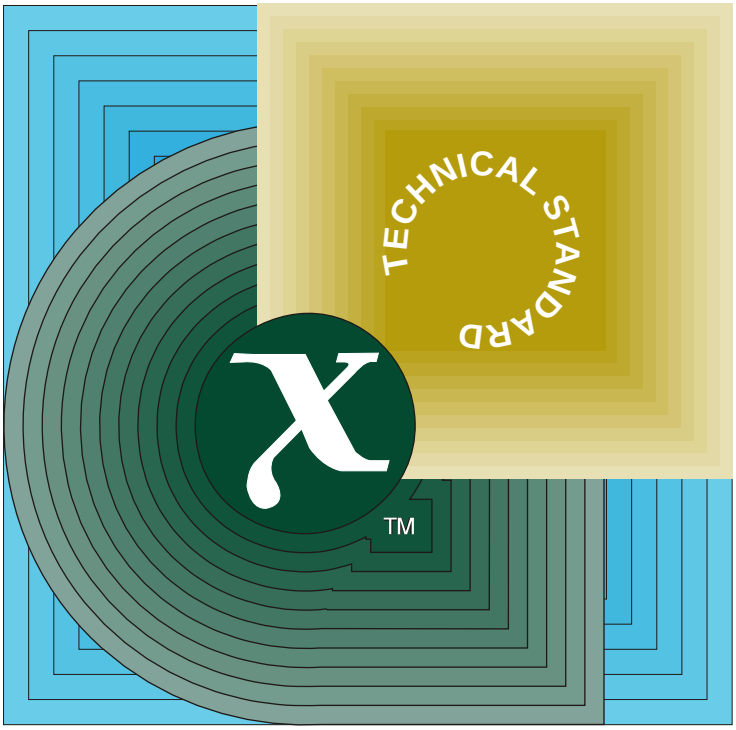
Technical Standard

Systems Management: Software License Use Management (XSLM)



THE Open GROUP

[This page intentionally left blank]

Technical Standard:

Systems Management:
Software License Use Management (XSLM)

The Open Group

This document is published by The Open Group under the terms and conditions of its agreement with GUIDE International, Inc., and the XSLM working group participants:

BC Government / ISM-BC
Boole & Babbage
Compuware Corporation
IBM Corporation
Isogon Corporation

Under this agreement, such permission is hereby granted to GUIDE International, Inc. and the XSLM working group participants.

This specification has not been verified for avoidance of possible third party proprietary rights. In implementing this specification, usual procedures to ensure the respect of possible third party intellectual property rights should be followed.

Technical Standard:

Systems Management: Software License Use Management (XSLM)

ISBN: 1-85912-270-1
Document Number: C806

Published in the U.K. by The Open Group, March 1999.

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

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

or by Electronic Mail to:

OGSpecs@opengroup.org

Contents

List of Figures

List of Tables

Preface

The Open Group

The Open Group is the leading vendor-neutral, international consortium for buyers and suppliers of technology. Its mission is to cause the development of a viable global information infrastructure that is ubiquitous, trusted, reliable, and as easy-to-use as the telephone. The essential functionality embedded in this infrastructure is what we term the *IT DialTone*. The Open Group creates an environment where all elements involved in technology development can cooperate to deliver less costly and more flexible IT solutions.

Formed in 1996 by the merger of the X/Open Company Ltd. (founded in 1984) and the Open Software Foundation (founded in 1988), The Open Group is supported by most of the world's largest user organizations, information systems vendors, and software suppliers. By combining the strengths of open systems specifications and a proven branding scheme with collaborative technology development and advanced research, The Open Group is well positioned to meet its new mission, as well as to assist user organizations, vendors, and suppliers in the development and implementation of products supporting the adoption and proliferation of systems which conform to standard specifications.

With more than 200 member companies, The Open Group helps the IT industry to advance technologically while managing the change caused by innovation. It does this by:

- Consolidating, prioritizing, and communicating customer requirements to vendors
- Conducting research and development with industry, academia, and government agencies to deliver innovation and economy through projects associated with its Research Institute
- Managing cost-effective development efforts that accelerate consistent multi-vendor deployment of technology in response to customer requirements
- Adopting, integrating, and publishing industry standard specifications that provide an essential set of blueprints for building open information systems and integrating new technology as it becomes available
- Licensing and promoting the Open Brand, represented by the ''X'' Device, that designates vendor products which conform to Open Group Product Standards
- Promoting the benefits of the IT DialTone to customers, vendors, and the public

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

Development of Product Standards

This process includes the identification of requirements for open systems and, now, the IT DialTone, development of Technical Standards (formerly CAE and Preliminary Specifications) through an industry consensus review and adoption procedure (in parallel with formal standards work), and the development of tests and conformance criteria.

This leads to the preparation of a Product Standard which is the name used for the documentation that records the conformance requirements (and other information) to which a vendor may register a product.

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

Open Group Publications

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

There are several types of specification:

- *Technical Standards* (formerly *CAE Specifications*)

  The Open Group Technical Standards form the basis for our Product Standards. These Standards are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement a Technical Standard can enjoy the benefits of a single, widely supported industry standard. Where appropriate, they can demonstrate product compliance through the Open Brand. Technical Standards are published as soon as they are developed, so enabling vendors to proceed with development of conformant products without delay.

- *CAE Specifications*

  CAE Specifications and Developers' Specifications published prior to January 1998 have the same status as Technical Standards (see above).

- *Preliminary Specifications*

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

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

- *Consortium and Technology Specifications*

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

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

In addition, The Open Group publishes:

- *Product Documentation*

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

- *Guides*

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

- *Technical Studies*

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

Versions and Issues of Specifications

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

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

Corrigenda

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

Ordering Information

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

This Document

The key factors driving the need for comprehensive license use management are escalating software costs, the high administrative burden of license compliance control, the lack of effective customer control over the use of their software, and asset protection. Customers and suppliers alike share these concerns, although customers are primarily interested in the first two, while suppliers tend to be more interested in the latter two. Being able easily and accurately to monitor the use of licensed software products in a business, and to relate the value of a licensed software product to its usage is a valuable measurement in enabling control and effective deployment of such products.

XSLM provides an interoperable solution for managing software licensing, accommodating diverse license management schemes and license certificate formats, in different operating environments. It does this through use of a logically centralized software license use management system. This enables a supplier to ship a single product that will operate under any license use management scheme that a customer may elect to install.

Document Structure

- **Chapter 1** explains the context and business reqirements for management of the use of software licenses.
- **Chapter 2** describes the licensing process, its workflow, and license types.
- **Chapter 3** describes the roles of the Application Broker and the Management Agent in the process.
- **Chapter 4** addresses security issues and features that are provided in XSLM for authentication and data integrity.
- **Chapter 5** defines the data types and data elements used in XSLM.
- **Chapter 6** explains the license certificate format.
- **Chapter 7** defines the API function calls, both in the "basic" set and in the "advanced" set.
- **Chapter 8** defines the Management API function calls.
- **Chapter 9** describes XSLM's recording and logging facilities.
- **Chapter 10** describes all the data elements defined in XSLM.
- **Appendixes A-D** give information on license types, implementation guidelines, functional sets and towers, and future directions.
- **Appendix E** gives the standard Java bindings for both the basic and the advanced XSLM API.

Typographical Conventions

The following typographical conventions are used throughout this document:

- **Bold** font is used in text for options to commands, filenames, keywords, type names, data structures and their members.
- *Italic* strings are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote:
  - — Command operands, command option-arguments, or variable names; for example, substitutable argument prototypes
  - — Environment variables, which are also shown in capitals

  - — Utility names
  - — External variables, such as *errno*
  - — Functions; these are shown as follows: *name*( ); names without parentheses are C external variables, C function family names, utility names, command operands, or command option-arguments.
- Normal font is used for the names of constants and literals.
- The notation **<file.h>** indicates a header.
- Syntax, code examples and user input in interactive examples are shown in `fixed width` font. Brackets shown in this font, `[ ]`, are part of the syntax and do *not* indicate optional items.

Trademarks

Motif®, OSF/1®, UNIX®, and the ‘‘X Device’’® are registered trademarks and IT DialTone™ and The Open Group™ are trademarks of The Open Group in the U.S. and other countries.

Acknowledgements

The Open Group gratefully acknowledges the work of the Distributed Systems Management Program Group in the development of this specification.

Many members of the Open Group's Management Working Group, and of GUIDE Inyternational, have contributed to this specification, either by providing input material or by reviewing drafts. In particular, thanks are due to:

Jim Lackey	BC Government / ISM-BC
Ron Higgin	Boole & Babbage
Wynn Pope	Burlington Industries
Andy Gordon	Compuware Corporation
Arnie Adelman	Gradient Technologies
Tom Cierech	IBM Corporation
Ron Falciani	IBM Corporation
Carlo Romano	IBM Corporation
Nancy Saipe	IBM Corporation
Paolo Squartini	IBM Corporation
Vittore Casarosa	Isogon Corporation
Per Hellberg	Isogon Corporation
Adam Bringhurst	Novell, Inc.
Jim Olsen	Novell, Inc.
Jim Keithley	National Computer Services

The X / Open Technical Specification Working Group would also like to acknowledge the valuable input provided by:

Teri Bahr	AT&T
Rolf Suter	Bell South
Denis Leghorn	CIGNA
David Chase	Digital Equipment Corporation
Dennis Deutsh	First Bank
Chris Germann	Gartner Group
Mary Welch	Gartner Group
Chris Davis	Texas Instruments

Referenced Documents

The following documents are referenced in this Technical Standard:

GUIDE LUM Requirements
: A Requirement for Software License Use Management; created by the License Use Management Project of GUIDE International Corporation; August 29, 1996.

IEEE Std 754
: ANSI / IEEE Std 754-1985: Standard for Binary Floating-Point Arithmetic.

LCF
: NetWare License Certificate Format; Novell, Inc.; December 18, 1995.

LM Standardization
: License Management Standardization; The Open Group Systems Management Task Group; July 31, 1992.

LSAPI
: License Service Application Programming Interface (LSAPI) Specification V1.1, published by Microsoft and others (available on MSDN CD under *Platform SDK | Setup and Systems Management Services | License Service Application Programming Interface*, January 28, 1993.

LUM Requirements
: Requirements for the Software License Management Work Group; Systems Management Workgroup; UNIX International System Management Workgroup; Revision 3; July 23, 1992.

Chapter 1

Introduction

The Software License Use Management (XSLM) specification provides a mechanism for addressing interoperability problems among different license management systems, license certificate formats, and operating environments. XSLM also provides a mechanism for the creation of user-oriented tools to aid in the management of software licenses and application use monitoring.

This chapter provides an introduction to the concepts of software license use management and introduces the XSLM specification components.

1.1 Business Requirements

The key factors driving the need for comprehensive license use management are escalating software costs, the high administrative burden of license compliance control, the lack of effective customer control of software usage, and the lack of adequate protection for software publishers.

Customers must deal with multiple products, from multiple software publishers, on multiple platforms, with multiple licensing models. Given this exponential growth in complexity, there is a clear requirement for an overall framework for license use management that is:

- Extensible, flexible and comprehensive
- Independent of software publisher
- Independent of platform
- Independent of standalone or connected operations
- Independent of implementation
- Adaptable to future technologies
- Meeting the needs of both customers and software publishers
- Cost effective for both customers and software publishers.

The primary goals of this specification are to:

- Provide a consistent and standard means for the management of software licensing
- Facilitate the availability of more flexible licensing terms
- Enable cost-effective license management and control
- Gather and provide access to licensing data in a heterogeneous and configuration independent environment.

License use management tools, processes, products, and systems must:

- Provide facilities to encode license terms and conditions
- Record and report use level data
- Determine, report on, and verify compliance to license terms

- Allow customers to control and optimize the use of licenses within the terms and conditions of the license policy
- Allow software publishers to ensure their assets are protected.

1.2 Implementation

There are currently thousands of software products in use that rely upon technical license managers (TLM) to ensure compliance with license terms and conditions.

The existence and widespread use of TLMs must be acknowledged, and in order to become generally accepted, any license use management system such as described in this document must allow for coexistence and an active integration with TLMs.

Today, a technical license manager, in general, does not provide extensive support for license use management as described in this document, especially on an enterprise-wide level. For example, a TLM often does not provide:

- The ability to run products from multiple software publishers using a single TLM implementation
- The ability for the customer to manage all licenses (using applications from different software publishers, each of which may be using different TLMs), in a consistent manner
- The ability for the customer to accept licensing terms from all software publishers in a common format that can readily be understood and verified
- The ability to provide usage feedback to the customer and the software publisher in a consistent manner for all installed products
- The ability to choose a license use management system that best fits a customer's particular needs, independent of the TLM the software publisher may have chosen
- The ability for the software publisher to deliver a solution that runs in a predictable manner, no matter which license use management system a particular customer has chosen.

An XSLM-compliant licensing system may, of course, provide more function than is specified in this document; this ability will serve as a way for licensing system publishers to differentiate themselves, thereby providing customer choice.

Appendix B on page 185 provides additional detail about implementation-specific areas.

1.3 Scope

This specification defines the functional components of a complete license use management system, and the rules by which those components interact with one another and with programs lying outside the framework domain (that is, programs that request XSLM services). The framework does not specify how the XSLM components are internally designed or implemented.

In general, the XSLM specification provides a vehicle for the implementation of the policy, not the enforcement of it. That is, the system provides an application with sufficient information about the policy to let it make an informed decision about whether or not to allow itself to be used. However, it is up to the application (and, thus, the software publisher) to enforce the license policy; the XSLM-compliant system should be viewed as a trusted vehicle by software publishers and customers, providing a repository for usage data supplied by the applications.

The XSLM-compliant system will manage licenses and license use policy information across multiple platforms in a consistent manner. The customer will be able to view policy information and administer software licenses from a single vantagepoint, even in cases of an application running on various platforms or multiple servers.

The key issue addressed by the license use management specification is consistency from the points of view of the application and manager clients, rather than that of the end-user. Achieving such consistency demands that licensing system publishers adhere to a strict set of interface rules. The specific solution implementation is irrelevant, providing that it adheres to a common, software publisher-independent set of interface protocols that preserves the clients black box view of the system. Adherence to this consistency will enable software publishers and customers to achieve the goals described in the Business Requirements section of this document.

During the development of this specification there have been many functions discussed and evaluated, some of which, while important, are not critical to the first release of the specification. Some of these functions, such as component licensing, network computing and server-to-server communications are to be included in the next release. Appendix D on page 193 discusses these functions in more detail.

1.4 XSLM Specification Overview

This specification establishes the points at which a program may request services of and receive responses from a licensing system component, and the external protocols for doing so. A licensing system component implementation that conforms to, and enforces, these external protocols complies with the licensing system specification, and is therefore said to be XSLM-compliant (or just compliant).

This specification facilitates coexistence among multiple TLMs from the same or different software publishers on the same or different platforms, and allows for emerging technologies. Compliance with this specification will ensure software publishers that their intellectual properties are protected against misuse.

The XSLM-compliant system provides for a separation of data between different software publishers. Therefore, the specification requires that each publisher of an application has a unique identification. Each software publisher must assign its own product-codes or product-numbers within the publisher-code. Once the terms and conditions of a license have been agreed upon between a software publisher and a customer, those terms and conditions are provided to the XSLM-compliant system so that they can be used to issue licenses.

The XSLM specification is extensible so that it can be implemented on virtually any kind of hardware or software and can encode virtually any kind of terms and conditions. This document includes a list (see Appendix A on page 175) of several license types, such as maximum number of concurrent users, that XSLM-compliant licensing systems must support. The XSLM-compliant system does not restrict a software publisher and a customer from negotiating any unusual license, such as the application can only be used on the first and third Monday of each month, by providing an escape mechanism, under which the policy data can be made available directly to the application, complementing any rule-based decision-making normally done by the XSLM-compliant system. The extensibility features of the specification will provide protection for software publishers in the evolving world of property-right legislation and enforcement.

Briefly, the XSLM software license management process is:

1. The customer and the application software publisher agree to license terms and conditions.
2. The application software publisher translates these terms and conditions into machine readable policies embedded in a license certificate, which is delivered to the customer along with the application program, into which are embedded calls to the XSLM licensing system.
3. The licensing system supplies the logic to interpret the license certificate, the logic to determine whether an application request to execute should be accepted, and a secure secret-sharing mechanism to ensure confidence in the validity of both the request and the response.

This combination of the license certificate, the enabled application and the licensing system provides the customer with:

- Transparency to the end-user of the product
- The ability to monitor product usage patterns
- The capability to maintain adherence to the terms and conditions.

1.5 Main Specification Components

The specification addresses four areas, all of vital concern in a software license use management system:

- A common license certificate format that makes it possible to define the licensing terms and conditions in an environment-independent way
- An application API that is used by applications which require the licensing capabilities provided by an XSLM-compliant system
- A management API that is used by XSLM applications which implement such functions as installing, updating, and deleting license certificates, and extracting usage data for reporting
- A recording and logging definition that specifies the minimum data that must be recorded by an XSLM-compliant system.

This specification defines the API syntax and semantics. It does not specify the method used to implement the API functions. This is left to the discretion of the licensing system publisher.

This specification does not specify the communications protocol to be used between an API implementation and its corresponding licensing server implementation.

1.5.1 A Common License Certificate Format

A central part of the XSLM specification is the concept of a license certificate specified in a common license certificate format. A license certificate is the encoding, into a standard format, of the terms and conditions that are contained within a license agreement governing the use of a particular product. This holds true, no matter how the product is licensed, whether via a direct negotiation between software publisher and customer (in which case there often exists a license agreement specific to a particular transaction), or as a shrink-wrap license.

The common license certificate format is such that an external license certificate can be created on a platform different from that where the license certificate will be installed (that is, the location of the license server), and different from the platform where the licensed application actually will be running. There is no requirement that the tools used to create a license certificate

and to install it into a license server are provided by the same licensing system software publisher. The only requirement is that the tools used by both licensing system software publishers and customers can create and accept the same external representation, that is the common license certificate format.

A license certificate will contain sufficient security and integrity data (such as digital signatures) to ensure that tampering is detectable by the XSLM system, or directly by the application when it requests a license.

Chapter 6 specifies the format of a license certificate and explains how application publishers can define their own, private data elements. Chapter 10 contains a list of all predefined data elements, both required and optional.

1.5.2 The Application API

The Application API (XAAPI) defines the interface that any licensing system-enabled application would use to interact with an XSLM-compliant license server, to include license verification and to handle all the activities associated with it. The intent is to standardize the way in which common functionality provided by existing licensing systems can be accessed. These API functions allow software publishers to enable applications in a way that is independent of the underlying (XSLM-compliant) licensing system.

Although there may be many physical servers, the application product communicates with just a single, logical license server that is embodied in the API library. Application products direct requests to a library that provides the XAAPI interface, not directly to a physical server.

The main goals for the XAAPI are to:

- Define a standard API, to be adopted by a variety of software publishers, that any application product may use on different operating systems and network environments
- Be easy to use
- Provide support for all commonly used licensing policies
- Provide support for different levels of application complexity and accommodate different levels of protection against licensing system tampering.

The XAAPI functions fall into two sets: a basic API and an advanced API. The basic set includes the minimum function required to provide support for a basic, yet complete, licensing activity. The advanced set includes and extends the functionality of the basic set. It provides the application developer with more flexibility and options in support of a more complex licensing scheme.

1.5.3 The Management API

The Management API (XMAPI) is the component that enables license use management systems provided by multiple software publishers to be managed as a single logical licensing system.

The XMAPI provides license management applications with an implementation independent means of:

- Installing and removing software license certificates
- Retrieving information about installed software license certificates, software license usage, and selected licensing system properties
- Updating the properties of installed software licenses.

This specification does not prescribe the physical structure of the data, or the means by which an implementation chooses to store the data it collects and maintains. It does, however, define the logical structure of the data passed to, and received from a license server, via the XMAPI. This logical (self-defining) data structure is defined in Chapter 6. This specification also defines the persistency requirements for data transmitted to the licensing system across selected management API functions.

This self-defining data architecture provides for additional implementation specific data to be transmitted across the XMAPI. To permit this flexibility while maintaining compatibility with other compliant licensing systems, XSLM mandates that each implementation ignores any data received across the API that does not carry that implementations unique publisher identification.

1.5.4 The Recording and Logging Services

A crucial function of an XSLM licensing system is to collect and record data about the usage of the licensed products and about relevant events related to license management. A compliant XSLM system will maintain three types of information: certificate data, status data, and historic (or logged) data.

The certificate data is the combination of information provided by the application software publisher (embodied in the license certificate), which cannot be modified by the licensing system or by the administrator; information provided by the customer's license administrator (to complement or override, when allowed, the licensing policy specified in the license certificate); and information created and maintained by the licensing system itself.

The status data is maintained by the licensing system while it is running, and at each point in time it provides information about the licenses presently in use, and the value of the meters maintained by the licensing system. Some applications can be licensed based on the count of some units whose meaning in general is known only to the application, and the licensing system keeps track of the units to be counted, acting as a "record keeper". The updating of the meter will be explicitly requested by the application with an API call. A change in the status information is triggered by external (to the licensing system) events, such as the request for a new license, or a change in policy setting (e.g. the administrator switching from soft stop to hard stop) or the expiration of a timer.

The historic data is the persistent log of events relevant to license management. All events related to license administration actions will always be logged, as they can constitute an audit trail (e.g. the addition or deletion of a certificate to/from the license database). The logging of events related to license usage (e.g. an application requesting or releasing a license, or a meter being updated) will usually be under administrator's control.

1.6 A Logical View of the Specification

Conceptually, this specification describes a single, logical, licensing system running on a single computing system. This logical view is independent of how a particular system is implemented. An actual implementation may provide capabilities for defining multiple, logical, and customer-defined domains that can be controlled independently of each other.

