

1 / *Preliminary Specification*

2 **Distributed Audit Service (XDAS)** |

3 **Company Review Version** |

4 *The Open Group* |

5



6

© February 1997, The Open Group

7

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the copyright owners.

8

9

10

Preliminary Specification

11

Distributed Audit Service (XDAS) Company Review Version

12

ISBN: 1-85912-139-X

13

Document Number: P441

14

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

15

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

16

The Open Group

17

Apex Plaza

18

Forbury Road

19

Reading

20

Berkshire, RG1 1AX

21

United Kingdom

22

or by Electronic Mail to:

23

OGSpecs@opengroup.org

# Contents

25	<b>Chapter 1</b>	<b>Introduction.....</b>	<b>1</b>
26	1.1	Functional Requirements .....	2
27	1.1.1	Audit Event Services.....	2
28	1.1.2	Audit Service Management .....	3
29	1.1.3	Audit Event Management.....	3
30	1.1.4	Audit Log Management .....	3
31	1.1.5	Audit Log Enquiry.....	4
32	1.2	Security Requirements.....	4
33	1.3	Distributed System Requirements .....	4
34	1.4	Non-functional Requirements .....	5
35	1.5	Out of Scope.....	5
36	<b>Chapter 2</b>	<b>Conformance Statement.....</b>	<b>7</b>
37	2.1	Basic XDAS Conformance.....	7
38	2.2	XDAS Import API Option Conformance .....	7
39	2.3	XDAS Event Submission API Option Conformance.....	8
40	2.4	XDAS Filter Management API Option Conformance.....	8
41	<b>Chapter 3</b>	<b>XDAS Model.....</b>	<b>9</b>
42	3.1	Introduction .....	9
43	3.2	Interfaces.....	11
44	3.3	Distributed Audit Service Model .....	12
45	3.3.1	XDAS Event Supplier Components .....	12
46	3.3.2	XDAS Event Consumer Components .....	13
47	<b>Chapter 4</b>	<b>XDAS Data Structures.....</b>	<b>15</b>
48	4.1	Audit Record Stream.....	15
49	4.2	Audit Event Record.....	15
50	4.3	Originator, Initiator and Target Information .....	17
51	4.3.1	Originator_Information.....	17
52	4.3.2	Initiator_Information.....	17
53	4.3.3	Target_Information.....	18
54	4.4	Identification of Audit Events.....	18
55	4.4.1	Event Numbers .....	19
56	4.4.2	XDAS Events.....	19
57	4.4.3	Event Classes .....	22
58	4.4.4	Outcomes.....	23
59	4.5	Event Selection .....	24
60	4.5.1	Event Submission Pre-Selection Filtering.....	25
61	4.5.2	Event Import Pre-Selection Filtering .....	25
62	4.5.3	Event Analysis Post Selection Filtering.....	25
63	4.5.4	Event Filters .....	25

64	4.5.5	Filter Expression List.....	26
65	4.5.6	Filter Action List.....	26
66	<b>Chapter 5</b>	<b>DAS Usage Model .....</b>	<b>27</b>
67	5.1	Authorisation Policy .....	27
68	5.2	General Audit Service API.....	28
69	5.3	Audit Event Service Client API .....	28
70	5.4	Audit Log Import API.....	29
71	5.5	Audit Event Management API.....	29
72	5.6	Audit Read API.....	29
73	<b>Chapter 6</b>	<b>Parameter Passing Conventions.....</b>	<b>31</b>
74	6.1	Structured Data Types .....	31
75	6.2	Integer Types.....	31
76	6.3	String Data and Similar Data.....	32
77	6.3.1	Byte Strings .....	32
78	6.3.2	Character Strings.....	32
79	6.3.3	Opaque.....	32
80	6.4	Common Audit Record Format .....	33
81	6.5	Filters.....	34
82	6.6	Status Values.....	35
83	6.6.1	XDAS Status Codes .....	35
84	6.6.2	Minor Status Codes .....	37
85	6.7	Optional Arguments .....	38
86	6.7.1	<i>xdas_buffer_t</i> Types ( <i>Input or Input,Output</i> ) .....	38
87	6.7.2	Integer Types.....	38
88	6.7.3	Pointer Types .....	38
89	6.8	Constants.....	39
90	6.9	Event Numbers .....	40
91	6.10	XDAS Event Classes.....	42
92	6.11	XDAS Event Outcome Codes.....	43
93	6.12	XDAS Action Codes .....	44
94	6.13	XDAS Filter Types.....	45
95	6.14	XDAS Filter Flags.....	46
96	6.15	XDAS Filter Attributes.....	47
97	6.16	XDAS Filter Operators.....	48
98	<b>Chapter 7</b>	<b>XDAS Application Program Interface (API).....</b>	<b>49</b>
99		<i>xdas_close_audit_stream()</i> .....	50
100		<i>xdas_commit_record()</i> .....	51
101		<i>xdas_create_filter()</i> .....	52
102		<i>xdas_delete_filter()</i> .....	54
103		<i>xdas_disable_filter()</i> .....	55
104		<i>xdas_discard_record()</i> .....	56
105		<i>xdas_enable_filter()</i> .....	57
106		<i>xdas_get_filter()</i> .....	58
107		<i>xdas_get_next()</i> .....	60
108		<i>xdas_import_event_records()</i> .....	62

## Contents

109		<i>xdas_initialise_session()</i> .....	64
110		<i>xdas_list_filters()</i> .....	66
111		<i>xdas_open_audit_stream()</i> .....	67
112		<i>xdas_put_event_info()</i> .....	68
113		<i>xdas_release_buffer()</i> .....	70
114		<i>xdas_release_filter_list()</i> .....	71
115		<i>xdas_rewind_audit_stream()</i> .....	72
116		<i>xdas_start_record()</i> .....	73
117		<i>xdas_terminate_session()</i> .....	75
118		<i>xdas_timestamp_record()</i> .....	76
119	<b>Appendix A</b>	<b>Mapping of DAS Events</b> .....	<b>77</b>
120	A.1	Mapping .....	79
121	A.2	IEEE P1003.1e -- Protection, Audit and Control Interfaces .....	81
122	<b>Appendix B</b>	<b>XDAS Naming Syntax</b> .....	<b>83</b>
123	B.1	Composite Name String Syntax .....	83
124	B.1.1	Encoding of XDAS Composite Name Strings .....	83
125	B.1.2	Backus-Naur Form (BNF) of XDAS Composite Names .....	83
126		<b>Glossary</b> .....	<b>85</b>
127		<b>Index</b> .....	<b>97</b>
128	<b>List of Figures</b>		
129	3-1	Distributed Audit Service Interfaces .....	9
130	3-2	Distributed Audit Service Model .....	12
131	<b>List of Tables</b>		
132	4-1	Event Filtering Criteria .....	24
133	6-1	Calling Errors .....	35
134	6-2	Routine Errors .....	36
135	6-3	Optional Parameter Constants .....	39
136	6-4	XDAS Event Field Separators .....	39
137	6-5	XDAS Event Numbers .....	40
138	6-6	XDAS Default Event Class Codes .....	42
139	6-7	XDAS Event Outcome Codes .....	43
140	6-8	XDAS Action Codes .....	44
141	6-9	XDAS Filter Types .....	45
142	6-10	XDAS Filter Flags .....	46
143	6-11	XDAS Filter Attributes .....	47
144	6-12	XDAS Filter Operators .....	48
145	A-1	Mapping of ORACLE Audit Events to XDAS Generic Audit Events .....	77
146	A-2	Mapping of Solaris BSM Audit Events to XDAS Generic Audit Events ...	79
147	A-3	Mapping of IEEE P1003.1e Audit Events to XDAS Generic Audit Events .....	81



**150 The Open Group**

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

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

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

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

**179 The X/Open Process**

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

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

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

#### 194 **Open Group Publications**

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

199 There are several types of specification:

- 200 • *CAE Specifications*

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

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

- 210 • *Preliminary Specifications*

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

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

- 222 • *Consortium and Technology Specifications*

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

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



230 In addition, The Open Group publishes:

231 • *Product Documentation*

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

236 • *Guides*

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

241 • *Technical Studies*

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

246 • *Snapshots*

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

251 **Versions and Issues of Specifications**

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

255 • A new *Version* indicates there is no change to the definitive information contained in the  
256 previous publication of that title, but additions/extensions are included. As such, it *replaces*  
257 the previous publication.

258 • A new *Issue* indicates there is substantive change to the definitive information contained in  
259 the previous publication of that title, and there may also be additions/extensions. As such,  
260 both previous and new documents are maintained as current publications.

261 **Corrigenda**

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

264 **Ordering Information**

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

267 **This Document**

268 This document is a Preliminary Specification (see above).

- 269 • Chapter 1 is an introduction to the GAS-API.
- 270 • Chapter 2 is a conformance statement.
- 271 • Chapter 3 describes the audit service model.
- 272 • Chapter 4 defines the logical data structures used within this specification.
- 273 • Chapter 5 provides an overview of the functions defined by this specification and how they  
274 are used.
- 275 • Chapter 6 describes the parameters required by the DAS API,
- 276 • Chapter 7 describes the XDAS API function definitions,
- 277 • Appendix A provides a mapping of domain specific events to the generic set of event classes  
278 identified within this specification,
- 279 • Appendix B describes the syntax used for names within this specification.
- 280 • A glossary of terms used within this specification is provided.

281 **Typographical Conventions**

282 The following typographical conventions are used throughout this document:

- 283 • **Bold** font is used in text for filenames, and C-language keywords, type names, data  
284 structures and their members.
- 285 • *Italic* strings are used for emphasis or to identify the first instance of a word requiring  
286 definition. Italics in text also denote:
  - 287 — C-language variable names, for example, substitutable argument prototypes
  - 288 — C-language functions; these are shown as follows: *name()*.
- 289 • Normal font is used for the names of constants and literals.
- 290 • The notation **<file.h>** indicates a header file.
- 291 • The notation [EABCD] is used to identify a C-language return code EABCD.
- 292 • Syntax, code examples and user input in interactive examples are shown in *fixed width*  
293 *font*.
- 294 • Variables within syntax statements are shown in *italic fixed width font*.
- 295 • Language-independent functions and arguments use ***bold italic*** font, for example, ***function()***  
296 and ***argument***.

297

## *Trade Marks*

298

OSF™ is a trademark of The Open Software Foundation, Inc.

299

X/Open® is a registered trademark, and the “X” device is a trademark, of X/Open Company Limited.

300

301

## *Acknowledgements*

302

The OpenGroup gratefully acknowledges the work of the OpenGroup Security Program Group

303

in the development of this specification.

## Referenced Documents

304

305 The following documents are referenced in this specification:

306 CESG Memo

307 CESG Memorandum No.1 Issue 1.2 Oct 1992, Glossary of Security Terminology.

308 Federal Criteria

309 Federal Criteria Version 1.0 Dec 1992, Federal Criteria for Information Technology Security.

310 ISO/IEC 7498-2

311 ISO/IEC 7498-2: 1989, Information Processing Systems — Open Systems Interconnection —  
312 Basic Reference Model — Part 2: Security Architecture.

313 ISO/IEC 10181

314 ISO/IEC 10181, Information Technology — Open Systems Interconnection — Security  
315 Frameworks in Open Systems —

316 10181-1: Part 1: Security Frameworks Overview

317 10181-2: Part 2: Authentication Framework

318 10181-3: Part 3: Access Control

319 10181-4: Part 4: Non-repudiation Framework

320 10181-5: Part 5: Integrity Framework

321 10181-6: Part 6: Confidentiality Framework

322 10181-7: Part 7: Security Audit Framework

323 ITSEC

324 Information Technology Security Evaluation Criteria, Provisional Harmonised Criteria, June  
325 1991, Version 1.2, published by the Commission of the European Communities.

326 POSIX.0

327 IEEE Std 1003.0/D15, June 1992, Draft Standard for Information Technology — Portable  
328 Operating System Interface (POSIX) — Part 0.

329 X.509

330 ISO/IEC 9594-8: 1990, Information Technology — Open Systems Interconnection — The  
331 Directory — Part 8: Authentication Framework, together with:

332 Technical Corrigendum 1: 1991 to ISO/IEC 9594-8: 1990.

333 ISO 8859-1:1987 Information processing -- 8-bit single-byte coded graphic character sets -- Part 1:  
334 Latin alphabet No. 1

335 The following X/Open documents are referenced in this specification:

336 XDSEF

337 Guide, December 1994, Distributed Security Framework (ISBN: 1-85912-071-7, G410).

338 Federated Naming

339 Preliminary Specification, August 1994, Federated Naming: The XFN Specification,  
340 (ISBN: 1-85912-458-8, P403).

341 XEMS

342 X/Open Preliminary Specification, June 1996, Systems Management: Event Management  
343 Service, Draft V0.3



# Introduction

1

2

The purpose of security audit services is to provide support for

3

- the principle of accountability, that is holding users of a system accountable for their actions within the system, and

4

5

- detection of security policy violations, that is the detection of attempts by unauthorised individuals to access the system and of attempts by authorised users to misuse their access to the system.

6

7

8

Many components of distributed systems now include some form of security auditing or event logging capability whereby the component records events deemed to have security relevance within the domain of that component. These services are provided via component specific interfaces and use component specific audit record formats.

9

10

11

12

However, within distributed systems security relevant activity is not isolated within individual components but spans many components. For example, an intrusion attempt may be made via multiple entry points to the distributed system. Such attempts are not necessarily focused through single points of entry. Also the purpose of a distributed system is to enable the end-users of the system to utilise the resources of components throughout the system and not just those of their local workstation.

13

14

15

16

17

18

Within a distributed system it is therefore necessary to monitor activity across and between components. This is made difficult by the current component specific approaches. It is not easy to compare activity across system components when the events monitored and the record formats may be different. It is especially difficult to do this in a timely manner to detect and respond to intrusion attempts.

19

20

21

22

23

The objective of the XDAS specification is to define

24

- a set of generic events of relevance at a global distributed system level, For example, end-user system sign-on and the initiation and termination of communication sessions between components.

25

26

27

- a common portable audit record format to facilitate the merging and analysis of audit information from multiple components at the distributed system level

28

29

- an API for use by applications to submit events to XDAS

30

- an API to import audit data from existing component specific audit services to XDAS

31

- an API to configure event pre-selection criteria for event submission to XDAS

32

- an API to read records from a XDAS audit trail

33

This service is intended to be a complement to existing system component specific audit services, not to replace them. Such local audit services are also likely to handle events and a level of detail that may be irrelevant at the global level of XDAS.

34

35

36

Interfaces are supported for use by four different types of applications:

37

- an API to submit events to the audit service, for use by applications that generate audit records and use XDAS to log such events

38

39

- an API and a common audit event record format for use by existing component specific audit services to import audit records into the XDAS audit stream for distributed system level analysis

40

41

- 42 • an API to support the configuration of event pre-selection criteria and event disposition  
43 actions, for use by XDAS audit event management applications
- 44 • an API together with a common audit event record format, for use by Audit Log Analysis  
45 applications

46 The XDAS-API provides the following benefits:

- 47 • Application developers have a common API, a generic set of audit events, and a common  
48 audit format regardless of the platform on which the XDAS service is running. This is of  
49 benefit to the developers of both applications that detect and wish to record security relevant  
50 events and of applications that analyse audit events.
- 51 • Platform and application infrastructure vendors are able to support the needs of users at the  
52 distributed system level within a heterogeneous environment without the necessity to re-  
53 engineer their current operating system or application specific audit service  
54 implementations, perhaps with resulting performance implications
- 55 • End-user organisations benefit through increased effectiveness in enforcing individual  
56 accountability within a distributed environment.

## 57 **1.1 Functional Requirements**

58 The business requirements for a distributed audit service are detailed in this section for  
59 completeness. Not all of these requirements are satisfied by the current scope of XDAS. The  
60 requirements are grouped according to audit event services, audit service management, audit  
61 log management and audit log retrieval facilities.

### 62 **1.1.1 Audit Event Services**

63 Security events are detected outside the XDAS by an operating system or applications. The  
64 requirements on a distributed audit service are as follows

- 65 • To handle event records newly generated at the local API level.
- 66 • The audit facility shall support the pre-selection of criteria for the detection of an event,  
67 thereby reducing the numbers of audit events generated and analysed.
- 68 • Filter and analyse records for instances or accumulations of pre-determined security events,  
69 and trigger timely notification. These filters shall be driven by parameters in a standard  
70 format. Three types of event or compound event are identified:
  - 71 • a single record selected by one or more fields
  - 72 • sequences of selected records
  - 73 • timed sequences of records
- 74 • Generate local alarms.
- 75 • Generate messages to be passed to the audit system management interface.
- 76 • Take pre-defined action on the occurrence of specific events.
- 77 • Receive records passed on from another system in a standard format and re-interpret them in  
78 the context of extra information available from event records arriving from other systems.



### 79 1.1.2 Audit Service Management

80 These generic requirements are out of scope for the XDAS:

- 81 • Support a consistent management interface.
- 82 • Integrate the audit system management interface with other elements in the system  
83 management infrastructure, including logs, protocols and databases and the management of  
84 authorisations.
- 85 • Support both Remote and Local Administration  
86 The XDAS must support role-based decentralised administration, such that individuals are  
87 only presented with the data that apply to their area of responsibility.
- 88 • Support both equivalent GUI and command line access so that the functions are available  
89 regardless of the mode of interaction.

### 90 1.1.3 Audit Event Management

91 The following are requirements on the Audit Event Management interface:

- 92 • Support the configuration of the disposition of audit alarms, such that the audit event source  
93 and type can be sent to a particular destination, and to a particular role at that destination to  
94 be actioned.
- 95 • Provide a set of standard calls to modify the parameters which define the filtering performed.  
96 These are used to configure the actions taken by the filtering and analysis component on each  
97 system. They may be originated by an operator or automatically as a result of event  
98 processing.
- 99 • Support two types of configuration: *static configuration* and *dynamic configuration*.  
100 With *static configuration*, the levels of audit data to be generated are pre-set by operator  
101 intervention. With *dynamic configuration*, the events or series of events detected are used to  
102 re-configure the filters on the monitor. Reconfiguration can involve increasing or decreasing  
103 the level of monitoring activity, as deemed appropriate by the analysis of the event or series  
104 of events.
- 105 • Determine and effect change to the configuration of security event detection on each of the  
106 platforms in a distributed environment. If several systems are monitored and all have a  
107 common requirement for maintaining a particular level of event logging, then a single  
108 definition should be applied to all.
- 109 • Record a security event message whenever a change to the configuration of the event  
110 discrimination service is made.

### 111 1.1.4 Audit Log Management

112 Audit Log Management requirements are:

- 113 • Log records to a protected audit record repository.
- 114 • Ensure that the sequence of events recorded is a reflection of what actually transpired. Thus,  
115 any mechanism which generates audit data should incorporate a *header* or common set of  
116 data which is co-ordinated with other systems with which it interacts. The header should  
117 contain a minimum set of information describing the date, time, location, initiator, target,  
118 message, etc., of the activity. Platforms, applications and network services should have the  
119 ability to add domain specific information to the information set.

### 120 1.1.5 Audit Log Enquiry

121 The Audit Log Enquiry requirements are:

- 122
- Provide a common format definition for the audit log for use by analysis applications.

## 123 1.2 Security Requirements

124 An implementation of the XDAS needs to meet the following security requirements:

- 125
- Prevent unauthorised modification of the audit service configuration data.
  - 126 • Prevent unauthorised modification of the event detection records.
  - 127 • Prevent unauthorised disclosure of the event records.
  - 128 • Support adequate separation of duties for users.
  - 129 • Provide appropriate measures in dealing with an unauthorised denial of service, for example,
  - 130 by suspending an offending process, if appropriate.
  - 131 • Protect audit service configuration data.
  - 132 • Protect the *audit log* and its contents from any unauthorised modification or deletions.
  - 133 • Protect the audit log by making it accessible only to principals acting in specific
  - 134 administrative or security roles.

135 The security requirements are met by using underlying distributed system security services and  
136 platform security services, wherever possible.

## 137 1.3 Distributed System Requirements

138 Two requirements need to be met by XDAS to support a distributed model. It must:

- 139
- Not hinder the achievement of adequate performance over the network.
  - 140 • Utilise trustworthy universal timestamps on event records. Because the XDAS cannot
  - 141 assume a trusted time service is available, there is a requirement that the audit records
  - 142 include a measure of the uncertainty of the time at which the recorded event occurred. This
  - 143 uncertainty information needs to be inserted into the records when they are imported to or
  - 144 exchanged between XDAS systems.

## 145 1.4 Non-functional Requirements

146 The following non-functional requirements have been identified:

- 147 1. the XDAS shall be application independent
- 148 2. the XDAS shall not impose a particular placement of access control to distributed audit  
149 services within an operating system kernel
- 150 3. The XDAS shall not constrain future extensibility. Nor shall it constrain the services of  
151 other audit systems, including operating system and site specific events types and  
152 associated data.

## 153 1.5 Out of Scope

154 The XDAS provides a set of primitives only, which are used by audit applications. The  
155 following facilities and services are deemed to be out of scope.

### 156 Event Detection

157 The detection of security relevant events is done outside the audit service. The specification  
158 assumes that that the applications responsible for even detection will prevent any  
159 unauthorised modification of those event detection services.

### 160 Audit Filter Propagation

161 XDAS defines interfaces for the creation and management of audit filters. This version of  
162 the specification does not define any protocols or data formats for the propagation of those  
163 filters between XDAS components.

### 164 Detection of sequences of events or compound events

165 XDAS provides the basic functionality for the submission and filtering of individual events  
166 together with a common audit event record format for audit event consolidation and  
167 analysis. An application capable of detecting complex sequences of events or combinations  
168 of events can be implemented over these basic XDAS services.

### 169 Dynamic Modification of Audit Filter Parameters

170 XDAS does not include functionality for the analysis of monitored security related events to  
171 determine whether modifications are needed to the filter parameters. This functionality falls  
172 within the scope of an audit administration application that can be implemented over the  
173 XDAS services provided.

### 174 Domain Specific Event

175 XDAS is not attempting to map all operating system or domain specific events to XDAS  
176 generic events, only those of significance at a distributed system level.

### 177 Graphical User Interface (GUI)

178 The XDAS provides support for GUI tools. The specification supports but does *not* address  
179 the definition of these tools.

### 180 Audit Log Analysis

181 The XDAS provides a set of interfaces for audit log analysis. It does not support queries on  
182 the audit log against a set of selection criteria. Nor does it define any of the audit log  
183 analysis tools.

184 It is assumed that the audit analysis tools will consolidate recorded security related events  
185 as part of their analysis of the audit logs.

