_template*
View/modify the template ACL used to initialize the product_template ACLs of future software distributions added to the hosts identified by the *target_selections*.

-M *acl_entry*
Adds a new ACL entry or changes the permissions of an existing entry. Multiple *-M* options can be specified.

-x *option=value*
Set the session option to value and override the default value (or a value in an alternate *option_file* specified with the *-X* option). Multiple *-x* options can be specified.

-X *option_file*
Read the session options and behaviors from *option_file*.

-t *target_file*
Read the list of *target_selections* from file instead of (or in addition to) the command line.

Only one of the *-M*, *-D*, or *-F* options can be specified for an invocation of **swacl**. For example, the *-M* and *-D* options cannot be specified together.

**OPERANDS**

When used with the *-l* product level, the **swacl** command allows the POSIX 1387.2 standard software specification syntax for each *software_selection* in order to specify one or more products. If the software selection refers to a bundle, then all products contained in that bundle are included. If the software selection refers to a subproduct or fileset, then the containing product is included.

The **swacl** command supports the following syntax for each *target_selection*:

**[host][:][/directory]**

The ‘‘:’’ (colon) is required if both a host and directory are specified.

**EXTERNAL INFLUENCES**

The **swacl** utility supports the following extended options:

*distribution_target_directory=implementation_defined_value*
Defines the default location of the target distribution.

*level=*
Defines the level of ACLs to view/modify. The supported levels are: *host*, *depot*, *root*, *product*, *product_template*, *global_soc_template* or *global_product_template*. See also the discussion of the *-l* option above.

*rpc_binding_info=ncadg_ip_udp:[2121]*
Defines the protocol sequence and endpoint which will be used to contact **swagentd**.

*rpc_timeout=5*
Relative length of the communications timeout.

*select_local=true*
If no *target_selections* are specified, select the local host (if the level is *host*, *global_soc_template* or *global_product_template*), the target directory ''/'' on the local host (if the level is *root*), or the default *distribution_target_directory* of the local host (if the level is *depot*, *product* or *product_template*) as the *target_selection* for the command.

*software=*
Defines the default of *software_selections*. There is no supplied default.

*targets=*
Defines the default *target_selections*. There is no supplied default (see *select_local* above). If there is more than target selection, they must be separated by spaces.

**EXTERNAL EFFECTS**

stdout
The **swacl** command prints ACL information to **stdout** when the user requests an ACL listing.

stderr
The **swacl** command writes events with a status of SW_WARNING or SW_ERROR to **stderr**.

logging
The **swacl** command does not log summary events. The daemon managing the ACLs logs events about each ACL which is modified, as well as any events associated with looking up an ACL, to the daemon logfile on each target host.

**EXTENDED DESCRIPTION**

The **swacl** utility contacts the daemon on the host targets specified. This is done using the DCE Security Service interfaces which interact indirectly with the RDACL interfaces served by the daemon.

In order to lookup an ACL or ACL entry, the full ACL is retrieved from the daemon. In order to replace a full ACL with the *-F* option, the new ACL is sent. In order to modify (change or add) and ACL entry, the full ACL is retrieved, the ACL is modified, then the full ACL is replaced.

When modifying a product ACL specified by a *software_selection*, the target distribution containing the product is opened as is done for other POSIX 1387.2 standard commands. The *software_selections* are then resolved against the distribution. Any problems resolving the *software_selection* generate the same events as other POSIX 1387.2 standard management commands:

- If the selection is not found, 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)

**EXIT VALUES**

The **swacl** command returns:

0 The *software_selections* and/or *target_selections* were successfully displayed or modified.

1 The display or modify operation failed on all *target_selections*.

2 The display or modify operation failed on some *target_selections*.

**CONSEQUENCES OF ERRORS**

If there are any errors, then the display or modification of ACLs fails.

## 8.2 DCE Security Service RPC use for XDSA-DCE

XDSA-DCE implements the distributed interface to ACL management by:

- providing the server support for the ACL manager by implementing the DCE RDACL interface in the XDSA-DCE daemon
- using the client support provided by DCE Security Service RPC calls to implement the manager utility, **swacl**
- defining the allowable values for the parameters of these interfaces.