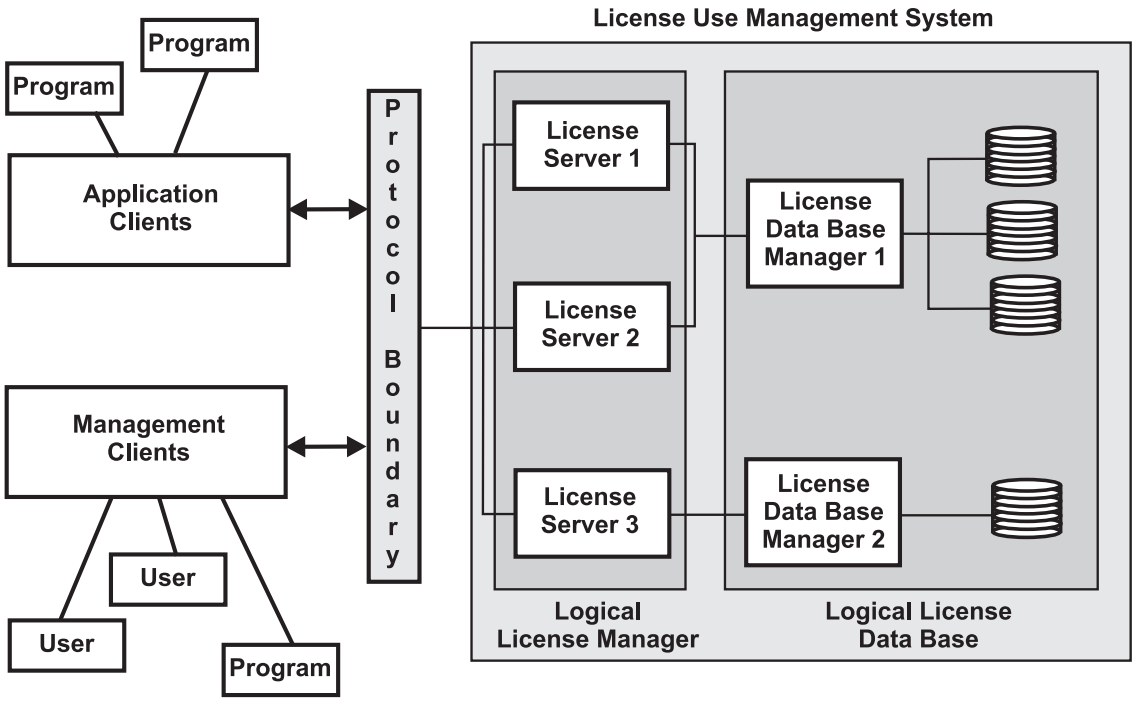


Figure 1-1 Logical View of a License Use Management System

1.7 Coexistence and Integration with Technical License Managers

There are several existing software technical license managers (TLMs) in common use today, each of which is based on a proprietary set of protocols defined solely by the TLM publisher. It is an XSLM goal to make it practical for these existing proprietary license managers to be adapted to form compliant implementations of this specification.

XSLM compliant licensing systems can always coexist with other XSLM and non-XSLM compliant licensing systems running on the same computer system, although there are no practical license management related benefits realized by doing so.

1.7.1 Providing XSLM Compliance

XSLM provides for several levels of compliance.

Legacy Functional Level compliance requires support for the Basic XMAPI API and the Basic XCLA data architecture — the Base Function Set[1]. A Legacy Functional Level compliant license manager is visible to XSLM-compliant license use management tools. This enables users to view license use information in an enterprise-wide systems management context; that is, as part of a single logical licensing system comprised of (potentially) multiple physical license servers from (potentially) multiple license system publishers.

Additional compliance levels require support for one or more of the documented optional functional towers, as defined in Appendix C on page 189.

1.7.2 Special Notes for LSAPI-Enabled Licensing Systems

Some licensing systems in common use today provide support for the licensing system protocols and APIs described by the LSAPI 1.1 specification (see referenced document **LSAPI**). This specification addresses the basic application requirement to acquire, validate, and release a software license through one of an unknown (to the application) set of licensing system servers.

The basic XSLM API is defined so as to make it possible for any TLM that has implemented the LSAPI specification to be easily transformed, at the application API level, into one providing equivalent functionality that is XSLM-compliant. However, to be labeled XSLM-compliant an existing licensing system must implement, in addition to the basic application API set, the appropriate XSLM management API set.

1.7.3 Providing Non-XSLM Defined Functionality

Licensing systems electing to operate at the Advanced compliance level can reasonably expect to represent most, if not all, existing TLM functionality in terms of XSLM defined APIs and related logical data structures. However, it is acknowledged that application publishers may require specific functionality offered by an existing proprietary TLM, but where it is not appropriate to define that functionality as part of a general license use management specification.

The XSLM specification addresses the potential requirement for implementation specific function and related data by providing for licensing system specific extensions. An application that depends on such extensions will be restricted to using only compliant licensing systems that implement those extensions. Even in the case where one or more applications require such extensions there are customer benefits to XSLM compliance; the principal benefit being the

1. See Appendix C on page 189 for additional information.

ability to generate, distribute, install, and manage standard XSLM-defined license certificates and related licensing system resources.

An XSLM-defined license certificate may contain licensing system specific information. This information is encoded in the form of an architected licensing system publisher specific section of the certificate, identifying that specific publisher. The licensing system publisher data itself appears within the licensing system section and is defined information encapsulated in XSLM-format license certificate data elements. The presence of a licensing system publisher specific section restricts the license certificate to being installed only in a license system server whose published id matches that identified in the license certificate. When multiple licensing system publisher sections appear in a certificate, the certificate may be installed in a license server whose id matches any one of those contained in that certificate.

An application publisher can thus distribute a standard XSLM-defined license certificate that conveys licensing system specific information to one or more specific licensing system server implementations. This in turn provides an existing TLM with the ability to offer extended (not defined by XSLM) services to those applications that require them, yet remain XSLM compliant. Similarly the XSLM defined data architecture provides existing TLMs with the means to expose implementation specific software license management data through the standard XSLM-defined management APIs.

Chapter 2

Licensing Workflow and License Types

There are always at least two, and often more, independent entities—software publisher and customer—involved in the process of licensing software, and it is important that these processes are clearly understood in order to fully comprehend the issues involved with software license use management.

This chapter provides an overview of the software licensing process and underlying license types as seen by the software publisher, the customer, and the licensing system publisher. A workflow model is used as the means of introducing the activities required to both enable and use a compliant software license use management system. This model is used to validate that this specification addresses the requirements of all the parties involved.

2.1 The Licensing Process

2.1.1 Application Software Publisher's View

In this context, an application software publisher is defined as an individual or company that both develops and licenses one or more commercially available software products, owns the software and generally sets the terms and conditions of the licensing on that software. The application software publisher may also manufacture, market and distribute the software.

Application software publishers have a need to protect their software assets. At the same time, they need the ability to be as flexible as possible when negotiating licensing terms and conditions with their customers. Figure 2-1 illustrates the license use management workflow and tasks from the application software publishers perspective.

A responsibility of the application software publisher is to decide which sales models are to be used and to translate these models, in conjunction with the negotiated terms and conditions of the business contracts, into machine-readable policies called license certificates.

By itself, the license certificate does not protect the publisher's software assets. Nor does it provide a useful management tool for the customer. The application software publisher enables these capabilities allowing the application to function according to an agreed upon contract by embedding calls to the XSLM-compliant licensing system as part of the product program logic.

The XSLM specification is comprehensive enough to satisfy the needs of most application software publishers, but does not include all the functions provided by all existing licensing systems. Consequently, an XSLM-compliant license use management system product may include non-XSLM functions. These additional capabilities are referred to as publisher-specific functions.

Use of publisher-specific functions by application software publishers will limit compatibility with XSLM-compliant licensing systems.

Once an application software publisher has enabled a product to work with an XSLM-compliant licensing system, a reliable product distribution and installation process must be developed that delivers the product code, a customer-specific (or generic) XSLM license certificate and, sometimes, an XSLM licensing system.

Finally, the application software publisher bears primary responsibility for providing customer support. If, for any reason, the XSLM-compliant licensing system rejects a publisher-provided license certificate, or incorrectly denies a customer access to the application software publishers product, the application software publisher must be able to quickly identify the source of the problem, regardless of which XSLM-compliant licensing system is in use at the customer's site and, in resolving the problem, potentially work directly with the XSLM-compliant licensing system publisher.

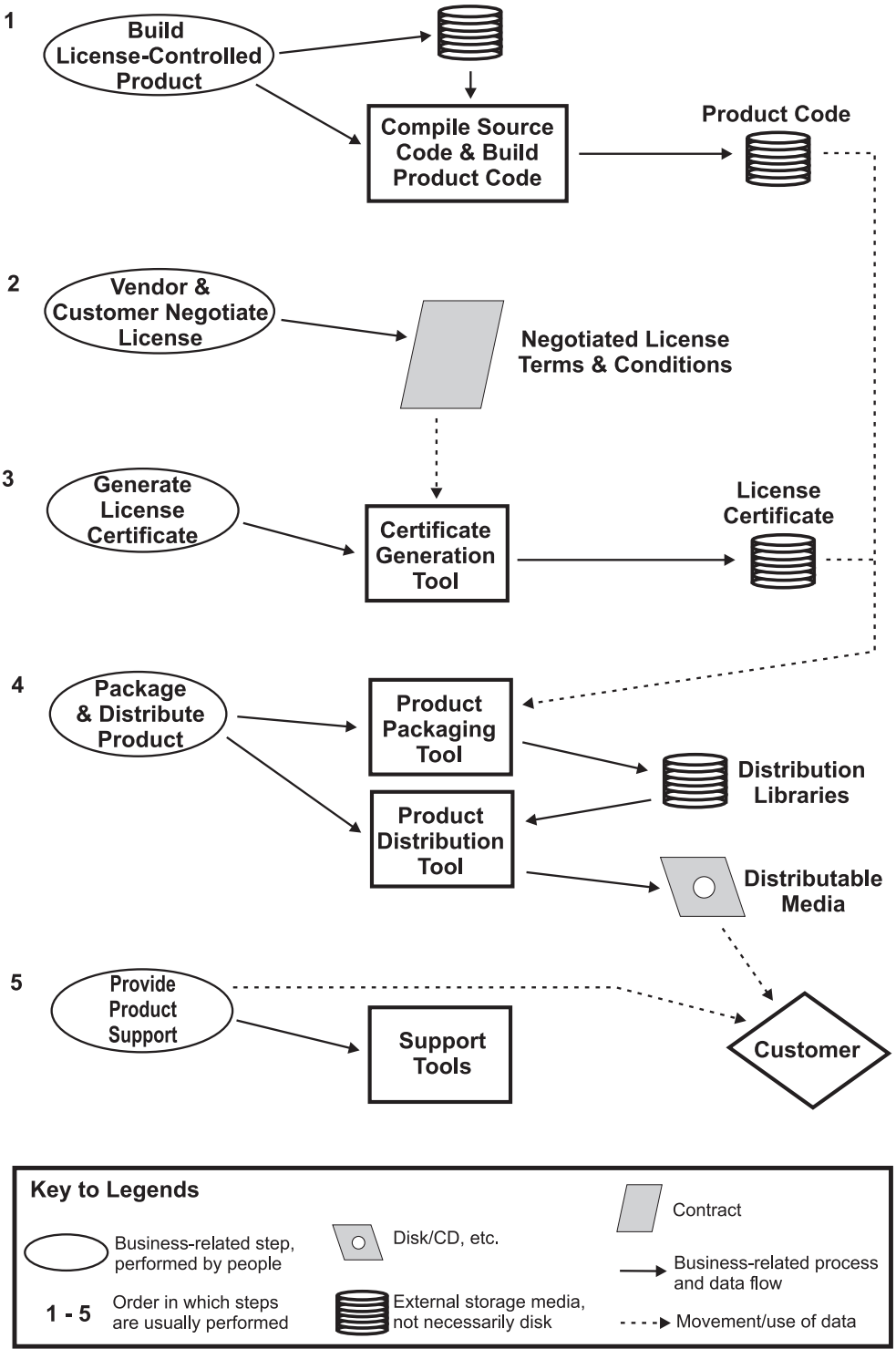


Figure 2-1 Licensing Workflow from the Publishers Perspective

2.1.2 Customer's View

The customer is the named licensee: a person or entity that acquires software. In this context, a customer is the person or persons responsible for installing and maintaining the XSLM-compliant licensing system and the application software publisher-provided products as well as monitoring use of those products to ensure adherence to pre-negotiated terms and conditions. Generally, the administrator within the customer's organization would handle these functions.

Customers will have a standard, easy to use method of allowing the use of products while maintaining strict adherence to the terms and conditions. Customers also will have tools that provide them with information that allows them to diagnose license use problems, and to monitor and determine patterns of license and product use. A properly implemented license use management system will be transparent to the end-user of the products whose licenses it manages, with the possible exception of allowing the application to tell the user when no license is available.

Figure 2-2 on page 15 illustrates the license use management workflow and tasks from the customer's perspective.

In order to use a product, a customer must first be able to install it. Installation consists of loading the product on the customer's system, and in installing the associated license certificate (perhaps as simple as loading a file into a directory) into an already installed XSLM licensing system. Therefore, an XSLM licensing system accepts a valid license certificate from a software publisher provided product installation tool, loading the license certificate into the licensing system as part of the normal product installation process.

When explicitly permitted by the license certificate, a customer will have the ability to assign licenses subject to the terms and conditions embodied in that license certificate. In other words, customers will have the ability to make the license policy for a given product more restrictive than the terms and conditions of the license agreement.

In addition to proactive license use data analysis, customers will be able to integrate license use management into their existing automation processes. This means, for many customers, choosing a licensing system that provides alert information which can be directed, via standard instrumentation interfaces, to one or more installed automation systems. Alerts in this context refer to informational reports such as license not available, license about to expire, licensing system terminated, and the like. In an ideal situation, the customer will be able to configure the automation system to automatically respond to most common alerts. For example, a license about to expire might result in an automatically generated electronic mail directed to the department responsible for negotiating/renewing license agreements.

Finally, in a "run time" context, a customer will be able to operate in a disaster recovery scenario. Many customers have off site computing facilities to be used in the event of a disaster where their primary computing resources are unusable. A license use management system should not prevent the customer from conducting business in a disaster recovery (real or test) situation, allowing grace periods or alternate-server capabilities, for instance.

A compliant licensing system will maintain a machine-readable log of significant licensing events, for example, license shortages. The customer will be required to manage one or more log files (for example, to archive a log file). During the normal course of use, customers need to be able to detect license shortages, to dynamically adjust license use policy (when permitted), and to diagnose problems where the license-use management system denies a user product access for reasons which are not evident. The customer will also need, on occasion, to communicate with the licensing system to perform unusual recovery procedures such as reclaiming a license known by the customer to be reclaimable.

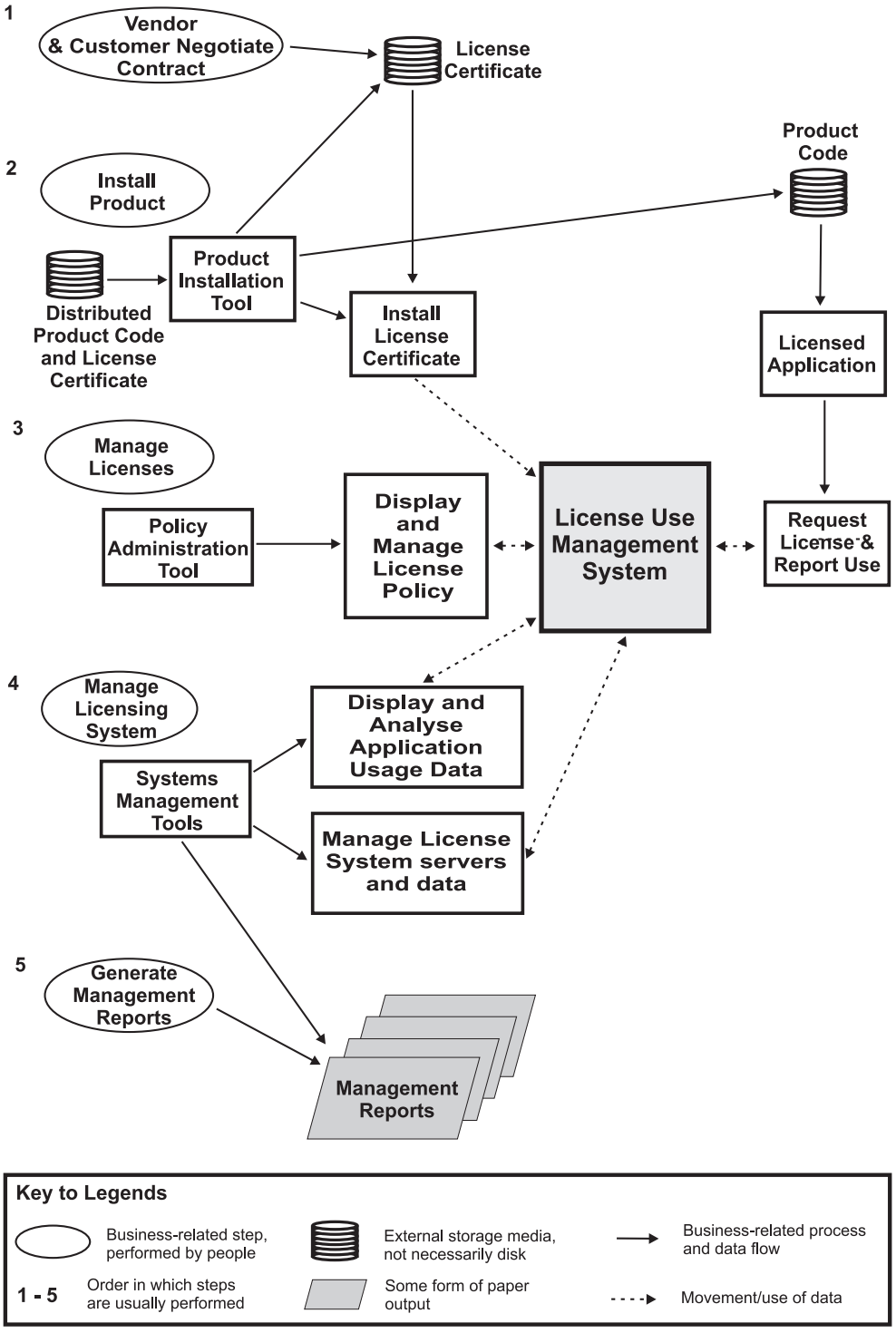


Figure 2-2 Licensing Workflow from the Customers Perspective

Customers will have the ability to view product license information, extract that information to external media (for safe keeping and/or off-line analysis), and to generate real-time or batch reports on historical and current license use. This information will be used by the customer to

analyze usage patterns for purposes such as determining the need to acquire additional licenses, detecting products which are no longer being used, and providing statistical data to be used when negotiating new license agreements.

2.1.3 Licensing System Publisher's View

The third partner in this scenario is the XSLM licensing system publisher: the provider of the product or function which accumulates the information from the calls embedded in the application product, compares it to the terms and conditions evident in the license certificate, and responds with direction to the application product.

the XSLM licensing system must consider and resolve platform dependencies. For instance, not all operating systems provide access control, and signal alarms are not common across all platforms.

Therefore, the specification clearly defines what functions the XSLM licensing system must accommodate.

An XSLM-compliant licensing system may provide more function than is defined as the basic set; for instance, an enhanced reporting capability or alternative common logging facilities may be provided. The specific implementation of most required management functions is not defined only the requirement that these functions must be available.

2.2 License Types

As mentioned earlier in the Licensing Process section, sales models along with terms and conditions are translated into license certificates. A subset of the license certificate contains the license type, which is defined as the scope of the use of a specific product. It specifies the restrictions that are defined in the agreement between the customer and the software publisher. This section defines the minimum number of license types which have to be supported in order to allow the implementation of the wide variety of licensing terms and conditions (many of which are commonly known). The license types are listed in Appendix A on page 175.

In the technical implementation of these specifications, a license system publisher may view these unique license types, which define the behavior of the licensing system, as either reusable or non-reusable, and modifiable by time, capacity, count and/or naming conventions. Reusable or non-reusable can also be modifiers. It is more useful in this document to define these license types in terms more commonly used in the customer community as follows:

- BASIC (Unrestricted)
- CAPACITY
- CONCURRENT
- CONSUMPTIVE
- CUMULATIVE
- NAMED

Each license type is modifiable by time. For instance, a DEMO implementation option might be a BASIC license that is useful only for 30 days from the date of installation or until a defined expiration date. These time attributes might be "start date/time," "end date/time," or "duration," for example.

A reusable license is one that, when its no longer required, is returned to the licensing system and becomes available for re-issuance against another license request; in contrast a non- reusable license type is one that once used, or counted, is not retrievable or reusable.

BASIC

This license type is the base line. It represents a license for which there are no restrictions. In contrast, all the other license types define restrictions within which the application is licensed and the customer is to abide.

CAPACITY

This license type compares the capacity of the operating environment against a predefined table, for instance, to assure the application is running in a machine whose computing capacity is not larger than that for which the product is licensed.

CONCURRENT

This is a license type for which the charges are based on counting the number of simultaneous demands or uses of a product, independent of who or what user is using the application quite the opposite from the Named concept. Further, these license uses are reusable: when the license use is no longer required it is returned to the licensing system and becomes available for re-issuance against another license request. The number and defined unit of measure may include a minimum or maximum number permitted per request. For instance, a Concurrent license may require that whenever a license request is made, five units of the defined measure (users, for instance), must be requested as a minimum.

CONSUMPTIVE

This is a license type for which the charges are based on counting the defined units executed, perhaps over a specified period of time, against those licensed. Of principal importance with this license type is that a license count once used is not retrievable or reusable. As with Concurrent licensing, the number and defined unit of measure may include a minimum or maximum number permitted per request. For instance, a Consumptive license may require that whenever a license request is made, five units of the defined measure (blocks of time or gigabytes of storage, for instance), must be requested as a minimum. This license type might be useful in a peak use situation.

CUMULATIVE

This is a license type for which the charges are based on counting a defined unit of measure against the number of units of that measure which were licensed. While Cumulative licensing merely accumulates the defined units of measure, as with Consumptive licensing, once used these units are not retrievable, or reusable. This license type might be useful in a post-pay term and condition.

NAMED

This is a license type which compares name or serial number or ID or node address (for example) against those licensed. The Named license type implies pre-definition of the name. However, to build the registered or named "authorization list," the Named license type can also allow for a "first come-first served" concept where license requesting users (for instance) are registered (accepted / defined) until the number of users licensed is reached.