### 186 Audit Log Management

187 The current XDAS specification views the audit log as a stream of time ordered audit event

188 records. No management structure is imposed on this stream and no functions are specified  
189 for the management of the system resources, for example files, used for the storage and  
190 processing of the stream

191

|

# Conformance Statement

192

193

The following XDAS implementation conformance categories are defined:

194

- **Basic XDAS Conformance**

195

This is applicable to an implementation of XDAS that supports the Common Audit Record Format and the Audit Read API in support of Audit Trail Analysis Applications. All implementations are required to comply with this basic conformance criteria.

196

197

198

- **XDAS Import API Option Conformance**

199

This is applicable to an implementation of XDAS that supports the Audit Log Import API.

200

- **XDAS Event Submission API Option Conformance**

201

This is applicable to an implementation of XDAS that supports the Audit Event Service Client API for direct use by applications.

202

203

- **XDAS Filter Management API Option Conformance**

204

This is applicable to an implementation of XDAS that supports a filtering capability and the Audit Event Management API.

205

206

## 2.1 Basic XDAS Conformance

207

An implementation of XDAS that conforms with this conformance category shall support the following interfaces:

208

209

`xdas_close_audit_stream`      `xdas_get_next`

210

`xdas_initialise_session`      `xdas_open_audit_stream`

211

`xdas_release_buffer`      `xdas_rewind_audit_stream`

212

`xdas_terminate_session`

213

## 2.2 XDAS Import API Option Conformance

214

An implementation of XDAS that conforms with this conformance category shall support the following interfaces in addition to those defined for Basic XDAS Conformance:

215

216

`xdas_import_event_records`

**217 2.3 XDAS Event Submission API Option Conformance**

218 An implementation of XDAS that conforms with this conformance category shall support the  
219 following interfaces in addition to those defined for Basic XDAS Conformance:

220	xdas_commit_record	xdas_discard_record
221	xdas_put_event_info	xdas_start_record
222	xdas_timestamp_record	

**223 2.4 XDAS Filter Management API Option Conformance**

224 An implementation of XDAS that conforms with this conformance category shall support the  
225 following interfaces in addition to those defined for Basic XDAS Conformance:

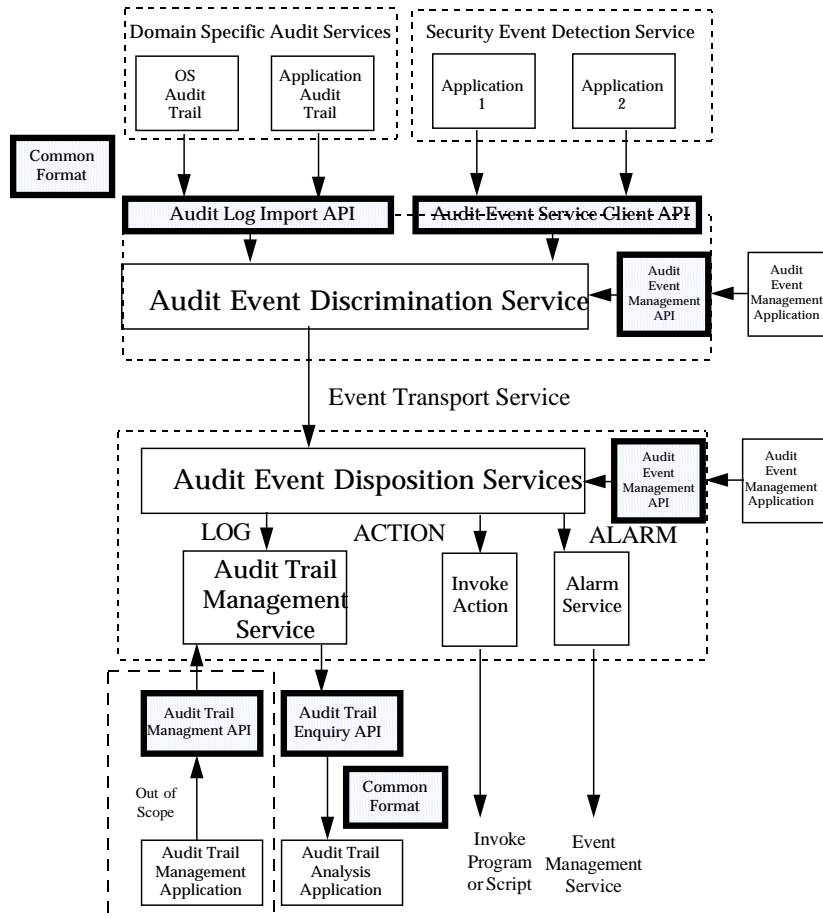
226	xdas_create_filter	xdas_delete_filter
227	xdas_disable_filter	xdas_enable_filter
228	xdas_get_filter	xdas_list_filters
229	xdas_release_filter_list	

230

231

232 **3.1 Introduction**

233



234

**Figure 3-1** Distributed Audit Service Interfaces

235

The XDAS Audit Service provides an API to support:

236

- the submission of audit events by applications

237

- the import of information from audit logs generated by domain specific audit services

238

- control of the filtering of audit events prior to submission or import

239

- control of the disposition of events as a combination of any of logging, action initiation and alarm triggering

240

- 241           • the analysis of audit logs.

242           The Distributed Audit Service model discussed in this section is illustrated in Figure 3-1 on page  
243           9. This is a logical representation and does not reflect a particular physical architecture. It  
244           comprises the following components:

245           **Security Event Detection Service**

246           The *Security Event Detection* service resides in the callers of the XDAS Audit Event Service  
247           Client API (shown in the diagram as applications 1 and 2.) An application is responsible for  
248           detecting security relevant activity in the context of its own local domain and to generate an  
249           audit event record which contains a description of the activity and information about the  
250           local security context. An application report the events it detects via the *Audit Event Service*  
251           *Client API*.

252           **Audit Event Import Service**

253           Many domains, in particular operating systems, provide their own audit service designed to  
254           meet their domain's specific needs in terms of event types and the information recorded  
255           about an event. The *Audit Event Import Service* provides for the import of audit events from  
256           a domain specific log for the purposes of merging with XDAS audit information into a time  
257           ordered sequence of records for the support of analysis of audit events across domains. In  
258           order to use the import service a local domain needs to provide a facility to translate its own  
259           audit records into the XDAS common audit event record format.

260           **Note:** The translation to the XDAS common audit event record format does not  
261           necessarily preserve all information in the original audit record. The XDAS  
262           common audit event record format includes information that can be used to locate  
263           the original record within the originating domain's audit trail.

264           **Audit Event Discrimination Service**

265           The *Audit Event Discrimination Service* discriminates all incoming events against pre-set  
266           criteria which are configured via the *Audit Event Management Service*. Those which do not  
267           meet the criteria are ignored. Those which do are passed to the *Audit Event Disposition*  
268           *Service*.

269           **Audit Event Disposition Service**

270           The *Audit Event Disposition Service* receives security relevant events from the *Audit Event*  
271           *Discrimination Service*. Based upon configuration data, the audit disposition service invokes  
272           one or more of the following services:

- 273           • an *Audit Trail Management Service* for logging the event,  
274           • an *Invoke Action Service* for invoking a command or application configured for  
275           invocation on the occurrence of the event.  
276           • an *Alarm Delivery Service* that submits the event to an Event Management Service for  
277           handling as a system alarm.

278           **Audit Trail Management Service**

279           The *Audit Trail Management Service* receives audit events and stores them in the *Audit*  
280           *Stream*, in an implementation defined format.

281           The *Audit Trail Management Service* supports:

- 282           • The *Audit Trail Management Service* supports configuration and management of the  
283           system resources used to store and process the audit records. For example, files which  
284           are often referred to as audit logs. The service allows the location of the audit logs to be  
285           defined, as well as how and when the service switches from one audit log to the next in  
286           the set. The service also supports the archiving of the audit log in the common audit



287 event record format and the retrieval of logs for analysis  
288 This version of XDAS is not defining an audit log management API. This is unnecessary  
289 for support of the primary objectives of XDAS. XDAS interfaces for recording audit  
290 event records and analysing audit event records perceive the audit log as a single time  
291 ordered stream of records.

- 292 • The *Audit Trail Enquiry API* provides query access to records on the audit log according  
293 to submitted post-selection criteria. The *Audit Trail Enquiry API* presents security audit  
294 event information in a common audit log format. See "Common Format" illustrated in  
295 Figure 3-1 on page 9.

## 296 3.2 Interfaces

297 Five application audit APIs are identified in the model but only four are of these are within the  
298 current scope of this specification. The four APIs within scope are:

### 299 **Audit Event Service Client API**

300 The *Audit Event Service Client API* is defined at the boundary to the *Audit Event*  
301 *Discrimination Service* for submission of audit events detected within application or platform  
302 services

### 303 **Audit Event Import API**

304 The *Audit Event Import API* is defined at the boundary to the *Audit Event discrimination*  
305 *service* for the merging of a set of audit records recorded by a domain specific audit service  
306 with the XDAS audit stream. It requires the definition of a common, portable audit log  
307 format to support interoperability. See *Common format* in Figure 3-1 on page 9.

### 308 **Audit Event Management API**

309 The *Audit Event Management API* is defined to support management applications to  
310 configure the Audit Event Discrimination and Audit Event Disposition Services.

### 311 **Audit Trail Enquiry API**

312 The *Audit Trail Enquiry API* is defined for the analysis of audit records in the audit stream.

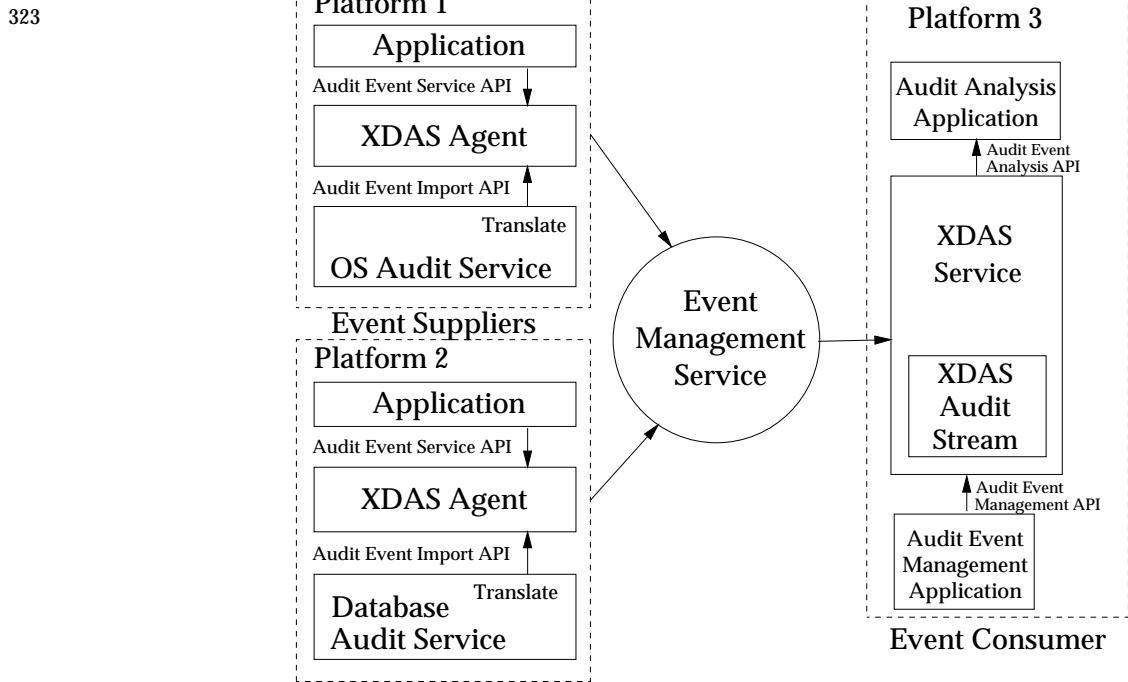
313 The fifth API, currently out of the scope of this specification is:

### 314 **Audit Trail Management API**

315 The *Audit Trail Management API* is defined to configure, manage and archive audit logs that  
316 comprise the XDAS audit stream.

317 **3.3 Distributed Audit Service Model**

318 The distributed aspect of an XDAS implementation is illustrated in Figure 3-2. For the purposes  
 319 of this illustration the XDAS implementation is shown as working over the X/Open Event  
 320 Management Service. Although this is a possible method of implementation, and one that is  
 321 capable of supporting interoperability between implementations (to the extent that XEMS  
 322 supports interoperability) it is not mandated by this specification.



324 **Figure 3-2** Distributed Audit Service Model

325 **3.3.1 XDAS Event Supplier Components**

326 An XDAS component executes on each platform within the distributed system. Those XDAS  
 327 components providing the **Audit Event Service API** and the **Audit Event Import API** are XEMS  
 328 Event Suppliers.

329 Applications may submit audit event records to the XDAS service via the *Audit Event Service*  
 330 *API*. Domain specific audit services, such as an operating system audit service, may submit  
 331 audit event records to the XDAS service for integration with the XDAS Audit Stream. In the case  
 332 of the *Audit Event Import API* then the caller is required to provide a translation service from the  
 333 domain specific format to the XDAS common audit event record format.

334 An XDAS Event Supplier uses the filtering rules to control the events that it submits to the Event  
 335 Management Service. No decisions regarding the disposition of XDAS events is made by an  
 336 XDAS Event Supplier.

**337 3.3.2 XDAS Event Consumer Components**

338 The XDAS components that handle the disposition of events are XEMS Event Consumers. The  
339 XEMS passes XDAS events submitted to it to XDAS Event Consumers. These components use  
340 the action part of the filter rules to control the disposition of the XDAS events received. The  
341 actions are to:

- 342 • Log the event
- 343 • Initiate an action by invoking a program or script
- 344 • Initiate an alarm by submitting the XDAS event to the Event Management System as a  
345 system alarm.

346 An audit analysis application is illustrated using the *Audit Event Analysis API* and an Audit  
347 Event Management Application using the *Audit Event Management API* from a central XDAS  
348 Management platform. The actual location and internal structuring of the XDAS Audit Stream is  
349 implementation defined.

350 The method and format for communicating filtering criteria to the individual XDAS Event  
351 Supplier components is not defined by this version of the specification.



353

354 This chapter presents a definition of the data structures needed for the Distributed Audit  
 355 Service.

## 356 **4.1 Audit Record Stream**

357 The XDAS API assumes that audit event records are inserted into and read from a time  
 358 sequenced stream of audit records in a common format. This stream of records is termed an  
 359 *Audit Stream*. The organisation and management of the system resources used to comprise the  
 360 audit stream is implementation defined.

## 361 **4.2 Audit Event Record**

362 Information regarding an audit event is recorded in an *Audit Event Record*. The following section  
 363 presents a definition of the portable *common exchange format* for audit event records. This is the  
 364 format in which records are submitted to, or retrieved from, the XDAS API.

365 The audit record contents are represented using the ISO LATIN1 character set. This does not  
 366 assume that the record contents are in a form that can be displayed as readable text. In addition,  
 367 manifest constants should not be localised by any internationalisation routines used within  
 368 XDAS implementations.

369 The *audit event record* comprises:

- 370 • firstly, a minimum set of common information needed to support the filtering of audit events  
 371 and a top level analysis across the distributed environment for the purposes of traceability  
 372 and assignment of accountability.
- 373 • secondly, for events originated within a domain specific audit service and imported into  
 374 XDAS, a pointer to the location and position of the original record within the originating  
 375 domain audit service to support more detailed analysis using domain specific audit tools if  
 376 required.
- 377 • thirdly, provision for recording detailed domain event specific information within the record  
 378 itself that can be used for more detailed analysis of activity within the context of the service  
 379 originating the event. This may be used instead of or in addition to the pointer to the original  
 380 record.

381 Thus, the detailed information from the source domain is not necessarily required for analysis in  
 382 the context of the distributed environment. For example, an agent may have created objects in a  
 383 database, the distributed environment may only be interested in the fact that database objects  
 384 have been created, and not specifically in the type of database object, say a trigger.

385 In order to be both portable and extensible, the format proposed here adopts an approach based  
 386 on self-defining attributes expressed in a textual format. See Chapter 6 for the actual format.

387 The structure of an audit record is as follows:

### 388 **header**

389 The header is a mandatory component of an audit event record and contain essential  
 390 information about the event to be recorded:

- 391 • The *length* of the audit record (generated by the implementation)
- 392 • The *version\_number* of the service, so that analysis tools can accurately interpret the
- 393 information to follow (generated by the implementation).
- 394 • The *date\_and\_time* of the event (generated by the implementation at the time at which
- 395 the caller commits an audit event record to the stream.)

396 The XDAS specification includes the date and time of the start of the current EPOCH  
 397 which applies to the current version of the XDAS record format. (Start of the day  
 398 January 1, 1970) Time is represented as the:

- 399 • The *offset* in milliseconds from the beginning of the EPOCH
- 400 • The *uncertainty interval* in milliseconds of offset
- 401 • The *uncertainty indicator* as a percentage of confidence in the uncertainty interval
- 402 • The *signal* or *source* of trusted time.
- 403 • The *timezone*

404 The uncertainty interval and uncertainty indicator shall default to NULL. These are  
 405 considered placeholders for future use.

- 406 • The *event\_number*, a number which uniquely identifies the event (provided by the caller)
- 407 • The *outcome* of the event, ie., its success or failure (provided by the caller)

#### 408 **originator\_information**

409 The originator of an event is defined as the service that detects and requests the recording of  
 410 an audit event. As such it defines the security domain in which the event occurs.

411 The *originator\_information* is a mandatory component of an audit event record. It is  
 412 generated by the implementation on the basis of information provided by the caller when  
 413 an association between the caller and the audit service is initialised.

#### 414 **initiator\_information**

415 The initiator of an event is defined as the principal that is accountable for the initiation of  
 416 the action that results in the audit event.

417 The *initiator\_information* is a mandatory component of an audit event record and is provided  
 418 by the caller.

#### 419 **target\_information**

420 This defines the target on which the initiator has acted. The target may be the identity of a  
 421 service with which a session has been initiated or terminated,

422 The *target\_information* is an optional component of an audit event record and is provided by  
 423 the caller.

#### 424 **source\_reference**

425 The *source\_reference* is a pointer to the original audit event record for those records that have  
 426 been imported to the XDAS service from a domain specific audit service. The intention is  
 427 that this information provides the location of the audit record within the original domain if  
 428 more detailed analysis is required. This information is provided by the original domain  
 429 when calling the XDAS import API.

#### 430 **event\_specific\_information**

431 The *event\_specific\_information* is provided for primary use by applications using XDAS as  
 432 their primary audit service. *Event\_specific\_information* varies from one event to the next and  
 433 is specific to the context of originating security domain identified by the *originator\_identity*

434 The *event\_specific\_information* may include the information pertaining to the security context  
435 of originator, initiator or target.

436 The structure of this field is required to be textual, that is, it cannot contain any binary data  
437 except in an encoded format. It is expected to comprise a number of *attribute=value* pairs.

## 438 4.3 Originator, Initiator and Target Information

### 439 4.3.1 Originator\_Information

440 The information associated with an originator, the service that detects and records an audit  
441 event, comprises:

- 442 • **Location\_Name**  
443 the name of the host/service defined using the syntax and quoting rules defined in Appendix  
444 B.
- 445 • **Location\_Address**  
446 This is a communication service end point address. Comparisons on this data should use  
447 bitwise comparison.
- 448 • **Service\_Type**  
449 The *service\_type* may include information about the particular subset of functions being  
450 provided by the originator. For example, a service provider may support different subsets of  
451 functions according to the port by which it is invoked. It is represented as a text string.
- 452 • **Authentication Authority**  
453 is defined using the syntax and quoting rules defined in Appendix B. Examples of an  
454 *authentication authority* are the name of a kerberos realm, an NIS domain, and a UNIX  
455 hostname.
- 456 • **Originator Name**  
457 the originator principal name as authenticated by authentication authority. Examples of  
458 principal names are a kerberos principal name, and a UNIX username.
- 459 • **Originator Identity**  
460 the originator principal identity. Examples are the DCE UUID and a UNIX uid.

461 It is not mandatory that both the *location\_name* and the *location\_address* are completed, but at  
462 least one of them must be.

463 The *authentication authority*, *originator name* and *originator identity* represent the authenticated  
464 identity of the originator. Some of this information may not be available for inclusion in the  
465 audit record.

### 466 4.3.2 Initiator\_Information

467 The information associated with an initiator comprises

- 468 • **Authentication Authority**  
469 defined using XFN syntax. Examples of an *authentication authority* are the name of a kerberos  
470 realm, an NIS domain, and a UNIX hostname.
- 471 • **Initiator Name**  
472 the initiator principal name. Examples of principal names are a kerberos principal name, and  
473 a UNIX username.

474           • **Initiator Identity**  
475           the initiator principal identity. Examples of principal identities are a DCE UUID and a UNIX  
476           uid.

477       **Note:** It should be noted that in some countries, for example, Germany, it is illegal to  
478           associate events directly with individual users without an additional reference stage in  
479           the analysis. This may influence the information that is actually stored in an XDAS  
480           record.

### 481 **4.3.3 Target\_Information**

482 The target of an activity that results in an auditable event may be:

- 483       • an "object" that may be identified by a name within the originating domain's namespace. For  
484       example a file on a UNIX platform, a record within a database.
- 485       • a service with which an association is established.

486       In the case of client-server operations, when an association is created then both ends may be  
487       considered to be the target of the other even though strictly speaking one side is the initiator.  
488       For events recording the creation of associations the *target\_information* therefore records  
489       information about the remote service component. The *initiator\_information* therefore always  
490       references the original (normally end-user) principal.

491       The service may assign its own representation of the principal identity to the *Initiator* (e.g.,  
492       using a local account database.) In this case the identity assigned needs to be recorded to  
493       support traceability at the distributed system level.

494       The target of an activity that results in an auditable event is represented as for  
495       *originator\_information*.

## 496 **4.4 Identification of Audit Events**

497       The identification of audit events is an important part of supporting requirements to filter and  
498       select audit events.

499       Audit Events may be specifically referenced by an *Event Number*. A set of Audit Events may be  
500       referenced by an *Event Class*. A potential set of generic Event Classes are listed at the end of this  
501       section.

502       The purpose of defining *Event Classes* is to facilitate the definition of filtering criteria for the  
503       control of the audit service and for facilitating the definition of search criteria for audit analysis.  
504       An audit event record only includes the *Event Number*. It does not include any reference to *Event*  
505       *Class*



#### 506 4.4.1 Event Numbers

507 XDAS uses the event numbering scheme defined by the DCE auditing service in OSF RFC 29.2.

508 X/Open will register an Open Group set id and a set of numbers under that set id for the XDAS  
509 events identified. It is possible for application developers to register their own set of event  
510 numbers if they wish to utilise the services of XDAS for more domain specific auditing not  
511 catered for by the generic set of XDAS events

#### 512 4.4.2 XDAS Events

513 The following generic events are registered by XDAS. Not all of these events are necessarily  
514 security significant within all domains. For example the querying of attributes or configuration  
515 data is not necessarily of security significance.