### 8.2.1 DCE Security Service RPC Server Interfaces

When implementing an ACL manager, the program needs to implement the entire RDACL interface.

The definition of each of these calls can be found in DCE Security Service documentation.

```
void rdacl_lookup(
        [in]    handle_t h,
        [in]    sec_acl_component_name_t component_name,
        [in]    uuid_t *manager_type,
        [in]    sec_acl_type_t sec_acl_type,
        [out]   error_status_t *st
);

void rdacl_replace(
        [in]    handle_t h,
        [in]    sec_acl_component_name_t component_name,
        [in]    uuid_t *manager_type,
        [in]    sec_acl_type_t sec_acl_type,
        [in]    sec_acl_list_t *sec_acl_list,
        [out]   error_status_t *st
);

boolean32 rdacl_test_access(
        [in]    handle_t h,
        [in]    sec_acl_component_name_t component_name,
        [in]    uuid_t *manager_type,
        [in]    sec_acl_permset_t desired_permset,
        [out]   error_status_t *st
);

boolean32 rdacl_test_access_on_behalf(
        [in]    handle_t h,
        [in]    sec_acl_component_name_t component_name,
        [in]    uuid_t *manager_type,
        [in]    sec_id_pac_t *subject,
        [in]    sec_acl_permset_t desired_permset,
        [out]   error_status_t *st
);

void rdacl_get_manager_types(
        [in]    handle_t h,
        [in]    sec_acl_component_name_t component_name,
        [in]    sec_acl_type_t sec_acl_type,
        [in]    unsigned32 size_avail,
        [out]   unsigned32 *size_used,
        [out]   unsigned32 *num_types,
```

```
        [out]   uuid_t manager_types[],
        [out]   error_status_t *st
);

void rdacl_get_mgr_types_semantics(
        [in]    handle_t h,
        [in]    sec_acl_component_name_t component_name,
        [in]    sec_acl_type_t sec_acl_type,
        [in]    unsigned32 size_avail,
        [out]   unsigned32 *size_used,
        [out]   unsigned32 *num_types,
        [out]   uuid_t manager_types[],
        [out]   sec_acl_posix_semantics_t posix_semantics[],
        [out]   error_status_t *st
);

void rdacl_get_printstring(
        [in]    handle_t h,
        [in]    uuid_t *manager_type,
        [in]    unsigned32 size_avail,
        [out]   uuid_t *manager_type_chain,
        [out]   sec_acl_printstring_t *manager_info,
        [out]   boolean32 *tokenize,
        [out]   unsigned32 *total_num_printstrings,
        [out]   unsigned32 *size_used,
        [out]   sec_acl_printstring_t printstrings[],
        [out]   error_status_t *st
);

void rdacl_get_referral(
        [in]    handle_t h,
        [in]    sec_acl_component_name_t component_name,
        [in]    uuid_t *manager_type,
        [in]    sec_acl_type_t sec_acl_type,
        [out]   sec_acl_tower_set_t *towers,
        [out]   error_status_t *st
);

void rdacl_get_access(
        [in]    handle_t h,
        [in]    sec_acl_component_name_t component_name,
        [in]    uuid_t *manager_type,
        [out]   sec_acl_permset_t *net_rights,
        [out]   error_status_t *st
);
```

### 8.2.2 DCE Security Service RPC Client Interfaces

The ‘‘manager swacl’’ uses DCE client Security Service interfaces which in turn call the RDACL server interfaces.