Chapter 3

Processing Flow and Linkages

This chapter describes the processing flow and linkages between an application and a licensing system, and between a management application and the licensing servers that together constitute the licensing system.

3.1 Application Broker

The Application Broker is, in effect, a routing function. It serves only to render the presence of multiple licensing system implementations (from one or more licensing system vendors) transparent to application clients; that is, to create a single logical license management system in those situations where the actual licensing system is comprised of two or more XSLM-compliant licensing system implementations.

Note that where an implementation provides for the concurrent existence of multiple physical license servers, the responsibility for providing server transparency (that is, server topology) lies with that specific implementation[2]. In other words, the Broker provides implementation rather than server transparency with respect to those application clients served by the Broker.

As illustrated in Figure 3-1 on page 20, the Application Broker is positioned between the application client and one or more XSLM-compliant license use management system implementations. The numbered steps in the figure indicate processing order. The primary functions of the Application Broker are to:

- Locate the Application Agents (DLMs) for each XSLM-compliant licensing system implementation defined to the Broker[3].
- Route application client service requests to one or more implementation Agents.
- Logically propagate implementation-Agent provided responses to "routed" service requests to the application client from which those requests were received. Note that for certain XAAPI functions (for example, *xslm_query_api_level*( )), the Broker is required to consolidate responses received from multiple implementation Agents to yield a single response, which is then delivered to the application client.

A licensing system implementation is exposed to the Broker in the form of an implementation dependent Application Agent. This is discussed further in Section 3.1.1 on page 20.

The Application Broker DLM must be named "XSLM_Broker" and must export the names of all XSLM application API (XAAPI) functions. This provides application clients with a means of accessing all supported services without requiring knowledge of the number or identity of the specific licensing system implementation(s) that are providing those services.

It is expected that licensing system publishers will assume responsibility for developing and delivering at least one Application Broker. However, this does not preclude other software publishers from delivering alternative Broker implementations providing additional features.

---

2. This Technical Standard does not detail the requirements for the server-to-server interactions required to provide transparency between servers from different licensing system publishers. See Appendix D for more information.
3. It is possible in some environments that a broker must be "refreshed" in order to recognize a newly-added Application Agent.

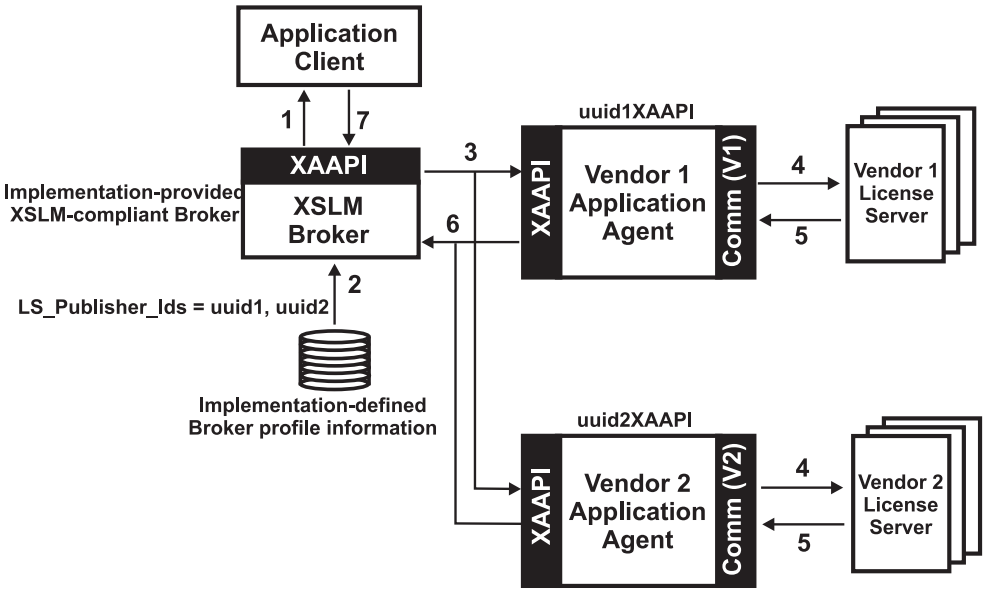


Figure 3-1 Application API Architecture

3.1.1 Application Agent

The Application Agent itself provides the Application Broker and (when dynamic client-to-Agent binding is employed) application clients with a means of accessing all supported XSLM-defined application services without requiring knowledge of the methods used to implement those services; that is, without knowing which implementation is being used, how function is distributed between the agent and its corresponding server(s), or which protocol is being used to effect physical communication between the agent and its server(s).

As illustrated in Figure 3-1, each Application Agent implementation must physically consist of at least one dynamically loadable module (DLM).

The Application Agent must export the names of all XSLM application (XAAPI) functions. This provides the means by which the Application Broker can bind to the implementation Agent.

An Application Client can:

- Dynamically or statically bind to the Application Broker.
- Directly (but dynamically) bind to the implementation Agent as an alternative to using an Application Broker. This capability is useful in operating environments where only a single compliant licensing system implementation is present and/or required, and thus where binding the Agent to an Application Broker would necessarily result in a performance penalty without offering any functional benefit.
- Be statically bound to a specific implementation Agent. This capability could be used, for example, to provide a software publisher with additional, but implementation specific, authentication features, albeit at the expense of a loss of interoperability with respect to alternative compliant licensing system implementations.

3.1.2 Application-to-Licensing System Communications

XSLM does not define the protocol over which an application client communicates with an XSLM-compliant licensing system.

The physical (wire level) communication protocol between XSLM-enabled application clients and license servers is implementation specific, yet transparent to the clients. This is accomplished by requiring clients to dynamically bind to, and direct all XSLM-defined application service requests (API calls) to, an implementation independent Application Broker (DLM) or, when appropriate, to an implementation dependent Application Agent.

Referring to the numbers in Figure 3-1 on page 20, the process flow between the application client and the licensing system server is as follows:

1. The Application Client calls a desired XSLM-defined function entry point within the Broker DLM. This is always possible since each licensing system implementation is required to provide a Broker DLM named "XSLM_Broker.DLM" that exports entry points for all XSLM- defined functions. The Application Client may be either statically bound (link edited with) or dynamically bound (at run time) to the Broker.
2. The Broker, via an implementation dependent method, locates profile (initialization) information that provides the Broker with the UUIDs of the licensing system publishers with which the Broker is to establish a dynamic binding. These UUIDs are used by the Broker to locate each licensing system's Application Agent (implementation) DLM. This is possible because each implementation is required to supply an Agent DLM named "uuid_XAAPI.DLM," where "uuid" is the implementation publisher's UUID.
3. The Broker, having located all required Application Agent DLMs, either invokes each implementation Agent (in an implementation determined order) until the first time an "OK" response is received, or the entire list of Agents as defined in the Broker profile information.
4. The implementation Agent (DLM) is invoked at the corresponding entry point to that at which the Broker was originally entered. The Agent prepares an implementation dependent representation of the function request and forwards it to a compatible license server belonging to the same publisher as the one that provided the Agent. XSLM does not specify the form of the function request representation, or the means by which that representation is communicated to, nor how the response to the represented request is received from, the licensing system server.
5. The license server receives the implementation specific representation of the XSLM function request over a licensing system (implementation) defined communications medium. The server processes the request, prepares an appropriate response, and sends the response back to the requesting Application Agent.
6. The Implementation Agent converts, as necessary, the server response information to a form compliant with that mandated by the requested XSLM function. The XSLM-compliant response data is then returned to the Broker function.
7. The Broker receives control back from the implementation Agent and either:
    a. Checks for an "OK" return, indicating that the requested function completed without error, and when received effects an immediate return to the requesting Application Client. In the event a "Not OK" return is received the Broker continues by invoking the same function in the Agent DLM for the next implementation (UUID) defined to the Broker.

or (for some functions):

b. Saves the information returned from the implementation, invokes the same function in the Agent DLM for the next implementation (UUID) defined to the Broker, saves the information returned by that Agent, and so on, until the requested function has been invoked for each defined implementation. At the conclusion of this process the Broker analyzes the responses, producing one consolidated response which is provided upon return to the requesting Application Client.

At this point the Application Client will have received control back from the Broker, along with an appropriate XSLM-defined return code and all output data defined for the requested XSLM function.

3.2 Management Agent

The Management Agent provides management clients with a means of accessing all supported XSLM-defined management services without requiring knowledge of the methods used to implement those services; that is, without knowing which implementation is being used, how function is distributed between the Agent and its corresponding server(s), or which protocol is being used to effect physical communication between the agent and its server(s).

As illustrated in Figure 3-2 on page 23, each conforming implementation must provide a Management Agent physically consisting of at least one dynamically loadable module). This DLM must be named "uuidXMAPI.DLM" where "uuid" is the publisher identification (UUID) for the licensing system that provides the DLM.

Each Management Agent must export the names of all XSLM management (XMAPI) functions. This provides the means by which a management client can dynamically bind to one or more Management Agents.

3.2.1 Communication Protocol

XSLM does not define the protocol over which a Management Client and a license server communicate. The physical (wire level) communication protocol between a given licensing system's Management Agent and corresponding license server(s) is implementation specific, and transparent to clients.

As illustrated in Figure 3-1 on page 20, each implementation's Management Agent must physically consist of at least one dynamically loadable module (DLM) that exports the names of all XSLM-defined management functions. This provides Management Clients with access to all XSLM-defined management functions without requiring client knowledge of the methods used to implement those functions; that is, without knowing how functionality is distributed between the Agent and its corresponding server(s), or what protocol is being used to effect physical communication between the Management Agent and the server(s).

There is no specified limit as to the number of Management Agents a Management Client may concurrently load. A client must, at a minimum, load an agent at least once for each licensing system implementation with which it intends to communicate.

Server-level addressability (XMAPI requests directed to a specific licensing system server) facilitates the creation of management tools that can provide a single point of control for a given logical licensing system comprised of multiple physical servers while maintaining an accurate topological view of the physical components of that licensing system.

The XMAPI provides for directed communication with any compliant license server through a unique communications handle provided by the Management Agent (DLM) associated with the

server implementation. The "XSLM_Query_Servers" function, directed to the implementation Management Agent, is used to obtain the communication handle for each active server for that implementation. A management application may subsequently direct XMAPI requests to a specific server running on the behalf of an implementation by invoking the desired XMAPI function (contained in the corresponding implementation Management Agent) and providing as an input parameter the communication handle for the desired target server.

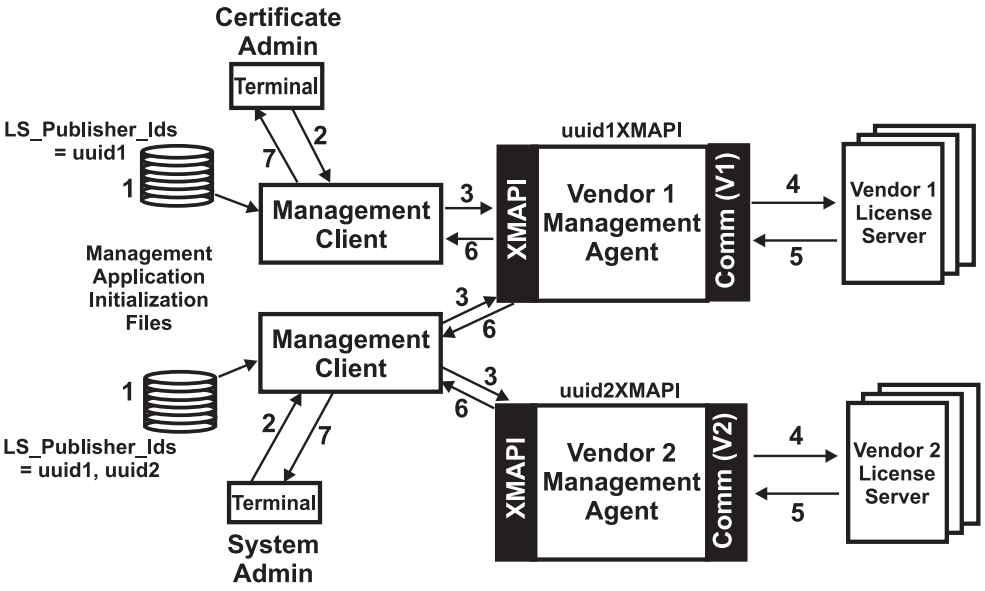


Figure 3-2 Management API Architecture

3.2.2 Management Client-to-Licensing System Communication

This specification does not define the protocol over which a Management Client communicates with an XSLM-compliant licensing system, or with the license servers that comprise it.

The physical (wire level) communication protocol between XSLM-enabled management clients and license servers is implementation specific, yet transparent to those clients. This is accomplished by requiring clients to dynamically bind to a unique implementation specific Management Agent for each licensing system implementation. This direct binding to multiple Agents is, unlike for Application Clients, required for Management Clients because management clients require complete knowledge of the physical topology of the logical licensing system.

Referring to the numbered steps in Figure 3-1, the process through which Management Client-to-licensing system server is effected is as follows:

1. The Management Client, via an implementation dependent method, locates its initialization information. This information includes a list of publisher UUIDs for the licensing system implementations with which the Client may establish a dynamic binding. These UUIDs are used by the Management Client to locate each licensing system's Management Agent (implementation) DLM. This is possible because each compliant implementation is required to supply an Agent DLM named "uuid_XMAPI.DLM", where "uuid" is the implementation publisher's UUID.

   The Client, having located all required Management Agent DLMs, invokes the "XSLM_Query_Servers" function for each Management Agent to obtain a list of the identities and attributes of the active license servers for each implementation. The output

of this function provides the Management Client with the information required to direct XSLM requests to each active server for a specific implementation. By consolidating the information obtained from the Management Agents for all specified implementations the Management Client is able to determine the identities and logical communication identifiers of all servers comprising the XSLM-compliant licensing system.

2. The Management Client receives an implementation specific management request from either a terminal user or management application program (tool).
3. The Management Client, based on the input received from the terminal user or program, calls the required XSLM-defined management function entry points contained within one or more Client selected Management Agent DLMs.
4. The Management Agent, upon receiving control at the Management Client-selected XSLM-defined function entry point, prepares a licensing system implementation-dependent representation of the function request and forwards it to a compatible license server belonging to the same publisher as the one that provided the Agent. XSLM does not specify the form of the function request representation, or the means by which that representation is communicated to, nor how the response to the represented request is received from, the licensing system server.
5. The license server receives the implementation specific representation of the XSLM management function request over a licensing system (implementation) defined communications medium. The server processes the request, prepares an appropriate response, and sends the response back to the requesting Management Agent.
6. The Management Agent converts, as necessary, the server response information to a form compliant with that mandated by the requested XSLM function. The XSLM-compliant response data is then returned to the Management Client.
7. The Management Client receives control back from, and processes the output data returned by the Management Agent. Depending on the terminal user or program request, the Management Client may choose to make subsequent calls to the same or to other Management Agent functions.

At this point the Management Client completes the terminal user or program request by preparing an appropriate (implementation dependent) response that is either delivered to (displayed upon) the user's terminal, or returned to the requesting (management) program (tool).

Chapter 4

Authentication and Data Integrity

This chapter defines the aspects of security that must be supported by a XSLM-compliant licensing system. The use of these security mechanisms is optional on the part of the application software publisher.

4.1 Scope

An XSLM-compliant licensing system provides the following security features:

- License certificate integrity
  to verify that a certificate submitted to the licensing system was not changed after its creation.
- License certificate authenticity
  to verify that a certificate submitted to the licensing system was created by the intended software application publisher.
- Licensing system authentication.
  to verify that the licensing system which an agent is communicating with is the intended licensing system.

4.2 Security Mechanisms Deployed

This XSLM Technical Standard relies on public-key technology to provide the integrity and authenticity features identified above.

There are a number of different public-key schemes that can be used. These algorithms share the common characteristic that keys exist in pairs, ''*k*'' and ''*K*''. Key *k* is kept secret by its owner and is called the *private key*, while key *K* is published and available to all interested parties and is called the *public key*. The operation called *signing* requires the possession of the private key and of the data being signed. The operation called *verification* requires the possession of the public key and of the signed data. If successful, the verification operation proves that the signature was created by an entity in possession of the private key at the time of signing, and that the data in question has not been altered. These signing and verification operations might involve the use of a cryptographically sound hash function, and the use of cryptographically sound encryption and decryption functions. The description and definition of these functions and operations is outside the scope of this Technical Standard. In XSLM, all other kinds of authentication, integrity, and security are the responsibility of the licensing system, and of the supporting environment in which the application executes.

The trust which can be placed in the verification operation resides in the trust which can be placed in the association between the public key and the owner of the private key who signed the data. This trust can be established by an external framework (referred to as Public Key Infrastructure, or PKI) in which one or more Certification Authorities (CA) can issue *authentication certificates* (or digital certificates), in which the association between a public key and the entity owning the corresponding private key is made. The authentication certificates are signed by the CA and then made available through public lists, so that anyone trusting the CA and in possession (in a trusted way) of the public key of the CA can verify the authenticity of the certificate and obtain, in a trusted way, the public key of the party of interest.

All of the above clearly relies on the basic assumption of security, that is, "the secret is secret", or in other words the private key is never compromised.

Additional aspects of the functionality needed for security are assumed to be provided by the licensing system in an implementation-dependent way, or by the environment in which the licensing system and/or the application execute. These are outside the scope of this Technical Standard.

4.3 License Certificate Integrity

The license certificate integrity (and authenticity) is assured by the application software publisher by signing the certificate at the time it is created. The certificate integrity (and authenticity) is verified by the licensing system at the time the certificate is installed in the license data base, by using the public key of the application software publisher which created the certificate.

It is assumed that the licensing system is a legitimate one and has not been compromised, that is, the binary code has not been patched in order to have a different behavior.

An optional data element on the license certificate, called *authentication section*, contains all the information needed by the licensing system for the verification operation. If the data element is not present, the licensing system is not required to perform any validation.

Three cases arise:

- The authentication section may contain an authorization certificate, issued by a CA trusted by the licensing system.

  In this case, the verification of the license certificate is done in two steps:

  1. The licensing system verifies the authorization certificate. How the licensing system acquires the public key of the CA is outside the scope of this specification.
  2. The licensing system verifies the license certificate with the public key of the application software publisher, which it has just found in the verified authorization certificate.

- The authentication section may contain the request that the licensing system should only validate the license certificate by using an authorization certificate issued by some trusted CA. How the authorization certificate and the CA public key are made available to the licensing system is outside the scope of this Technical Standard. There is however the requirement that the application software publisher information contained in the authentication certificate must be exactly the same information which is provided in the license certificate.

- The authentication section may contain only the public key of the application software publisher (that is, it does not contain any authorization certificate) which will be directly used by the licensing system to verify the certificate.

  This simple verification scheme can detect "naive" tampering with the certificate, but cannot detect the replacement of the entire certificate with a forged one created and signed by a malicious party, and therefore can not ensure that the license certificate is a valid one, issued by the intended software application publisher.

  It is recommended that this scheme is only used in conjunction with the verification of authenticity, described below.

4.4 License Certificate Authenticity

At run time, whenever a license is requested (by means of an *xslm_basic_request_license*( ) or *xslm_adv_request_license*( ) API call), an application can dictate the behavior of the licensing system by setting the appropriate value of an input parameter:

- In the simple case, the licensing system is not required to perform any authenticity check of the license certificate.
- In the second case, the licensing system is directed to grant licenses only if they come from certificates whose authenticity was verified by the licensing system through the use of an authentication certificate.
- In the final case, the licensing system is directed to verify that a key received from the application (through another parameter of the same request call) is equal to the public key contained in the license certificate. Since the value of the key provided by the application is defined by the application software publisher, this check can ensure that the certificate installed in the license data base was also created by the same application software publisher.

In order for this validation to be trusted, in addition to the integrity of the licensing system, two further assumptions have to be made:

- The application must be integer, that is, the binary code of the application has not been patched in order to change the value of these parameters.
- The communication network between the application agent and the licensing system is secure, so that no "man in the middle" can intercept the call and change the values which are actually provided to the licensing system.

Otherwise the licensing system could be directed not to perform any check, or could be provided with a false key, equal to the one provided in a completely forged license certificate.

4.5 Licensing System Authentication

The goal of this authentication scheme is to allow an application (or a management tool) to check both the integrity and the authenticity of each message (that is, input and output parameters) passed between the application and the licensing system which the agent is communicating with. If an application software publisher chooses to implement this scheme into an application, they will restrict the application to work only with those licensing system which have provided to the application the needed information prior to a license request (see below).

The authentication is based on a signature generated by the licensing system, where the data being signed is all the parameters received from the application and all the data returned to it, and returned to the application together with the licensing system's unique identifier. This identifier is used by the application to select the appropriate public key to be used for the verification of the signature.

The trust in this scheme is based on the association between the licensing system unique identifier and the public key available to the application for the verification, and therefore is reliable also in the presence of a non-secure communication network between the application and the licensing system. How this trusted association is achieved is outside the scope of this Technical Standard.

Note: This scheme is only available to applications using the advanced application API. Applications using the basic application API can use, if available, an application broker statically linked with the application (see Section 3.1 on page 19), that

implements this scheme on behalf of the application. This Technical Standard does not require that an implementation must provide such a broker.

4.6 Process Description

All licensing system publishers that want to be authenticated by an application using this scheme must provide to the application software publisher (in a trusted way, outside the scope if this Technical Standard) the information needed to successfully authenticate the data returned by the licensing system, namely:

- The algorithms used for signing
- The corresponding public key to be used for verification
- A unique identifier of this specific licensing system.

Note: It is recommended that licensing system publishers provide a library containing the appropriate authentication functions, to make the use of this scheme easier for an application publisher. To improve security, it is recommended that this library should be statically linked with the application. The public key could be embedded in this library or, better, could be provided on external removable media accessed by the application at run time.

The application software publisher embeds in the application the knowledge of the licensing system unique identifiers that the application will support, and for each API call will follow the steps described below.