##### 516 Account Management Events

517 This set of events is applicable to the management of principal accounts. A principal may  
518 be an end-user or a service within the system, a psuedo-user.

- 519 • **Create account**

520 The creation of an account representing a principal within a domain.

- 521 • **Delete account**

522 The deletion of an account representing a principal from a domain.

- 523 • **Disable account**

524 An action the prevents a principal account from being used within a domain.

- 525 • **Enable account**

526 An action that permits a principal account to be used within a domain.

- 527 • **Query account attributes**

528 The requesting of the attributes associated with a principal within a domain.

- 529 • **Modify account attributes**

530 The modification of the attributes associated with a principal within a domain.

##### 531 User Session Events

532 This set of events is relevant to the creation and use of user sessions on the system.

- 533 • **Create a user session**

534 The establishment of a processing environment to service an end user.

- 535 • **Terminate a user session**

536 The dismantling of a processing environment associated with servicing an end user.

- 537 • **Query user session attributes**

538 The requesting of the attributes associated with a user session.

- 539 • **Modify user session attributes**

540 The modification of security significant attributes of the context of a processing  
541 environment servicing an end user.

##### 542 Data item and Resource Element Management Events

543 This set of events relate to the creation and management of data items and resource  
544 elements within a domain. The type of data item or resource element is dependent upon the  
545 domain, e.g., files and directories, device special files, shared memory segments, within an  
546 operating system, tables and records within a database, messages within an email system.  
547 The term data item is used to refer to any type of resource element.

- 548           • **Create data item**  
549            Creation of a data item within a domain.
- 550           • **Delete data item**  
551            Deletion of a data item from a domain.
- 552           • **Query data item attributes**  
553            Request the attributes associated with a domain data itemt.
- 554           • **Modify data item attributes**  
555            Modification of the security attributes of a domain data item such as access control  
556            attributes, ownership, aliases
- 557           **Service or Application Management Events**  
558            This set of events relate to the management of system services and applications.
- 559           • **Install service or application**  
560            The installation of additional or updated software on a system., e.g., an application or  
561            system service.
- 562           • **Remove service or application**  
563            The deinstallation of software on a system.
- 564           • **Configure service or application**  
565            The modification of the configuration data associated with a software component.
- 566           • **Query configuration of service or application**  
567            The requesting of information about the configuration of a service or application.
- 568           • **Disable service or application**  
569            An action that prevents an application or system service from being used, for example,  
570            inhibiting responses to service requests. It may also involve the termination (shutdown)  
571            of application processing components that are currently providing the service.
- 572           • **Enable service or application**  
573            An action that permits an application or system service to be used, for example,  
574            allowing responses to service requests. This may also involve the invocation of specific  
575            application processing components (startup).
- 576           **Service and Application Utilisation Events**  
577            These events relate to the use of service and applications. They typically map to the  
578            execution of a program or a procedure and manipulation of the processing environment.
- 579           • **Invoke service or application**  
580            Invocation of a service or application (exec), e.g., operating system utility, database,  
581            accounting application, etc.
- 582           • **Terminate service or application component**  
583            Terminate (exit) the use of a service or application. This could be at the instigation of the  
584            application itself or by the intervention of the domain in response to user or  
585            administrative action.
- 586           • **Query processing context**  
587            Query the attributes associated with the current processing environment.
- 588           • **Modify processing context**  
589            Modify the attributes associated with the current processing environment.
- 590           **Peer Association Management Events**

- 591           • **Create an association with a peer**
- 592            The creation of a communication channel and the processing context between system
- 593            components.
- 594           • **Terminate an association with a peer**
- 595            The closure of a communications channel and destruction of processing context between
- 596            system components.
- 597           • **Query an association context**
- 598            The query of the attributes of a context associated with a communications channel
- 599            between peers.
- 600           • **Modify an association context**
- 601            The modification of the attributes of a processing context associated with a
- 602            communications channel.
- 603           • **Receive data via an association**
- 604            Receive data from associated peer within current association context.
- 605           • **Send data via an association**
- 606            Send data to associated peer within current association context.

#### **Data Item or Resource Element Content Access Events**

- 607           These events relate to the formation of an association between a service or application and a
- 608           data item or resource element for the purpose of using its contents or services. For example,
- 609           a file or directory, device special file, memory segment, communications port, etc.
- 610
- 611           • **Create association with data item**
  - 612            Create an association with (open) a data item. This creates a binding between the caller
  - 613            and the data item.
  - 614           • **Terminate association with data item**
  - 615            Terminate an existing association with (close) a data item.
  - 616           • **Query context of association with data item**
  - 617            Query the context of an association with a data item, e.g., mode of access, size limits,
  - 618            access path, etc.
  - 619           • **Modify context of association with a data item**
  - 620            Modify the context of an association with a data item or resource element.
  - 621           • **Query data item contents**
  - 622            Requesting the contents of a domain data item (read).
  - 623           • **Modify data item contents**
  - 624            Modification of the contents of a domain data item (write, append etc).

#### **Exceptional Events**

- 625           These are events that are considered to be outside the generalised events listed above.
- 626
- 627           • **Start system**
  - 628            The action of booting a system host or of changing the processing state of a system host
  - 629            to an operational mode.
  - 630           • **Shutdown System**
  - 631            The action of halting the processing by a system host or of changing the processing state
  - 632            of a system host to a maintenance mode.
  - 633           • **Resource exhaustion**
  - 634            The detection of resource exhaustion which has a potential impact on system operations,

635 perhaps based upon a configurable threshold, e.g., data storage resources,  
636 communication end points.

### 7 **Notes to Reviewers**

8 *This section with side shading will not appear in the final copy. - Ed.*

9 This could alternatively be called service exhaustion or service availability failure"

- 640 • **Resource corruption**

641 The detection of an integrity failure of a system resource, for example data storage  
642 resource.

### 3 **Notes to Reviewers**

4 *This section with side shading will not appear in the final copy. - Ed.*

5 This could alternatively be called service integrity failure"

- 646 • **Backup datastore**

647 The action of making a backup copy of a datastore for the purposes of protecting  
648 availability and integrity of the data it contains.

- 649 • **Recover datastore**

650 The action of restoring the contents of a datastore from a previously made backup copy  
651 for the purposes of restoring the availability of the contents, or the integrity of the  
652 contents, or both.

### 653 **Audit Service Management Events**

654 These are events of specific relevance to the audit service itself.

- 655 • **Configure audit service**

656 The modification of the parameters controlling the operation of the audit service, for  
657 example, audit event filtering criteria.

- 658 • **Audit datastore full**

659 The detection of resource exhaustion for the particular instance of the resource used to  
660 store the log of audit event records.

- 661 • **Audit datastore corrupted**

662 The detection of a datastore integrity failure for the particular instance of the resource  
663 used to store the log of audit event records.

### 664 **4.4.3 Event Classes**

665 Audit Events may be specifically referenced by an *Event Number*. A set of Audit Events may be  
666 referenced by an *Event Class*. The concept of an *Event Class* is included in the XDAS solely as an  
667 administrative convenience. It provides an efficient and convenient reference to sets of audit  
668 events so that audit filters can be easily defined. An audit event record only includes the *Event*  
669 *Number*. It does not include any reference to *Event Class* for two reasons: its inclusion leads to  
670 redundant information in the audit record; and the mapping of event classes across  
671 administrative domains is problematic. When specified in filtering selection criteria, an *event*  
672 *class* is translated internally into the individual event numbers.

673 **Default Event Classes**

674 The XDAS defines a default set of event classes. Others can be defined by the implementation  
 675 and configured by a system administrator to group together XDAS event numbers in a  
 676 meaningful way. The default set of event classes defined by the XDAS are listed below:

- 677 • Account management events
- 678 • User session events
- 679 • Data item and resource element management events
- 680 • Service and application management events
- 681 • Peer association management
- 682 • Data item or resource element content access events
- 683 • Exceptional events
- 684 • Audit service management events

685 The default mapping of events to these event classes is as listed in Section 4.4.2

686 **4.4.4 Outcomes**

687 An event may be identified by both its event number and outcome. The following outcome  
 688 codes and sub-codes are defined by this specification:

Outcome	Outcome Description
Successful	XDAS_OUT_SUCCESS XDAS_OUT_PRIV_USED XDAS_OUT_PRIV_GRANTED XDAS_OUT_PRIV_DENIED XDAS_OUT_PRE_SELECT_CRITERIA_SET XDAS_OUT_THRESHOLDS_SET XDAS_OUT_ACTIONS_SET XDAS_OUT_THRESHOLD_EXCEEDED
Failure	XDAS_OUT_FAILURE XDAS_OUT_SERVICE_UNAVAILABLE XDAS_OUT_SERVICE_FAILURE XDAS_OUT_HARDWARE_FAILURE XDAS_OUT_LOST_ASSOCIATION XDAS_OUT_ALREADY_ENABLED XDAS_OUT_ALREADY_DISABLED XDAS_OUT_SERVICE_ERROR XDAS_OUT_BUSY XDAS_OUT_DISABLED XDAS_OUT_INVALID_INPUT XDAS_OUT_ENTITY_EXISTS XDAS_OUT_ENTITY_NON-EXISTENT
Denial	XDAS_OUT_DENIAL XDAS_OUT_INSUFFICIENT_AUTHORIZATION

714  
715  
716  
717

Outcome	Outcome Description
	XDAS_OUT_INVALID_IDENTITY XDAS_OUT_INVALID_CREDENTIALS

718 **4.5 Event Selection**

719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748

Field	Event Submission	Event Import	Event Analysis
Header:			
Length	-	-	-
Version	X	X	X
Date	-	X	X
Event Number	X	X	X
Outcome	X	X	X
Originator_Information:			
location_name	-	X	X
location_address	-	X	X
service_type	-	X	X
auth_authority	-	X	X
name	-	X	X
identity	-	X	X
Initiator_information:			
auth_authority	X	X	X
name	-	X	X
identity	-	X	X
Target_information:			
location_name	-	X	X
location_address	-	X	X
service_type	-	X	X
auth_authority	-	X	X
name	-	X	X
identity	-	X	X
Source:	-	-	X
Event_Specific:	-	-	X

**Table 4-1** Event Filtering Criteria

749 Event selection may be applied at three places within the XDAS architecture:

- 750 • Pre-selection criteria may be applied when an event is detected to determine whether an
- 751 event is to be logged or an action initiated, or both.
- 752 • Selection criteria may be applied when an event is imported from a domain specific audit
- 753 service to determine whether the event is to be imported and if so whether the event is
- 754 logged or an action initiated, or both.
- 755 • Post-selection criteria may be applied by audit analysis applications to control the selection
- 756 of audit event records from the XDAS audit stream.

757 Table 4-1 on page 24 sets out the filtering criteria for pre-and post-selection criteria. An "X"  
758 indicates that the field is available for filtering; a "-" that it is not.

#### 759 **4.5.1 Event Submission Pre-Selection Filtering**

760 The filtering criteria for pre-selection of events on event submission is constrained by  
761 considerations of limiting the performance impact of evaluating the criteria on the calling  
762 application and the system as a whole.

763 Whilst date and time of day are valid requirements for filtering on event submission, they are  
764 not included as mandatory requirements in the table. This is because this selection can be  
765 achieved more efficiently using a scheduling service to switch event filtering criteria as a whole.

766 The event originator is not included in the table, even though it is a valid requirement for  
767 filtering. This filtering can be achieved more easily as an application level facility which turns  
768 auditing on or off for the application as a whole, or for subservices within an application. It is  
769 not considered to be a valid XDAS function.

770 Filtering by initiator *auth\_authority* is a requirement as an *auth\_authority* may be compromised or  
771 otherwise untrusted. However, controlling filtering by individual identity impacts performance  
772 significantly and thus, it is not a mandatory requirement in the XDAS. Such filtering is more  
773 efficiently performed on import or post-selection analysis.

#### 774 **4.5.2 Event Import Pre-Selection Filtering**

775 XDAS supports a much richer set of filter criteria for controlling the selection of records for  
776 import to XDAS as the performance impact is of lesser concern in this case.

#### 777 **4.5.3 Event Analysis Post Selection Filtering**

778 Post selection filtering is the responsibility of the analysis application. XDAS does not itself  
779 apply filtering to the audit records returned by the Audit Read API and therefore does not  
780 include interfaces to support this.

#### 781 **4.5.4 Event Filters**

782 An event filter comprises the following information:

##### 783 **Version Number**

784 The XDAS version number.

##### 785 **Filter Name**

786 A name by which the filter is referenced.

##### 787 **Filter Type**

788 The filter applies to the event submission or event import interface, or both interfaces.

##### 789 **Flag**

790 The flag which indicates whether the filter is enabled or disabled.

##### 791 **Expression List**

792 A set of expressions AND'd to establish the complete filter to be applied.

##### 793 **Action List**

794 The actions to be taken when the event is detected.

#### 795 4.5.5 Filter Expression List

796 A filter expression list comprises a set of expressions that are ANDed to establish the complete  
797 expression to be applied. This specification does not assume any precedence or ordering of the  
798 evaluation of a set of filters (although an implementation may apply one for performance  
799 reasons.) If an event requires auditing under the filtering criteria of any individual filter then it  
800 shall be audited, even if excluded by other filters. In the circumstance that an event is required  
801 to be audited by multiple filters then duplicate audit event records shall not be created.

802 An expression comprises:

##### 803 **Include/Exclude Flag**

804 Events matching this expression are to be included or excluded from selection. When a  
805 filter is evaluated all inclusions are processed first and followed by all exclusions.

##### 806 **Attribute**

807 The event attribute or field.

##### 808 **Operator**

809 The operator defines the boolean operation to be performed on the attribute. Operators are  
810 equal, greater than, less than, greater than or equal, less than or equal, not equal, bitwise  
811 ANDed, substring.

##### 812 **Value**

813 The value against which the attribute value in the event is tested.

#### 814 4.5.6 Filter Action List

815 The action list is a list comprising a constant to indicate the action and a text string.

##### 816 **XDAS\_CONSTANT**

817 The action to be taken. This can be LOG, ALARM or ACTION.

##### 818 **Text String**

819 A text string that provides additional information pertinent to the action to be taken.

820 Examples of the *filter action list* are

- 821 • LOG + NULL string
- 822 • ALARM + Severity Code
- 823 • ACTION + Pathname of executable or script to invoke and input parameters

824



# DAS Usage Model

825

826 The XDAS comprises both operational and management services. The operational XDAS  
 827 services are those available to applications in support of the logging of audit records. The  
 828 management services support the configuration and management of audit events, the audit  
 829 service itself, as well as providing interfaces for the analysis of audit records.

830 The XDAS places a dependency on an Event Management Service such that the intermediate  
 831 event management components do not modify the filtering or routing of audit events, thereby  
 832 ensuring that an audit alarm, for example, is not filtered out part way to its destination

833 Operational services include:

- 834 • *General Audit Service API*, used by all callers of the XDAS.

835 All callers are required to initiate a session with the XDAS audit service. This authenticates  
 836 the caller's identity and establishes a session between the caller and the XDAS. Thereafter,  
 837 callers may use the XDAS APIs to log events, configure the audit service, or analyse audit  
 838 streams subject to the XDAS authorities assigned to them.

- 839 • The *Audit Event Service Client API*, used by applications to submit security relevant events to  
 840 the Audit Service.

841 These allow audit records to be created, filled and committed to the implementation defined  
 842 audit log in common format.

- 843 • The *Audit Log Import API*, used by domain specific audit services to import audit records in  
 844 the XDAS common audit event record format into the XDAS audit stream.

845 Management services include:

- 846 • The *Audit Event Management API*, used by applications to configure the pre-selection criteria  
 847 for the *Audit Event Discrimination Service* and the *Audit Event Disposition Service*

- 848 • The *Audit Read API*, used by applications to retrieve events from the audit stream for the  
 849 purposes of analysis.

## 850 5.1 Authorisation Policy

851 The authorisation policy inherent in the XDAS-API is defined on the principle of the separation  
 852 of duties. The granting of XDAS authorities is under the control of authorisation security  
 853 services. The following XDAS authorities have been defined.

### 854 XDAS\_AUDIT\_SERVICE

855 required to initialise a session with the XDAS audit service

### 856 XDAS\_AUDIT\_SUBMIT

857 for using the audit logging interfaces of the Audit Event Service Client

### 858 XDAS\_AUDIT\_IMPORT

859 required to import audit events records from a domain specific audit service.

### 860 XDAS\_AUDIT\_CONTROL

861 for use of the Audit Event Management APIs

862 **XDAS\_AUDIT\_READ**  
863 for access to the Audit Read API

864 **XDAS\_AUDIT\_PURGE**  
865 to authorise the removal of records from the XDAS audit stream

866 Each interface specification includes the XDAS authority required to be possessed by a caller in  
867 order to utilise the interface. The mechanism for enforcement of the authorisation policy is  
868 implementation specific. Support is included in this specification for the initialisation of a  
869 session between a caller and the XDAS service whereby the identity of the caller can be  
870 authenticated and appropriate authorisation attributes established.

## 871 **5.2 General Audit Service API**

872 **Initialise Session**  
873 Initialise a session with the XDAS. This call will fail unless the caller possesses at least one  
874 XDAS authority.

875 **Terminate Session**  
876 Terminate a session with the XDAS

877 All callers must initiate a session with the XDAS before they can use any of the services it  
878 provides. The initialisation of the session supports the mutual authentication of the audit client  
879 and audit service components and establishes the audit client's XDAS authorities. The caller is  
880 returned a handle to the XDAS service which is then used for all XDAS API functions. On  
881 completion, the caller must terminate the XDAS session.

882 The behaviour if a client dies or exits without calling *terminate session* is implementation defined.  
883 An implementation may take specific action to try and detect and terminate such sessions itself  
884 to address any potential denial of service risks.

## 885 **5.3 Audit Event Service Client API**

886 **Start Record**  
887 Allocate and initialise an audit record descriptor. The return from this indicates to the caller  
888 whether the event requires auditing or not under the current filtering criteria.

889 **Put Event Information**  
890 Add event specific information to the initialised audit record

891 **Commit Record**  
892 Write the audit record to the audit log

893 **Discard Record**  
894 Discard the audit record

895 **Time Stamp Record**  
896 Control the time at which the record is timestamped

897 Callers submit security relevant events to the *Audit Event Service Client API*. The functions build  
898 the record from the information given by the caller and from the processing environment. The  
899 interfaces cover the creation, filling and committing of an audit record to the audit trail.

## 900 5.4 Audit Log Import API

### 901 Import\_Event\_Records

902 This function supports the import to XDAS by another audit service of multiple audit event  
903 records formatted in the XDAS common audit event record format.

904 This service permits domain specific audit services to import their own audit records into the  
905 XDAS service for consolidation and analysis at the distributed system level. Only callers with  
906 the XDAS\_AUDIT\_IMPORT authority are permitted to use this function.

## 907 5.5 Audit Event Management API

### 908 Create Filter

909 Create or modify an audit filter defining the selection criteria and the action to be taken on  
910 detection.

### 911 List Filters

912 Get a list of the names of filters which have been defined

### 913 Release Filter List

914 Release the list of filter names returned by List Filters

### 915 Get Filter

916 Get the specified audit filter

### 917 Delete Filter

918 Delete the specified audit filter

### 919 Enable Filter

920 Enable the specified filter

### 921 Disable Filter

922 Disable the specified filter

923 The *Audit Event Management API* provides the means whereby the *Audit Event Discrimination*  
924 *Service* and the *Audit Event Disposition Service* are configured. Only callers with the  
925 XDAS\_AUDIT\_CONTROL authority are permitted to use these interfaces.

## 926 5.6 Audit Read API

### 927 Open Audit Stream

928 Open the XDAS audit stream for read

### 929 Rewind Audit Stream

930 Rewind the audit stream

### 931 Close Audit Stream

932 Close the XDAS audit stream

### 933 Get Next

934 Read the next set of audit records from the specified audit trail into buffer. The caller  
935 supplies the buffer length and the maximum number of records to be returned. The  
936 implementation may return as many records as will fit into the buffer up to the specified

937           maximum. The caller can then parse the buffer to extract individual records.  
938           The *Audit Read API* is used to extract records from the XDAS audit stream for analysis. The  
939           interface supports the copying of a record into a buffer where the contents may be examined by  
940           the caller. The interfaces are available to privileged callers who possess the  
941           XDAS\_AUDIT\_ANALYSIS authority.

942

## Parameter Passing Conventions

943

944 This chapter describes the data types and constants used by the the XDAS functions. It also  
945 explains calling conventions for these functions.

### 946 6.1 Structured Data Types

947 Wherever these XDAS-API C-bindings describe structured data, only fields that must be  
948 provided by all XDAS-API implementations are documented. Individual implementations may  
949 provide additional fields, either for internal use within XDAS-API routines, or for use by non-  
950 portable applications.

### 951 6.2 Integer Types

952 XDAS-API defines the following integer data type

953 `OM_uint32` 32-bit unsigned integer

954 Where guaranteed minimum bit-count is important, this portable data type is used by the  
955 XDAS-API routine definitions. Individual XDAS-API implementations include appropriate  
956 `typedef` definitions to map this type onto a built-in data type.

## 957 6.3 String Data and Similar Data

### 958 6.3.1 Byte Strings

959 Many of the XDAS-API routines take arguments and return values that describe contiguous  
 960 multi-byte data. All such data are passed between the XDAS-API and the caller using the  
 961 **xdas\_buffer\_t** data type. This data type is a pointer to a buffer descriptor consisting of a **length**  
 962 field, which contains the total number of bytes in the data, and a **value** field, which contains a  
 963 pointer to the actual data:

```
964     typedef struct xdas_buffer_desc_struct{
965         size_t    length;
966         void     *value;
967     } xdas_buffer_desc, *xdas_buffer_t;
```

968 Storage for data passed to the application by a XDAS-API routine using the **xdas\_buffer\_t**  
 969 conventions is allocated by the XDAS-API routine. The application may free this storage by  
 970 invoking the *xdas\_release\_buffer()* routine. Allocation of the **xdas\_buffer\_desc** object is always  
 971 the responsibility of the application; unused **xdas\_buffer\_desc** objects may be initialised to the  
 972 value **XDAS\_C\_EMPTY\_BUFFER**.

### 973 6.3.2 Character Strings

974 Certain multi-octet data items may be regarded as simple Latin-1 character strings as defined in  
 975 the ISO/IEC 8859-1 standard. Character strings are passed between the application and the  
 976 XDAS-API using the **xdas\_buffer\_t** data type, defined earlier.