The definition of each of these calls can be found in DCE Security Service documentation.

```
void sec_acl_bind_to_addr(
        [in]    idl_char *site_addr,
        [in]    sec_acl_component_name_t component_name,
        [out]   sec_acl_handle_t *h,
        [out]   error_status_t *status
);

void sec_acl_get_manager_types(
        [in]    sec_acl_handle_t h,
        [in]    sec_acl_type_t sec_acl_type,
        [in]    unsigned32 size_avail,
        [out]   unsigned32 *size_used,
        [out]   unsigned32 *num_types,
        [out]   uuid_t manager_types[],
        [out]   error_status_t *st
);

void sec_acl_lookup(
        [in]    sec_acl_handle_t h,
        [in]    uuid_t *manager_type,
        [in]    sec_acl_type_t sec_acl_type,
        [out]   sec_acl_list_t *sec_acl_list,
        [out]   error_status_t *st
);

void sec_acl_replace(
        [in]    sec_acl_handle_t h,
        [in]    uuid_t *manager_type,
        [in]    sec_acl_type_t sec_acl_type,
        [in]    sec_acl_list_t *sec_acl_list,
        [out]   error_status_t *st
);
```

### 8.2.3 DCE Security Service RPC Type Values

This section lists the supported values for parameters of the client and server RPC calls.

#### Component Naming

The *component_name* (with type *sec_acl_component_name_t*) identifies the name of the target object with which an ACL is associated. If the *authentication_service* option is set to *dce-secret*, the component name uses the following syntax:

**_key**

The *acl_key* is a string representing the type of object the **acl** is associated with, and if needed, a installed software collection or distribution path and a software specification of a product. The following defines are used:

| | |
|---|---|
| **SW_SEC_HOST** | **host** |
| **SW_SEC_ROOT** | **root** |
| **SW_SEC_DEPOT** | **depot** |
| **SW_SEC_PRODUCT** | **product** |

The *acl_key* is constructed as follows, depending on the level of the ACL specified by the *-l* option to **swacl**:

| | |
|---|---|
| **host** | **SW_SEC_HOST** |
| **global_soc_template** | **SW_SEC_HOST** |
| **global_product_template** | **SW_SEC_HOST** |
| **root** | **SW_SEC_ROOT ':' root_path [ ':' root_catalog ]** |
| **depot** | **SW_SEC_DEPOT ':' depot_path** |
| **product_template** | **SW_SEC_DEPOT ':' depot_path** |
| **product** | **SW_SEC_PRODUCT ':' depot_path ':' product_catalog** |

The *root_path* variable is the pathname to the installed software collection. The *depot_path* is the pathname to the distribution. The *root_catalog* is the added if an *installed_software_catalog* option was set to a value other than the default. The *product_catalog* is the *control_directory* attribute of the product.

If the *authentication_service* option is set to *internal*, the *acl_key* in component name is preceded by the principal and internal secret information needed to authenticate the user before accessing the ACL. This information is provide in this manner because the DCE Security Service interfaces do not provide a way (such as options for the XDSA-DCE interface) to pass this additional information:

**username.groupname[.othergroups]@realm;encrypted_secret,acl_key**

The *username* is the name of the user. The *groupname* is the current primary group of the user. The *othergroups* is a dot-separated (''.'') list of secondary groups the user belongs to. The realm is the NIS domain name, internet name, or network address of the manager. The *encrypted_secret* is a concatenation of the 2-character salt used by the manager to encrypt its secret and the encrypted secret itself. The manager and daemon encrypt using **crypt(3)**, and the daemon uses the salt provided in the first two characters of the *encrypted_secret*.

### ACL Types

The *sec_acl_type* parameter (with type of *sec_acl_type_t*) also depends on the level of the ACL specified with the *-l* option to **swacl**. The values are DCE type values:

| | |
|---|---|
| **host** | **sec_acl_type_object** |
| **root** | **sec_acl_type_object** |
| **depot** | **sec_acl_type_object** |
| **product** | **sec_acl_type_object** |
| **product_template** | **sec_acl_type_default_object** |
| **global_soc_template** | **sec_acl_type_default_container** |
| **global_product_template** | **sec_acl_type_default_object** |