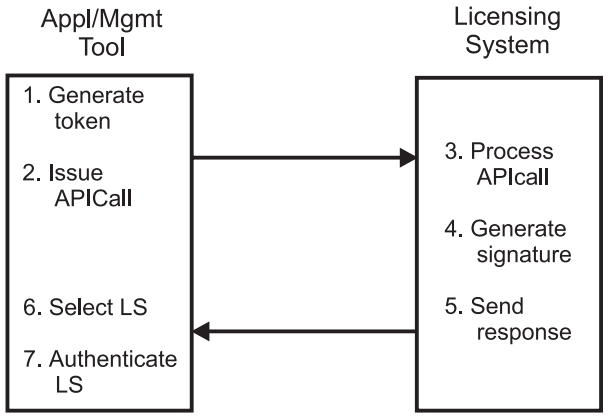


Figure 4-1 Licensing System Authentication Process

Referring to the numbers in Figure 4-1:

1. The application generates an arbitrary *token*, known only to the application, which can be any arbitrary number (for example, a timestamp, or a randomly generated number) which is different in each API call. The data signed by the licensing system will include this variable element, to provide protection against a possible "replay" of old messages by an imposter licensing system.
2. The token is passed to the licensing system as one of the input parameters of the API call. All the management API functions and most of the advanced application API functions contain this parameter.

3. The licensing system processes the request.
4. If the value of the token was not set to 0 (meaning that no authentication was needed), the licensing system signs all the data transmitted in the API call (that is, all the input parameters as received by the application and all the output parameters just computed) using the private key of the licensing system publisher.
5. The licensing system returns to the application the signature and the licensing system identifier, along with all the other output parameters.
6. The application uses the value of the returned licensing system identifier to determine the verification technique and the public key to be used for verification.
7. The application verifies the signature returned by the licensing system.

If the *verification* operation is successful then the application is assured that the licensing system is the intended one and that the data returned by the licensing system was received with integrity.

Chapter 5

Data Types and Data Elements

This chapter defines the primary data types and the different types of data elements used within an XSLM license certificate, as well as for the external representation of data elements maintained internally by a licensing system, such as certificate-instance data, status data, and logging data.

All "normal" API parameters (those not described as being represented as XSLM-specific data elements), both for the Application API and the Management API, use the data representation commonly used within the particular environment in which the application is executing — it is up to the licensing agents to transform the data as needed.

The API definitions use several type definitions, loosely defined later in this chapter. The formal definitions are provided in Chapter 10 on page 121.

5.1 Data Types

This section describes the detailed representation of all primary data types used within an XSLM licensing system.

5.1.1 Bit and Byte Numbering and Order

Within XSLM, the eight bits within a byte are numbered from left to right, starting with 0 and ending with 7. Bit 0 is the most-significant position.

Multi-byte values are stored as a sequence of bytes, in such an order that the left-most byte is the most significant byte. Within such a multi-byte value, the bytes are numbered from the left, starting with 0. Individual bits within such a multi-byte value can be referred to either by its bit position within a particular byte, or as its bit position within the multi-byte value, again starting the numbering from the left-most bit as bit 0[4].

5.1.2 Fixed-Point Binary Numbers

All fixed-point binary numbers are (except where noted) stored as non-negative numbers, 4 bytes wide, with a maximum value of 2**31-1[5].

This data type is referred to as DTP_FIXED.

4. This detailed description is needed to ensure that data can be created and manipulated within different computing platforms, even if they normally use different methods of representing such data (for example, little-endian versus big-endian)
5. Thus, in most programming languages and computing environments, they may be represented as either unsigned or signed binary numbers.

5.1.3 Floating-Point Numbers

All floating-point numbers are (except where noted) stored in IEE754 Double Float format, that is, as a 64-bit multi-byte value containing (starting from the left) 1 bit for the sign, 11 bits for the exponent, and 52 bits for the base-2 fraction.

This data type is referred to as DTP_FLOAT.

5.1.4 Character Strings

Character (text) strings are stored in Unicode, using a slightly modified version of the UCS Transformation Format 8 bit (UTF-8) format[6].

All compliant licensing systems must be able to correctly process and display the Unicode Basic Latin character range. Other UCS characters must be correctly processed, but need not be displayable. However, any non-displayable character must still be included as part of the text string, whenever that string is being passed to an API caller.

Character strings are not 0-terminated. Instead, each character string is preceded by two length values: a 4-byte non-negative fixed-point binary number equal to the character count (indicating, in most cases, the display length) and a 4-byte non-negative fixed-point binary number equal to the byte count (indicating the number of bytes used to store the string) These counts do not include their own lengths, and may both be 0 to represent an empty string.

This data type is referred to as DTP_TEXT.

5.1.5 Byte Strings

Byte (binary) strings are stored as a sequence of bytes, numbered from the left, starting with the first byte being assigned the position 0.

Byte strings are not 0-terminated. Instead, each byte sequence is preceded by a length value: a 4-byte non-negative fixed-point binary number equal to the byte count (indicating the number of bytes used to store the string). This count does not include its own length, and may be 0 to indicate an empty string.

This data type is referred to as DTP_BSTR.

5.1.6 Date/Time and Time-Interval Values

Unless otherwise specified, all date and time values, as well as interval values, are stored in character format,[7]. as described below.

---

6. In this modified format, each character in the Basic Latin character range except the code U+0000 (that is, U+0001-U+007F) is represented as a single byte with a numeric code value equal to its UCS code value. All other UCS characters are represented by two or more bytes. This encoding ensures that all of the 0x01 through 0x7f ASCII codes can be represented in a single-byte format with no conversion required between the ASCII and Unicode formats.

7. Using the modified UTF-8 format described above, but without any length fields (the date/time format uses a fixed-length format).

Representation of a Point in Time

A particular point in time is defined as follows:

```
YYYYMMDDhhmmss.mmmmmm[±]UUU
```

where YYYY represents the century and year; MM represents the month ordinal (01...12); DD represents the day of month ordinal (00...31); hhmmss represents the hour, minute, and second; mmmmmm represents the fractional time in micro-seconds; ± indicates that the time value is offset from UTC time by a positive (+) or negative (–) value (UUU), given in minutes (000...720).

If less precision is desired, then the characters to the left of ± can be replaced (from right to left) with one or more asterisks ("*")). Any omitted part is assumed to take on its lowest value.

In addition, if ±UUU is given as +***, then no time offset is used, representing local time at the licensing system server, and if ±UUU is given as ****, then the time given represents local time at the licensing system agent.

This data type is referred to as DTP_TIME.

Representation of a Time Interval

A time interval is defined as follows:

```
DDDDDDDDhhmmss.mmmmmm:000
```

where DDDDDDDD represents the number of 24-hour days; hhmmss represents the number of hours, minutes, and seconds; mmmmmm represents fractional elapsed time in micro-seconds; :000 (which must be specified exactly as given) indicates that the time value represents a time interval.

This data type is referred to as DTP_INTVL.

5.1.7 Universally Unique Identifiers (UUIDs)

Some license certificate elements, such as PUBLISHER-ID and LICENSING-SYSTEM-PUBLISHER-ID must contain values that are guaranteed to be unique in the global name space. Such values are defined as 16-byte long byte strings containing Universal Unique Identifiers (UUIDs)[8].

A UUID is created by combining the current (local or UTC) date and time, a unique value such as a network card ID, and a serial number (unique within the organization creating the UUID).

The creation of a UUID is left at the discretion of the organization that needs to define one. There is no global directory of UUIDs in use and their corresponding "owners"instead, each occurrence of a UUID within a license certificate is accompanied by a character string representation of the creating organization. (There is no requirement that all such representations for a given UUID be identical, although in general this is the preferred method.)

This data type is referred to as DTP_UUID.

8. Also known as Globally Unique Identifier (GUID).

5.2 Data Elements

Each data element is of one of two types: *simple* or *compound*. Simple data elements, such as integers and strings, are used to represent individual values; compound data elements, such as structures and lists, are used to represent a collection of related data elements (which can themselves be either simple or compound).

5.2.1 Simple Data Elements

A simple data element consists of three control fields, and the actual data value (whose format depends upon the data type).

Data Type	Data Element ID	Data Element Sequence Number	Data Element Value
4 bytes	4 bytes	4 bytes	Variable

Data Type
provides syntactic information sufficient to parse the data element, even if the particular data element definition is not understood.

Data Element ID
provides semantic information needed to properly process the element data. Without prior knowledge of the meaning of this identifier, the data element cannot be understood by a licensing system; however, the Data Type syntactic information may enable a management tool to provide some limited form of management capability even for such data elements.

Data Element Sequence Number
is a certificate-unique, arbitrary, number that can be used by the management API functions to update a particular data element, once its sequence number is known (by first retrieving the complete certificate). Once a data element sequence number has been assigned to a data element (including certificate-related state information), it can never change.

Data Element Value
provides the actual value of the particular data element.

In addition to the primitive data types described above, many simple data elements may also be of type DTP_NULL, or empty[9]. This data type may be used in place of any data type normally associated with a particular data element ID to indicate that the element has a "null" (or unspecified) value. An empty data element contains only the data type, data element ID, and data element sequence number fields; the data element value is omitted.

9. The detailed data element descriptions in Chapter 10 indicate whether a particular data element may take on the "null" value.

5.2.2 Compound Data Elements

There are two kinds of compound data elements: *structures* and *lists*. These are syntactically identical, but are used to represent two very different types of data elements.

A *structure* data element consists of a data element header, followed by zero or more data elements (simple or compound), that can be of the same or different types. A list data element, on the other hand, consists of a data element header, followed by zero or more occurrences of one data element (simple or compound). The data elements included within a structure data element can be either required or optional; required ones must be present, while optional ones may be omitted .

A *compound* data element consists of three control fields, a component count field, and the total length in bytes of the nested data elements that follow.

Data Type	Data Element ID	Data Element Sequence Number	Component Count	Length of Nested Data Elements
4 bytes	4 bytes	4 bytes	4 bytes	4 bytes

Data Type
: provides syntactic information sufficient to parse the data element, even if the particular data element definition isn't understood.

Data Element ID
: provides semantic information needed to properly process the element data. Without prior knowledge of the meaning of this identifier, the data element can't be understood by a licensing system; however, the Data Type syntactic information may enable a management tool to provide some limited form of management capability even for such data elements.

Data Element Sequence Number
: is a certificate-unique, arbitrary, number that can be used by the management API functions to update a particular data element, once its sequence number is known (by first retrieving the complete certificate). Once a data element sequence number has been assigned to a data element (including certificate-related state information), it can never change.

Component Count
: indicates the number of data elements that are included within this compound data element. Note that these data elements may themselves be compound data elements. If the value of component count is 0, then the compound data element is effectively a "null" element.

Length of Nested Data Elements
: is the total length in bytes of all the data elements that are part of the current compound data element.

5.2.3 API Data Types

The following data types are used within the API definitions, with meanings as described. Actual definitions will be provided by licensing system vendors.

Name	Description
xslm_uint32	A 32-bit unsigned binary integer. The value-range is such that the high-order bit is always 0.
xslm_handle	A data structure containing a licensing-system-created handle corresponding to an instance of a license, or to an active session between an application and a licensing system. The internal data format of this structure may vary from implementation to implementation — an application should not have any reason to inspect the structure details. However, in all implementations, this data structure must be represented by exactly 64 bits.
xslm_bin_string	Binary string of unspecified length (length specified as a separate parameter, unless the length is fixed by the architecture).
xslm_string	Text string of unspecified length (length specified as a separate parameter, unless the length is fixed by the architecture).
xslm_uuid	A 128-bit string containing a UUID data structure.
xslm_tod	A 25-character string containing date and time values as specified in Section 5.1.6 on page 32.
xslm_float	A double-precision 64-bit floating-point type, representing format IEEE 754 values, as specified in IEEE Standard 754-1985, that is, 64-bit multi-byte values containing (starting from the left) 1 bit for the sign, 11 bits for the exponent, and 52 bits for the base-2 fraction.

Chapter 6

License Certificate Format

The primary purpose of a license certificate is to encode the terms and conditions contained within a license agreement in a machine-readable representation. In addition, a license certificate may also contain certain management-related information that is not directly included within the license agreement, such as designations of specific users that may use a product; whether exceeding the available number of concurrent uses should result in rejection of further license requests; and so on.

The license certificate format is such that an external license certificate can be created in an environment different from that where the license certificate will be installed, and different from the environment where the licensed application will be running. There is not even any requirement that the tools used to create a license certificate and to install it into a license server are provided by the same publisher. The only requirement is that the tools used by publishers and customers both can create and accept the same external representation.

A license certificate issuer may deliver a license certificate to a customer in many different ways, for example as a binary file on a diskette; as a binary-file attachment via electronic mail; or as a textual representation (for example, by displaying each binary byte of data as two hexadecimal characters) via fax. Some publishers may choose to include a limited-function license certificate on the shipping media for a product; others may require the customer to contact the publisher to receive a license certificate. Any such delivery mechanism is acceptable. However, all compliant license servers must be able to process a license certificate as a pure binary file, as described in this chapter.

A license certificate may contain security and integrity data (such as a digital signature) to ensure that any tampering is detectable by the licensing system, or directly by the application when it requests a license.

This chapter specifies the format of a license certificate and defines how application and licensing system publishers can define their own, private, data elements. Chapter 10 contains a list of all predefined data elements, both required and optional ones.

6.1 Overall Certificate Structure

In its simplest form, a license certificate consists of a compound data element (as described in Section 5.2 on page 34) containing one or more sections, each in turn made up of one or more simple and/or compound data elements. Some of these predefined data elements are required, while others are optional. In addition, both application publishers and licensing system publishers can define their own custom data elements within special sections. However, it's also possible to combine several independent certificates in one certificate group, for example when licensing several products together in a bundle or suite. Table 6-1 shows the overall structure of a certificate (the detailed definition is given in Chapter 10).

GROUP_CERTIFICATE
  GROUP_TYPE
  CERTIFICATE_LIST
  CERTIFICATE
  BASE_SECTION
  PUBLISHER_SECTION
  LICENSING_SYSTEM_SECTION_LIST
  LICENSING_SYSTEM_SECTION
  AUTHENTICATION_SECTION
  GROUP_AUTHENTICATION_SECTION
	A certificate group consists of one or more related certificates (if there's only one certificate, the group element may be omitted). Each certificate contains a required base section, may have an application-publisher-defined section and/or one or more licensing system-dependent sections, and/or an authentication section. Finally, the certificate group (if present) must have its own authentication section.

BASE_SECTION
  FUNCTIONAL_LEVEL
  CERTIFICATE_CREATED
  CERTIFICATE_ID
  CERTIFICATE_DESCRIPTION
  PUBLISHER_USE
  REPLACE_CERTIFICATE
  LIFE
  DURATION
  LICENSED_UNITS
  PUBLISHER_CAPACITY_LIMITS_LIST
  PUBLISHER_ASSIGNMENTS_LIST
  CERTIFICATE_TARGET_NODES
  CUSTOMER_ASSIGNABLE_LIMITS
  COUNTERS_CONSUMPTIVE
  COUNTERS_CUMULATIVE
  CONFIRM_INTERVAL
  NON_MASKABLE_EVENTS
  RESETTING_FREQUENCY
  LOCALLY_AVAILABLE
  DEFAULT_UNITS_TO_GRANT
  FORCE_RELEASE_OK
  ADVANCE_EXPIRATION_NOTIFICATION
  DISASTER_RECOVERY
  MULTI_USE_ALLOWED
	The base section contains all the data elements normally used for granting license requests. Of particular importance are CERTYIFICATE_ID whose components uniquely identify a particular certificate; REPLACE_CERTIFICATE which allows an already installed certificate to be replaced with an updated one; LIFE and DURATION which define the date/time interval during which the certificate is valid; and UNITS which specifies the number of license units (for example, number of concurrent users of an application, or how many times an application may be executed).
	
	The remaining components further qualify and quantify how license request can be granted from this certificate, and also define such items as the counters used for application-initiated metering; control of which events must be logged; and some management-related elements such as whether the customer may make use of this certificate in a disaster recovery situation.

Table 6-1 License Certificate Structure

6.1.1 Required and Optional License Certificate Sections

The *base* section contains all data elements that are required within a certificate, as well as the data elements that are optional but for which the certificate issuer wants to specify values. This section is required.

The publisher *section* contains data elements that the certificate issuer has defined for use directly by the application. These data elements are not understood by the licensing system; the only processing it will perform on this section is to store it and make it available to an application upon request. This section is optional.

The *licensing system* sections contain settings for data elements defined by a particular licensing system publisher. These data elements are only understood by the licensing system that defines them. A certificate that contains licensing system sections may only be installed on a licensing system that understands one of the licensing system sections. This section is optional, and there

may be more than one such section (but at most one for each unique licensing system).

One possible use of a licensing system section is to encapsulate the complete license password and data pertaining to a license managed by a technical license manager, thus providing a way to provide some management capabilities also for such licenses without requiring major changes to their format.

Finally, an *authentication* section may optionally be part of each certificate as well as of the group of certificates, to ensure that the data created by the certificate issuer is unaltered when it reaches the licensing system and, eventually, the licensed application.

6.2 License Certificate State Data

In addition to the data contained within a license certificate, each licensing system must maintain some license certificate-related data that, while not physically part of a license certificate, logically can be seen as belonging to it. This includes the following types of data:

- Local copies of certain data elements maintained by the licensing system to permit modifications by an administrator or application program (for example, choice of hard stop-policy instead of soft-stop policy; default license re-confirm time).
- License certificate-related specifications made by an administrator (for example, user- and node-assignments).
- Certain types of data elements that can never occur within a license certificate, but from the outside appear as if they are part of it (for example, application-recorded usage data; number of concurrent license requests granted at this point in time).

These types of data can be considered collected into a state section. The status section is not part of the certificate as created by a certificate issuer — it exists only within the data maintained by a licensing system and its contents is made available externally via the management API.

6.3 Base and Optional Data Element Sets

The certificate data elements defined in this specification are arranged in sets. The basic set contains all data elements that must be supported by all compliant licensing systems. Other sets contain data elements that need not be supported by all licensing systems. However, any licensing system that supports one data element within a particular set must also support all other data elements within that set. (The current specification defines the basic set and one optional set.)

A certificate that contains data elements defined in one or more optional sets can only be installed in a licensing system that supports all those sets.

Chapter 7

Application Program API

This chapter contains a detailed description of the callable functions included in the basic and advanced sets of the Application Program API (XAAPI).

An application can concurrently use both the basic application API set and the advanced application API set for different licenses. However, a license obtained via *xslm_basic_request_license*( ) can only be manipulated via the basic application API set, and a license obtained via *xslm_adv_request_license*( ) can only be manipulated via the advanced application API set.

7.1 Application API - Common Functions

In order to provide exploitation of licensing system functions available in a licensing system implementation, an application must have the ability to determine the XSLM functional level of the licensing system. The *xslm_query_api_level*( ) function is defined to satisfy this requirement.

A customer's licensing system may be comprised of servers from multiple implementations and/or (in the future) servers conforming to different levels of the XSLM specification. In responding to a query of the licensing system functional level, an Application Broker must return the highest implementation level supported by all license servers, as well as by the Broker itself.

XAAPI Function	Usage
xslm_query_api_level( )	Returns the maximum API level supported by the licensing system.

7.2 Application API - The Basic Set

XAAPI Function	Usage
xslm_basic_confirm( )	Confirms that a license is still in use.
xslm_basic_release_license( )	Releases a previously acquired license.
xslm_basic_request_license( )	Requests a license to run.

7.3 Application API - The Advanced Set

XAAPI Function	Usage
xslm_adv_begin_session( )	Starts a license use session.
xslm_adv_confirm( )	Confirms that a license is still in use.
xslm_adv_end_session( )	Ends a license use session.
xslm_adv_log( )	Logs an application-specified message.
xslm_adv_query( )	Requests certificate-related information.
xslm_adv_record( )	Records application-collected usage data.
xslm_adv_release_license( )	Releases a previously acquired license.
xslm_adv_request_license( )	Requests a license to run.

NAME

xslm_adv_begin_session

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_adv_begin_session(
    xslm_handle * session_handle,
    xslm_uint32 * status
);
```

DESCRIPTION

xslm_adv_begin_session( ) establishes a reference which is used to keep track of composite licensing activities (that is, multiple license requests identified by different handles) as a whole. A session may include license requests for products from one or more publishers.

OUTPUT PARAMETERS

session_handle
An identifier representing the newly created session.

status
Completion status. Detailed error code directly processable by the caller. This value is set to XSLM_STATUS_OK if no error occurred. For other values, see ERRORS.

RETURN VALUE

XSLM_OK
Session created

XSLM_COMM_ERR
Communications problem

XSLM_RESRC_UNAVL
Local resources unavailable

XSLM_PARM_ERR
Parameter error

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred
XSLM_COMM_ERR	XSLM_LIC_SYS_NOT_RESP	The licensing system does not respond
XSLM_RESRC_UNAVL	XSLM_NO_RES	Platform dependent
XSLM_PARM_ERR	XSLM_BAD_PARM	One or more parameters were not correct

SEE ALSO

xslm_adv_end_session( ).

NAME

xslm_adv_confirm

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_adv_confirm(
    xslm_handle     session_handle,
    xslm_handle     lic_handle,
    xslm_uint32     * confirm_time,
    xslm_uint32     * status,
    xslm_uint32     auth_token,
    xslm_bin_string * auth_signature,
    xslm_uuid       * auth_lic_sys_id,
);
```

DESCRIPTION

xslm_adv_confirm( ) confirms that a license is currently in use. Should the license server not receive confirmation from the product within the default time period specified in the certificate, the license server will assume the license is no longer in use and will terminate the license and, if it is of the CONCURRENT type, return the respective number of units. This call allows the specification of the *confirm_time* value.

INPUT PARAMETERS

session_handle
A reference returned by *xslm_adv_begin_session*( )

lic_handle
A reference returned by *xslm_adv_request_license*( )

auth_token
A 32-bit arbitrary value created by the application and used as part of the licensing system authentication process. See Section 4.6 on page 28 for more information.

INPUT/OUTPUT PARAMETERS

confirm_time
Elapsed time (in seconds) within which the license server will expect the next *xslm_adv_confirm*( ) call. Failure to confirm continued license use will cause the license server to assume that the product is no longer active and, if the license is reusable, return it to the pool of available licenses. If *confirm_time* is specified as 0, the license server will return the current value.

A *confirm_time* value specified by an application overrides any value contained within the license certificate, or any value set by the administrator.