### 977 6.3.3 Opaque

978 Certain multi-octet data items are considered opaque data types at the XDAS-API, because their  
 979 internal structure only has significance to the implementation. Examples of such opaque data  
 980 types are

#### 981 *audit service handle*

982 This is opaque to the caller and returned to the caller on initialisation of a session between  
 983 the caller and the XDAS audit service. It is subsequently passed as a parameter to each  
 984 XDAS-API call as a **xdas\_audit\_ref\_t** data type.

#### 985 *audit stream handle*

986 This is opaque to the caller and is returned to a caller of the *xdas\_open\_audit\_stream()*  
 987 function. It is subsequently passed as a parameter to those functions that manipulate an  
 988 audit stream as a **xdas\_audit\_stream\_t** data type.

#### 989 *audit record descriptor*

990 This is opaque to the caller and is returned to a caller of the *xdas\_start\_record()* function. It  
 991 is subsequently passed as a parameter to those functions that manipulate an audit record  
 992 for submission to the XDAS service as a **xdas\_audit\_rec\_desc\_t** data type.

993 **6.4 Common Audit Record Format**

994 The audit record format is defined as an ISO LATIN-1 character set in an `xdas_buffer_t`  
 995 structure. Fields are delineated with colons (:); where a colon is part of the alphanumeric string,  
 996 a % should be used as an escape character. Empty strings are represented by two adjacent  
 997 separator characters. Note that this is an ordered sequence. The common audit record format is  
 998 set out below:  
 999

1000	<b>field</b>	<b>Type</b>
1001	Header:	HDR
1002	<length in bytes>	Digits 0-9
1003	<ver#>	Digits 0-9
1004	<date/time>	Hexadecimal
1005	<offset>	Hexadecimal
1006	<uncertainty interval>	Hexadecimal
1007	<uncertainty indicator>	Hexadecimal
1008	<time source>	Alphanumeric
1009	<time zone>	Alphanumeric
1010	<event_number>:	Hexadecimal
1011	<outcome>	Hexadecimal
1012	Originator	ORG
1013	<location_name>	Alphanumeric
1014	<location_address>	Alphanumeric
1015	<service-type>	Alphanumeric
1016	<auth_authority>	Alphanumeric
1017	<principal_name>	Alphanumeric
1018	<principal_id>	Alphanumeric
1019	Initiator	INR
1020	<auth_authority>	Alphanumeric
1021	<domain_specific_name>	Alphanumeric
1022	<domain_specific_id>	Alphanumeric
1023	Target	TGT
1024	<location_name>	Alphanumeric
1025	<location_address>	Alphanumeric
1026	<service-type>	Alphanumeric
1027	<auth_authority>	Alphanumeric
1028	<principal_name>	Alphanumeric
1029	<principal_id>	Alphanumeric
1030	Source	SRC
1031	<pointer_to_source_domain>	Alphanumeric
1032	Event	EVT
1033	<event_specific_information>	Alphanumeric
1034	END	END

1035 The strings HDR, ORG, INR, TGT, SRC and EVT are included to support syntax checking. All  
 1036 fields should be included in the audit record, with separators, even if they are blank.

1037 **6.5 Filters**

1038 Filters are used to set the criteria for pre-selecting events to be recorded, or for selecting records  
 1039 to be imported from an audit stream.

1040 A filter comprises a name and a set of filter information. It is defined as:

```

1041     typedef struct xdas_filter_desc_struct{
1042         xdas_buffer_t          filter_name;
1043         OM_unit32              filter_type;
1044         xdas_bool_t           flag;
1045         xdas_buffer_t          expression_list;
1046         xdas_buffer_t          action_list;
1047     } xdas_filter_desc, *xdas_filter_t;
  
```

1048 A filter expression is defined as a **xdas\_buffer\_t** data type. It is a sequence of variable length  
 1049 ASCII Fields, separated by a ":" delimiter, as set out below. Note that if a colon is part of an  
 1050 alphanumeric string, the % should be used as an escape character. Empty strings are  
 1051 represented by two adjacent separator characters. The format for a filter expression is set out  
 1052 below:

	<b>field</b>	<b>Type</b>
1055	Include/Exclude Flag	Alphanumeric
1056	Attribute	Alphanumeric
1057	Operator	Alphanumeric
1058	Value	alphanumeric



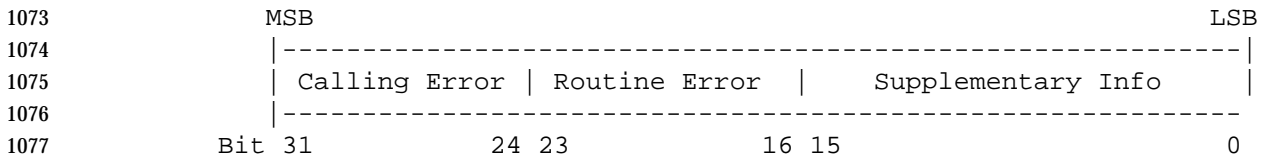
1059 **6.6 Status Values**

1060 One or more status codes are returned by each XDAS-API routine. Two distinct sorts of status  
 1061 code are returned. These are termed XDAS status codes and minor status codes. An  
 1062 implementation of XDAS functions shall return XDAS\_S\_COMPLETE and other status values  
 1063 appropriate for the implementation of the function. The characteristics of a particular  
 1064 implementation may make some status returns inappropriate for that implementation.

1065 **6.6.1 XDAS Status Codes**

1066 XDAS-API routines return XDAS status codes as their **OM\_uint32** function value. These codes  
 1067 indicate major status errors that are independent of the underlying mechanism used to provide  
 1068 the security service.

1069 A XDAS status code can indicate a single fatal generic API error from the routine and a single  
 1070 calling error. In addition, supplementary status information may be indicated by setting bits in a  
 1071 **Supplementary Info** field in a XDAS status code. These errors are encoded into the 32-bit XDAS  
 1072 status code as follows:



1078 Hence if a XDAS-API routine returns a XDAS status code whose upper 16 bits contain a non-  
 1079 zero value, the call failed. If the **Calling Error** field is non-zero, the invoking application's call of  
 1080 the routine was erroneous. Calling errors are defined in Table 6-1. If the **Routine Error** field is  
 1081 non-zero, the routine failed for one of the routine-specific reasons listed in Table 6-2 on page 36.  
 1082 Whether or not the upper 16 bits indicate a failure or a success, the routine may indicate  
 1083 additional information by setting bits in the **Supplementary Info** field of the status code. This  
 1084 specification does not currently define any supplementary information but it is included to  
 1085 accommodate a possible future expansion in scope that might require such information.

Name	Value in Field	Meaning
[XDAS_S_CALL_INACCESSIBLE_READ]	1	A required input argument cannot be read.
[XDAS_S_CALL_INACCESSIBLE_WRITE]	2	A required output argument cannot be written.
[XDAS_S_CALL_BAD_STRUCTURE]	3	An argument is malformed.

1094 **Table 6-1** Calling Errors

Table 6-2 Routine Errors

1095  
1096  
1097  
1098  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137

Name	Value in Field	Meaning
[XDAS_S_COMPLETE]	0	Successful completion.
[XDAS_S_FAILURE]	1	An implementation specific error or failure has occurred.
[XDAS_S_AUTHORISATION_FAILURE]	2	The caller does not possess the required authority.
[XDAS_S_END]	3	The end of the audit stream has been reached.
[XDAS_S_INVALID_ACTION_LIST]	40	The action list supplied is not valid.
[XDAS_S_INVALID_AUDIT_STREAM]	5	The audit stream supplied is not valid.
[XDAS_S_INVALID_DAS_REF]	6	The audit daemon handle supplied does not point to the audit daemon.
[XDAS_S_INVALID_EVENT_INFO]	7	The specified audit event information is not valid.
[XDAS_S_INVALID_EVENT_NO]	8	The event number supplied is not valid.
[XDAS_S_INVALID_FILTER]	9	The filter name supplied is not valid.
[XDAS_S_INVALID_FILTER_EXPR]	10	The filter expression supplied is not valid.
[XDAS_S_INVALID_FILTER_LIST]	11	The list of filter names supplied is not valid.
[XDAS_S_INVALID_FILTER_TYPE]	12	The filter type supplied is not valid.
[XDAS_S_INVALID_INITIATOR_INFO]	13	The initiator information has a syntax error.
[XDAS_S_INVALID_ORIG_INFO]	14	The originator information has a syntax error.
[XDAS_S_INVALID_OUTCOME]	15	The specified outcome is invalid.
[XDAS_S_INVALID_RECORD_DESCRIPTOR]	16	The specified audit record descriptor is not valid.
[XDAS_S_INVALID_SECURITY_CONTEXT]	17	The security context supplied is invalid.
[XDAS_S_INVALID_TARGET_INFO]	18	The target information has a syntax error.
[XDAS_S_NO_AUDIT]	19	The event does not need to be audited.
[XDAS_S_RECORD_SYNTAX_ERROR]	20	A syntax error has been detected in an input record
[XDAS_S_STORAGE_FAILURE]	21	The audit record cannot be written to stable storage.
[XDAS_S_SERVICE_FAILURE]	22	There has been an audit service failure
[XDAS_S_UNCERTAIN_AUDIT]	23	It is not certain whether the event should be audited.

1138 The function specifications also use the name [XDAS\_S\_COMPLETE], which is a zero value, to  
1139 indicate an absence of any API errors or supplementary information bits.

1140 All [XDAS\_S\_\*] symbols equate to complete **OM\_uint32** status codes, rather than to bit-field  
1141 values. For example, the actual value of the symbol [XDAS\_S\_BAD\_SIZE] (value 3 in the  
1142 **Routine Error** field) is  $3 \ll 16$ .

1143 The macros:

1144 XDAS\_CALLING\_ERROR()  
1145 XDAS\_ROUTINE\_ERROR()  
1146 XDAS\_SUPPLEMENTARY\_INFO()

1147 are provided, each of which takes a XDAS status code and removes all but the relevant field. For  
1148 example, the value obtained by applying XDAS\_ROUTINE\_ERROR() to status code removes  
1149 the **Calling Errors** and **Supplementary Info** fields, leaving only the **Routine Errors** field. The  
1150 values delivered by these macros may be directly compared with a [XDAS\_S\_\*] symbol of the  
1151 appropriate type. The macro XDAS\_ERROR() is also provided, which when applied to a XDAS  
1152 status code returns a non-zero value if the status code indicates a calling or routine error, and a  
1153 zero value otherwise.

1154 A XDAS-API implementation may choose to signal calling errors in a platform-specific manner  
1155 instead of, or in addition to the routine value; routine errors and supplementary information  
1156 should be returned by means of routine status values only.

## 1157 **6.6.2 Minor Status Codes**

1158 XDAS-API C-language functions return a *minor\_status* argument, which is used to indicate  
1159 specialised errors from the underlying security mechanism. This argument may contain a single  
1160 mechanism-specific error, indicated by an **OM\_uint32** value.

1161 The *minor\_status* argument is always set by a XDAS-API function, even if it returns a calling  
1162 error or one of the generic API errors indicated above as fatal, although other output arguments  
1163 may remain unset in such cases. However, output arguments that are expected to return  
1164 pointers to storage allocated by a function must always be set by the function, even in the event  
1165 of an error, although in such cases the XDAS-API function may elect to set the returned  
1166 argument value to NULL to indicate that no storage was actually allocated. Any length field  
1167 associated with such pointers (as in a **xdas\_buffer\_desc** structure) should also be set to zero in  
1168 such cases. The XDAS status code [XDAS\_S\_FAILURE] is used to indicate that the underlying  
1169 mechanism detected an error for which no specific XDAS status code is defined. The minor  
1170 status code provides more details about the error.

**1171 6.7 Optional Arguments**

1172 Various arguments are described as optional. This means that they follow a convention whereby  
1173 a default value may be requested. The following conventions are used for omitted arguments.  
1174 These conventions apply only to those arguments that are explicitly documented as optional.

**1175 6.7.1 `xdas_buffer_t` Types (*Input or Input, Output*)**

1176 Specify `XDAS_C_NO_BUFFER` as a value. For an input argument this signifies that default  
1177 behaviour is requested, while for an input,output argument it indicates that the information that  
1178 would be returned by the argument is not required by the application

**1179 6.7.2 Integer Types**

1180 Individual argument documentation lists values to be used to indicate default actions. These are  
1181 passed by value.

**1182 6.7.3 Pointer Types**

1183 Specify `NULL` as the value.

1184 **6.8 Constants**

1185 The tables below set out the constants defined by the specification, and the value to which they  
 1186 are set.  
 1187

1188

Name	Value	Meaning
[XDAS_C_EMPTY_BUFFER]	NULL	Empty buffer
[XDAS_C_NO_BUFFER]	NULL	No buffer is supplied or returned.

1191 **Table 6-3** Optional Parameter Constants

1192 **Table 6-4** XDAS Event Field Separators

1193

Separator	Purpose
HDR	Start of header data
ORG	Start of originator data
INR	Start of initiator data
TGT	Start of target data
SRC	Start of pointer to source record
EVT	Start of event specific data
END	End of record

1194  
1195  
1196  
1197  
1198  
1199  
1200  
1201

1202 **6.9 Event Numbers**

1203 The following table defines the initial set of XDAS events numbers. These numbers will be  
 1204 converted into OpenGroup assigned numbers by addition to a root number once that number  
 1205 has been assigned.

1206 **Table 6-5 XDAS Event Numbers**

Event Description	Event Number
Create account	1
Delete account	2
Disable account	3
Enable account	4
Query account attributes	5
Modify account attributes	6
Create a user session	7
Terminate a user session	8
Query a user session attributes	9
Modify user session attributes	10
Create data item	11
Delete data item	12
Query data item attributes	13
Modify data item attributes	14
Install service or application	15
Remove service or application	16
Query configuration of service or application	17
Modify configuration of service or application	18
Disable service or application	19
Enable service or application	20
Invoke service or application	21
Terminate service or application	22
Query processing context	23
Modify processing context	24
Create an association with a peer	25
Terminate an association with a peer	26
Query an association context	27
Modify an association context	28
Receive data via an association	29
Send data via an association	30
Create association with data item	31
Terminate association with data item	32
Query context of association with data item	33
Modify context of association with data item	34
Query data item contents	35
Modify data item contents	36
Start system	37
Shutdown system	38
Resource exhaustion	39

1248  
1249  
1250  
1251  
1252  
1253  
1254  
1255

<b>Event Description</b>	<b>Event Number</b>
Resource corruption	40
Backup datastore	41
Recover datastore	42
Configure audit service	43
Audit datastore full	44
Audit datastore corrupted	45

1256 **6.10 XDAS Event Classes**

1257 The default set of event classes are:

	<b>Event Class Description</b>	<b>Event Class Code</b>
1260	Account management events	1
1261	User session events	2
1262	Data item and resource element management events	3
1263	Service or application management events	4
1264	Service and application utilisation events	5
1265	Peer association management events	6
1266	Data item or resource element content access events	7
1267	Exceptional events	8
1268	Audit service management events	9

1269 **Table 6-6** XDAS Default Event Class Codes



1270 **6.11 XDAS Event Outcome Codes**

1271 The XDAS outcome codes are:

1272	Name	Value	Meaning
1273	[XDAS_OUT_SUCCESS]	"0x00000000"	Successful Event
1274	[XDAS_OUT_PRIV_USED]	"0x00000100"	Privilege used
1275	[XDAS_OUT_PRIV_GRANTED]	"0x00000200"	Privilege granted
1276	[XDAS_OUT_PRIV_REVOKED]	"0x00000400"	Privilege revoked
1277	[XDAS_OUT_PRE_SELECT_CRITERIA_SET]	"0x00000800"	Pre-selection criteria set or modified
1278	[XDAS_OUT_THRESHOLDS_SET]	"0x00000800"	Thresholds set
1279	[XDAS_OUT_ACTIONS_SET]	"0x00001000"	Actions set for alarms
1280	[XDAS_OUT_THRESHOLD_EXCEEDED]	"0x00002000"	Pre-set thresholds exceeded
1281	[XDAS_OUT_FAILURE]	"0x00000001"	Non security relevant failure
1282	[XDAS_OUT_SERVICE_UNAVAILABLE]	"0x00000101"	Service not available
1283	[XDAS_OUT_SERVICE_FAILURE]	"0x00000201"	Service failure
1284	[XDAS_OUT_HARDWARE_FAILURE]	"0x00000401"	Hardware failure or exception condition
1285	[XDAS_OUT_LOST_ASSOCIATION]	"0x00000801"	Association lost
1286	[XDAS_OUT_ALREADY_ENABLED]	"0x00001001"	Service, user or device already enabled
1287	[XDAS_OUT_ALREADY_DISABLED]	"0x00002001"	Service, user or device already disabled
1288	[XDAS_OUT_SERVICE_ERROR]	"0x00004001"	Service returns an error
1289	[XDAS_OUT_BUSY]	"0x00008001"	Service or device busy
1290	[XDAS_OUT_DISABLED]	"0x00010001"	Service or device disabled
1291	[XDAS_OUT_INVALID_INPUT]	"0x00020001"	Input supplied invalid
1292	[XDAS_OUT_ENTITY_EXISTS]	"0x00040001"	Attempt to create an entity which already exists
1293	[XDAS_OUT_ENTITY_NON-EXISTENT]	"0x00080001"	Attempt to access a non-existent entity
1294	[XDAS_OUT_DENIAL]	"0x00000002"	Security relevant failure
1295	[XDAS_OUT_INSUFFICIENT_PRIVILEGE]	"0x00000102"	Not sufficient privilege
1296	[XDAS_OUT_INVALID_IDENTITY]	"0x00000202"	Identity supplied not valid
1297	[XDAS_OUT_INVALID_USER_CREDENTIALS]	"0x00000402"	User credentials supplied are not valid

1300 **Table 6-7 XDAS Event Outcome Codes**

1303

1304 **6.12 XDAS Action Codes**

1305 The XDAS action codes are:

1306

1307

1308

1309

1310

1311

Name	Value	Meaning
[XDAS_ACT_LOG]	1	Record in Audit Stream
[XDAS_ACT_ALARM]	2	Submit event to Event Management System
[XDAS_ACT_ACTION]	3	Take specified action

**Table 6-8** XDAS Action Codes

1312 **6.13 XDAS Filter Types**

1313 The XDAS filter types are:

1314

1315

1316

1317

1318

1319

Name	Value	Meaning
XDAS_C_SUBMIT	1	Filters for event submission interface
XDAS_C_IMPORT	2	Filters for event import interface
XDAS_C_ALL	3	All filters

**Table 6-9** XDAS Filter Types

1320 **6.14 XDAS Filter Flags**

1321 The XDAS filter flags are:  
1322

Name	Value	Meaning
XDAS_C_INCLUDE	1	include events matching the following rule
XDAS_C_EXCLUDE	2	exclude events matching the following rule

1323  
1324  
1325  
1326 **Table 6-10** XDAS Filter Flags

1327 **6.15 XDAS Filter Attributes**

1328 The XDAS filter attributes are:

1329  
1330

1331

1332

1333

1334

1335

1336

1337

1338

1339

1340

1341

1342

1343

1344

1345

1346

1347

1348

1349  
1350

Name	Value
XDAS_VERSION	1
XDAS_DATE_TIME	2
XDAS_EVENT_NUMBER	3
XDAS_OUTCOME	4
XDAS_ORG_LOC_NAME	5
XDAS_ORG_LOC_ADD	6
XDAS_ORG_SERV_TYPE	7
XDAS_ORG_AUTH_AUTH	8
XDAS_ORG_NAME	9
XDAS_ORG_IDENTITY	10
XDAS_INR_AUTH_AUTH	11
XDAS_INR_NAME	12
XDAS_INR_IDENTITY	13
XDAS_TRT_LOC_NAME	14
XDAS_TRT_LOC_ADD	15
XDAS_TRT_SERV_TYPE	16
XDAS_TRT_AUTH_AUTH	17
XDAS_TRT_NAME	18
XDAS_TRT_IDENTITY	19

**Table 6-11** XDAS Filter Attributes

1351 **6.16 XDAS Filter Operators**

1352 The XDAS filter operators are:

Operator	Value	Meaning
XDAS_O_EQ	1	Equal
XDAS_O_NE	2	Not equal
XDAS_O_GT	3	Greater than
XDAS_O_LT	4	Less than
XDAS_O_GE	5	Greater than or equal
XDAS_O_LE	6	Less than or equal
XDAS_O_BA	7	Bitwise AND
XDAS_O_SS	8	Substring

1363 **Table 6-12** XDAS Filter Operators

1364

## *XDAS Application Program Interface (API)*

1365

1366

1367

This chapter presents the functions to be used by callers of the XDAS application programming interfaces

1368 **NAM**

1369 `xdas_close_audit_stream` — close the specified `audit_stream`

1370 **SYNOPSIS**

```
1371     OM_uint32 xdas_close_audit_stream (
1372         OM_uint32          *minor_status
1373         xdas_audit_ref_t   *das_ref,
1374         xdas_audit_stream_t *audit_stream_ref
1375     );
```

1376 **DESCRIPTION**

1377 The `xdas_close_audit_stream` function closes the audit stream, previously opened for reading,  
1378 specified by the `audit_stream_ref` handle. The caller must possess the `XDAS_AUDIT_READ`  
1379 authority.

1380 If successful, the function returns `[XDAS_S_COMPLETE]`

1381 The arguments for `xdas_close_audit_stream()` are:

1382 *minor\_status* (out)

1383 An implementation specific return status that provides additional information when  
1384 `[XDAS_S_FAILURE]` is returned by the function.

1385 *das\_ref* (in)

1386 Handle to the audit service obtained from a previous call to `xdas_initialise_session()`.

1387 *audit\_stream\_reference* (in)

1388 Handle to the audit stream which is to be closed.

1389 **RETURN VALUE**

1390 The following XDAS status codes shall be returned:

1391 `[XDAS_S_COMPLETE]`

1392 Successful completion.

1393 `[XDAS_S_INVALID_AUDIT_STREAM]`

1394 The specified audit stream is not valid.

1395 `[XDAS_S_FAILURE]`

1396 An implementation specific error or failure has occurred.

1397 `[XDAS_S_INVALID_DAS_REF]`

1398 The handle to the audit service is not valid.

1399 `[XDAS_S_AUTHORISATION_FAILURE]`

1400 The caller does not possess the required authority.

1401 **ERROR**

1402 No other errors are defined.



1403 **NAME**

1404       xdas\_commit\_record — write a completed audit record to the audit stream

1405 **SYNOPSIS**

```
1406     OM_uint32 xdas_commit_record (
1407         OM_uint32          *minor_status
1408         xdas_audit_ref_t   *das_ref,
1409         xdas_audit_rec_desc_t   *audit_record_descriptor
1410     );
```

1411 **DESCRIPTION**

1412       The XDAS implementation writes the audit record identified by *audit\_record\_descriptor* to the  
1413       current audit stream controlled by the audit service and accessed by *das\_ref*. The caller must  
1414       have the XDAS\_AUDIT\_SUBMIT authority.