Additionally, for use with lookup, the owner of the object is designated by the DCE type value:

| | |
|---|---|
| **owner** | **sec_acl_type_unspecified_3** |

### ACL Permissions

The supported ACL permissions for the perms member (of type *sec_acl_permset_t*) of the *sec_acl_entries* array include the XDSA-DCE permissions and the corresponding DCE type values:

| | |
|---|---|
| **r(ead)** | **sec_acl_perm_read** |
| **w(rite)** | **sec_acl_perm_write** |
| **i(nsert)** | **sec_acl_perm_insert** |
| **t(est)** | **sec_acl_perm_test** |
| **c(ontrol)** | **sec_acl_perm_control** |

### ACL Entry Types

The supported ACL entry types for the *entry_type* member (of type *sec_acl_entry_type_t*) of the *entry_info* member of the *all_entries* array include the XDSA-DCE entry types and the corresponding DCE type values:

| | |
|---|---|
| **object_owner** | **sec_acl_e_type_user_obj** |
| **object_group** | **sec_acl_e_type_group_obj** |
| **user** | **sec_acl_e_type_user** |
| **group** | **sec_acl_e_type_group** |
| **other** | **sec_acl_e_type_other** |
| **user:@realm** | **sec_acl_e_type_foreign_user** |
| **group:@realm** | **sec_acl_e_type_foreign_group** |
| **other:@realm** | **sec_acl_e_type_foreign_other** |
| **host** | **sec_acl_e_type_user** |
| **any_other** | **sec_acl_e_type_any_other** |

# Glossary

This specification assumes familiarity with the terminology used in the POSIX 1387.2 standard. Some key POSIX 1387.2 terms are included here along with terms unique to XDSA-DCE.

**ACL**
Acronym for Access Control List.

**catalog**
The meta-data describing the software objects in a software collection. Each software collection has a catalog.

**control file**
A control script, or in the case of the exported form of a software collection (see software packaging layout), an "INDEX" or "INFO" file describing the software objects in the software collection.

**control script**
A file associated with a software object that is executed by a POSIX 1387.2 standard command at some point in the operation.

**depot**
Keyword used in the interface coding for the POSIX 1387.2 term ‘‘distribution’’.

**distribution**
A software collection containing software available for distribution or installation. A distribution can be on a removable media or a file system, and can be in a serial or filesystem format.

**files**
Files must be grouped into software filesets before they can be managed.

**host**
The system that contains software managed by the POSIX 1387.2 standard utilities. The host contains software collections.

**installed software**
A software collection containing installed software. This software may also be configured (activated, or ready for use) or act as a file server to another installation.

**root**
Keyword used in the interface coding for POSIX 1387.2 term ‘‘installed software collection’’.

**SOC**
XDSA-DCE acronym for POSIX 1387.2 term ‘‘software collection’’.

**software collection**
A grouping of software objects to be managed. POSIX 1387.2 standard commands apply tasks to software objects within software collections. Distributions are software collections of available software. Installed software collections contain software installed and ready for use or file serving to another installation.

**software object**
A software product or fileset that contains files and control files, as well as descriptive and behavioral attributes. (There are also grouping of products and filesets called bundles and subproducts respectively). Software objects are managed by the POSIX 1387.2 standard commands within the context of software collections.

**software packaging layout**
Describes the layout of the software files in a distribution, as well as the syntax of the files describing the attributes of the software in the distribution. This is an exported form of a software collection. INDEX files contain the attributes of the software objects. INFO files contain the attributes of the software file and control file objects.

**software product**
A group of related software. Products contain filesets and control scripts.

**software fileset**
A set of software files and control scripts. This is the smallest manageable object.

**software selections**
The set of software objects that are applied to each software collection in the course of executing a command.

**source**
A software collection on a particular host. The source of a command is the collection from which the software objects are retrieved.

**target**
A software collection on a particular host. The target of a command is the collection to which the software objects are applied (for example, for installing software) or otherwise operated on (for example, verified, removed).

**target selections**
The set of software collections that the software selections are applied to in the course of executing a command.

**XDSA-DCE**
Acronym for identifying this specification, derived from its subject matter which addresses Distributed Software Administration in an environment which uses DCE RPC technology to provide interoperability between implementations.

# Index