OUTPUT PARAMETERS

status
Completion status. Detailed error code directly processable by the caller. This value is set to XSLM_STATUS_OK if no error occurred. For other messages, see ERRORS.

auth_signature
An area large enough to contain a 16-byte digital signature created by the licensing system from the input parameters and auth_token. Used as part of the licensing system authentication process.

auth_lic_sys_id
A unique identifier for the particular type of licensing system handling the current license instance. Used as part of the licensing system authentication process.

RETURN VALUE

XSLM_OK
License still valid

XSLM_COMM_ERR
Communications problem

XSLM_CERT_ERR
Problem with license and/or certificate

XSLM_RESRC_UNAVL
Local resources unavailable

XSLM_PARM_ERR
Parameter error

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred
XSLM_COMM_ERR	XSLM_LIC_SYS_NOT_RESP	The licensing system does not respond
XSLM_CERT_ERR	XSLM_CAPACITY_LIMIT	A capacity limit attached to the license has been exceeded
XSLM_RESRC_UNAVL	XSLM_NO_RES	Platform dependent
XSLM_PARM_ERR	XSLM_BAD_LICENSE_HANDLE	The handle is invalid, perhaps because the *xslm_adv_confirm*( ) call was issued too late
	XSLM_BAD_PARM	One or more parameters were not correct
	XSLM_BAD_SESSION_HANDLE	The specified session handle is invalid
	XSLM_INVALID_API_USE	Attempt to combine basic and advanced API for same handle

SEE ALSO

xslm_adv_begin_session( ), *xslm_adv_request_license*( ).

NAME

xslm_adv_end_session

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_adv_end_session(
    xslm_handle session_handle,
    xslm_uint32 * status
);
```

DESCRIPTION

xslm_adv_end_session( ) terminates the session established by a call to *xslm_adv_begin_session*( ). All active licenses requested under the session to be terminated will be released.

INPUT PARAMETERS

session_handle
A reference returned by *xslm_adv_begin_session*( ).

OUTPUT PARAMETERS

status
Completion status. Detailed error code directly processable by the caller. This value is set to XSLM_STATUS_OK if no error occurred. For other messages, see ERRORS.

RETURN VALUE

XSLM_OK
Session terminated

XSLM_COMM_ERR
Communications problem

XSLM_RESRC_UNAVL
Local resources unavailable

XSLM_PARM_ERR
Parameter error

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred
XSLM_COMM_ERR	XSLM_LIC_SYS_NOT_RESP	The licensing system does not respond
XSLM_RESRC_UNAVL	XSLM_NO_RES	Platform dependent
XSLM_PARM_ERR	XSLM_BAD_PARM	One or more parameters were not correct
	XSLM_BAD_SESSION_HANDLE	The specified session handle is invalid

SEE ALSO

xslm_adv_begin_session( ).

NAME

xslm_adv_log

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_adv_log(
    xslm_handle     session_handle,
    xslm_handle     lic_handle,
    xslm_uint32     * msg_length,
    xslm_string     msg,
    xslm_uint32     * status,
    xslm_uint32     auth_token,
    xslm_bin_string * auth_signature,
    xslm_uuid       * auth_lic_sys_id
);
```

DESCRIPTION

xslm_adv_log( ) logs an application-specified text message into the license server's log. Publisher-specific message logging can usually be disabled by an administrator. When application message logging is disabled, XSLM_MASK_APPLIED status value is returned and no message is logged.

A return code of XSLM_OK merely indicates that the message to be logged has been accepted by the license server; it may not have been committed to a physical log file.

INPUT PARAMETERS

session_handle
A reference returned by *xslm_adv_begin_session*( ).

lic_handle
A reference returned by *xslm_adv_request_license*( ).

msg
Text of the message. This can be of any length; however, a licensing system is not required to accept more than 4,096 bytes.

auth_token
A 32-bit arbitrary value created by the application and used as part of the licensing system authentication process. See Section 4.6 on page 28 for more information.

INPUT/OUTPUT PARAMETERS

msg_length
Length in bytes of the msg parameter. If specified as 0, or a value higher than the licensing system can handle, this value is updated to indicate the maximum number of bytes that can be accepted (in these cases, no data is written).

OUTPUT PARAMETERS

status
Completion status. Detailed error code directly processable by the caller. This value is set to XSLM_STATUS_OK if no error occurred. For other messages, see ERRORS.

auth_signature
An area large enough to contain a 16-byte digital signature created by the licensing system from the input parameters and *auth_token*. Used as part of the licensing system authentication process.

auth_lic_sys_id
A unique identifier for the particular type of licensing system handling the current license instance. Used as part of the licensing system authentication process.

RETURN VALUE

XSLM_OK
Message accepted

XSLM_COMM_ERR
Communications problem

XSLM_CERT_ERR
Problem with license and/or certificate

XSLM_RESRC_UNAVL
Local resources unavailable

XSLM_PARM_ERR
Parameter error

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred (including the case when the input length is 0)
	XSLM_MASK_APPLIED	Log record not written due to log masking as set by an administrator
XSLM_COMM_ERR	XSLM_LIC_SYS_NOT_RESP	The licensing system does not respond
XSLM_CERT_ERR	XSLM_LOG_ERROR	Error accessing log
XSLM_RESRC_UNAVL	XSLM_NO_RES	Platform dependent
XSLM_PARM_ERR	XSLM_BAD_LICENSE_HANDLE	The specified license handle is invalid
	XSLM_BAD_PARM	One or more parameters were not correct
	XSLM_BAD_SESSION_HANDLE	The specified session handle is invalid
	XSLM_INVALID_API_USE	Attempt to combine basic and advanced API for same handle
	XSLM_MSG_TOO_LONG	The specified message length exceeds the maximum length that the licensing system can handle. No data is written.

Return Value	Status Value	Explanation
		The maximum length supported is returned in the *msg_length* parameter.

SEE ALSO

xslm_adv_begin_session( ), *xslm_adv_request_license*( ).

NAME

xslm_adv_query

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_adv_query(
    xslm_handle     session_handle,
    xslm_handle     lic_handle,
    xslm_uint32     query_type,
    xslm_uint32     * query_buffer_length,
    xslm_bin_string * query_buffer,
    xslm_uint32     * status,
    xslm_uint32     auth_token,
    xslm_bin_string * auth_signature,
    xslm_uuid       * auth_lic_sys_id
);
```

DESCRIPTION

xslm_adv_query( ) returns various types of license certificate information associated with the specified license handle.

INPUT PARAMETERS

session_handle

A reference returned by *xslm_adv_begin_session*( ).

lic_handle

A reference returned by *xslm_adv_request_license*( ).

query_type

Value which identifies the information to be returned:

Query Type	Explanation
XSLM_QUERY_PUBLISHER_INFO	Return the publisher-unique section of the license certificate publisher defined data element structure. A returned *query_buffer_length* value of null indicates that the certificate contains no publisher-unique section.
XSLM_QUERY_CUST_DEF_INFO	Return application-related information specified by the customer. A returned *query_buffer_length* value of null indicates that the certificate does not contain the CUSTOMER_ASSIGNED_APPL_INFO data element.
XSLM_QUERY_CERTIFICATE	Return the certificate as it has been created by the application publisher.
XSLM_QUERY_CERT_RELATED_INFO	Return the certificate related information maintained by the Licensing System.

auth_token

A 32-bit arbitrary value created by the application and used as part of the licensing system authentication process. See Section 4.6 on page 28 for more information.

INPUT/OUTPUT PARAMETERS

query_buffer_length
As an input parameter, the length of the buffer to receive the results of the query. A value of zero indicates a request for the size of the buffer required to contain the entire results of the query. No data is returned in the key buffer when a value of zero is passed.

As an output parameter, the length of data returned, or the size of the buffer required to contain the entire query result if the input value was zero.

Note that the buffer must be large enough to contain the entire result of the query. If it is not, no data is placed in the buffer and the output value of *query_buffer_length* contains the buffer length needed to contain the entire result. A value of zero on output indicates that the requested information was not available on the certificate.

Note that because the certificate related information may change at any time, the actual size needed may be different from the returned value in cases other than XSLM_QUERY_PUBLISHER_INFO and XSLM_QUERY_CERTIFICATE query types.

OUTPUT PARAMETERS

query_buffer
Buffer in which the query results are returned. The results are returned as a set of one or more data elements.

status
Completion status. Detailed error code directly addressable by the caller. This value is set to XSLM_STATUS_OK if no error occurred. For other messages, see ERRORS.

auth_signature
An area large enough to contain a 16-byte digital signature created by the licensing system from the input parameters and auth_token. Used as part of the licensing system authentication process.

auth_lic_sys_id
A unique identifier for the particular type of licensing system handling the current license instance. Used as part of the licensing system authentication process.

RETURN VALUE

XSLM_OK
Query completed successfully

XSLM_COMM_ERR
Communications problem

XSLM_RESRC_UNAVL
Resources unavailable

XSLM_PARM_ERR
Bad parameters passed

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred
XSLM_COMM_ERR	XSLM_COMM_UNAVAIL	Communications problem
XSLM_RESRC_UNAVL	XSLM_NO_RES	Local platform specific environmental problems
	XSLM_SERVER_ERROR	Unrecoverable environmental error was encountered by the license server
XSLM_PARM_ERR	XSLM_BAD_BUFFER_LENGTH	The non-zero value for the buffer length was too small for the certificate being retrieved.
	XSLM_BAD_LICENSE_HANDLE	Invalid license handle passed.
	XSLM_BAD_PARM	One or more parameters were not correct
	XSLM_BAD_SESSION_HANDLE	The specified session handle is invalid
	XSLM_INVALID_API_USE	Attempt to combine standard and advanced API for the same handle

OUTPUT DATA

- For XSLM_QUERY_PUBLISHER_INFO, the data element PUBLISHER_SECTION, if present, followed by the data element PUBLISHER_USE, if present.
- For XSLM_QUERY_CUST_DEF_INFO, the data element CUSTOMER_ASSIGNED_APPL_INFO, if present.
- For XSLM_QUERY_CERTIFICATE, the data element CERTIFICATE.
- For XSLM_QUERY_CERT_RELATED_INFO, the data element CERTIFICATE_RELATED_INFORMATION.

See Chapter 10 on page 121 for details on the data elements.

Note that if the *query_buffer_length* parameter does not specify a large enough buffer to contain the entire query result, no data is returned

SEE ALSO

xslm_adv_begin_session( ), *xslm_adv_request_license*( ).

NAME

xslm_adv_record

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_adv_record(
    xslm_handle      session_handle,
    xslm_handle      lic_handle,
    xslm_uint32      counter_id,
    xslm_float       counter_incr,
    xslm_float       * counter_value,
    xslm_uint32      * status,
    xslm_uint32      auth_token,
    xslm_byte_string * auth_signature,
    xslm_uuid        * auth_lic_sys_id
);
```

DESCRIPTION

xslm_adv_record( ) is used to record resource usage in one of the set of certificate-related counters defined by the application publisher.

INPUT PARAMETERS

session_handle
A reference returned by *xslm_adv_begin_session*( ).

lic_handle
A reference returned by *xslm_adv_request_license*( ).

counter_id
A number which identifies one of the counters defined within the license certificate.

counter_incr
The value that the licensing system should add to or subtract from (depending upon the counter type) the current counter value. This value must be greater than or equal to 0. If it is set to 0, then the current counter value is returned in the *counter_value* parameter and the event is not logged.

auth_token
A 32-bit arbitrary value created by the application and used as part of the licensing system authentication process. See Section 4.6 on page 28 for more information.

OUTPUT PARAMETERS

counter_value
The value of the counter following processing of the current request. Note that if *counter_incr* is set to 0, then the only processing done is the return of the current counter value.

status
Completion status. Detailed error code directly processable by the caller. This value is set to XSLM_STATUS_OK if no error occurred. For other messages, see ERRORS.

auth_signature
An area large enough to contain a 16-byte digital signature created by the licensing system from the input parameters and auth_token. Used as part of the licensing system authentication process.

auth_lic_sys_id
A unique identifier for the particular type of licensing system handling the current license instance. Used as part of the licensing system authentication process.

RETURN VALUE

XSLM_OK
Data recorded

XSLM_COMM_ERR
Communications problem

XSLM_CERT_ERR
Problem with license and/or certificate

XSLM_RESRC_UNAVL
Local resources unavailable

XSLM_PARM_ERR
Parameter error

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred
XSLM_COMM_ERR	XSLM_LIC_SYS_NOT_RESP	The licensing system does not respond
XSLM_CERT_ERR	XSLM_INV_COUNTER_ID	Counter not defined within license certificate
	XSLM_ZERO_REACHED	Counter has reached the value 0
	XSLM_COUNT_OVERFLOW	Counter has wrapped around
	XSLM_COUNT_UNDERFLOW	Counter has wrapped around
XSLM_RESRC_UNAVL	XSLM_NO_RES	Platform dependent
XSLM_PARM_ERR	XSLM_BAD_LICENSE_HANDLE	The license handle is invalid
	XSLM_BAD_PARM	One or more parameters were not correct
	XSLM_BAD_SESSION_HANDLE	The specified session handle is invalid
	XSLM_INVALID_API_USE	Attempt to combine basic and advanced API for same handle

SEE ALSO

xslm_adv_begin_session( ), *xslm_adv_request_license*( ).

NAME

xslm_adv_release_license

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_adv_release_license(
    xslm_handle     session_handle,
    xslm_handle     lic_handle,
    xslm_uint32     * status,
    xslm_uint32     auth_token,
    xslm_bin_string * auth_signature,
    xslm_uuid       * auth_lic_sys_id
);
```

DESCRIPTION

xslm_adv_release_license( ) releases all license units related to and acquired via a prior call to *xslm_adv_request_license*( ).

INPUT PARAMETERS

session_handle
A reference returned by *xslm_adv_begin_session*( ).

lic_handle
A reference returned by *xslm_adv_request_license*( ). It is used to store information about a request/release transaction for a particular product.

auth_token
A 32-bit arbitrary value created by the application and used as part of the licensing system authentication process. See Section 4.6 on page 28 for more information.

OUTPUT PARAMETERS

status br Completion status. Detailed error code directly processable by the caller. This value is set to XSLM_STATUS_OK if no error occurred. For other messages, see ERRORS.

auth_signature
An area large enough to contain a 16-byte digital signature created by the licensing system from the input parameters and auth_token. Used as part of the licensing system authentication process.

auth_lic_sys_id
A unique identifier for the particular type of licensing system handling the current license instance. Used as part of the licensing system authentication process.

RETURN VALUE

XSLM_OK
License returned

XSLM_COMM_ERR
Communications problem

XSLM_RESRC_UNAVL
Local resources unavailable

XSLM_PARM_ERR
Parameter error

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred
XSLM_COMM_ERR	XSLM_LIC_SYS_NOT_RESP	The licensing system does not respond
XSLM_RESRC_UNAVL	XSLM_NO_RES	Platform dependent
XSLM_PARM_ERR	XSLM_BAD_LICENSE_HANDLE	The specified license handle is invalid
	XSLM_BAD_PARM	One or more parameters were not correct
	XSLM_BAD_SESSION_HANDLE	The specified session handle is invalid
	XSLM_INVALID_API_USE	Attempt to combine basic and advanced API for the same handle

SEE ALSO

xslm_adv_begin_session( ), *xslm_adv_request_license*( ).

NAME

xslm_adv_request_license

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_adv_request_license(
    xslm_handle     session_handle,
    xslm_uuid       publisher_id,
    xslm_uint32     product_id,
    xslm_uint32     version_id,
    xslm_uint32     feature_id,
    xslm_uint32     named_user_length,
    xslm_string     * named_user,
    xslm_uint32     num_units_req,
    xslm_uint32     * num_units_granted,
    xslm_uint32     force_num_units,
    xslm_uint32     confirm_time,
    xslm_uint32     cert_auth_type
    xslm_uint32     publisher_key_length,
    xslm_bin_string * publisher_key,
    xslm_handle     * lic_handle,
    xslm_uint32     * status,
    xslm_uint32     auth_token,
    xslm_bin_string * auth_signature,
    xslm_uuid       * auth_lic_sys_id
);
```

DESCRIPTION

xslm_adv_request_license( ) asks the license server for *num_units_req* license units, for the product identified by *publisher_id*, *product_id*, *version_id*, and *feature_id*. This call can also provide confirm time information, via *confirm_time*, to the server to specify the time interval within which an *xslm_confirm*( ) call must be issued in order for the license to remain valid.

INPUT PARAMETERS

session_handle
A reference returned by *xslm_adv_begin_session*( ).

publisher_id
The software publisher identification.

product_id
The identification of the product whose license is requested.

version_id
The identification of the version of the product whose license is requested.

feature_id
The product and versions associated feature number; if not used, must be set to 0.

named_user_length
Length of *named_user* in bytes. If this value is 0, no name is specified, and the licensing system should use the current system-dependent name, if required.

named_user
A text string containing the id of the named user for which a license is requested. An empty

string (that is, *named_user_length* equal to 0) indicates that the licensing system should use the current system-dependent name, if required.

num_units_req
The number of license units requested. A value of XSLM_DEFAULT_UNITS indicates that the default number of units, as defined by the certificate data element DEFAULT_UNITS_TO_GRANT, will be requested.

force_num_units
A value of XSLM_GRANT_PARTIAL indicates that the license request may be satisfied by less than the number of units requested via the *num_units_req* parameter. A value of XSLM_GRANT_FULL indicates that the request may only be satisfied by at least as many units as indicated by the *num_units_req* parameter. No other values may be specified.

cert_auth_type
Certificate authorization request type. If this value is XSLM_CERT_AUTH_NONE, the licensing system is not required to perform any specific action to authenticate the license certificate issuer before granting a license. If this value is XSLM_CERT_AUTH_PUB_KEY, the licensing system must use the value provided via the *publisher_key* parameter to authenticate the license certificate issuer. If this value is XSLM_CERT_AUTH_CA, then the licensing system must perform the authentication using an authentication certificate associated with the license certificate, as described in Section 4.4 on page 27.

publisher_key_len
Length of *publisher_key* in bytes. If this value is 0, no certificate authentication will take place, and the certificate is assumed to have been created by the application publisher.

publisher_key
Publisher-specified public key. Used to confirm the authenticity of a license certificate. The value passed must match the value of the publisher's public key contained in the license certificate.

auth_token
A 32-bit arbitrary value created by the application and used as part of the licensing system authentication process. See Section 4.6 on page 28 for more information.

INPUT/OUTPUT PARAMETERS

confirm_time
Elapsed time (in seconds) within which the license server will expect the next *xslm_adv_confirm*( ) call. This value is an integer greater than or equal to one. Failure to confirm continued license use will cause the license server to assume that the product is no longer active and, if the license is reusable, return it to the pool of available licenses.

The *confirm_time* value overrides any other confirmation value. If set to 0, the current value will be returned by the license.

OUTPUT PARAMETERS

num_units_granted
Number of license units granted.

lic_handle
A reference returned by this call. It is used to maintain information about the currently granted license.

status
Completion status. Detailed error code directly processable by the caller. This value is set to XSLM_STATUS_OK if no error occurred. For other messages, see ERRORS.

auth_signature
An area large enough to contain a 16-byte digital signature created by the licensing system from the input parameters and *auth_token*. Used as part of the licensing system authentication process.

auth_lic_sys_id
A unique identifier for the particular type of licensing system handling the current license instance. Used as part of the licensing system authentication process.

RETURN VALUE

XSLM_OK
License granted

XSLM_COMM_ERR
Communications problem

XSLM_CERT_ERR
Problem with license and/or certificate

XSLM_RESRC_UNAVL
Local resources unavailable

XSLM_PARM_ERR
Parameter error

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred
	XSLM_IN_SOFT_STOP	License granted due to soft-stop policy
	XSLM_IN_RECOVERY_MODE	License granted due to disaster-recovery mode
XSLM_COMM_ERR	XSLM_LIC_SYS_NOT_RESP	The licensing system does not respond
	XSLM_NO_SVRS_FOUND	No license servers responding
XSLM_CERT_ERR	XSLM_NO_CERTIFICATES	No certificate found for the specified *publisher_id*, *product_id*, *version_id* and *feature_id*
	XSLM_CERT_NOT_STARTED	Certificate validity period not yet begun.
	XSLM_NOT_ENOUGH_CAPACITY	Capacity requested exceeds that available.

Return Value	Status Value	Explanation
	XSLM_NOT_ENOUGH_LICS	Number of requested licenses is more than the number available.
	XSLM_NO_LICS	The number of available license units is currently less than the number needed to satisfy the request.
	XSLM_NO_MATCHING_NODE	No certificate found for current node.
	XSLM_NO_MATCHING_USERID	No certificate found for current userID.
	XSLM_INVALID_PUBLIC_KEY	Public keys do not match
XSLM_RESRC_UNAVL	XSLM_NO_RES	Platform dependent
XSLM_PARM_ERR	XSLM_BAD_PARM	One or more parameters were not correct
	XSLM_BAD_SESSION_HANDLE	The specified session handle is invalid

SEE ALSO

xslm_adv_begin_session( ).

NAME

xslm_basic_confirm

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_basic_confirm(
    xslm_handle lic_handle,
    xslm_uint32 * status
);
```

DESCRIPTION

xslm_basic_confirm( ) confirms that a license is currently in use. Should the license server not receive any confirmation from the application within the default time period specified in the certificate, the license server will assume the license is no longer in use and will release the license, and, if it is of the CONCURRENT type, return the respective number of units.

INPUT PARAMETERS

lic_handle
A reference returned by *xslm_basic_request_license*( ).

OUTPUT PARAMETERS

status
Completion status. Detailed error code directly processable by the caller. This value is set to XSLM_STATUS_OK if no error occurred. For other messages, see ERRORS.

RETURN VALUE

XSLM_OK
License obtained

XSLM_COMM_ERR
Communications problem

XSLM_CERT_ERR
Problem with license and/or certificate

XSLM_RESRC_UNAVL
Local resources unavailable

XSLM_PARM_ERR
Parameter error

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred
XSLM_COMM_ERR	XSLM_LIC_SYS_NOT_RESP	The licensing system does not respond
XSLM_RESRC_UNAVL	XSLM_NO_RES	Platform dependent
XSLM_PARM_ERR	XSLM_BAD_LICENSE_HANDLE	The handle is invalid, perhaps because the *xslm_confirm*( ) call was issued too late
	XSLM_BAD_PARM	One or more parameters were not correct
	XSLM_INVALID_API_USE	Attempt to combine basic and advanced API for same handle

SEE ALSO

xslm_basic_request_license( ).

NAME

xslm_basic_release_license

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_basic_release_license(
    xslm_handle lic_handle,
    xslm_uint32 * status
);
```

DESCRIPTION

xslm_basic_release_license( ) releases all license units related to and acquired via a prior call to *xslm_basic_request_license*( )

INPUT PARAMETERS

lic_handle

A reference returned by *xslm_basic_request_license*( ). It is used to store information about a request/release transaction for a particular product.