1415       If successful, the function returns [XDAS\_S\_COMPLETE]. The arguments for  
1416       *xdas\_commit\_record()* are:

1417       *minor\_status* (out)

1418       An implementation specific return status that provides additional information when  
1419       [XDAS\_S\_FAILURE] is returned by the function.

1420       *das\_ref* (in)

1421       Handle to the XDAS service daemon, and the means by which the caller accesses the audit  
1422       stream.

1423       *audit\_record\_descriptor* (in)

1424       A descriptor referencing a completed audit record to be written to the audit stream. On  
1425       successful completion the *audit\_record\_descriptor* is no longer a valid reference to an audit  
1426       record.

1427 **RETURN VALUE**

1428       The following XDAS status codes shall be returned:

1429       [XDAS\_S\_COMPLETE]

1430       Successful completion.

1431       [XDAS\_S\_INVALID\_RECORD\_DESCRIPTOR]

1432       The specified audit record descriptor is not valid.

1433       [XDAS\_S\_INVALID\_DAS\_REF]

1434       The handle to the audit service is not valid.

1435       [DAS\_S\_STORAGE\_FAILURE]

1436       The audit record cannot be written to stable storage.

1437       [XDAS\_S\_SERVICE\_FAILURE]

1438       There has been an audit service failure.

1439       [XDAS\_S\_FAILURE]

1440       An implementation specific error or failure has occurred.

1441       [XDAS\_S\_AUTHORISATION\_FAILURE]

1442       The caller does not possess the required authority

1443 **ERRORS**

1444       No other errors are defined.

1445 **NAME**

1446 `xdas_create_filter` — create the specified audit filter

1447 **SYNOPSIS**

```

1448     OM_uint32 xdas_create_filter (
1449         OM_uint32          *minor_status,
1450         xdas_audit_ref_t   *das_ref,
1451         xdas_buffer_t      *name,
1452         OM_uint32          *filter_type,
1453         xdas_buffer_t      *filter_exp,
1454         xdas_buffer_t      *filter_action_list,
1455     );
    
```

1456 **DESCRIPTION**

1457 The `xdas_create_filter` function creates a filter for the `filter_name` specified. If a filter with the  
 1458 specified name already exists the call fails. On creation the filter is in a disabled state.

1459 The caller must possess the XDAS\_AUDIT\_CONTROL authority.

1460 If successful, the function returns [XDAS\_S\_COMPLETE].

1461 The arguments for `xdas_create_filter()` are:

1462 `minor_status` (out)

1463 An implementation specific return status that provides additional information when  
 1464 [XDAS\_S\_FAILURE] is returned by the function.

1465 `das_ref` (in)

1466 The handle to the XDAS server, obtained from a previous call to `xdas_initialise_session`.

1467 `filter_type` (optional,in)

1468 The type of filter. This may be either XDAS\_C\_SUBMIT or XDAS\_C\_IMPORT or  
 1469 XDAS\_C\_ALL.

1470 `name` (optional,in)

1471 The name of the filter.

1472 `filter_exp` (optional,in)

1473 The expression list which defines the criteria for detection of the event.

1474 `filter_action_list` (optional,in) The list defining the actions to be taken on detecting the event.

1475 **RETURN VALUE**

1476 The following XDAS status codes shall be returned:

1477 [XDAS\_S\_COMPLETE]

1478 Successful completion.

1479 [XDAS\_S\_INVALID\_DAS\_REF]

1480 The audit daemon handle supplied does not point to the audit daemon.

1481 [XDAS\_S\_INVALID\_FILTER]

1482 The filter name supplied already exists.

1483 [XDAS\_S\_INVALID\_FILTER\_TYPE]

1484 The filter type supplied is not recognized.

1485 [XDAS\_S\_INVALID\_FILTER\_EXP]

1486 The filter expression supplied is not valid.

- 1487 [XDAS\_S\_INVALID\_ACTION\_LIST]
- 1488 The filter type supplied is not recognized.
- 1489 [XDAS\_S\_FAILURE]
- 1490 An implementation specific error or failure has occurred.
- 1491 [XDAS\_S\_AUTHORISATION\_FAILURE]
- 1492 The caller does not possess the required authority.
- 1493 **ERRORS**
- 1494 No other errors are defined.

1495 **NAME**

1496 `xdas_delete_filter` — delete the specified audit filter

1497 **SYNOPSIS**

```
1498     OM_uint32 xdas_delete_filter (
1499         OM_uint32          *minor_status
1500         xdas_audit_ref_t   *das_ref,
1501         xdas_buffer_t      *name,
1502     );
```

1503 **DESCRIPTION**

1504 The `xdas_delete_filter` function deletes the filter defined by *name* from the XDAS system. This  
 1505 may involve deleting copies of the filter from all agents managed via a particular instance of the  
 1506 XDAS interface. The function does not wait upon the successful deletion of all instances of the  
 1507 filter maintained by XDAS agents. The caller must possess the XDAS\_AUDIT\_CONTROL  
 1508 authority.

1509 If successful, the function returns [XDAS\_S\_COMPLETE].

1510 The arguments for `xdas_delete_filter()` are:

1511 *minor\_status* (out)

1512 An implementation specific return status that provides additional information when  
 1513 [XDAS\_S\_FAILURE] is returned by the function.

1514 *das\_ref* (in)

1515 The handle to the XDAS server, obtained from a previous call to `xdas_initialise_session`.

1516 *name* (in)

1517 The name of the filter.

1518 **RETURN VALUE**

1519 The following XDAS status codes shall be returned:

1520 [XDAS\_S\_COMPLETE]

1521 Successful completion.

1522 [XDAS\_S\_INVALID\_DAS\_REF]

1523 The audit daemon handle supplied does not point to the audit daemon.

1524 [XDAS\_S\_INVALID\_FILTER\_TYPE]

1525 The filter type supplied is not valid.

1526 [XDAS\_S\_INVALID\_FILTER]

1527 The filter name supplied is not valid.

1528 [XDAS\_S\_FAILURE]

1529 An implementation specific error or failure has occurred.

1530 [XDAS\_S\_AUTHORISATION\_FAILURE]

1531 The caller does not possess the required authority

1532 **ERRORS**

1533 No other errors are defined.

1534 **NAME**

1535        xdas\_disable\_filter — disable the specified filter

1536 **SYNOPSIS**

```

1537     OM_uint32 xdas_disable_filter (
1538         OM_uint32          *minor_status
1539         xdas_audit_ref_t   *das_ref,
1540         xdas_buffer_t      *name,
1541     );

```

1542        The *xdas\_disable\_filter* function disables the filter specified by *name*. It sets the state of the filter to disabled. If necessary the disabled state of the filter may require propagation to all XDAS agents managed by a particular instance of the XDAS Interface. The function does not wait upon the successful disabling of all instances of the filter maintained by XDAS agents. The caller must possess the XDAS\_AUDIT\_CONTROL authority.

1547        If successful, the function returns [XDAS\_S\_COMPLETE].

1548        The arguments for *xdas\_disable\_filter( )* are:

1549        *das\_ref* (in)

1550            The handle to the XDAS server, obtained from a previous call to *xdas\_initialise\_session*.

1551        *filter\_type* (in)

1552            The type of filter. This may be either XDAS\_C\_SUBMIT or XDAS\_C\_IMPORT.

1553        *name* (in)

1554            The name of the filter to be disabled.

1555        *minor\_status* (out)

1556            An implementation specific return status that provides additional information when [XDAS\_S\_FAILURE] is returned by the function.

1558 **RETURN VALUE**

1559        The following XDAS status codes shall be returned:

1560        [XDAS\_S\_COMPLETE]

1561            Successful completion.

1562        [XDAS\_S\_INVALID\_DAS\_REF]

1563            The audit daemon handle supplied does not point to the audit daemon.

1564        [XDAS\_S\_INVALID\_FILTER]

1565            The filter name supplied is not known.

1566        [XDAS\_S\_FAILURE]

1567            An implementation specific error or failure has occurred.

1568        [XDAS\_S\_AUTHORISATION\_FAILURE]

1569            The caller does not possess the required authority.

1570 **ERRORS**

1571        No other errors are defined.

1572 **NAME**1573 `xdas_discard_record` — discard a previously created audit record1574 **SYNOPSIS**

```
1575     OM_uint32 xdas_discard_record (
1576         OM_uint32          *minor_status
1577         xdas_audit_ref_t   *das_ref,
1578         xdas_audit_desc_t  *audit_record_descriptor,
1579     );
```

1580 **DESCRIPTION**

1581 The *xdas\_discard\_record* function clears the buffer specified by *audit\_record\_descriptor* and releases  
1582 the memory used by it. The caller must have the XDAS\_AUDIT\_SUBMIT authority.

1583 If successful, the function returns [XDAS\_S\_COMPLETE]. The arguments for  
1584 *xdas\_discard\_record*( ) are:

1585 *minor\_status* (out)

1586 An implementation specific return status that provides additional information when  
1587 [XDAS\_S\_FAILURE] is returned by the function.

1588 *das\_ref* (in)

1589 Handle to the XDAS service, obtained from a previous call to *xdas\_initialise\_session*.

1590 *audit\_record\_descriptor* (in)

1591 The audit record descriptor returned from a previous call to *xdas\_start\_record*.

1592 **RETURN VALUE**

1593 The following XDAS status codes shall be returned:

1594 [XDAS\_S\_COMPLETE]

1595 Successful completion.

1596 [XDAS\_S\_INVALID\_RECORD\_DESCRIPTOR]

1597 The specified audit record descriptor is not valid.

1598 [XDAS\_S\_INVALID\_DAS\_REF]

1599 The audit daemon handle supplied does not point to the audit daemon.

1600 [XDAS\_S\_FAILURE]

1601 An implementation specific error or failure has occurred.

1602 [XDAS\_S\_AUTHORISATION\_FAILURE]

1603 The caller does not possess the required authority

1604 **ERRORS**

1605 No other errors are defined.

1606 **NAME**

1607       xdas\_enable\_filter — enable the specified audit filter

1608 **SYNOPSIS**

```

1609     OM_uint32 das_get_filter (
1610         OM_uint32          *minor_status
1611         xdas_audit_ref_t   *das_ref,
1612         xdas_buffer_t      *name,
1613     );

```

1614 **DESCRIPTION**

1615       The *xdas\_enable\_filter* function enables the filter corresponding to the *name* specified. If  
 1616       necessary the enabled state of the filter may require propagation to all XDAS agents managed by  
 1617       a particular instance of the XDAS Interface. The function does not wait upon the successful  
 1618       enabling of all instances of the filter maintained by XDAS agents. The caller must possess the  
 1619       XDAS\_AUDIT\_CONTROL authority.

1620       If successful, the function returns [XDAS\_S\_COMPLETE].

1621       The arguments for *xdas\_enable\_filter()* are:

1622       *minor\_status* (out)

1623       An implementation specific return status that provides additional information when  
 1624       [XDAS\_S\_FAILURE] is returned by the function.

1625       *das\_ref* (in)

1626       The handle to the XDAS server, obtained from a previous call to *xdas\_initialise\_session*.

1627       *name* (in)

1628       The name of the filter to be enabled.

1629 **RETURN VALUE**

1630       The following XDAS status codes shall be returned:

1631       [XDAS\_S\_COMPLETE]

1632       Successful completion.

1633       [XDAS\_S\_INVALID\_DAS\_REF]

1634       The audit daemon handle supplied does not point to the audit daemon.

1635       [XDAS\_S\_INVALID\_FILTER]

1636       The filter name supplied is not known.

1637       [XDAS\_S\_FAILURE]

1638       An implementation specific error or failure has occurred.

1639       [XDAS\_S\_AUTHORISATION\_FAILURE]

1640       The caller does not possess the required authority.

1641 **ERRORS**

1642       No other errors are defined.

1643 **NAME**1644 `xdas_get_filter` — get audit filters for a specified name1645 **SYNOPSIS**

```
1646     OM_uint32 xdas_get_filter (
1647         OM_uint32          *minor_status,
1648         xdas_audit_ref_t   *das_ref,
1649         xdas_buffer_t      *name,
1650         OM_uint32          *filter_type,
1651         xdas_buffer_t      *filter_exp,
1652         xdas_buffer_t      *filter_action_list,
1653         OM_uint32          *filter_status
1654     );
```

1655 **DESCRIPTION**

1656 The `xdas_get_filter` function returns the components of the filter referenced by *name*. The caller  
1657 must possess the XDAS\_AUDIT\_CONTROL authority.

1658 If successful, the function returns [XDAS\_S\_COMPLETE].

1659 The arguments for `xdas_get_filter()` are:

1660 *minor\_status* (out)

1661 An implementation specific return status that provides additional information when  
1662 [XDAS\_S\_FAILURE] is returned by the function.

1663 *das\_ref* (in)

1664 The handle to the XDAS server, obtained from a previous call to `xdas_initialise_session`.

1665 *name* (in)

1666 The name of the filter to be returned.

1667 *filter\_type* (in)

1668 The type of filter. This may be either XDAS\_C\_SUBMIT or XDAS\_C\_IMPORT.

1669 *filter\_exp* (out)

1670 The contents of the expression list that determines the events to be selected by this filter.

1671 *filter\_exp* (out)

1672 The contents of the expression list that determines the events to be selected by this filter.

1673 *filter\_status* (out)

1674 The enabled or disabled state of the filter. If the filter is enabled then a value of 0 is returned  
1675 in this parameter, otherwise a value of 1 is returned.

1676 **RETURN VALUE**

1677 The following XDAS status codes shall be returned:

1678 [XDAS\_S\_COMPLETE]

1679 Successful completion.

1680 [XDAS\_S\_INVALID\_DAS\_REF]

1681 The audit daemon handle supplied does not point to the audit daemon.

1682 [XDAS\_S\_INVALID\_FILTER]

1683 The filter name supplied is not known.

1684 [XDAS\_S\_FAILURE]

1685 An implementation specific error or failure has occurred.



1686 [XDAS\_S\_AUTHORISATION\_FAILURE]  
1687 The caller does not possess the required authority.

1688 **ERRORS**

1689 No other errors are defined.

1690 **NAME**

1691 `xdas_get_next` — read next set of records from a previously opened audit stream

1692 **SYNOPSIS**

```

1693     OM_uint32 xdas_get_next (
1694         OM_uint32             *minor_status
1695         xdas_audit_ref_t      *das_ref,
1696         xdas_audit_stream_t    *audit_stream_ref,
1697         OM_unit32             max_records,
1698         xdas_buffer_t          *audit_record_buffer,
1699         OM_unit32             *no_of_records,
1700     );

```

1701 **DESCRIPTION**

1702 The `xdas_get_next()` function copies up to *max-records* complete records from the audit stream  
 1703 accessed by *das\_ref* into the buffer *audit\_record\_buffer*. The actual number of records copied is  
 1704 returned in *no\_of\_records*.

1705 If the function successfully reads a record or records from the audit stream, the cursor associated  
 1706 with the audit stream referenced by *das\_ref* will be advanced to the next record in the audit  
 1707 stream.

1708 If the call is unsuccessful, the position of the cursor is not changed. The caller must have the  
 1709 XDAS\_AUDIT\_READ authority

1710 If successful, the function returns [XDAS\_S\_COMPLETE].

1711 The arguments for `xdas_get_next()` are:

1712 *minor\_status* (out)

1713 An implementation specific return status that provides additional information when  
 1714 [XDAS\_S\_FAILURE] is returned by the function.

1715 *das\_ref* (in)

1716 The handle to the XDAS server, obtained from a previous call to `xdas_initialise_session()`.

1717 *audit\_stream\_ref* (in)

1718 The handle to the XDAS audit stream, obtained from a previous call to  
 1719 `xdas_open_audit_stream()`.

1720 *max\_records* (in)

1721 The maximum number of records to be returned by the function in any one call.

1722 *audit\_record\_buffer* (in)

1723 Pointer to the buffer to which the audit records are to be copied.

1724 *no\_of\_records* (out)

1725 the number of records actually copied into *audit\_record\_buffer*.

1726 **RETURN VALUE**

1727 The following XDAS status codes shall be returned:

1728 [XDAS\_S\_COMPLETE]

1729 Successful completion.

1730 [XDAS\_S\_INVALID\_DAS\_REF]

1731 The audit daemon handle supplied does not point to the audit daemon.

1732 [XDAS\_S\_INVALID\_STREAM\_REF]

1733 The audit stream handle supplied is invalid.

- 1734 [XDAS\_S\_END]
- 1735 The end of the audit stream has been reached.
- 1736 [XDAS\_S\_FAILURE]
- 1737 An implementation specific error or failure has occurred
- 1738 [XDAS\_S\_AUTHORISATION\_FAILURE]
- 1739 The caller does not possess the required authority.
- 1740 **ERRORS**
- 1741 No other errors are defined.

1742 **NAME**

1743 xdas\_import\_event\_records — imports records from an external audit service into XDAS in  
1744 XDAS common format

1745 **SYNOPSIS**

```
1746 OM_uint32 xdas_import_event_records (
1747     OM_uint32          *minor_status
1748     xdas_audit_ref_t   das_ref,
1749     xdas_buffer_t      *audit_record_buffer,
1750     OM_uint32          position_in_buffer,
1751 );
```

1752 **DESCRIPTION**

1753 The *xdas\_import\_event\_records* function allows a caller to submit audit event records in the XDAS  
1754 format directly to the XDAS service. The caller places one or more complete audit event records  
1755 into the buffer referenced by *audit\_record\_buffer* from which they are copied by XDAS and  
1756 integrated into the XDAS audit stream. The implementation may select the records that are  
1757 actually imported based upon some selection criteria. The caller is not advised of the disposition  
1758 of the audit records it submits.

1759 The caller must possess the XDAS\_AUDIT\_IMPORT authority.

1760 If successful, the function returns [XDAS\_S\_COMPLETE].

1761 The arguments for *xdas\_import\_event\_records()* are:

1762 *minor\_status* (out)

1763 An implementation specific return status that provides additional information when  
1764 [XDAS\_S\_FAILURE] is returned by the function.

1765 *das\_ref* (in)

1766 Handle to the XDAS service obtained by a previous call to *xdas\_initialise\_session()*.

1767 *audit\_record\_buffer* (in)

1768 Buffer into which the caller places the audit records to be imported into the XDAS audit  
1769 stream.

1770 *position\_in\_buffer* (out)

1771 If a record syntax error is detected this parameter contains the position in the buffer at  
1772 which the syntax error was detected.

1773 **RETURN VALUE**

1774 The following XDAS status codes shall be returned:

1775 [XDAS\_S\_COMPLETE]

1776 Successful completion.

1777 [XDAS\_S\_INVALID\_DAS\_REF]

1778 The audit daemon handle supplied does not point to the audit daemon.

1779 [XDAS\_S\_FAILURE]

1780 An implementation specific error or failure has occurred

1781 [XDAS\_S\_RECORD\_SYNTAX\_ERROR]

1782 A syntax error has been detected in an input record.

1783 [XDAS\_S\_AUTHORISATION\_FAILURE]

1784 The caller does not possess the required authority.

1785 **ERRORS**

1786       No other errors are defined.

1787 **NAME**1788 `xdas_initialise_session` — initialise a session with the distributed audit service1789 **SYNOPSIS**

```
1790     OM_uint32 xdas_initialise_session (
1791         OM_uint32             *minor_status
1792         xdas_buffer_t         *security_context,
1793         xdas_buffer_t         *org_info,
1794         xdas_audit_ref_t      *das_ref,
1795     );
```

1796 **DESCRIPTION**

1797 The `xdas_initialise_session` function initiates a session between the `server_identity` and the  
1798 distributed audit service. It validates the `security_context` provided to ensure that caller has been  
1799 authenticated and is authorised to use the XDAS.

1800 If successful, the function returns `das_ref`, a handle to the XDAS server. The caller must have the  
1801 XDAS\_AUDIT\_SERVICE authority.

1802 If successful, the function returns [XDAS\_S\_COMPLETE].

1803 The use of this function must itself be audited by the XDAS service.

1804 The arguments for `xdas_initialise_session( )` are:

1805 `minor_status` (out)

1806 An implementation specific return status that provides additional information when  
1807 [XDAS\_S\_FAILURE] is returned by the function.

1808 `security_context` (in)

1809 An opaque structure containing defining the security context of the caller requesting use of  
1810 the audit service. This is used to authenticate the caller to the XDAS and establish the  
1811 callers XDAS authorisations.

1812 `org_info` (in)

1813 This buffer includes the originator information that is to be included with each audit event  
1814 subsequently submitted by this caller. The XDAS service uses this information to populate  
1815 the originator information of an audit record when `xdas_start_record( )` is invoked.

1816 `das_ref` (out)

1817 The handle to the XDAS server is returned in `das_ref`.

1818 **RETURN VALUE**

1819 The following XDAS status codes shall be returned:

1820 [XDAS\_S\_COMPLETE]

1821 Successful completion.

1822 [XDAS\_S\_INVALID\_SECURITY\_CONTEXT]

1823 The security context supplied is not valid.

1824 [XDAS\_S\_INVALID\_ORIG\_INFO]

1825 The originator information supplied has a syntax error.

1826 [XDAS\_S\_FAILURE]

1827 An implementation specific error or failure has occurred.

1828 [XDAS\_S\_AUTHORISATION\_FAILURE]

1829 The caller does not possess the required authority.

1830 **ERRORS**

1831       No other errors are defined.

1832 **NAM**

1833 `xdas_list_filters` — list the audit filters that have been defined

1834 **SYNOPSIS**

```
1835     OM_uint32 xdas_list_filters (
1836         OM_uint32          *minor_status
1837         xdas_audit_ref_t   *das_ref,
1838         xdas_buffer_t      **filter_list,
1839     );
```

1840 **DESCRIPTION**

1841 The *xdas\_list\_filters* function returns a pointer to a NULL terminated list of the names of the  
 1842 filters that exist within the XDAS service. The caller must possess the  
 1843 XDAS\_AUDIT\_CONTROL authority.

1844 If successful, the function returns [XDAS\_S\_COMPLETE].

1845 The arguments for *xdas\_list\_filters()* are:

1846 *minor\_status* (out)

1847 An implementation specific return status that provides additional information when  
 1848 [XDAS\_S\_FAILURE] is returned by the function.

1849 *das\_ref* (in)

1850 The handle to the XDAS server, obtained from a previous call to *xdas\_initialise\_session*.

1851 *filter\_name\_list* (out)

1852 A pointer to the list of the names of the filters that exist within the XDAS service.

1853 **RETURN VALUE**

1854 The following XDAS status codes shall be returned:

1855 [XDAS\_S\_COMPLETE]

1856 Successful completion.

1857 [XDAS\_S\_INVALID\_DAS\_REF]

1858 The handle to the XDAS server supplied does not point to the audit daemon.

1859 [XDAS\_S\_FAILURE]

1860 An implementation specific error or failure has occurred.

1861 [XDAS\_S\_AUTHORITY\_FAILURE]

1862 The caller does not possess the required authority.

1863 **ERRORS**

1864 No other errors are defined



1865 **NAM**1866        *xdas\_open\_audit\_stream* — open the *audit\_stream*1867 **SYNOPSIS**

```

1868     OM_uint32 xdas_close_audit_stream (
1869         OM_uint32          *minor_status
1870         xdas_audit_ref_t   *das_ref,
1871         xdas_audit_stream_t *audit_stream_ref
1872     );

```

1873 **DESCRIPTION**

1874        The *xdas\_open\_audit\_stream* function opens the audit stream for reading and returns a handle to  
 1875        the audit stream in *audit\_stream\_ref* handle. A caller may obtain more than one handle to the  
 1876        audit stream, each of which is independent of any other handles. The caller must possess the  
 1877        XDAS\_AUDIT\_READ authority.

1878        If successful, the function returns [XDAS\_S\_COMPLETE]

1879        The arguments for *xdas\_open\_audit\_stream( )* are:

1880        *minor\_status* (out)

1881            An implementation specific return status that provides additional information when  
 1882            [XDAS\_S\_FAILURE] is returned by the function.

1883        *das\_ref* (in)

1884            Handle to the audit service obtained from a previous call to *xdas\_initialise\_session*.

1885        *audit\_stream\_reference* (out)

1886            Handle to the audit stream returned by the function.

1887 **RETURN VALUE**

1888        The following XDAS status codes shall be returned:

1889        [XDAS\_S\_COMPLETE]

1890            Successful completion.

1891        [XDAS\_S\_FAILURE]

1892            An implementation specific error or failure has occurred.

1893        [XDAS\_S\_INVALID\_DAS\_REF]

1894            The handle to the audit service is not valid.

1895        [XDAS\_S\_AUTHORISATION\_FAILURE]

1896            The caller does not possess the required authority.

1897 **ERROR**

1898        No other errors are defined.

1899 **NAME**

1900 `xdas_put_event_info` — add specific event information to an audit record buffer

1901 **SYNOPSIS**

```

1902     OM_uint32 xdas_put_event_info (
1903         OM_uint32          *minor_status,
1904         xdas_audit_ref_t   *das_ref,
1905         xdas_audit_desc_t  *audit_record_descriptor,
1906         OM_uint32          *event_number,
1907         OM_uint32          *outcome,
1908         xdas_buffer_t      *initiator_information,
1909         xdas_buffer_t      *target_information,
1910         xdas_buffer_t      *event_info
1911     );

```

1912 **DESCRIPTION**

1913 The *xdas\_put\_event\_info* function adds event specific information to an audit record. If the  
 1914 optional parameters are supplied, it also checks whether the specified event should be audited  
 1915 and returns an XDAS\_AUDIT\_UNCERTAIN or XDAS\_NO\_AUDIT code to the caller. Multiple  
 1916 calls to *xdas\_put\_event\_info* may be made. For any individual parameter, information supplied in  
 1917 a call will overwrite any previous information supplied. The order of the event information is  
 1918 preserved. The caller must have the XDAS\_AUDIT\_SUBMIT authority.

1919 If successful, the function returns [XDAS\_S\_COMPLETE].

1920 The arguments for *xdas\_put\_event\_info()* are:

1921 *minor\_status* (out)

1922 An implementation specific return status that provides additional information when  
 1923 [XDAS\_S\_FAILURE] is returned by the function.

1924 *das\_ref* (in)

1925 The handle to the XDAS server, obtained from a previous call to *xdas\_initialise\_session()*.

1926 *audit\_record\_descriptor* (in)

1927 The handle to the audit record, obtained from a previous call to *xdas\_start\_record()*.

1928 *event\_number* (optional,in)

1929 The event number of the detected event. This is specified only if it has not already been set  
 1930 in the *audit\_record\_descriptor* supplied.

1931 *outcome* (optional,in)

1932 The outcome of the event determined by the caller. This is specified only if it has not  
 1933 already been set in the *audit\_record\_descriptor*  
 1934 supplied.

1935 *initiator\_information* (optional,in)

1936 The information describing the initiator in the format required by the XDAS common audit  
 1937 format. Again, this is optional, and is included only if it has not already been set in the  
 1938 *audit\_record\_descriptor* supplied.

1939 *target\_information* (optional,in)

1940 The information on the target of the event in the format required by the XDAS common  
 1941 audit format. This is specified only if it has not already been set in the *audit\_record\_descriptor*  
 1942 supplied.

1943 *event\_info* (in)

1944 The event specific information that is to be added to the audit record specified by

1945            *audit\_record\_descriptor*.

1946 **RETURN VALUE**

1947            The following XDAS status codes shall be returned:

1948            [XDAS\_S\_COMPLETE]

1949            Successful completion.

1950            [XDAS\_S\_INVALID\_DAS\_REF]

1951            The audit daemon handle supplied does not point to the audit daemon.

1952            [XDAS\_S\_INVALID\_INITIATOR\_INFO]

1953            The initiator information supplied has a syntax error.

1954            [XDAS\_S\_INVALID\_TARGET\_INFO]

1955            The specified target information has a syntax error.

1956            [XDAS\_S\_INVALID\_EVENT\_NO]

1957            The specified event number is not valid.

1958            [XDAS\_S\_INVALID\_OUTCOME]

1959            The specified outcome is not valid.

1960            [XDAS\_S\_INVALID\_RECORD\_DESCRIPTOR]

1961            The specified audit record descriptor is not valid.

1962            [XDAS\_S\_NO\_AUDIT]

1963            The event specified does not need to be audited.

1964            [XDAS\_S\_UNCERTAIN\_AUDIT]

1965            There is uncertainty as to whether the event specified needs to be audited.

1966            [XDAS\_S\_INVALID\_EVENT\_INFO]

1967            The specified audit event information is not valid.

1968            [XDAS\_S\_FAILURE]

1969            An implementation specific error or failure has occurred.

1970            [XDAS\_S\_AUTHORISATION\_FAILURE]

1971            The caller does not possess the required authority.

1972 **ERRORS**

1973            No other errors are defined.

1974 **NAME**1975        *xdas\_release\_buffer* — free storage associated with a buffer1976 **SYNOPSIS**

```
1977        OM_uint32 xdas_release_buffer(  
1978            OM_uint32                    *minor_status  
1979            xdas_audit_ref_t            *das_ref,  
1980            xdas_buffer_t                *buffer,  
1981        );
```

1982 **DESCRIPTION**

1983        This function frees storage associated with a buffer. The storage must have been allocated by a  
1984        XDAS-API function. In addition to freeing the associated storage, the function zeros the length  
1985        field in the *buffer* argument. If successful, the function returns [XDAS\_S\_COMPLETE]. The  
1986        arguments for *xdas\_release\_buffer()* are:

1987        *minor\_status* (out)

1988        An implementation specific return status that provides additional information when  
1989        [XDAS\_S\_FAILURE] is returned by the function.

1990        *das\_ref* (in)

1991        Handle to the XDAS service obtained by a previous call to *xdas\_initialise\_session()*.

1992        *buffer* (in,out)

1993        The storage associated with the *buffer* is deleted. The *xdas\_buffer\_t* object is not freed, but  
1994        its length field is zeroed.

1995 **RETURN VALUE**

1996        The following GCS status codes shall be returned:

1997        [GCS\_S\_COMPLETE]

1998        Successful completion

1999        [XDAS\_S\_INVALID\_DAS\_REF]

2000        The audit daemon handle supplied is invalid.

2001        [GCS\_S\_FAILURE]

2002        An implementation specific error or failure has occurred.

2003 **ERRORS**

2004        No other errors are defined.

2005 **NAME**

2006            xdas\_release\_filter\_list — release the list of filter names

2007 **SYNOPSIS**

```
2008            OM_uint32 dас_release_filter_list (
2009                    OM_uint32                    *minor_status
2010                    xdas_audit_ref_t             *das_ref,
2011                    xdas_buffer_t                **filter_list,
2012            );
```

2013 **DESCRIPTION**

2014            The *xdas\_release\_filter\_list* function releases the list of filter names which were obtained by a  
2015            previous call to *xdas\_list\_filters*. The caller must possess the XDAS\_AUDIT\_CONTROL  
2016            authority.

2017            If successful, the function returns [XDAS\_S\_COMPLETE].

2018            The arguments for *xdas\_release\_filter\_list* () are:

2019            *das\_ref* (in)

2020                The handle to the XDAS server, obtained from a previous call to *xdas\_initialise\_session*.

2021            *filter\_list* (in)

2022                A pointer to the list of filter names obtained from a previous call to *xdas\_list\_filters* ().

2023            *minor\_status* (out)

2024                An implementation specific return status that provides additional information when  
2025                [XDAS\_S\_FAILURE] is returned by the function.

2026 **RETURN VALUE**

2027            The following XDAS status codes shall be returned:

2028            [XDAS\_S\_COMPLETE]

2029                Successful completion.

2030            [XDAS\_S\_INVALID\_DAS\_REF]

2031                The audit daemon handle supplied does not point to the audit daemon.

2032            [XDAS\_S\_INVALID\_FILTER\_LIST]

2033                The list of filter names is not valid.

2034            [XDAS\_S\_FAILURE]

2035                An implementation specific error or failure has occurred.

2036            [XDAS\_S\_AUTHORISATION\_FAILURE]

2037                The caller does not possess the required authority.

2038 **ERRORS**

2039            No other errors are defined.

2040 **NAME**2041 `xdas_rewind_audit_stream` — rewind the audit stream2042 **SYNOPSIS**

```
2043     OM_uint32 xdas_rewind_audit_stream (
2044         OM_uint32          *minor_status
2045         xdas_audit_ref_t   *das_ref,
2046         xdas_audit_stream_t *audit_stream_ref,
2047     );
```

2048 **DESCRIPTION**

2049 The `xdas_rewind_audit_stream` function rewinds the audit stream referenced by `xdas_stream_ref` so  
2050 that the cursor associated with the `xdas_stream_ref` points to the first record in the audit stream.  
2051 The caller must possess the XDAS\_AUDIT\_READ authority.

2052 If successful, the function returns [XDAS\_S\_COMPLETE].

2053 The arguments for `xdas_rewind_audit_stream()` are:

2054 `minor_status` (out)

2055 An implementation specific return status that provides additional information when  
2056 [XDAS\_S\_FAILURE] is returned by the function.

2057 `das_ref` (in)

2058 The handle to the XDAS server, obtained from a previous call to `xdas_initialise_session()`.

2059 `audit_stream_ref` (in/out)

2060 Handle to the audit stream which is to be rewound.

2061 **RETURN VALUE**

2062 The following XDAS status codes shall be returned:

2063 [XDAS\_S\_COMPLETE]

2064 Successful completion.

2065 [XDAS\_S\_INVALID\_DAS\_REF]

2066 The audit daemon handle supplied does not point to the audit daemon.

2067 [XDAS\_S\_INVALID\_AUDIT\_STREAM]

2068 The specified audit stream is not valid.

2069 [XDAS\_S\_FAILURE]

2070 An implementation specific error or failure has occurred.

2071 [XDAS\_S\_AUTHORISATION\_FAILURE]

2072 The caller does not possess the required authority.

2073 **ERRORS**

2074 No other errors are defined.

2075 **NAME**

2076        xdas\_start\_record — initialise an audit record

2077 **SYNOPSIS**

```

2078     OM_uint32 xdas_start_record (
2079         OM_uint32          *minor_status
2080         xdas_audit_ref_t   *das_ref,
2081         xdas_audit_desc_t  *audit_record_descriptor,
2082         xdas_buffer_t      *event_number,
2083         xdas_buffer_t      *outcome,
2084         xdas_buffer_t      *initiator_information,
2085         xdas_buffer_t      *target_information,
2086         xdas_buffer_t      *event_info,
2087     );

```

2088 **DESCRIPTION**

2089        The *xdas\_start\_record* function returns a *audit\_record\_descriptor* handle to the audit record to the  
 2090 caller. If the optional parameters are not specified in the call, then the audit record is initialised  
 2091 but requires fully populating by subsequent calls to *xdas\_put\_event\_info*.

2092        If the optional parameters are specified, *xdas\_start\_record* determines whether a specified event  
 2093 should be audited given the *event\_number*, *outcome* and *initiator\_informations* supplied. If the  
 2094 event should be audited a valid *audit\_record\_descriptor* is returned to the caller. If the audit event  
 2095 does not require auditing then *audit\_record\_descriptor* is set to NULL. The caller must have the  
 2096 XDAS\_AUDIT\_SUBMIT authority.

2097        If successful, the function returns [XDAS\_S\_COMPLETE].

2098        The arguments for *xdas\_start\_record()* are:

2099        *minor\_status* (out)

2100            An implementation specific return status that provides additional information when  
 2101 [XDAS\_S\_FAILURE] is returned by the function.

2102        *das\_ref* (in)

2103            Handle to the XDAS service, obtained from a previous call to *xdas\_initialise\_session()*.

2104        *event\_number* (optional,in)

2105            The *event\_number* of the detected event.

2106        *outcome* (optional,in)

2107            The outcome of the event as determined by the caller.

2108        *initiator\_information* (optional,in)

2109            The available information describing the initiator in the format required by the XDAS  
 2110 common audit format.

2111        *target\_information* (optional,in)

2112            Information on the target of the event in the format required by the XDAS common audit  
 2113 format.

2114        *event\_info* (optional,in)

2115            Information specific to the event

2116        *audit\_record\_descriptor* (out)

2117            Pointer to an audit record, populated as defined by the optional input parameters. If the  
 2118 event does not need to be audited, a NULL pointer is returned.

**2119 RETURN VALUE**

2120 The following XDAS status codes shall be returned:

2121 [XDAS\_S\_COMPLETE]

2122 Successful completion.

2123 [XDAS\_S\_INVALID\_INITIATOR\_INFO]

2124 The initiator information specified has a syntax error. |

2125 [XDAS\_S\_INVALID\_EVENT\_NO]

2126 The event number specified is not valid.

2127 [XDAS\_S\_INVALID\_OUTCOME]

2128 The outcome supplied is not valid.

2129 [XDAS\_S\_INVALID\_TARGET\_INFO]

2130 The target information specified has a syntax error. |

2131 [XDAS\_S\_INVALID\_EVENT\_INFO]

2132 The event information specified is not valid.

2133 [XDAS\_S\_NO\_AUDIT]

2134 The specified event does not need to be audited.

2135 [XDAS\_S\_UNCERTAIN\_AUDIT]

2136 There is uncertainty as to whether the specified event requires auditing.

2137 [XDAS\_S\_FAILURE]

2138 An implementation specific error or failure has occurred.

2139 [XDAS\_S\_AUTHORISATION\_FAILURE]

2140 The caller does not possess the required authority.

**2141 ERRORS**

2142 No other errors are defined.



2143 **NAME**2144 `xdas_terminate_session` — terminate a session with the distributed audit service2145 **SYNOPSIS**

```
2146     OM_uint32 xdas_terminate_session (
2147         OM_uint32          *minor_status
2148         xdas_audit_ref_t    *das_ref,
2149     );
```

2150 **DESCRIPTION**

2151 The `xdas_terminate_session` closes a session between the caller and the distributed audit service.  
2152 The caller must have the XDAS\_AUDIT\_SERVICE authority.

2153 If successful, the function returns [XDAS\_S\_COMPLETE].

2154 The arguments for `xdas_terminate_session()` are:

2155 *minor\_status* (out)

2156 An implementation specific return status that provides additional information when  
2157 [XDAS\_S\_FAILURE] is returned by the function.

2158 *das\_ref* (in)

2159 The handle to the XDAS server, obtained from a previous call to `xdas_initialise_session`.

2160 **RETURN VALUE**

2161 The following XDAS status codes shall be returned:

2162 [XDAS\_S\_COMPLETE]

2163 Successful completion.

2164 [XDAS\_S\_INVALID\_DAS\_REF]

2165 The audit daemon handle supplied does not represent a valid audit service session.

2166 [XDAS\_S\_FAILURE]

2167 An implementation specific error or failure has occurred.

2168 [XDAS\_S\_AUTHORISATION\_FAILURE]

2169 The caller does not possess the required authority.

2170 **ERRORS**

2171 No other errors are defined.

2172 **NAME**2173 `xdas_timestamp_record` — timestamp the supplied audit record2174 **SYNOPSIS**

```
2175     OM_uint32 xdas_timestamp_record (
2176         OM_uint32          *minor_status
2177         xdas_audit_ref_t   *das_ref,
2178         xdas_audit_desc_t   *audit_record_descriptor,
2179     );
```

2180 **DESCRIPTION**2181 The *xdas\_timestamp\_record* puts a timestamp on the audit record supplied. The caller must have  
2182 the XDAS\_AUDIT\_SUBMIT authority.

2183 If successful, the function returns [XDAS\_S\_COMPLETE].

2184 The arguments for *xdas\_timestamp\_record()* are:2185 *minor\_status* (out)2186 An implementation specific return status that provides additional information when  
2187 [XDAS\_S\_FAILURE] is returned by the function.2188 *das\_ref* (in)2189 The handle to the XDAS server, obtained from a previous call to *xdas\_initialise\_session*.2190 *audit\_record\_descriptor* (in)2191 The handle to the audit record returned from a previous call to *xdas\_start\_record()*2192 **RETURN VALUE**

2193 The following XDAS status codes shall be returned:

2194 [XDAS\_S\_COMPLETE]

2195 Successful completion.

2196 [XDAS\_S\_INVALID\_DAS\_REF]

2197 The audit daemon handle supplied does not represent a valid audit service session.

2198 [XDAS\_S\_INVALID\_RECORD\_DESCRIPTOR]

2199 The specified audit record descriptor is not valid.

2200 [XDAS\_S\_FAILURE]

2201 An implementation specific error or failure has occurred.

2202 [XDAS\_S\_AUTHORISATION\_FAILURE]

2203 The caller does not possess the required authority.

2204 **ERRORS**

2205 No other errors are defined.

# Mapping of DAS Events

2206

## 07 Notes to Reviewers

08 *This section with side shading will not appear in the final copy. - Ed.*

09 These mappings need have been reworked using the revised set of XDAS events. The Oracle  
10 mappings are suspect as the editor is not familiar with the semantics of some of the statements.

2211 The following events have been taken from the Oracle Database Administrator's Manual. The  
2212 table below presents an illustrative mapping to XDAS events.

2213 **Table A-1** Mapping of ORACLE Audit Events to XDAS Generic Audit Events

Oracle Event Description	XDAS-API Event(s)
Alter system	configure service or application
Create/drop cluster	create/delete data item
Alter/truncate cluster	modify data item
	Modify data item contents ??
Create/drop database link	create/delete data item
Create/delete index	create/delete data item
Alter index	modify data item
Not exists	THIS IS REPRESENTED BY AN OUTCOME CODE
Create/replace function	configure service or application
Create/replace package/package body	configure service or application
Create/replace procedure	configure service or application
Drop function, package, procedure	configure service or application
Create/drop public database link	configure service or application
Create/drop public synonym	configure service or application
Create/drop role	configure service or application
Set/alter role	configure service service or application
Create/drop rollback segment	create/delete data item
Alter rollback segment	configure service
Create/drop sequence	create/delete data item
Session connect/disconnect	create/terminate an association
Set system audit	configure audit service
System grant	modify account attributes
Create/drop table	create/delete data item
Truncate table	modify data item contents
Create/drop tablespace	configure service or application
Alter tablespace	configure service or application
Create trigger	configure service or application
Alter trigger enable/disable	modify data item
Create/drop/alter user	create/delete/modify account
Create/drop view	create/delete data item
Alter sequence	modify data item

	<b>Oracle Event Description</b>	<b>XDAS-API Event(s)</b>
2247		
2248		
2249	Alter table, comment on table	modify data item
2250	Execute procedure	invoke service or application
2251	Grant/revoke privilege on procedure	configure service or application
2252	Grant/revoke privilege on sequence	configure service or application
2253	Grant/revoke privilege on table	modify data item attributes
2254	Insert into table	modify data item
2255	Lock table	modify data item attributes
2256	Select sequence, table	create association with data item
2257	Update table, view	modify data item
2258	Upgrade data	modify data item attributes
2259	Downgrade data	modify data item attributes
2260	Upgrade higher level rows	modify data item attributes
2261	Insert, update, delete lower level rows	create/delete data items,
2262		modify data item attributes
2263	Lower DBMS label	modify data item attributes
2264	Raise DBMS label	modify data item attributes
2265	Alter DBMS label to a non-comparable label	modify data item attributes
2266	Grant MAC privileges	modify account attributes,
2267		modify an association context
2268	Switch modes	modify an association context

2269 **A.1 Mapping**

2270 The following events have been taken from the SUN Solaris BSM Manual for audit records. The  
2271 table below shows where they map to the suggested GASAPI events.