OUTPUT PARAMETERS

status

Completion status. Detailed error code directly processable by the caller. This value is set to XSLM_STATUS_OK if no error occurred. For other error codes, see ERRORS.

RETURN VALUE

XSLM_OK
License obtained

XSLM_COMM_ERR
Communications problem

XSLM_RESRC_UNAVL
Local resources unavailable

XSLM_PARM_ERR
Parameter error

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred
XSLM_COMM_ERR	XSLM_LIC_SYS_NOT_RESP	The licensing system does not respond
XSLM_RESRC_UNAVL	XSLM_NO_RES	Platform dependent
XSLM_PARM_ERR	XSLM_BAD_LICENSE_HANDLE	The specified handle is invalid
	XSLM_BAD_PARM	One or more parameters were not correct
	XSLM_INVALID_API_USE	Attempt to combine basic and advanced API for same handle

SEE ALSO

xslm_basic_request_license( ).

NAME

xslm_basic_request_license

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_basic_request_license(
    xslm_uuid       publisher_id,
    xslm_uint32     product_id,
    xslm_uint32     version_id,
    xslm_uint32     feature_id,
    xslm_uint32     cert_auth_type
    xslm_uint32     publisher_key_length,
    xslm_bin_string * publisher_key,
    xslm_handle     * lic_handle,
    xslm_uint32     * status
);
```

DESCRIPTION

xslm_basic_request_license( ) requests a license, using the default number of license units (as specified within a license certificate), for a *publisher_id* product identified by *product_id*, *version_id*, and additionally by the *feature_id* parameter.

INPUT PARAMETERS

publisher_id
The software publisher identification.

product_id
The identification of the product whose license is requested.

version_id
The identification of the version of the product whose license is requested.

feature_id
The product and version's associated feature number; if not used, must be set to 0.

cert_auth_type
Certificate authorization request type. If this value is XSLM_CERT_AUTH_NONE, the licensing system is not required to perform any specific action to authenticate the license certificate issuer before granting a license. If this value is XSLM_CERT_AUTH_PUB_KEY, the licensing system must use the value provided via the *publisher_key* parameter to authenticate the license certificate issuer. If this value is XSLM_CERT_AUTH_CA, then the licensing system must perform the authentication using an authentication certificate associated with the license certificate, as described in Section 4.4 on page 27.

publisher_key_len
Length of *publisher_key* in bytes. If this value is 0, no certificate authentication will take place, and the certificate is assumed to have been created by the application publisher.

publisher_key
Publisher-specified public key. Used to confirm the authenticity of a license certificate. The value passed must match the value of the publisher's public key contained in the license certificate.

OUTPUT PARAMETERS

lic_handle
A reference returned by this call. It is used to maintain information about the currently granted license.

status
Completion status. Detailed error code directly processable by the caller. This value is set to XSLM_STATUS_OK if no error occurred. For other messages, see ERRORS.

RETURN VALUE

XSLM_OK
License granted

XSLM_COMM_ERR
Communications problem

XSLM_CERT_ERR
Problem with license and/or certificate

XSLM_RESRC_UNAVL
Local resources unavailable

XSLM_PARM_ERR
Parameter error

ERRORS

The function's return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred
	XSLM_IN_SOFT_STOP	License granted due to soft-stop policy
	XSLM_IN_RECOVERY_MODE	License granted due to disaster-recovery mode
XSLM_COMM_ERR	XSLM_LIC_SYS_NOT_RESP	The licensing system does not respond
	XSLM_NO_SVRS_FOUND	No license servers responding
XSLM_CERT_ERR	XSLM_NO_CERTIFICATES	No certificate found for the specified *publisher_id*, *product_id*, *version_id*, and *feature_id*
	XSLM_CERT_NOT_STARTED	Certificate validity period not yet begun
	XSLM_NOT_ENOUGH_CAPACITY	Capacity requested exceeds that available

Return Value	Status Value	Explanation
	XSLM_NO_LICS	The number of available license units is currently less than the number needed to satisfy the request
	XSLM_NO_MATCHING_NODE	No certificate found for current node
	XSLM_NO_MATCHING_USERID	No certificate found for current userID
	XSLM_INVALID_PUBLIC_KEY	Public keys do not match
XSLM_RESRC_UNAVL	XSLM_NO_RES	Platform dependent.
XSLM_PARM_ERR	XSLM_BAD_PARM	One or more parameters were not correct

SEE ALSO

xslm_basic_confirm( ), *xslm_basic_release_license*( ).

NAME

xslm_query_api_level

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_query_api_level(
    xslm_uint32 * func_level,
    xslm_uint32 * func_tower_count,
    xslm_uint32 [] func_towers,
    xslm_uint32 * status
);
```

DESCRIPTION

xslm_query_api_level( ) returns the highest API levels supported by all license servers in the licensing system. (Individual license servers may support a higher API level.)

INPUT/OUTPUT PARAMETERS

func_tower_count

On input, contains the number of elements in the *func_towers* array, available for use by the licensing system to return information. On output, contains the number of elements actually used within the *func_towers* array. If the return value is XSLM_PARM_ERR and the status value is XSLM_TOO_SMALL, this word contains the number of elements required.

OUTPUT PARAMETERS

func_level

The highest functional level supported by all license servers within the licensing system. Individual license servers may support a higher functional level.

func_towers

An array of optional functional towers that are supported by all license servers within the licensing system.

status

Completion status. Detailed error code directly addressable by the caller. This value is set to XSLM_STATUS_OK if no error occurred. For other messages, see ERRORS.

RETURN VALUE

XSLM_OK

Query completed successfully

XSLM_COMM_ERR

Communications problem

XSLM_CERT_ERR

Problems with the license and/or certificate

XSLM_RESRC_UNAVL

Resources unavailable

XSLM_PARM_ERR

Bad parameters passed

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred
XSLM_COMM_ERR	XSLM_COMM_UNAVAIL	Communications problem
XSLM_RESRC_UNAVL	XSLM_NO_RES	Local platform-specific environmental problems
	XSLM_SERVER_ERROR	Unrecoverable environmental error was encountered by the license server
XSLM_PARM_ERR	XSLM_TOO_SMALL	The number of available space in the func_towers array is insufficient

SEE ALSO

none.

Chapter 8

Management API

This chapter describes all license use management-related API functions. First, there is a listing of all functions, ordered by category. Following this, there is an alphabetical listing of all management API functions.

Note that there is a notion of functional level of the Management API. Refer to Appendix C on page 189 for more information on levels of the Management API.

8.1 Server-Related Functions

XMAPI Function	Usage
xslm_query_servers( )	Provides a list of all addressable servers.
xslm_query_server_info( )	Provides a list of all data elements supported by a specific server.

8.2 Certificate-Related Functions

XMAPI Function	Usage
xslm_get_certificate( )	Retrieves a specific license certificate.
xslm_install_certificate( )	Installs a license certificate, making it available for use.
xslm_query_cert_ids( )	Requests certificate key-information.
xslm_query_next_level_cert_names( )	Requests certificate key-information in external format.
xslm_remove_certificate( )	Removes specific license certificate.
xslm_set_admin_policy( )	Sets administrator-specified certificate-related values.

8.3 License Instance-Related Functions

XMAPI Function	Usage
xslm_get_license_instances( )	Requests all instance-related data for specified certificate key data elements.
xslm_release_license_instance( )	Releases a license-use instance.

8.4 Log-Related Functions

XMAPI Function	Usage
xslm_get_log_data( )	Retrieve one or more log records.

NAME

xslm_get_certificate

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_get_certificate(
    xslm_uuid       server_id,
    xslm_uuid       publisher_id,
    xslm_uint32     product_id,
    xslm_uint32     version_id,
    xslm_uint32     feature_id,
    xslm_uint32     cert_serial_number,
    xslm_uint32     cert_update_seq,
    xslm_uint32     * cert_buffer_length,
    xslm_bin_string * cert_buffer,
    xslm_uint32     * status,
    xslm_uint32     auth_token,
    xslm_bin_string * auth_signature,
    xslm_uuid       * auth_lic_sys_id
);
```

DESCRIPTION

xslm_get_certificate( ) asks the license server to return the license certificate that matches the input parameters, including licensing system-maintained status information and settings provided by the administrator.

INPUT PARAMETERS

server_id
Identification of server to which this request is being directed.

publisher_id
The Software Publisher UUID of the certificate being requested. It unequivocally identifies the Publisher. A non-zero value must be provided.

product_id
ID number of the products certificate being requested. A non-zero value must be provided.

version_id
The version of the products certificate being requested. A non-zero value must be provided.

feature_id
The feature number of the products certificate being requested.

cert_serial_number
The instance id of the products certificate being requested. A non-zero value must be provided.

auth_token
A 32-bit arbitrary value created by the application and used as part of the licensing system authentication process. See Section 4.6 on page 28 for more information.

INPUT/OUTPUT PARAMETERS

cert_buffer_length
As an input parameter, the length of the buffer to receive the results of the call. A value of zero on input indicates a request for the size of the buffer required to contain the entire

results of the call. No data is returned in the buffer when a value of zero is passed.

As an output parameter, the length of data returned, or the size of the buffer required to contain the entire result if the input value was zero.

Note that the buffer must be large enough to contain the entire certificate data area if it is not, no data is placed in the buffer and the output value of *cert_buffer_length* contains the buffer length required to contain the entire result (note that because the certificate state data may change at any time, the actual size needed may be different from the returned value).

OUTPUT PARAMETERS

cert_update_seq
The certificate's update counter. This counter is incremented every time the license certificate values are changed via *xslm_set_admin_policy*( ). To avoid concurrent updates from taking place, the license server will only permit an update via *xslm_set_admin_policy*( ) if the cert_update_seq value passed on that call matches the current value of the license certificates counter.

cert_buffer
Buffer in which the results will be returned.

status
Completion status. Detailed error code directly addressable by the caller. This value is set to XSLM_STATUS_OK if no error occurred. For other messages, see ERRORS.

auth_signature
An area large enough to contain a 16-byte digital signature created by the licensing system from the input parameters and *auth_token*. Used as part of the licensing system authentication process.

auth_lic_sys_id
A unique identifier for the particular type of licensing system handling the current license instance. Used as part of the licensing system authentication process.

RETURN VALUE

XSLM_OK
Query completed successfully

XSLM_COMM_ERR
Communications problem

XSLM_CERT_ERR
Problems with the license and/or certificate

XSLM_RESRC_UNAVL
Resources unavailable

XSLM_PARM_ERR
Bad parameters passed

XLSM_AUTH_ERROR
Requester is not authorized

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred
XSLM_COMM_ERR	XSLM_COMM_UNAVAIL	Communications problem
XSLM_CERT_ERR	XSLM_NO_CERTIFICATES	No certificate matched the get request.
XSLM_RESRC_UNAVL	XSLM_NO_RES	Local platform specific environmental problems
	XSLM_SERVER_ERROR	Unrecoverable environmental error was encountered by the license server
XSLM_PARM_ERR	XSLM_BAD_PARM	One or more parameters were not correct
	XSLM_BAD_BUFFER_LENGTH	The non-zero value for the buffer length was too small for the certificate being retrieved
	XSLM_BAD_SERVER_ID	The *server_id* specified does not correspond to a known server
XSLM_AUTH_ERROR	XSLM_NOT_AUTHORIZED	The requester is not authorized to perform the requested action

PROGRAMMING NOTE

A license certificate key (*publisher_id, product_id, version_id, feature_id,* and *cert_serial_number*) needed to get a certificate can, for example, be obtained by combining a call to *xslm_query_next_level_cert_names*( ) with a call to *xslm_query_cert_ids*( ).

OUTPUT DATA

A data element structure, CERTIFICATE, followed by a data element structure, CERTIFICATE_RELATED_INFORMATION (see Chapter 10 on page 121 for details), unless the buffer is not large enough in which case no data is returned.

SEE ALSO

xslm_query_cert_ids( ).

NAME

xslm_get_license_instances

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_get_license_instances(
    xslm_uuid       server_id,
    xslm_uuid       publisher_id,
    xslm_uint32     product_id,
    xslm_uint32     version_id,
    xslm_uint32     feature_id,
    xslm_uint32     * instance_buffer_length,
    xslm_bin_string * instance_buffer,
    xslm_uint32     * status,
    xslm_uint32     auth_token,
    xslm_bin_string * auth_signature,
    xslm_uuid       * auth_lic_sys_id
);
```

DESCRIPTION

xslm_get_license_instances( ) asks the license server to return the complete set of license instance records which match the specified key fields. When all of the input key parameters have values of zero, the server returns all valid license instance records.

INPUT PARAMETERS

server_id

Identification of server to which this request is being directed.

publisher_id

The software publisher identification for all license instance records to be returned. It uniquely identifies the publisher. A value of zero indicates that instance records for all publishers within the specified *server_id* are to be returned. Whenever a value of zero is used for this parameter, the *product_id*, *version_id*, and *feature_id* parameters must all also have values of zero.

product_id

ID number of the product for which all license instance records are to be returned. A value of zero indicates that all the instance records, which match the *server_id* and *publisher_id* are to be returned. Note that whenever a value of zero is used for this parameter, the *version_id* and *feature_id* parameters must all also have values of zero.

version_id

The version of the product for which all license instance records are to be returned. A value of zero indicates that all the instance records, which match the *server_id*, *publisher_id*, and *product_id*, are to be returned. Note that whenever a value of zero is used for this parameter, the feature_id parameter must also have a value of zero.

feature_id

The associated feature number of the product for which all license instance records are to be returned. A value of zero indicates that all the instance records, which match the *server_id*, *publisher_id*, *product_id*, and *version_id*, are to be returned.

auth_token

A 32-bit arbitrary value created by the application and used as part of the licensing system authentication process. See Section 4.6 on page 28 for more information.

INPUT/OUTPUT PARAMETERS

instance_buffer_length
As an input parameter, the length of the buffer to receive the results of the query. A value of zero on input indicates a request for the size of the buffer required to contain the all the results of the query. No data is returned in the key buffer when a value of zero is passed. As an output parameter the length of usable data returned in the key buffer or the size of the buffer required to contain the entire query result if the input value was zero.

Note that the buffer must be large enough to contain the entire license instances data area. If it is not, no data is placed in the buffer and the output value of this parameter contains the size of the buffer length needed to contain the entire result. Note also that because the license instances data may change at any time, the actual size needed may be different from the returned value.

OUTPUT PARAMETERS

instance_buffer
Pointer to the buffer into which the query results are returned.

status
Completion status. Detailed error code directly addressable by the caller. This value is set of XSLM_OK if no error occurred. For other messages, see ERRORS.

auth_signature
An area large enough to contain a 16-byte digital signature created by the licensing system from the input parameters and *auth_token*. Used as part of the licensing system authentication process.

auth_lic_sys_id
A unique identifier for the particular type of licensing system handling the current license instance. Used as part of the licensing system authentication process.

RETURN VALUE

XSLM_OK
Query completed successfully

XSLM_COMM_ERR
Communications problem

XSLM_CERT_ERR
Problems with the license and/or certificate

XSLM_RESRC_UNAVL
Resources unavailable

XSLM_PARM_ERR
Bad parameters passed

XLSM_AUTH_ERROR
Requester is not authorized

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred
	XSLM_PARTIAL_DATA	Invalid license instance record(s) encountered. Only valid instance records returned.
XSLM_COMM_ERR	XSLM_COMM_UNAVAIL	Communications problem
XSLM_CERT_ERR	XSLM_NO_CERTIFICATES	No certificates matched the query.
XSLM_RESRC_UNAVL	XSLM_NO_RES	Local platform specific environmental problems
	XSLM_SERVER_ERROR	Unrecoverable environmental error was encountered by the license server
XSLM_PARM_ERR	XSLM_BAD_BUFFER_LENGTH	The non-zero value for the buffer length was too small for the data being retrieved
	XSLM_BAD_PARM	One or more parameters were not correct
	XSLM_BAD_SERVER_ID	The *server_id* specified does not correspond to a known server
XSLM_AUTH_ERROR	XSLM_NOT_AUTHORIZED	The requester is not authorized to perform the requested action

OUTPUT DATA

A sequence of 1 or more LICENSE_INSTANCES_INFORMATION data elements (see Chapter 10 on page 121 for details), unless the output buffer is not large enough to contain all data elements matching the query in which case no data is returned.

SEE ALSO

xslm_release_license_instance( ).

NAME

xslm_get_log_data

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_get_log_data(
    xslm_uuid       server_id,
    xslm_uint32     tod_type,
    xslm_tod        log_start,
    xslm_tod        log_end,
    xslm_uint32     log_class,
    xslm_uint32     log_record_type,
    xslm_uint32     log_record_subtype,
    xslm_handle     * query_handle,
    xslm_uint32     log_buffer_length,
    xslm_bin_string * log_buffer,
    xslm_uint32     * status,
    xslm_uint32     auth_token,
    xslm_bin_string * auth_signature,
    xslm_uuid       * auth_lic_sys_id
);
```

DESCRIPTION

xslm_get_log_data( ) requests the license server to return all the log records which match the input parameters.

INPUT PARAMETERS

server_id
Identification of server to which this request is being directed.

tod_type
Indicates whether the time/date values on *log_start* and *log_end* refer to server or application-client local time. A value of XSLM_LOGTOD_SERVER indicates server time; a value of XSLM_LOGTOD_APPL indicates application client time.

log_start
The earliest date for which log records are to be retrieved.

log_end
The last date for which log records are to be retrieved.

log_class
The class ID (XSLM_LOGCLASS_ADMIN, XSLM_LOGCLASS_APPL, or XSLM_LOGCLASS_SYSTEM) of the log records to be retrieved. Use XSLM_LOGCLASS_ANY to retrieve log records of any class.

log_record_type
The value of the type data element of the log records to be retrieved. Use XSLM_LOGTYPE_ANY to retrieve log records for any event type. If XSLM_LOGCLASS_ANY is specified for the log record class ID, then XSLM_LOGTYPE_ANY must be specified for this parameter.

log_record_subtype
The value of the subtype data element of the log records to be retrieved. Use XSLM_LOGSUBTYPE_ANY to retrieve log records for any event subtype. If

XSLM_LOGTYPE_ANY is specified for the log record type, then XSLM_LOGSUBTYPE_ANY must be specified for this parameter.

auth_token
A 32-bit arbitrary value created by the application and used as part of the licensing system authentication process. See Section 4.6 on page 28 for more information.

INPUT/OUTPUT PARAMETERS

log_buffer_length
As an input parameter, the length of the buffer to receive the results of the query. As an output parameter, the length of data returned.

The buffer must be large enough to contain the longest record to be returned. If a record does not fit within the specified buffer size, then a return code of XSLM_PARM_ERR and a status code of XSLM_BAD_BUFFER_LENGTH is returned, and *log_buffer_length* is set to the size required to hold the record.

query_handle
This parameter allows the caller to use a buffer smaller than that required to hold all the results from the query. On the first query, a value of zero must be passed to indicate that this is the first call for this query. If a value of zero is returned on output, all the query results were placed in the key buffer. A non-zero value indicates that the buffer was too small and this value must be used as the input query_handle value to obtain the next part of the query results.

In order to receive all the results of the query, the caller must continue to call *xslm_get_log_data*( ), passing the last returned *query_handle* value until a value of zero is finally returned. Note that the results of the query may not be internally consistent, because changes can occur between calls, whenever multiple calls are required to obtain all the results from a single query.

OUTPUT PARAMETERS

log_buffer
Buffer into which the log records are returned. Each log record is returned as a set of data elements, and the log record itself is identified by a data element. Log records are returned in random order. A server may return more than one log record at the same time, if they all fit within the buffer. See for a list of all data elements.

status
Completion status. Detailed error code directly addressable by the caller. This value is set to XSLM_STATUS_OK if no error occurred. For other messages, see ERRORS.

auth_signature
An area large enough to contain a 16-byte digital signature created by the licensing system from the input parameters and auth_token. Used as part of the licensing system authentication process.

auth_lic_sys_id
A unique identifier for the particular type of licensing system handling the current license instance. Used as part of the licensing system authentication process.

RETURN VALUE

XSLM_OK
Query completed successfully

XSLM_COMM_ERR
Communications problem

XSLM_CERT_ERR
Problems with the license and/or certificate

XSLM_RESRC_UNAVL
Resources unavailable

XSLM_PARM_ERR
Bad parameters passed

XLSM_AUTH_ERROR
Requester is not authorized

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred
	XSLM_PARTIAL_DATA	Invalid license instance record(s) encountered. Only valid instance records returned
XSLM_COMM_ERR	XSLM_COMM_UNAVAIL	Communications problem
XSLM_CERT_ERR	XSLM_NO_CERTIFICATES	No certificates matched the query
XSLM_RESRC_UNAVL	XSLM_NO_RES	Local platform specific environmental problems
	XSLM_SERVER_ERROR	Unrecoverable environmental error was encountered by the license server
XSLM_PARM_ERR	XSLM_BAD_PARM	One or more parameters were not correct
	XSLM_INVALID_TOKEN	*query_handle* contains a value which is not currently valid
	XSLM_BAD_BUFFER_LENGTH	The length of the buffer is too small to hold at least one complete record
	XSLM_BAD_SERVER_ID	*server_id* specified does not correspond to a known server
XSLM_AUTH_ERROR	XSLM_NOT_AUTHORIZED	The requester is not authorized to perform the requested action

OUTPUT DATA

A sequence of 1 or more LOGGED_EVENT data elements (see Chapter 10 on page 121 for details), each containing data about one logged event.

SEE ALSO

None.

NAME

xslm_install_certificate

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_install_certificate(
    xslm_uuid       server_id,
    xslm_uint32     certificate_length,
    xslm_cert_t     * certificate,
    xslm_uint32     * annotation_length,
    xslm_string     * annotation,
    xslm_uint32     data_element_error_offset,
    xslm_uint32     value_error_offset,
    xslm_uint32     * status,
    xslm_uint32     auth_token,
    xslm_bin_string * auth_signature,
    xslm_uuid       * auth_lic_sys_id
);
```

DESCRIPTION

xslm_install_certificate( ) installs a certificate into a license server. The certificate is installed only when all of the following are true: the certificate structure is valid; all the certificate values are valid; this specific certificate is not already installed; except for publisher-specific data elements, all other certificate functions are supported by the server; the certificate has a valid digital signature.