2272 **Table A-2** Mapping of Solaris BSM Audit Events to XDAS Generic Audit Events

	<b>BSM Kernel-level Audit Events</b>	<b>XDAS-API Event</b>
2273		
2274		
2275	access(2)	query data item attributes
2276	acct(2)	configure audit service
2277	adjtime(2)	configure service or application
2278	chdir(2)	modify processing context
2279	chmod(2)	modify data item attributes
2280	chown(2)	modify data item attributes
2281	chroot(2)	modify processing context
2282	close(2)	terminate association with data item
2283	creat(2)	create data item
2284	exec(2)	invoke service or application component
2285	execve(2)	as exec(2)
2286	exit(2)	terminate service or application component
2287	fchdir(2)	modify processing context
2288	fchmod(2)	modify data item attributes
2289	fchown(2)	modify data item attributes
2290	fchroot(2)	modify processing context
2291	fcntl(2)	modify data item attributes
2292	fork(2)	invoke service or application
2293	fstat(2)	query data item attributes
2294	fstatfs(2)	query configuration of service or application
2295	ioctl(2)	modify data item attributes
2296	kill(2)	modify data item contents
2297	link(2)	modify data item attributes
2298	lstat(2)	query data item attributes
2299	mkdir(2)	create data item
2300	mknod(2)	create data item
2301	mmap(2)	create a data item
2302	mount(2)	invoke service or application or enable service
2303		
2304	msgctl(2)	modify data item attributes
2305	msgget(2)	create data item, or create an association with peer
2306		
2307	msgrcv(2)	query data item contents
2308	msgsnd(2)	modify data item contents
2309	munmap(2)	delete data item
2310	open(2)	create an association with a data item
2311	pathconf(2)	query context of association with data item
2312	pipe(2)	create a data item
2313	process dumped core	resource corruption
2314	readlink(2)	query data item contents

	<b>BSM Kernel-level Audit Events</b>	<b>XDAS-API Event</b>
2315		
2316		
2317	rename(2)	modify data item,
2318		modify data item attributes
2319	rmdir(2)	delete data item
2320	semctl(2)	modify data item attributes
2321	semget(2)	create data item
2322		or create an association with peer
2323	semop(2)	query/modify data item contents
2324	setgroups(2)	modify user session attributes
2325	setspgrp(2)	modify user session attributes
2326	setrlimit(2)	query/modify configuration of service or application
2327	shmat(2)	create association with peer
2328	shmctl(2)	query/modify data item attributes
2329	shmdt(2)	terminate association with peer
2330	shmget(2)	create data item
2331	stat(2)	query data item attributes
2332	statfs(2)	query configuration of service or application
2333	symlink(2)	modify data item attributes
2334	system(2)	invoke a service or application
2335	umount(2)	terminate a service or application
2336	unlink	modify data item attributes
2337	utimes	modify data item attributes
2338	vfork(2)	invoke service or application
2339	vtrace(2)	invoke service or application
2340	/usr/sbin/allocate	startup system components/services
2341		shutdown system components/services
2342		enable or disable devices
2343	/usr/sbin/halt	shutdown system
2344	/usr/sbin/inetd	create an association with a peer
2345	/usr/sbin/in.ftpd	creat an association with a peer
2346	/usr/bin/login	create user session
2347	/usr/lib/nfs/mountd	modify configuration of service or application
2348	/usr/bin/passwd	modify account attributes
2349	/usr/sbin/reboot	start system
2350	/usr/sbin/in.rshd	creat an association with peer
2351		or create user session
2352	/usr/bin/su	create user session
2353		or modify user session attributes

2354 **A.2 IEEE P1003.1e -- Protection, Audit and Control Interfaces**

2355 This table maps the audit events defined in IEEE P1003.1e Draft 15 with the generic XDAS  
 2356 events.

	<b>P1003.1e Audit Event</b>	<b>XDAS-API Event</b>
2357		
2358		
2359	AUD_AET_AUD_SWITCH	Configure audit service
2360	AUD_AET_AUD_WRITE	access to other services
2361	AUD_AET_CHDIR	modify processing context
2362	AUD_AET_CHMOD	modify data item attributes
2363	AUD_AET_CHOWN	modify data item attributes
2364	AUD_AET_CREAT	create a data item
2365	AUD_AET_DUP	create association with a data item
2366	AUD_AET_EXEC	invoke service or application
2367	AUD_AET_EXIT	terminate service or application
2368	AUD_AET_FORK	invoke service or application
2369	AUD_AET_KILL	terminate service or application
2370	AUD_AET_LINK	modify data item attributes
2371		modify data item contents
2372	AUD_AET_MKDIR	create data item
2373	AUD_AET_MKFIFO	create data item
2374	AUD_AET_OPEN	create association with data item
2375	AUD_AET_PIPE	create data item
2376	AUD_AET_RENAME	amodify data item attributes
2377		modify data item contents
2378	AUD_AET_RMDIR	delete data item
2379	AUD_AET_SETGID	modify user session attributes
2380	AUD_AET_SETUID	modify user session attributes
2381	AUD_AET_UNLINK	modify data item attributes
2382		modify data item contents
2383	AUD_AET_UTIME	modify data item attributes
2384	AUD_AET_ACL_DELETE_DEF_FILE	modify data item attributes
2385	AUD_AET_ACL_SET_FD	modify data item attributes
2386	AUD_AET_ACL_SET_FILE	modify data item attributes
2387	AUD_AET_CAP_SET_FD	modify data item attributes
2388	AUD_AET_CAP_SET_FILE	modify data item attributes
2389	AUD_AET_CAP_SET_PROC	modify processing context
2390	AUD_AET_INF_SET_FD	modify data item attributes
2391	AUD_AET_INF_SET_FILE	modify data item attributes
2392	AUD_AET_INF_SET_PROC	modify processing context
2393	AUD_AET_MAC_SET_FD	modify data item attributes
2394	AUD_AET_MAC_SET_FILE	modify data item attributes
2395	AUD_AET_MAC_SET_PROC	modify processing context

2396 **Table A-3** Mapping of IEEE P1003.1e Audit Events to XDAS Generic Audit Events

2397

|



# XDAS Naming Syntax

2398

2399 The XDAS name syntax is based upon the composite name syntax defined by the XFN  
 2400 Preliminary Specification. Although no divergence of syntax definition is planned, this  
 2401 specification will not necessarily be updated to reflect changes in the XFN specification as they  
 2402 may occur.

## 03 Notes to Reviewers

04 *This section with side shading will not appear in the final copy. - Ed.*

05 I have changed the reference to ISO 646 as used in XFN to ISO 8859-1.

## 2406 B.1 Composite Name String Syntax

2407 An *XDAS composite name* consists of an ordered list of zero or more components. Each  
 2408 component is a string name from the namespace of a single naming system and uses the naming  
 2409 syntax of that naming system. A component may be an atomic or a compound name from that  
 2410 namespace. XFN does not specify any syntax for regular expressions at the composite name  
 2411 level. However, an individual naming system may allow a component to contain expressions  
 2412 (for example, wildcard characters).

2413 This form is the concatenation of the components of a composite name from left to right with the  
 2414 *XDAS component separator* character ('/') separating each component.

### 2415 B.1.1 Encoding of XDAS Composite Name Strings

2416 Special characters used in the XDAS composite name syntax, such as the component separator  
 2417 or escape characters, have the same encoding as they would in ISO 8859-1 .

2418 The minimum requirement for all XDAS implementations is to support the portable  
 2419 representation of ISO 8859-1 for communication of name strings.

### 2420 B.1.2 Backus-Naur Form (BNF) of XDAS Composite Names

2421 This section defines the standard string form of XDAS composite names in BNF. Note that all  
 2422 the characters of the string representation of one name must uniformly use the same encoding  
 2423 and locale information.

2424 The notations used are as follows:

2425

2426

2427

2428

2429

2430

2431

2432

Symbol	Meaning
::=	Is defined to be
	Alternatively
<text>	Non-terminal element
""	Literal expression
*	The preceding syntactic unit can appear 0 or more times.
+	The preceding syntactic unit can appear 1 or more times.

```

2433   | {} | The enclosed syntactic units are grouped as a single syntactic unit (can be nested). |
2434   The XFN composite name syntax in BNF is as follows.
2435   NULL ::=          // Empty set
2436   <PCS> ::=         // Portable Character Set
2437                   // The set consists of the glyphs:
2438                   // !"#$%&'()*+,-./0123456789;<=>?
2439                   // @ABCDEFGHIJKLMNQRSTUUVWXYZ[\]^_
2440                   // 'abcdefghijklmnopqrstuvwxyz{|}~
2441   <CharSet> ::=     <PCS>
2442                   | Characters from the repertoire of a string representation
2443   <EscapeChar> ::=  \
2444   <ComponentSep> ::= /
2445   <Quote1> ::=     "
2446   <Quote2> ::=     '
2447   <MetaChar> ::=   <EscapeChar> | <ComponentSep>
2448   <SimpleChar> ::= // any character from <CharSet> with <ComponentSep>, <Quote1>,
2449                   // and <Quote2> excluded. An <EscapeChar> <MetaChar>, or
2450                   // <EscapeChar> <Quote1>, or <EscapeChar> <Quote2> is
2451                   // substituted by the corresponding unescaped character and
2452                   // is equivalent to a <SimpleChar>.
2453   <Component> ::=  <SimpleChar>*
2454                   | <SimpleChar>+ {<Quote1> | <Quote2> | <SimpleChar>}*
2455                   | <Quote1> <CharSet>* {<EscapeChar><Quote1>}* <CharSet>*
2456                   <Quote1>
2457                   // <CharSet> must not contain unescaped <Quote1>
2458                   // (note that <Quote2> can appear unescaped)
2459                   | <Quote2> <CharSet>* {<EscapeChar><Quote2>}* <CharSet>*
2460                   <Quote2>
2461                   // <CharSet> must not contain unescaped <Quote2>
2462                   // (note that <Quote1> can appear unescaped)
2463   <CompositeName> ::= NULL
2464                   | <Component> {<ComponentSep> <Component>}*
2465

```

**67 Notes to Reviewers**

68 *This section with side shading will not appear in the final copy. - Ed.*

69 This glossary is the glossary from the XDSF. It needs those terms that are irrelevant to this  
70 specification removing and any other relevant terms adding.

**2471 access control**

2472 The prevention of unauthorised use of a resource including the prevention of use of a  
2473 resource in an unauthorised manner (see ).

**2474 access control certificate**

2475 ADI in the form of a security certificate (see ).

**2476 access control decision function**

2477 (ADF) — a specialised function that makes access control decisions by applying access  
2478 control policy rules to a requested action, ACI (of initiators, targets, actions, or that retained  
2479 from prior actions), and the context in which the request is made (see ).

**2480 access control decision information**

2481 (ADI) — the portion (possibly all) of the ACI made available to the ADF in making a  
2482 particular access control decision (see ).

**2483 access control enforcement function**

2484 (AEF) — a specialised function that is part of the access path between an initiator and a  
2485 target on each access that enforces the decisions made by the ADF (see ).

**2486 access control information**

2487 (ACI) — any information used for access control purposes, including contextual  
2488 information (see ).

**2489 access control list**

2490 A list of entities, together with their access rights which are authorised to have access to a  
2491 resource (see ).

**2492 access control policy**

2493 The set of rules that define the conditions under which an access may take place (see ).

**2494 accountability**

2495 The property that ensures that the actions of an entity may be traced to that entity (see ).

**2496 ACI**

2497 Access control information.

**2498 ACL**

2499 Access control list.

**2500 action**

2501 The operations and operands that form part of an attempted access (see ).

**2502 action ADI**

2503 Action decision information associated with the action (see ).

**2504 active threat**

2505 The threat of a deliberate unauthorised change to the state of the system

2506	<b>ADF</b>
2507	Access control decision function.
2508	<b>ADI</b>
2509	Access control decision information.
2510	<b>administrative security information</b>
2511	Persistent information associated with entities; it is conceptually stored in the Security
2512	Management Information Base. Examples are:
2513	<ul style="list-style-type: none"> <li>• security attributes associated with users and set up on user account installation, which is</li> </ul>
2514	used to configure the user's identity and privileges within the system
2515	<ul style="list-style-type: none"> <li>• information configuring a secure interaction policy between one entity and another</li> </ul>
2516	entity, which is used as the basis for the establishment of operational associations
2517	between those two entities.
2518	<b>AEF</b>
2519	Access control enforcement function.
2520	<b>alarm collector function</b>
2521	A function that collects the security alarm messages, translates them into security alarm
2522	records, and writes them to the security alarm log (see ).
2523	<b>alarm examiner function</b>
2524	A function that interfaces with a security alarm administrator (see ).
2525	<b>API</b>
2526	Application Programming Interface.
2527	The interface between the application software and the application platform, across which
2528	all services are provided.
2529	The application programming interface is primarily in support of application portability,
2530	but system and application interoperability are also supported by a communication API
2531	(see <b>Procurement Guide</b> ).
2532	<b>assertion</b>
2533	Explicit statement in a system security policy that security measures in one security domain
2534	constitute an adequate basis for security measures (or lack of them) in another (see ).
2535	<b>association-security-state</b>
2536	The collection of information that is relevant to the control of communications security for a
2537	particular application-association (see ).
2538	<b>audit</b>
2539	See Security Audit (see ).
2540	<b>audit analysis</b>
2541	The <i>analysis</i> of audit data comprises manual or automated processes which scrutinize the
2542	audit data to identify in them real or potential security threats or to track system activity for
2543	the purpose of assigning accountability. Several approaches are possible including:
2544	<ul style="list-style-type: none"> <li>• to compare activity with a profile based on <i>normal</i> behaviour;</li> </ul>
2545	<ul style="list-style-type: none"> <li>• to seek out unacceptable or suspicious events by establishing a rules base for</li> </ul>
2546	inappropriate system activity.
2547	Analysis can generate filtering requirements which can be fed back into the discrimination
2548	process and provide strong reporting utilities.

- 2549           **audit authority**  
2550           The manager responsible for defining those aspects of a security policy applicable to  
2551           maintaining a security audit (see ).
- 2552           **audit event detector function**  
2553           A function that detects the occurrence of security-relevant events. This function is normally  
2554           an inherent part of the functionality implementing the event (see ).
- 2555           **audit event discriminator function**  
2556           A function that filters audit events against pre-configured criteria. The filter mechanism is  
2557           parameter driven, based on policies or rules. This function may be invoked prior to event  
2558           generation, to determine whether a detected audit event is required to be audited, or after  
2559           event generation to determine how a generated event is to be handled, for example logged  
2560           or an alarm generated.
- 2561           **audit recorder function**  
2562           A function that records the security-relevant messages in a security audit trail (see ).
- 2563           **audit trail**  
2564           See Security Audit Trail (see ).
- 2565           **audit trail analyser function**  
2566           A function that checks a security audit trail in order to produce, if appropriate, security  
2567           alarm messages (see ).
- 2568           **audit trail archiver function**  
2569           A function that archives a part of the security audit trail (see ).
- 2570           **audit trail collector function**  
2571           A function that collects individual audit trail records into a security audit trail (see ).
- 2572           **audit trail examiner function**  
2573           A function that builds security reports out of one or more security audit trails (see ).
- 2574           **audit trail provider function**  
2575           A function that provides security audit trails according to some criteria (see ).
- 2576           **authenticated identity**  
2577           An identity of a principal that has been assured through authentication (see ).
- 2578           **authentication**  
2579           Verify claimed identity; see data origin authentication, and peer entity authentication (see ).
- 2580           **authentication certificate**  
2581           Authentication information in the form of a security certificate which may be used to assure  
2582           the identity of an entity guaranteed by an authentication authority (see ).
- 2583           **authentication exchange**  
2584           A sequence of one or more transfers of exchange authentication information (AI) for the  
2585           purposes of performing an authentication (see ).
- 2586           **authentication information (AI)**  
2587           Information used to establish the validity of a claimed identity (see ).
- 2588           **authentication initiator**  
2589           The entity which starts an authentication exchange (see ).
- 2590           **authentication method**  
2591           Method for demonstrating knowledge of a secret. The quality of the authentication method,  
2592           its strength is determined by the cryptographic basis of the key distribution service on

- 2593 which it is based. A symmetric key based method, in which both entities share common  
 2594 authentication information, is considered to be a weaker method than an asymmetric key  
 2595 based method, in which not all the authentication information is shared by both entities.
- 2596 **authorisation**  
 2597 The granting of rights, which includes the granting of access based on access rights (see ).
- 2598 **authorisation policy**  
 2599 A set of rules, part of an access control policy, by which access by security subjects to  
 2600 security objects is granted or denied. An authorisation policy may be defined in terms of  
 2601 access control lists, capabilities or attributes assigned to security subjects, security objects or  
 2602 both (see ).
- 2603 **availability**  
 2604 The property of being accessible and usable upon demand by an authorised entity (see ).
- 2605 **capability**  
 2606 A token used as an identifier for a resource such that possession of the token confers access  
 2607 rights for the resource (see ).
- 2608 **ciphertext**  
 2609 Data produced through the use of encipherment. The semantic content of the resulting data  
 2610 is not available (see ).
- 2611 **Note:** Ciphertext may itself be input to encipherment, such that super-enciphered output  
 2612 is produced.
- 2613 **claim authentication information**  
 2614 (Claim AI) — information used by a claimant to generate exchange AI needed to  
 2615 authenticate a principal (see ).
- 2616 **claimant**  
 2617 An entity which is or represents a principal for the purposes of authentication. A claimant  
 2618 includes the functions necessary for engaging in authentication exchanges on behalf of a  
 2619 principal (see ).
- 2620 **clear text**  
 2621 Intelligible data, the semantic content of which is available (see ).
- 2622 **client-server**  
 2623 These operations occur between a pair of communicating independent peer processes. The  
 2624 peer process initiating a service request is termed the client. The peer process responding to  
 2625 a service request is termed the server. A process may act as both client and server in the  
 2626 context of a set of transactions.
- 2627 **confidentiality**  
 2628 The property that information is not made available or disclosed to unauthorised  
 2629 individuals, entities, or processes (see ).
- 2630 **contextual information**  
 2631 Information derived from the context in which an access is made (for example, time of day)  
 2632 (see ).
- 2633 **corporate security policy**  
 2634 The set of laws, rules and practices that regulate how assets including sensitive information  
 2635 are managed, protected and distributed within a user organisation (see ).
- 2636 **countermeasure**  
 2637 The deployment of a set of security services to protect against a security threat.

- 2638           **credentials**  
2639           Data that is transferred to establish the claimed identity of an entity (see ).
- 2640           **cryptanalysis**  
2641           The analysis of a cryptographic system and its inputs and outputs to derive confidential  
2642           variables and/or sensitive data including clear text (see ).
- 2643           **cryptographic algorithm**  
2644           A method of performing a cryptographic transformation (see cryptography) on a data unit.  
2645           Cryptographic algorithms may be based on symmetric key methods (the same key is used  
2646           for both encipher and decipher transformations) or on asymmetric keys (different keys are  
2647           used for encipher and decipher transformations).
- 2648           **cryptographic checkvalue**  
2649           Information that is derived by performing a cryptographic transformation (see  
2650           cryptography) on a data unit (see ).
- 2651           **Note:**   The derivation of the checkvalue may be performed in one or more steps and is a  
2652           result of a mathematical function of the key and data unit. It is usually used to  
2653           check the integrity of a data unit.
- 2654           **cryptography**  
2655           The discipline that embodies principles, means, and the methods for the transformation of  
2656           data in order to hide its information content, prevent its undetected modification and/or  
2657           prevent its unauthorised use (see ).
- 2658           **Note:**   The choice of cryptography mechanism determines the methods used in  
2659           encipherment and decipherment. An attack on a cryptographic principle, means or  
2660           methods is cryptanalysis.
- 2661           **data integrity**  
2662           The property that data has not been altered or destroyed in an unauthorised manner (see ).
- 2663           **data origin authentication**  
2664           The corroboration that the entity responsible for the creation of a set of data is the one  
2665           claimed.
- 2666           **decipherment**  
2667           The reversal of a corresponding reversible encipherment (see ).
- 2668           **decryption**  
2669           See decipherment (see ).
- 2670           **denial of service**  
2671           The unauthorised prevention of authorised access to resources or the delaying of time-  
2672           critical operations (see ).
- 2673           **digital fingerprint**  
2674           A characteristic of a data item, such as a cryptographic checkvalue or the result of  
2675           performing a one-way hash function on the data, that is sufficiently peculiar to the data  
2676           item that it is computationally infeasible to find another data item that possesses the same  
2677           characteristics (see ).
- 2678           **digital signature**  
2679           Data appended to, or a cryptographic transformation (see cryptography) of, a data unit that  
2680           allows a recipient of the data unit to prove the source and integrity of the data unit and  
2681           protect against forgery for example, by the recipient (see ).

- 2682        **discretionary access control**  
 2683            A discretionary authorisation scheme is one under which any principal using the domain  
 2684            services may be authorised to assign or modify ACI such that he may modify the  
 2685            authorisations of other principals under the scheme. A typical example is an ACL scheme  
 2686            which is often referred to as Discretionary Access Control (DAC).
- 2687        **distinguishing identifier**  
 2688            Data that unambiguously distinguishes an entity in the authentication process. Such an  
 2689            identifier shall be unambiguous at least within a security domain (see ).
- 2690        **distributed application**  
 2691            A set of information processing resources distributed over one or more open systems which  
 2692            provides a well-defined set of functionality to (human) users, to assist a given (office) task  
 2693            (see ).
- 2694        **encapsulated subsystem**  
 2695            A collection of procedures and data objects that is protected in a domain of its own so that  
 2696            the internal structure of a data object is accessible only to the procedures of the  
 2697            encapsulated subsystem and that those procedures may be called only at designated  
 2698            domain entry points. Encapsulated subsystem, protected subsystem and protected  
 2699            mechanisms of the TCB are terms that may be used interchangeably (see ).
- 2700        **encipherment**  
 2701            The cryptographic transformation of data (see cryptography) to produce ciphertext (see ).  
 2702            **Note:** Encipherment may be irreversible, in which case the corresponding decipherment  
 2703            process cannot feasibly be performed. Such encipherment may be called a one-  
 2704            way-function or cryptochecksum.
- 2705        **encryption**  
 2706            See encipherment (see ).
- 2707        **end-to-end encipherment**  
 2708            Encipherment of data within or at the source end system, with the corresponding  
 2709            decipherment occurring only within or at the destination end system (see ).
- 2710        **exchange authentication information**  
 2711            (Exchange AI) — information exchanged between a claimant and a verifier during the  
 2712            process of authenticating a principal (see ).
- 2713        **identification**  
 2714            The assignment of a name by which an entity can be referenced. The entity may be high  
 2715            level (such as a user) or low level (such as a process or communication channel).
- 2716        **identity-based security policy**  
 2717            A security policy based on the identities or attributes of users, a group of users, or entities  
 2718            acting on behalf of the users and the resources or targets being accessed (see ).
- 2719        **initiator**  
 2720            An entity (for example, human user or computer based entity) that attempts to access other  
 2721            entities (see ).
- 2722        **initiator access control decision information**  
 2723            (Initiator ADI) — ADI associated with the initiator (see ).
- 2724        **initiator access control information**  
 2725            (Initiator ACI) — access control information relating to the initiator (see ).



- 2726 **integrity**  
2727 See Data Integrity (see ).
- 2728 **key**  
2729 A sequence of symbols that controls the operations of encipherment and decipherment (see  
2730 ).
- 2731 **key management**  
2732 The generation, storage, distribution, deletion, archiving and application of keys in  
2733 accordance with a security policy (see ).
- 2734 **masquerade**  
2735 The unauthorised pretence by an entity to be a different entity (see ).
- 2736 **messaging application**  
2737 An application based on a store and forward paradigm; it requires an appropriate security  
2738 context to be bound with the message itself.
- 2739 **non-discretionary access control**  
2740 A non-discretionary authorisation scheme is one under which only the recognised security  
2741 authority of the security domain may assign or modify the ACI for the authorisation scheme  
2742 such that the authorisations of principals under the scheme are modified.
- 2743 **off-line authentication certificate**  
2744 A particular form of authentication information binding an entity to a cryptographic key,  
2745 certified by a trusted authority, which may be used for authentication without directly  
2746 interacting with the authority (see ).
- 2747 **on-line authentication certificate**  
2748 A particular form of authentication information, certified by a trusted authority, which may  
2749 be used for authentication following direct interaction with the authority (see ).
- 2750 **operational security information**  
2751 Transient information related to a single operation or set of operations within the context of  
2752 an operational association, for example, a user session. Operational security information  
2753 represents the current security context of the operations and may be passed as parameters  
2754 to the operational primitives or retrieved from the operations environment as defaults.
- 2755 **organisational security policy**  
2756 Set of laws, rules, and practices that regulates how an organisation manages, protects, and  
2757 distributes sensitive information (see ).
- 2758 **password**  
2759 Confidential authentication information, usually composed of a string of characters (see ).
- 2760 **peer-entity authentication**  
2761 The corroboration that a peer entity in an association is the one claimed (see ).
- 2762 **physical security**  
2763 The measures used to provide physical protection of resources against deliberate and  
2764 accidental threats (see ).
- 2765 **platform domain**  
2766 A security domain encompassing the operating system, the entities and operations it  
2767 supports and its security policy.
- 2768 **policy**  
2769 See security policy (see ).

- 2770 **primary service**  
 2771 An independent category of service such as operating system services, communication  
 2772 services and data management services. Each primary service provides a discrete set of  
 2773 functionality. Each primary service inherently includes generic qualities such as usability,  
 2774 manageability and security.
- 2775 Security services are therefore not primary services but are invoked as part of the provision  
 2776 of primary services by the primary service provider.
- 2777 **principal**  
 2778 An entity whose identity can be authenticated (see ).
- 2779 **privacy**  
 2780 The right of individuals to control or influence what information related to them may be  
 2781 collected and stored and by whom and to whom that information may be disclosed.
- 2782 **Note:** because this term relates to the right of individuals, it cannot be very precise and  
 2783 its use should be avoided except as a motivation for requiring security (see ).
- 2784 **private key**  
 2785 A key used in an asymmetric algorithm. Possession of this key is restricted, usually to only  
 2786 one entity (see ).
- 2787 **public key**  
 2788 The key, used in an asymmetric algorithm, that is publicly available (see ).
- 2789 **quality of protection**  
 2790 A label that implies methods of security protection under a security policy. This normally  
 2791 includes a combination of integrity and confidentiality requirements and is typically  
 2792 implemented in a communications environment by a combination of cryptographic  
 2793 mechanisms.
- 2794 **repudiation**  
 2795 Denial by one of the entities involved in a communication of having participated in all or  
 2796 part of the communication (see ).
- 2797 **rule-based security policy**  
 2798 A security policy based on global rules imposed for all users. These rules usually rely on a  
 2799 comparison of the sensitivity of the resources being accessed and the possession of  
 2800 corresponding attributes of users, a group of users, or entities acting on behalf of users (see  
 2801 ).
- 2802 **seal**  
 2803 A cryptographic checkvalue that supports integrity but does not protect against forgery by  
 2804 the recipient (that is, it does not support non-repudiation). When a seal is associated with a  
 2805 data element, that data element is *sealed* (see ).
- 2806 **secondary discretionary disclosure**  
 2807 An example of the misuse of access rights. It occurs when a principal authorised to access  
 2808 some information copies that information and authorises access to the copy by a second  
 2809 principal who is not authorised to access the original information.
- 2810 **secret key**  
 2811 In a symmetric cryptographic algorithm the key shared between two entities (see ).
- 2812 **secure association**  
 2813 An instance of secure communication (using communication in the broad sense of space  
 2814 and/or time) which makes use of a secure context.

2815	<b>secure context</b>
2816	The existence of the necessary information for the correct operation of the security
2817	mechanisms at the appropriate place and time.
2818	<b>secure interaction policy</b>
2819	The common aspects of the security policies in effect at each of the communicating
2820	application processes (see ).
2821	<b>security architecture</b>
2822	A high level description of the structure of a system, with security functions assigned to
2823	components within this structure (see ).
2824	<b>security attribute</b>
2825	A security attribute is a piece of security information which is associated with an entity.
2826	<b>security audit</b>
2827	An independent review and examination of system records and operations in order to test
2828	for adequacy of system controls, to ensure compliance with established policy and
2829	operational procedures, to detect breaches in security and to recommend any indicated
2830	changes in control, policy and procedures (see ).
2831	<b>security audit message</b>
2832	A message generated following the occurrence of an auditable security-related event (see ).
2833	<b>security audit record</b>
2834	A single record in a security audit trail corresponding to a single security-related event (see
2835	).
2836	<b>security audit trail</b>
2837	Data collected and potentially used to facilitate a security audit (see ).
2838	<b>security auditor</b>
2839	An individual or a process allowed to have access to the security audit trail and to build
2840	audit reports (see ).
2841	<b>security aware</b>
2842	The caller of an API that is aware of the security functionality and parameters which may be
2843	provided by an API.
2844	<b>security certificate</b>
2845	A set of security-relevant data from an issuing security authority that is protected by
2846	integrity and data origin authentication, and includes an indication of a time period of
2847	validity (see ).
2848	<b>Note:</b> All certificates are deemed to be security certificates (see the relevant definitions in
2849	) adopted in order to avoid terminology conflicts with (that is the directory
2850	authentication standard).
2851	<b>security domain</b>
2852	A set of elements, a security policy, a security authority and a set of security-relevant
2853	operations in which the set of elements are subject to the security policy, administered by
2854	the security authority, for the specified operations (see ).
2855	<b>security event manager</b>
2856	An individual or process allowed to specify and manage the events which may generate a
2857	security message and to establish the action or actions to be taken for each security message
2858	type (see ).

2859	<b>security label</b>
2860	The marking bound to a resource (which may be a data unit) that names or designates the
2861	security attributes of that resource (see ).
2862	<b>Note:</b> The marking may be explicit or implicit.
2863	<b>security policy</b>
2864	The set of criteria for the provision of security services (see also identity-based and rule-
2865	based security policy).
2866	<b>security service</b>
2867	A service which may be invoked directly or indirectly by functions within a system that
2868	ensures adequate security of the system or of data transfers between components of the
2869	system or with other systems.
2870	<b>security state</b>
2871	State information that is held in an open system and which is required for the provision of
2872	security services.
2873	<b>security token</b>
2874	A set of security-relevant data that is protected by integrity and data origin authentication
2875	from a source that is not considered a security authority (see ).
2876	<b>security unaware</b>
2877	The caller of an API that is unaware of the security functionality and parameters which may
2878	be provided by an API.
2879	<b>sensitivity</b>
2880	The characteristic of a resource that implies its value or importance, and may include its
2881	vulnerability (see ).
2882	<b>separation</b>
2883	The concept of keeping information of different security classes apart in a system (see ).
2884	<b>Note:</b> Separation may be implemented by temporal, physical, logical or cryptographic
2885	techniques.
2886	<b>service domain</b>
2887	A security domain encompassing an application, the entities and operations it supports and
2888	its security policy.
2889	<b>signature</b>
2890	See digital signature (see ).
2891	<b>strength of mechanism</b>
2892	An aspect of the assessment of the effectiveness of a security mechanism, namely the ability
2893	of the security mechanism to withstand direct attack against deficiencies in its underlying
2894	algorithms, principles and properties (see ).
2895	<b>system security function</b>
2896	A capability of an open system to perform security-related processing (see ).
2897	<b>target</b>
2898	An entity to which access may be attempted (see ).
2899	<b>target ADI</b>
2900	ADI associated with the target (see ).
2901	<b>target ACI</b>
2902	Access control information relating to the target (see ).

2903	<b>threat</b>
2904	A potential violation of security (see ).
2905	An action or event that might prejudice security (see ).
2906	<b>traffic analysis</b>
2907	The inference of information from observation of traffic flows (presence, absence, amount,
2908	direction and frequency) (see ).
2909	<b>traffic flow confidentiality</b>
2910	A confidentiality service to protect against traffic analysis (see ).
2911	<b>traffic padding</b>
2912	The generation of spurious instances of communication, spurious data units or spurious
2913	data within data units (see ).
2914	<b>trap door</b>
2915	A hidden software or hardware mechanism that permits system protection mechanisms to
2916	be circumvented. It is activated in some non-apparent manner (for example, special
2917	“random” key sequence at a terminal) (see ).
2918	<b>trojan horse</b>
2919	Computer program containing an apparent or actual useful function that contains
2920	additional (hidden) functions that allow unauthorised collection, falsification or destruction
2921	of data (see ).
2922	<b>trust</b>
2923	A relationship between two elements, a set of operations and a security policy in which
2924	element X trusts element Y if and only if X has confidence that Y behaves in a well defined
2925	way (with respect to the operations) that does not violate the given security policy (see ).
2926	<b>trusted computing base (TCB)</b>
2927	The totality of protection mechanisms within an IT system, including hardware, firmware,
2928	software and data, the combination of which is responsible for enforcing the security policy.
2929	<b>trusted functionality</b>
2930	That which is perceived to be correct with respect to some criteria, for example, as
2931	established by a security policy (see ).
2932	<b>trusted path</b>
2933	Mechanism by which a person using a terminal can communicate directly with the TCB (see
2934	).
2935	<b>Note:</b> Trusted path can only be activated by the person or the TCB and cannot be
2936	imitated by untrusted software.
2937	<b>trusted third party</b>
2938	A security authority or its agent, trusted by other entities with respect to security-related
2939	operations (see ).
2940	<b>verification AI</b>
2941	Information used by a verifier to verify an identity claimed through exchange AI (see ).
2942	<b>verifier</b>
2943	An entity which is or represents the entity requiring an authenticated identity. A verifier
2944	includes the functions necessary for engaging in authentication exchanges (see ).
2945	<b>virus</b>
2946	Self replicating, malicious program segment that attaches itself to an application or other
2947	executable system component and leaves no external signs of its presence (see ).

2948  
2949  
2950  
2951

**vulnerability**

Weakness in an information system or components (for example, system security procedures, hardware design, internal controls) that could be exploited to produce an information-related misfortune (see ).

# Index

1

2	access control.....	85	authentication.....	87
3	access control certificate.....	85	authentication certificate .....	87
4	access control decision function.....	85	authentication exchange .....	87
5	access control decision information .....	85	authentication information (AI) .....	87
6	access control enforcement function .....	85	authentication initiator.....	87
7	access control information.....	85	authentication method .....	87
8	access control list.....	85	authorisation.....	88
9	access control policy .....	85	authorisation policy .....	88
10	accountability .....	85	availability.....	88
11	ACL.....	85	bnf.....	83
12	ACL.....	85	C-language	
13	action.....	85	names .....	39
14	action ADI.....	85	calling convention	
15	active threat .....	85	names .....	39
16	ADF.....	86	status value.....	35
17	ADI .....	86	calling conventions	
18	administrative security information.....	86	optional arguments.....	38
19	AEF .....	86	calling errors.....	35
20	alarm collector function .....	86	capability .....	88
21	alarm examiner function.....	86	ciphertext.....	88
22	API.....	86	claim authentication information .....	88
23	APIs .....	49	claimant .....	88
24	argument		clear text .....	88
25	optional.....	38	client-server .....	88
26	assertion.....	86	confidentiality .....	88
27	association-security-state.....	86	Conformance .....	7
28	audit.....	86	constants.....	39
29	Audit		contextual information.....	88
30	Read.....	29	corporate security policy .....	88
31	audit analysis.....	86	countermeasure .....	88
32	audit authority .....	87	credentials .....	89
33	audit event detector function.....	87	cryptanalysis.....	89
34	audit event discriminator function.....	87	cryptographic algorithm.....	89
35	Audit Event Record.....	15	cryptographic checkvalue.....	89
36	Audit Event Service Client API.....	28	cryptography.....	89
37	Audit Log Import API.....	29	DAS_S_STORAGE_FAILURE	
38	audit record		in xdas_commit_record() .....	51
39	format.....	33	data integrity .....	89
40	audit recorder function.....	87	data origin authentication .....	89
41	audit trail.....	87	data type	
42	audit trail analyser function .....	87	character strings.....	32
43	audit trail archiver function .....	87	integer .....	31
44	audit trail collector function.....	87	OM_uint32.....	35, 37
45	audit trail examiner function .....	87	string .....	32
46	audit trail provider function .....	87	structured.....	31
47	authenticated identity.....	87	xdas_buffer_t.....	32

48	data type filter .....	34	principal.....	92
49	decipherment.....	89	privacy .....	92
50	decryption .....	89	private key .....	92
51	denial of service .....	89	public key .....	92
52	digital fingerprint .....	89	quality of protection.....	92
53	digital signature .....	89	repudiation.....	92
54	discretionary access control .....	90	return value.....	35
55	distinguishing identifier.....	90	rule-based security policy.....	92
56	distributed application .....	90	seal .....	92
57	encapsulated subsystem .....	90	secondary discretionary disclosure .....	92
58	encipherment.....	90	secret key.....	92
59	encoding .....	83	secure association .....	92
60	encryption .....	90	secure context.....	93
61	end-to-end encipherment .....	90	secure interaction policy .....	93
62	error		security architecture.....	93
63	calling.....	35	security attribute.....	93
64	exchange authentication information .....	90	security audit.....	93
65	GCS_S_COMPLETE		security audit message .....	93
66	in xdas_release_buffer( .....	70	security audit record.....	93
67	GCS_S_FAILURE		security audit trail .....	93
68	in xdas_release_buffer( .....	70	security auditor .....	93
69	General		security aware .....	93
70	Audit .....	28	security certificate .....	93
71	identification.....	90	security domain .....	93
72	identity-based security policy.....	90	security event manager .....	93
73	initiator .....	90	security label.....	94
74	initiator access control decision information.....	90	security policy .....	94
75	initiator access control information.....	90	security service.....	94
76	integrity .....	91	security state .....	94
77	Introduction .....	1	security token .....	94
78	key.....	91	security unaware .....	94
79	key management.....	91	sensitivity .....	94
80	masquerade.....	91	separation.....	94
81	messaging application.....	91	service domain .....	94
82	minor status code .....	37	signature.....	94
83	non-discretionary access control.....	91	status code.....	35
84	off-line authentication certificate .....	91	minor.....	37
85	OM_uint32.....	35, 37	status value .....	35
86	on-line authentication certificate.....	91	strength of mechanism .....	94
87	operational security information .....	91	system security function .....	94
88	optional arguments.....	38	target .....	94
89	organisational security policy.....	91	target ACI.....	94
90	parameter		target ADI.....	94
91	(argument) .....	38	threat .....	95
92	Parameter Passing Conventions.....	31	traffic analysis .....	95
93	password .....	91	traffic flow confidentiality .....	95
94	peer-entity authentication .....	91	traffic padding.....	95
95	physical security .....	91	trap door.....	95
96	platform domain.....	91	trojan horse .....	95
97	policy.....	91	trust.....	95
98	primary service .....	92	trusted computing base (TCB).....	95



## Index

99	trusted functionality .....	95	XDas_OUT_PRE_SELECT_CRITERIA_SET .....	43
100	trusted path.....	95	XDas_OUT_PRIV_GRANTED.....	43
101	trusted third party.....	95	XDas_OUT_PRIV_REVOKED .....	43
102	verification AI.....	95	XDas_OUT_SUCCESS .....	43
103	verifier.....	95	XDas_OUT_THRESHOLDS_EXCEEDED.....	43
104	virus.....	95	XDas_OUT_THRESHOLDS_SET.....	43
105	vulnerability .....	96	xdas_put_event_info().....	68
106	XDas		xdas_release_buffer( .....	70
107	status code .....	35	xdas_release_filter_list().....	71
108	XDas Data Structures .....	15	xdas_rewind_audit_stream() .....	72
109	XDas Model.....	9	XDas_ROUTINE_ERROR().....	37
110	XDas Usage Model.....	27	xdas_start_record().....	73
111	XDas_ACT_ACTION.....	44	XDas_SUPPLEMENTARY_INFO() .....	37
112	XDas_ACT_ALARM.....	44	XDas_S_AUTHORISATION_FAILURE.....	36
113	XDas_ACT_LOG.....	44	in xdas_close_audit_stream().....	50
114	xdas_buffer.....	32	in xdas_commit_record().....	51
115	xdas_buffer_t.....	32	in xdas_create_filter() .....	53
116	types.....	38	in xdas_delete_filter() .....	54
117	XDas_CALLING_ERROR().....	37	in xdas_disable_filter() .....	55
118	xdas_close_audit_stream() .....	50	in xdas_discard_record().....	56
119	xdas_commit_record().....	51	in xdas_enable_filter() .....	57
120	xdas_create_filter() .....	52	in xdas_get_filter().....	59
121	XDas_C_EMPTY_BUFFER.....	39	in xdas_get_next() .....	61
122	XDas_C_IMPORT .....	45	in xdas_import_event_records() .....	62
123	XDas_C_NO_BUFFER.....	39	in xdas_initialise_session() .....	64
124	XDas_C_SUBMIT.....	45	in xdas_list_filters().....	66
125	XDas_DATE_TIME.....	47	in xdas_put_event_info() .....	69
126	xdas_delete_filter().....	54	in xdas_release_filter_list().....	71
127	xdas_disable_filter() .....	55	in xdas_rewind_audit_stream().....	72
128	xdas_discard_record() .....	56	in xdas_start_record() .....	74
129	xdas_enable_filter() .....	57	in xdas_terminate_session().....	75
130	XDas_EVENT_NUMBER.....	47	in xdas_timestamp_record() .....	76
131	xdas_get_filter() .....	58	XDas_S_CALL_BAD_STRUCTURE.....	35
132	xdas_get_next() .....	60	XDas_S_CALL_INACCESSIBLE_READ .....	35
133	xdas_import_event_records() .....	62	XDas_S_CALL_INACCESSIBLE_WRITE.....	35
134	xdas_initialise_session().....	64	XDas_S_COMPLETE.....	36
135	XDas_INR_AUTH_AUTH.....	47	in xdas_close_audit_stream().....	50
136	XDas_INR_IDENTITY.....	47	in xdas_commit_record().....	51
137	XDas_INR_NAME.....	47	in xdas_create_filter() .....	52
138	xdas_list_filters().....	66	in xdas_delete_filter() .....	54
139	xdas_open_audit_stream() .....	67	in xdas_disable_filter() .....	55
140	XDas_ORG_AUTH_AUTH.....	47	in xdas_discard_record().....	56
141	XDas_ORG_IDENTITY .....	47	in xdas_enable_filter() .....	57
142	XDas_ORG_LOC_ADD.....	47	in xdas_get_filter().....	58
143	XDas_ORG_LOC_NAME.....	47	in xdas_get_next() .....	60
144	XDas_ORG_NAME.....	47	in xdas_import_event_records() .....	62
145	XDas_ORG_SERV_TYPE .....	47	in xdas_initialise_session() .....	64
146	XDas_OUTCOME.....	47	in xdas_list_filters().....	66
147	XDas_OUT_ACTIONS_SET .....	43	in xdas_put_event_info() .....	69
148	XDas_OUT_DENIAL.....	43	in xdas_release_filter_list().....	71
149	XDas_OUT_FAILURE.....	43	in xdas_rewind_audit_stream().....	72

150	in xdas_start_record()	74	XDAS_S_INVALID_FILTER_EXP	
151	in xdas_terminate_session()	75	in xdas_create_filter()	52
152	in xdas_timestamp_record()	76	XDAS_S_INVALID_FILTER_LIST	36
153	XDAS_S_END		in xdas_release_filter_list()	71
154	in xdas_get_next()	61	XDAS_S_INVALID_FILTER_TYPE	36
155	XDAS_S_FAILURE	36	in xdas_create_filter()	52
156	in xdas_close_audit_stream()	50	in xdas_delete_filter()	54
157	in xdas_commit_record()	51	XDAS_S_INVALID_INITIATOR_INFO	36
158	in xdas_create_filter()	53	in xdas_put_event_info()	69
159	in xdas_delete_filter()	54	in xdas_start_record()	74
160	in xdas_disable_filter()	55	XDAS_S_INVALID_NAME	36
161	in xdas_discard_record()	56	XDAS_S_INVALID_ORIG_INFO	
162	in xdas_enable_filter()	57	in xdas_initialise_session()	64
163	in xdas_get_filter()	58	XDAS_S_INVALID_OUTCOME	36
164	in xdas_get_next()	61	in xdas_put_event_info()	69
165	in xdas_import_event_records()	62	in xdas_start_record()	74
166	in xdas_initialise_session()	64	XDAS_S_INVALID_RECORD_DESCRIPTOR	36
167	in xdas_list_filters()	66	in xdas_commit_record()	51
168	in xdas_put_event_info()	69	in xdas_discard_record()	56
169	in xdas_release_filter_list()	71	in xdas_put_event_info()	69
170	in xdas_rewind_audit_stream()	72	in xdas_timestamp_record()	76
171	in xdas_start_record()	74	XDAS_S_INVALID_SECURITY_CONTEXT	36
172	in xdas_terminate_session()	75	in xdas_initialise_session()	64
173	in xdas_timestamp_record()	76	XDAS_S_INVALID_TARGET_INFO	36
174	XDAS_S_INVALID_ACTION_LIST		in xdas_put_event_info()	69
175	in xdas_create_filter()	53	in xdas_start_record()	74
176	XDAS_S_INVALID_AUDIT_STREAM	36	XDAS_S_NO_AUDIT	36
177	in xdas_close_audit_stream()	50	in xdas_put_event_info()	69
178	in xdas_rewind_audit_stream()	72	in xdas_start_record()	74
179	XDAS_S_INVALID_DAS_REF	36	XDAS_S_RECORD_SYNTAX_ERROR	
180	in xdas_close_audit_stream()	50	in xdas_import_event_records()	62
181	in xdas_commit_record()	51	XDAS_S_SERVICE_FAILURE	36
182	in xdas_create_filter()	52	in xdas_commit_record()	51
183	in xdas_delete_filter()	54	XDAS_S_STORAGE_FAILURE	36
184	in xdas_disable_filter()	55	XDAS_S_UNCERTAIN_AUDIT	36
185	in xdas_discard_record()	56	in xdas_put_event_info()	69
186	in xdas_enable_filter()	57	in xdas_start_record()	74
187	in xdas_get_filter()	58	xdas_terminate_session()	75
188	in xdas_get_next()	60	xdas_timestamp_record()	76
189	in xdas_import_event_records()	62	XDAS_TRT_AUTH_AUTH	47
190	in xdas_list_filters()	66	XDAS_TRT_IDENTITY	47
191	in xdas_put_event_info()	69	XDAS_TRT_LOC_ADD	47
192	in xdas_release_filter_list()	71	XDAS_TRT_LOC_NAME	47
193	in xdas_rewind_audit_stream()	72	XDAS_TRT_NAME	47
194	in xdas_terminate_session()	75	XDAS_TRT_SERV_TYPE	47
195	in xdas_timestamp_record()	76	XDAS_VERSION	47
196	XDAS_S_INVALID_EVENT_INFO	36		
197	in xdas_put_event_info()	69		
198	in xdas_start_record()	74		
199	XDAS_S_INVALID_EVENT_NO	36		
200	in xdas_start_record()	74		