INPUT PARAMETERS

server_id
Identification of server to which this request is being directed.

certificate_length
Length of the certificate structure to be installed.

certificate
The certificate structure to be installed.

annotation
Arbitrary textual data to be maintained together with the certificate.

auth_token
A 32-bit arbitrary value created by the application and used as part of the licensing system authentication process. See Section 4.6 on page 28 for more information.

INPUT/OUTPUT PARAMETERS

annotation_length
Length in bytes of the annotation data. Specify a value of 0 to indicate that no annotation data should be saved. A licensing server must support a length of at least 4,096 bytes, but may support any length longer than this. If the actual value passed in this parameter exceeds the maximum value the licensing server will reject the certificate installation request, and will place in this field the maximum value it can accept.

OUTPUT PARAMETERS

data_element_error_offset
Offset from the start of the certificate structure to the certificate data element where the license server detected the error reported with the status value.

value_error_offset
Offset from the start of the certificate data element to the value where the license server detected the error value reported with the status value. Note that a null value indicates that the data element itself is in error rather than the values which follow the data element.

status
Completion status. Detailed error code directly addressable by the caller. This value is set of XSLM_OK if no error occurred. For other messages, see ERRORS.

auth_signature
An area large enough to contain a 16-byte digital signature created by the licensing system from the input parameters and *auth_token*. Used as part of the licensing system authentication process.

auth_lic_sys_id
A unique identifier for the particular type of licensing system handling the current license instance. Used as part of the licensing system authentication process.

RETURN VALUE

XSLM_OK
Query completed successfully

XSLM_COMM_ERR
Communications problem

XSLM_CERT_ERR
Problems with the license and/or certificate

XSLM_RESRC_UNAVL
Resources unavailable

XSLM_PARM_ERR
Bad parameters passed

XLSM_AUTH_ERROR
Requester is not authorized

PROGRAMMING NOTE

The *xslm_install_certificate*( ) call requests the license server to install the license certificate being passed on the call. For a select set of XSLM_CERT_ERR errors, the license server will return the value of the offset from the beginning of the certificate to the first data element whose associated value represents an invalid value, a certificate structure problem, or a function that is not supported by this license server.

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred. Certificate installed
XSLM_COMM_ERR	XSLM_COMM_UNAVAIL	Communications problem
XSLM_CERT_ERR	XSLM_INVALID_STRUCTURE	Invalid certificate structure encountered
	XSLM_INVALID_VALUES	Invalid certificate value encountered
	XSLM_CERT_VALIDITY_FAILURE	Certificate validity check failed. Certificate content is different to that provided by the license certificate issuer.
	XSLM_DUPLICATE_CERT	License certificate already installed on license server
	XSLM_CERT_NOT_SUPPORTED	License certificate not supported by license server
XSM_RESRC_UNAVAIL	XSLM_NO_RES	Local platform specific environmental problems
	XSLM_SERVER_ERROR	Unrecoverable environmental error was encountered by the license server
XSLM_PARM_ERR	XSLM_BAD_PARM	One or more parameters were not correct
	XSLM_BAD_BUFFER_LENGTH	The value specified for *annotation_length* exceeds the maximum value the server can support
	XSLM_BAD_SERVER_ID	The *server_id* specified does not correspond to a known server
XSLM_AUTH_ERROR	XSLM_NOT_AUTHORIZED	The requester is not authorized to perform the requested action

SEE ALSO

xslm_get_certificate( ), *xslm_remove_certificate*( ).

NAME

xslm_query_cert_ids

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_query_cert_ids(
    xslm_uuid       server_id,
    xslm_uuid       publisher_id,
    xslm_uint32     product_id,
    xslm_uint32     version_id,
    xslm_uint32     feature_id,
    xslm_uint32     * certid_buffer_length,
    xslm_bin_string * certid_buffer,
    xslm_uint32     * status,
    xslm_uint32     auth_token,
    xslm_bin_string * auth_signature,
    xslm_uuid       * auth_lic_sys_id
);
```

DESCRIPTION

xslm_query_cert_ids( ) requests a license server to return the IDs of all certificates belonging to the fully qualified *publisher_id*, *product_id*, *version_id*, and *feature_id*. Note that, unlike for the *xslm_query_next_level_cert_names*( ) call, the text strings for the key fields are not returned as part of the query results.

INPUT PARAMETERS

server_id
Identification of server to which this request is being directed.

publisher_id
The software publisher UUID being queried. It uniquely identifies the publisher.

product_id
ID number of the product being queried.

version_id
The version of the product being queried.

feature_id
The associated feature number of the product being queried.

auth_token
A 32-bit arbitrary value created by the application and used as part of the licensing system authentication process. See Section 4.6 on page 28 for more information.

INPUT/OUTPUT PARAMETERS

certid_buffer_length
As an input parameter, the length of the buffer to receive the results of the query. A value of zero on input indicates a request for the size of the buffer required to contain all the results of the query. No data is returned in the key buffer when a value of zero is passed.

As an output parameter, the length of data returned in the buffer or the size of the buffer required to contain the entire query result if the input value is zero.

Note that the buffer must be large enough to contain the entire output data. If it is not, no data is placed in the buffer and the output value of this parameter contains an estimate of

the buffer length needed to contain the entire result. Note also that because the certificate data may change at any time, the actual size needed may be different from the returned value.

OUTPUT PARAMETERS

certid_buffer
Buffer in which the query results are returned.

status
Completion status. Detailed error code directly addressable by the caller. This value is set to XSLM_OK if no error occurred. For other messages, see ERRORS.

auth_signature
An area large enough to contain a 16-byte digital signature created by the licensing system from the input parameters and *auth_token*. Used as part of the licensing system authentication process.

auth_lic_sys_id
A unique identifier for the particular type of licensing system handling the current license instance. Used as part of the licensing system authentication process.

RETURN VALUE

XSLM_OK
Query completed successfully

XSLM_COMM_ERR
Communications problem

XSLM_CERT_ERR
Problems with the license and/or certificate

XSLM_RESRC_UNAVL
Resources unavailable

XSLM_PARM_ERR
Bad parameters passed

XLSM_AUTH_ERROR
Requester is not authorized

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK XSLM_PARTIAL_DATA	No errors occurred Invalid certificate(s) encountered. Key fields from those certificates were not placed in the key buffer.

Return Value	Status Value	Explanation
XSLM_COMM_ERR	XSLM_COMM_UNAVAIL	Communications problem
XSLM_CERT_ERR	XSLM_NO_CERTIFICATES	No certificates matched the query
XSLM_RESRC_UNAVL	XSLM_NO_RES	Local platform specific environmental problems
	XSLM_SERVER_ERROR	Unrecoverable environmental error was encountered by the license server
XSLM_PARM_ERR	XSLM_BAD_BUFFER_LENGTH	The non-zero value for the buffer length was too small for the data being retrieved
	XSLM_BAD_PARM	One or more parameters were not correct
	XSLM_BAD_SERVER_ID	The server_id specified does not correspond to a known server
XSLM_AUTH_ERROR	XSLM_NOT_AUTHORIZED	The requester is not authorized to perform the requested action

OUTPUT DATA

A sequence of data elements, CERTIFICATE_ID (see Chapter 10 on page 121 for details), describing all certificates matching the specified parameters, that have been installed and not deleted or replaced. The order of the returned data elements is unspecified.

SEE ALSO

xslm_query_next_level_cert_names( ).

NAME

xslm_query_next_level_cert_names

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_query_next_level_cert_names(
    xslm_uuid       server_id,
    xslm_uuid       publisher_id,
    xslm_uint32     product_id,
    xslm_uint32     version_id,
    xslm_uint32     feature_id,
    xslm_uint32     certname_buffer_length,
    xslm_bin_string * certname_buffer,
    xslm_uint32     * status,
    xslm_uint32     auth_token,
    xslm_bin_string * auth_signature,
    xslm_uuid       * auth_lic_sys_id
);
```

DESCRIPTION

xslm_query_next_level_cert_names( ) asks the license server to return the next level of certificate keys under the field that is specified with a non-zero input parameter. When all of the input parameters are zero, the publisher_id and publisher_name field pairs (the top-level key fields) for all certificates are returned. Note that, unlike *xslm_query_cert_ids*( ), this function returns not only the numeric key fields, but also their textual representations.

INPUT PARAMETERS

server_id

Identification of server to which this request is being directed.

publisher_id

The Software Publisher UUID being queried. It unequivocally identifies the Publisher. A value of zero indicates that a list of all publisher IDs, together with the corresponding publisher text strings, which are installed on this license server, be returned. Note that whenever a value of zero is used for this parameter, the *product_id*, *version_id* and *feature_id* parameters must all also have values of zero.

product_id

ID number of the product being queried. A value of zero indicates that a list of all the *product_id*s and product text strings, within the *publisher_id* installed on this license server, be returned. Note that whenever a value of zero is used for this parameter, the *version_id* and *feature_id* parameters must all have values of zero.

version_id

The version of the product being queried. A value of zero indicates that a list of all the version_ids and version text strings, within the *publisher_id* and *product_id* installed on this license server, be returned. Note that whenever a value of zero is used for this parameter, the *feature_id* parameter must also have a value of zero.

feature_id

The associated feature number of the product being queried. A value of zero indicates that a list of all the *feature_id*s and feature text strings, within the *publisher_id*, *product_id* and *version_id* installed on this license server, be returned.

auth_token
A 32-bit arbitrary value created by the application and used as part of the licensing system authentication process. See Section 4.6 on page 28 for more information.

INPUT/OUTPUT PARAMETERS

certname_buffer_length
As an input parameter: the length of the buffer to receive the results of the query. A value of zero on input indicates a request for the size of the buffer required to contain all the results of the query. No data is returned in the key buffer when a value of zero is passed.

As an output parameter: the length of usable data returned in the key buffer or the size of the buffer required to contain the entire query result if the input value were zero.

Note that the buffer must be large enough to contain the entire certificate data area. If it is not, no data is placed in the buffer and the output value of this parameter contains an estimate of the buffer length needed to contain the entire result. Note also that because the certificate state data may change at any time, the actual size needed may be different from the returned value.

OUTPUT PARAMETERS

certname_buffer
Pointer to the buffer into which the query results are returned.

status
Completion status. Detailed error code directly addressable by the caller. This value is set to XSLM_STATUS_OK if no error occurred. For other messages, see ERRORS.

auth_signature
An area large enough to contain a 16-byte digital signature created by the licensing system from the input parameters and *auth_token*. Used as part of the licensing system authentication process.

auth_lic_sys_id
A unique identifier for the particular type of licensing system handling the current license instance. Used as part of the licensing system authentication process.

RETURN VALUE

XSLM_OK
Query completed successfully

XSLM_COMM_ERR
Communications problem

XSLM_CERT_ERR
Problems with the license and/or certificate

XSLM_RESRC_UNAVL
Resources unavailable

XSLM_PARM_ERR
Bad parameters passed

XLSM_AUTH_ERROR
Requester is not authorized

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred. The key buffer contains data.
	XSLM_PARTIAL_DATA	Invalid certificate(s) encountered. Key fields from those certificates were not placed in the key buffer.
XSLM_COMM_ERR	XSLM_COMM_UNAVAIL	Communications problem
XSLM_CERT_ERR	XSLM_NO_CERTIFICATES	No certificates matched the query
XSLM_RESRC_UNAVL	XSLM_NO_RES	Local platform specific environmental problems
	XSLM_SERVER_ERROR	Unrecoverable environmental error was encountered by the license server
XSLM_PARM_ERR	XSLM_BAD_BUFFER_LENGTH	The non-zero value for the buffer length was too small for the data being retrieved
	XSLM_BAD_PARM	One or more parameters were not correct
	XSLM_BAD_SERVER_ID	The *server_id* specified does not correspond to a known server
XSLM_AUTH_ERROR	XSLM_NOT_AUTHORIZED	The requester is not authorized to perform the requested action

OUTPUT DATA

A sequence of data elements, PUBLISHER, PRODUCT, VERSION, or FEATURE (depending upon the input parameters as described above; see Chapter 10 on page 121 for details of the data elements), unless the buffer area is not large enough to contain all output data in which case no data is returned.

SEE ALSO

xslm_query_cert_ids( ).

NAME

xslm_query_server_info

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_query_server_info(
    xslm_uuid       server_id,
    xslm_uint32     * server_buffer_length,
    xslm_bin_string * server_buffer,
    xslm_uint32     * status,
    xslm_uint32     auth_token,
    xslm_bin_string * auth_signature,
    xslm_uuid       * auth_lic_sys_id
);
```

DESCRIPTION

xslm_query_server_info( ) returns the list of XSLM data element IDs supported by the specified server.

INPUT PARAMETERS

server_id

Identification of server to which this request is being directed.

auth_token

A 32-bit arbitrary value created by the application and used as part of the licensing system authentication process. See Section 4.6 on page 28 for more information.

INPUT/OUTPUT PARAMETERS

server_buffer_length

As an input parameter, the length of the buffer to receive the results of the call. A value of zero on input indicates a request for the size of the buffer required to contain the all the results of the call. No data is returned in the server buffer when a value of zero is passed.

As an output parameter the length of usable data returned in the server buffer or the size of the buffer required to contain the entire result if the input value is zero.

Note that the buffer must be large enough to contain the entire result of the call. If it is not large enough, no data is placed in the buffer and the output value of *server_buffer_length* contains the buffer length needed to contain the entire result.

OUTPUT PARAMETERS

server_buffer

Pointer to the buffer which contains a list of all the data element IDs supported by the specified server.

status

Completion status. Detailed error code directly addressable by the caller. This value is set to XSLM_STATUS_OK if no error occurred. For other messages, see ERRORS.

auth_signature

An area large enough to contain a 16-byte digital signature created by the licensing system from the input parameters and auth_token. Used as part of the licensing system authentication process.

auth_lic_sys_id
A unique identifier for the particular type of licensing system handling the current license instance. Used as part of the licensing system authentication process.

RETURN VALUE

XSLM_OK
Query completed successfully

XSLM_COMM_ERR
Communications problem

XSLM_CERT_ERR Problems with the license and /or certificate

XSLM_RESRC_UNAVL
Resources unavailable

XSLM_PARM_ERR
Bad parameters passed

XLSM_AUTH_ERROR
Requester is not authorized

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred
XSLM_COMM_ERR	XSLM_COMM_UNAVAIL	Communications problem
XSLM_RESRC_UNAVL	XSLM_NO_RES	Local platform specific environmental problems
	XSLM_SERVER_ERROR	Unrecoverable environmental error was encountered by the license server
XSLM_PARM_ERR	XSLM_BAD_PARM	One or more parameters were not correct
	XSLM_BAD_BUFFER_LENGTH	The non-zero value for the buffer length was too small for the certificate being retrieved.
	XSLM_BAD_SERVER_ID	The *server_id* specified does not correspond to a known server
XSLM_AUTH_ERROR	XSLM_NOT_AUTHORIZED	The requester is not authorized to perform the requested action

OUTPUT DATA

A data element, LICENSE_SERVER_INFORMATION (see Chapter 10 on page 121 for details).

SEE ALSO

xslm_query_servers( ).

NAME

xslm_query_servers

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_query_servers(
    xslm_uint32     * server_buffer_length,
    xslm_bin_string * server_buffer,
    xslm_uint32     * status
);
```

DESCRIPTION

xslm_query_servers( ) returns server-related information for every server in the licensing system.

INPUT/OUTPUT PARAMETERS

server_buffer_length

As an input parameter, the length of the buffer to receive the results of the call. A value of zero on input indicates a request for the size of the buffer required to contain the all the results of the call. No data is returned in the server buffer when a value of zero is passed.

As an output parameter, the length of data returned in the server buffer or the size of the buffer required to contain the entire result if the input value was zero.

Note that the buffer must be large enough to contain the entire result of the call otherwise, no data is placed in the buffer and the output value of *server_buffer_length* contains the buffer length needed to contain the entire result. Note also that since the server configuration may change at any time, the returned value is approximate only.

OUTPUT PARAMETERS

server_buffer

Pointer to the buffer into which the license systems publisher id and publisher name are returned together with information about each license server.

status

Completion status. Detailed error code directly addressable by the caller. This value is set to XSLM_STATUS_OK if no error occurred. For other messages, see ERRORS.

RETURN VALUE

XSLM_OK
Query completed successfully

XSLM_COMM_ERR
Communications problem

XSLM_CERT_ERR
Problems with the license and/or certificate

XSLM_RESRC_UNAVL
Resources unavailable

XSLM_PARM_ERR
Bad parameters passed

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred
XSLM_COMM_ERR	XSLM_COMM_UNAVAIL	Communications problem
	XSLM_NO_SVRS_FOUND	License server not responding
XSLM_RESRC_UNAVL	XSLM_NO_RES	Local platform specific environmental problems
	XSLM_SERVER_ERROR	Unrecoverable environmental error was encountered by the license server
XSLM_PARM_ERR	XSLM_BAD_PARM	One or more parameters were not correct
	XSLM_BAD_BUFFER_LENGTH	The non-zero value for the buffer length was too small for the certificate being retrieved

OUTPUT DATA

A sequence of data elements, LICENSE_SERVER_ID (see Chapter 10 on page 121 for details), unless the buffer is not large enough to contain the complete results of the query, in which case no data is returned.

SEE ALSO

xslm_query_server_info( ).

NAME

xslm_release_license_instance

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_release_license_instance(
    xslm_uuid       server_id,
    xslm_uuid       publisher_id,
    xslm_uint32     product_id,
    xslm_uint32     version_id,
    xslm_uint32     feature_id,
    xslm_uint32     cert_serial_number,
    xslm_uint32     license_instance_id,
    xslm_uint32     * status,
    xslm_uint32     auth_token,
    xslm_bin_string * auth_signature,
    xslm_uuid       * auth_lic_sys_id
);
```

DESCRIPTION

xslm_release_license_instance( ) asks the license server to release (that is, delete) the license instance record which matches the specified key fields.

INPUT PARAMETERS

server_id
Identification of server to which this request is being directed.

publisher_id
The Software Publisher UUID for the license instance record to be released. It unequivocally identifies the Publisher. A non-zero value must be provided.

product_id
ID number of the product for the license instance record to be released. A non-zero value must be provided.

version_id
The version of the product for the license instance record to be released. A non-zero value must be provided.

feature_id
The feature number of the product for the license instance record to be released.

cert_serial_number
The publisher provided license certificate instance id of the license instance record to be released. A non-zero value must be provided.

license_instance_id
The server created instance id of the license instance record to be released. A non-zero value must be provided.

auth_token
A 32-bit arbitrary value created by the application and used as part of the licensing system authentication process. See Section 4.6 on page 28 for more information.

OUTPUT PARAMETERS

status
Completion status. Detailed error code directly addressable by the caller. This value is set to XSLM_STATUS_OK if no error occurred. For other messages, see ERRORS.

auth_signature
An area large enough to contain a 16-byte digital signature created by the licensing system from the input parameters and *auth_token*. Used as part of the licensing system authentication process.

auth_lic_sys_id
A unique identifier for the particular type of licensing system handling the current license instance. Used as part of the licensing system authentication process.

RETURN VALUE

XSLM_OK
Query completed successfully

XSLM_COMM_ERR
Communications problem

XSLM_CERT_ERR
Problems with the license and/or certificate

XSLM_RESRC_UNAVL
Resources unavailable

XSLM_PARM_ERR
Bad parameters passed

XLSM_AUTH_ERROR
Requester is not authorized

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred
XSLM_COMM_ERR	XSLM_COMM_UNAVAIL	Communications problem
XSLM_CERT_ERR	XSLM_NO_MATCHING_INSTANCE	No license instance matched the request
XSLM_RESRC_UNAVL	XSLM_NO_RES	Local platform specific environmental problems
	XSLM_SERVER_ERROR	Unrecoverable environmental error was encountered by the license server

Return Value	Status Value	Explanation
XSLM_PARM_ERR	XSLM_BAD_PARM	One or more parameters were not correct
	XSLM_BAD_SERVER_ID	The *server_id* specified does not correspond to a known server
XSLM_AUTH_ERROR	XSLM_NOT_AUTHORIZED	The requester is not authorized to perform the requested action

SEE ALSO

xslm_get_license_instances( ).

NAME

xslm_remove_certificate

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_remove_certificate(
    xslm_uuid       server_id,
    xslm_uuid       publisher_id,
    xslm_uint32     product_id,
    xslm_uint32     version_id,
    xslm_uint32     feature_id,
    xslm_uint32     cert_serial_number,
    xslm_uint32     annotation_length,
    xslm_string     * annotation,
    xslm_uint32     * status,
    xslm_uint32     auth_token,
    xslm_bin_string * auth_signature,
    xslm_uuid       * auth_lic_sys_id
);
```

DESCRIPTION

Remove specified license certificate from license server.

INPUT PARAMETERS

server_id
Identification of server to which this request is being directed.

publisher_id
The Software Publisher UUID.

product_id
ID number of the product whose license is requested.

version_id
The version of the product whose license is being installed.

feature_id
The products associated feature number. If not used must be set to binary zeros.

cert_serial_number
The unique serial number assigned to the certificate by the publisher.

annotation
Arbitrary textual data to be logged together with the deletion event data.

auth_token
A 32-bit arbitrary value created by the application and used as part of the licensing system authentication process. See Section 4.6 on page 28 for more information.

INPUT/OUTPUT PARAMETERS

annotation_length
Length in bytes of the annotation data. Specify a value of 0 to indicate that no annotation data should be logged.

A licensing server must support a length of at least 4,096 bytes, but may support any length longer than this. If the actual value passed in this parameter exceeds the maximum value

the licensing server will reject the certificate deletion request, and will place in this field the maximum value it can accept.

OUTPUT PARAMETERS

status
Completion status. Detailed error code directly addressable by the caller. This value is set of XSLM_OK if no error occurred. For other messages, see ERRORS.

auth_signature
An area large enough to contain a 16-byte digital signature created by the licensing system from the input parameters and *auth_token*. Used as part of the licensing system authentication process.

auth_lic_sys_id
A unique identifier for the particular type of licensing system handling the current license instance. Used as part of the licensing system authentication process.

RETURN VALUE

XSLM_OK
Query completed successfully

XSLM_COMM_ERR
Communications problem

XSLM_CERT_ERR
Problems with the license and/or certificate

XSLM_RESRC_UNAVL
Resources unavailable

XSLM_PARM_ERR
Bad parameters passed

XLSM_AUTH_ERROR
Requester is not authorized

PROGRAMMING NOTE

xslm_remove_certificate( ) requests that the license server remove the license certificate identified on the call from the license servers certificate database. Note that certificates can not be removed while any license associated with that certificate is still in use. Also note that consumptive license certificates can not be fully removed from the license server once they are installed, until the certificates expiration date has occurred (that is, the licensing system is required to maintain enough information about such certificates to prevent them from being reinstalled and thus reused).

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred. Certificate removed from license server database.
XSLM_COMM_ERR	XSLM_COMM_UNAVAIL	Communications problem
XSLM_CERT_ERR	XSLM_CERT_NOT_FOUND	License certificate not found
	XSLM_CERT_IN_USE	Licenses associated with certificate still in use
	XSLM_CERT_NOT_REMOVABLE	License certificate type does not support removal
XSLM_RESRC_UNAVL	XSLM_NO_RES	Local platform specific environmental problems
	XSLM_SERVER_ERROR	Unrecoverable environmental error was encountered by the license server
XSLM_PARM_ERR	XSLM_BAD_PARM	One or more parameters were not correct
	XSLM_BAD_BUFFER_LENGTH	The value specified for *annotation_length* exceeds the maximum value the server can support
	XSLM_BAD_SERVER_ID	The *server_id* specified does not correspond to a known server
XSLM_AUTH_ERROR	XSLM_NOT_AUTHORIZED	The requester is not authorized to perform the requested action

SEE ALSO

xslm_get_certificate( ), *xslm_install_certificate*( ).

NAME

xslm_set_admin_policy

SYNOPSIS

```
#include <libxslm.h>

xslm_uint32 xslm_set_admin_policy(
    xslm_uuid       server_id,
    xslm_uuid       publisher_id,
    xslm_uint32     product_id,
    xslm_uint32     version_id,
    xslm_uint32     feature_id,
    xslm_uint32     cert_serial_number,
    xslm_uint32     cert_update_seq,
    xslm_uint32     update_type,
    xslm_uint32     update_element_id,
    xslm_uint32     update_element_seq,
    xslm_uint32     admin_data_length,
    xslm_bin_string * admin_data,
    xslm_uint32     * status,
    xslm_uint32     auth_token,
    xslm_bin_string auth_signature,
    xslm_uuid       * auth_lic_sys_id
);
```

DESCRIPTION

Sets those license-related values, which the license certificate allows an administrator to set or reset (assign nodeID, userID, set confirm time interval, set hard/soft stop policy, etc.).

INPUT PARAMETERS

server_id
Identification of server to which this request is being directed.

publisher_id
The Software Publisher UUID.

product_id
ID number of the product whose license is requested. Unique within the *publisher_id* domain.

version_id
The version of the product whose license is being installed.

feature_id
The products associated feature number. If not used must be set to binary zeros.

cert_serial_number
The unique serial number assigned to the certificate by the publisher.

cert_update_seq
The certificates current update value. This value is updated every time the license certificate values are updated. The license server will only update the certificate, if the *cert_update_seq* value passed on the call matches the license certificates current value. This value is returned as part of *xslm_get_certificate*( ).

update_type
The type of change to be made (XSLM_SET_POLICY_ADD, XSLM_SET_POLICY_DELETE, or XSLM_SET_POLICY_REPLACE).

update_element_id
The data element ID of the element to be operated on.

update_element_seq
The data element sequence number of the element to be operated on, or 0 if unknown.

admin_data_length
Length of administrative policy data.

admin_data
Structure containing a single or multiple sets of self-describing administrative policy data elements. See Chapter 10 on page 121 for the format of the administrative policy fields.

auth_token
A 32-bit arbitrary value created by the application and used as part of the licensing system authentication process. See Section 4.6 on page 28 for more information.

OUTPUT PARAMETERS

status
Completion status. Detailed error code directly addressable by the caller. This value is set of XSLM_OK if no error occurred. For other messages, see ERRORS.

auth_signature
An area large enough to contain a 16-byte digital signature created by the licensing system from the input parameters and *auth_token*. Used as part of the licensing system authentication process.

auth_lic_sys_id
A unique identifier for the particular type of licensing system handling the current license instance. Used as part of the licensing system authentication process.

RETURN VALUE

XSLM_OK
Query completed successfully

XSLM_COMM_ERR
Communications problem

XSLM_CERT_ERR
Problems with the license and/or certificate

XSLM_RESRC_UNAVL
Resources unavailable

XSLM_PARM_ERR
Bad parameters passed

XLSM_AUTH_ERROR
Requester is not authorized

PROGRAMMING NOTE

xslm_set_admin_policy( ) requests that the license server change one or more administrative policies associated with the specified certificate. Policies can be added, changed or removed. Only one policy element can be changed at a time.

ERRORS

The function return value gives an overall indication of the success or failure of the call. In addition, the status parameter will be set to one of the following values, to further help point to the reason for a failed request.

Note: In the future, additional values may be defined for the *status* variable. An application must not rely upon these being the only possible values.

Return Value	Status Value	Explanation
XSLM_OK	XSLM_STATUS_OK	No errors occurred. Certificate successfully updated.
XSLM_COMM_ERR	XSLM_COMM_UNAVAIL	Communications problem
XSLM_CERT_ERR	XSLM_CERT_NOT_FOUND	License certificate not found
	XSLM_UNRECOGNIZED_SEQ	Data element sequence number not found within certificate to be updated
	XSLM_UNRECOGNIZED_ID	Data element ID not recognized by license server
	XSLM_INVALID_VALUE	Value not valid for associated policy data element
	XSLM_UNCHANGABLE_POLICY	Publisher does not permit changes to this policy
	XSLM_NO_LONGER_CHANGABLE	Certificate field has already been changed and can only be changed once
XSLM_RESRC_UNAVL	XSLM_UPDATE_ID_ERROR	The *certificate_update_id* does not match the certificate's current *update_id* value (that is, the certificate has been updated since the time the last *update_id* was obtained)
	XSLM_NO_RES	Local platform specific environmental problems
XSLM_PARM_ERR	XSLM_SERVER_ERROR	Unrecoverable environmental error was encountered by the license server
	XSLM_BAD_PARM	One or more parameters were not correct
	XSLM_BAD_SERVER_ID	The *server_id* specified does not correspond to a known server

Return Value	Status Value	Explanation
XSLM_AUTH_ERROR	XSLM_NOT_AUTHORIZED	The requester is not authorized to perform the requested action

SEE ALSO

xslm_get_certificate( ).

Chapter 9

Recording and Logging

A crucial function of a XSLM licensing system is to collect and record data about the usage of the licensed products and about relevant events related to license management. The information maintained by a XSLM licensing system can be divided in two categories: certificate related data (that is data based on the information present on the original certificate and on the subsequent manipulation of some of these) and historic data. In a different perspective, the information maintained by a XSLM licensing system can be divided into "transient data" and "persistent data". The transient data is the one which is valid only as long as the licensing system is running, and is lost when the licensing system stops, while the persistent data is committed to permanent storage and is valid across license server runs. From this point of view, the certificate related data is made of both transient and persistent data, while the historic data is always persistent.

The certificate-related data is the combination of information provided by the application software publisher (the certificate), information created and maintained by the licensing system, and information provided by the customers license administrator (through a license management tool).

Changes in some of the certificate related data may be triggered by licensing system events such as the request for a new license, or a change in the policy setting (e.g. the administrator switching from soft stop to hard stop) or the expiration of a timer.

The historic data is the persistent log of events relevant to license management. Most of the events related to the license administrator actions are always logged, as they may constitute an audit trail. The logging of events related to license usage is usually under an administrators control, unless the publisher, through information on the certificate, enforces logging.

The physical format of the log file is not specified here. It is up to each licensing system implementation to determine the form (text, binary, or a combination of the two) of the log records, as well as the type of file system(s) supported for storing them.

Likewise, an implementation may choose to implement any kind of proprietary integrity check to insure the records have not been tampered with. However, ultimate responsibility for the integrity and security of the log file, and of the file system of which it is a part, lies with the underlying (operating system and hardware) platform.

9.1 Certificate Related Data

As described above, the certificate related data is comprised of:

- Original certificate data (described earlier)
- Licensing System generated data (described below)
- Administrator defined data (described below).

In a more general meaning, this data is kept alive by the licensing system during the normal operational time and always represents the latest available information. This may or may not be logged (becoming historic data) depending on the administrator, within the limits imposed by the publisher.

9.1.1 Licensing System Generated Data

Each data element described below is related to a specific certificate and uniquely identifies a specific instance of a certificate.

- Number of licensed units available. This is the total number of licensed units acquired by the customer (consolidated from all the valid certificates for this product), minus the number of licensed units presently in use or consumed.
- Number of licensed units in use. For the licensed units in use the licensing system will maintain also the user ID of the application to which they had been granted.
- Publisher High Water Mark (PHWM). The PHWM is the maximum number of licensed units which are (or were) concurrently in use since the last time that the PHWM was reset. In other words, every time that the number of "licensed units in use" is increased, the new value is compared with the PHWM, and if the former is higher, then it becomes the new value for the PHWM. The PHWM is reset to zero by the licensing system with a frequency specified on the certificate, and at the same time this event (PHWM reset to zero) is logged, together with the value of the PHWM. This value is maintained for the possible usage of the vendor, to support licensing policies where the customer can be billed after usage, based on the logged values. This value should be contrasted with AHWM (described below).
- Administrator High Water Mark (AHWM). The AHWM is the maximum number of licensed units which are (or were) concurrently in use since the last time that the AHWM was reset. In other words, every time that the number of "licensed units in use" is increased, the new value is compared with the AHWM, and if the former is higher, then it becomes the new value for the AHWM. The AHWM can be explicitly reset to zero by the customers license administrator. This value is maintained for the convenience of the administrator, to get statistics and usage patterns. This value should be contrasted with PHWM (described above).
- Counters. A certificate may specify that some counters have to be maintained by the Licensing System. A counter can be specified as either cumulative (the counter units will always be added) or consumptive (the counter units will always be subtracted). The meaning of the units is in general known only to the application, and the licensing system keeps track of the units, only acting as a "record keeper". The updating of the counters will be explicitly requested by the application with a *xslm_adv_record*( ) call. An XSLM compliant implementation must support at least eight counters.

9.1.2 Administrator-Defined Data

Each data element described below is related to a specific certificate and uniquely identifies a specific instance of a certificate.

- Assignments.
  Some certificates may require the assignment of the licenses to some entity before they can be used. The certificate may also further specify whether the assignment can be modified after the first assignment has been done. The entities that an administrator may be required to specify are the following:

  — Number of licensed units.
  The administrator may be required to specify the number of "units of use" up to the maximum number specified in the certificate.

  — Node(s).
  The administrator may be required to specify one or more nodes (machines) where the application will be licensed to run.

  — User(s).
  The administrator may be required to specify one or more users which are allowed to use the application. If the certificate so requires, the administrator may have to further specify to which node(s) are the users assigned.

  — Capacity limits.
  The administrator may be required to specify the upper limits of some capacity (resource) for which the application is licensed to run, up the maximum capacity specified in the certificate. The capacities presently defined are the "power" of the processor (expressed in MIPS), the memory size, and the disk(s) size.

  — Initial value for consumptive counters.
  The administrator may be required to specify an initial value for consumptive counters, up to the maximum value specified in the certificate.

- Policy setting.
  Some application software vendors may allow the customer's license administrator to set or override the licensing policy specified in the certificate, and in some cases to reset the value maintained by the licensing system for particular licensing system-generated data. The actions which can be taken by the administrator are the following:

  — Set confirmation interval.
  The time, in seconds, by which the application must issue an *xslm_basic_confirm*( ) or *xslm_adv_confirm*( ) call. The value of this time can be specified by the application, by the certificate or by the administrator (if so allowed in the certificate). See Chapter 10 on page 121 for a detailed description of the order of precedence between conflicting values. In the absence of any specification, a default value of no-limit will be used.

  — Set Hard-Soft Stop policy.
  In some cases the publisher may allow the customer to exceed the licensed values (either in quantity or in time or both) by some additional quantities specified in the certificate. The administrator can instruct the Licensing System (by setting the policy to "Hard Stop") not to make use of these additional quantities and deny a license whenever the licensed values are exceeded. Please note that the setting of this policy to "Hard Stop" does not imply that the application will actually stop. The behavior of an application upon denial of a license is defined by the application publisher.

— Release Licensed Units.
If so allowed in the certificate, the administrator can instruct the Licensing System to release (that is, to make them available) the reusable units presently in use by an application.

— Reset High Water Mark.
At any time the administrator can instruct the Licensing System to reset to zero the value of AHWM.

— Reset Counters.
The administrator can instruct the Licensing System to reset to their initial value the counters (either cumulative or consumptive) which have been specified as "administrator resettable" in the certificate.

— Mask logging of events.
The administrator can instruct the Licensing System to stop the logging of those events which are not mandatory (see below) and which have not been specified as "non-maskable" in the certificate.

— Set Disaster Recovery mode.
The administrator can instruct the Licensing System to enter "Disaster Recovery Mode", if so allowed in the certificate. While in this mode, all restrictions specified on the certificate will be waived. The Licensing System will resume enforcement of the restrictions specified on this certificate when the maximum duration of Disaster Recovery Mode (specified on the certificate) is elapsed or when Disaster Recovery Mode is terminated by the administrator, whichever occurs first.

9.2 Historic Data

A Licensing System will log a number of events, in order for the customer's license administrator (and also for the application software publisher) to derive useful information about the use of the application. The events are classified either as mandatory (that is, the licensing system will always log the event) or optional, that is, the licensing system will always log the event unless instructed not to do so by the administrator:

The logged events are divided into three classes:

- ADMINISTRATION.
  Actions of the administrator (e.g. install certificate, delete certificate)
- APPLICATION.
  Actions of the applications (e.g. request license, release license)
- LICENSING_SYSTEM.
  Actions of the system (e.g. start server, stop server).

In the tables below are described the events logged for each class, and the data logged for each event. The event type and subtype further define which event has occurred, within a class, and can be used as filters for the retrieval of logged events.

9.2.1 Logged Events of Class ADMINISTRATION

Event Type	Event Subtype	Logged Data
INSTALL	NEW The installation of a new Certificate by an Administrator through a management tool into the License Data Base	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO ADMINISTRATOR ADMINISTRATOR_NODE CERTIFICATE_ID INSTALL_ANNOTATION
	REPLACE The replacement of an existing Certificate in the License database with a new one overriding it	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO ADMINISTRATOR ADMINISTRATOR_NODE CERTIFICATE_ID REPLACE_CERTIFICATE INSTALL_ANNOTATION
DELETE	NULL The deletion of a Certificate from the License Data Base	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO ADMINISTRATOR ADMINISTRATOR_NODE CERTIFICATE_ID REMOVE_ANNOTATION
ASSIGN	UNITS The assignment, as performed by an Administrator, of the number of available *units of use* for a specific Certificate, when requested by the publisher	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO ADMINISTRATOR ADMINISTRATOR_NODE CERTIFICATE_ID ASSIGNED_LICENSED_UNITS
	NODES The assignment, as performed by an Administrator, of the Certificate to one or more nodes (machines) to which the usage of the usage of the application will be restricted, if requested by the publisher	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO ADMINISTRATOR ADMINISTRATOR_NODE CERTIFICATE_ID ASSIGNED_NODE_LIST
	USERS The assignment, as performed by an Administrator, of the Certificate to one or more users which will be allowed to use the application, if requested by the publisher	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO ADMINISTRATOR ADMINISTRATOR_NODE CERTIFICATE_ID ASSIGNED_NODE_USER_LIST

Event Type	Event Subtype	Logged Data
	CAPACITY The assignment, as performed by an Administrator, of the Certificate to specific Capacity limits to which the application will be restricted, if allowed by the publisher	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO ADMINISTRATOR ADMINISTRATOR_NODE CERTIFICATE_ID ASSIGNED_CAPACITY_LIST
	CONSUMPTIVE_COUNTERS The assignment, as performed by an Administrator, of the Certificate to a specific initial value for Consumptive Counters, if required by the publisher	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO ADMINISTRATOR ADMINISTRATOR_NODE CERTIFICATE_ID ASSIGNED_CNSMPTV_CNTRS
SET_POLICY	HARD_SOFT_STOP The setting performed by an Administrator of the current policy in effect for a specific Certificate from Soft Stop to Hard Stop and *vice versa*	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO ADMINISTRATOR ADMINISTRATOR_NODE CERTIFICATE_ID HARD_SOFT_STOP_POLICY
	CONFIRM_INTERVAL The setting of the Confirmation Interval, performed by an Administrator, for a specific Certificate within the range (if present) specified in the Certificate	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO ADMINISTRATOR ADMINISTRATOR_NODE CERTIFICATE_ID CONFIRM_INTERVAL_VALUE
	RESET_ADMINISTRATOR _HIGH_WATER_MARK The resetting (to zero), performed by an Administrator, of the value maintained by the licensing system for the maximum number of licenses concurrently in use for a specific Certificate	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO ADMINISTRATOR ADMINISTRATOR_NODE CERTIFICATE_ID HIGH_WATER_MARK_VALUE
	RESET_COUNTERS The resetting performed by an Administrator, of the value maintained by the licensing system for one or more counters, if allowed by the publisher	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO ADMINISTRATOR ADMINISTRATOR_NODE CERTIFICATE_ID ADMIN_RESET_COUNTER_LIST
	MASK_EVENTS The setting, performed by an Administrator, to stop (mask) the logging of one or more events, if allowed by the publisher	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO ADMINISTRATOR

Event Type	Event Subtype	Logged Data
		ADMINISTRATOR_NODE CERTIFICATE_ID MASKED_EVENTS
	RELEASE_UNITS The (forced) release of one or more license units in use, as performed by the Administrator	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO ADMINISTRATOR ADMINISTRATOR_NODE CERTIFICATE_ID FORCED_RELEASE_UNITS
	DISASTER_RECOVERY The setting or resetting, performed by an Administrator, of *"Disaster Recovery Mode*, if allowed by the publisher	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO ADMINISTRATOR ADMINISTRATOR_NODE DISASTER_RECOVERY_MODE

9.2.2 Logged Events of Class APPLICATION

Event Type	Event Subtype	Logged Data
LOG_MESSAGE	NULL A free-form text message as issued by the application and stored in the log file	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO REQUESTOR_ID CERTIFICATE_ID SESSION_HANDLE TRANSACTION_HANDLE LOGGED_MESSAGE RETURN_STATUS
REQUEST_LICENSE	GRANTED The successful granting of one or more licenses by the Licensing System to the requesting application	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO REQUESTOR_ID CERTIFICATE_ID SESSION_HANDLE TRANSACTION_HANDLE REQUESTED_UNITS GRANTED_UNITS RETURN_STATUS
	DENIED The denial by the Licensing System of the license(s) requested by the application	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO REQUESTOR_ID CERTIFICATE_ID SESSION_HANDLE TRANSACTION_HANDLE REQUESTED_UNITS RETURN_STATUS
RELEASE_LICENSE	NULL The successful returning to the pool of available licenses (releasing) of one or more licenses, as performed by the application	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO REQUESTOR_ID CERTIFICATE_ID SESSION_HANDLE TRANSACTION_HANDLE RETURNED_UNITS RETURN_STATUS
CONFIRM	NULL The confirmation performed by the application to the Licensing System, of the in-use status for one or more licenses	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO REQUESTOR_ID CERTIFICATE_ID SESSION_HANDLE TRANSACTION_HANDLE CONFIRM_INTERVAL_VALUE RETURN_STATUS

Event Type	Event Subtype	Logged Data
RECORD	CONSUMPTIVE The successful updating performed by the Licensing System of a consumptive counter, as requested by the application	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO REQUESTOR_ID CERTIFICATE_ID SESSION_HANDLE TRANSACTION_HANDLE COUNTER_UNITS RETURN_STATUS
	CUMULATIVE The successful updating performed by the Licensing System of a cumulative counter, as requested by the application	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO REQUESTOR_ID CERTIFICATE_ID SESSION_HANDLE TRANSACTION_HANDLE COUNTER_UNITS RETURN_STATUS
BEGIN_SESSION	NULL The successful start of a licensing session, as requested by the application	SERVER_TIME_STAMP CLIENT_TIME_STAMP REQUESTOR_ID SESSION_HANDLE RETURNED_STATUS
END_SESSION	NULL The completion of a licensing session, as requested by the application	SERVER_TIME_STAMP CLIENT_TIME_STAMP REQUESTOR_ID SESSION_HANDLE RETURNED_STATUS

9.2.3 Logged Events of Class LICENSING_SYSTEM

Event Type	Event Subtype	Logged Data
LICENSE_SERVER_START	NULL The successful start of the license server activity	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO ADMINISTRATOR_ID
LICENSE_SERVER_STOP	NULL The normal stop of the license server activity	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO ADMINISTRATOR_ID
RESET	PUBLISHER_HIGH_WATER_MARK The resetting (to zero), as performed by the licensing system, of the value maintained for the maximum number of licenses concurrently in use for a specific certificate	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO CERTIFICATE_ID PUBLISHER_HWM_VALUE
	COUNTERS The resetting (to zero), as performed by the licensing system, of the value maintained for one or more counters	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO SYSTEM_RESET_COUNTR_LIST
ERRORS	SYSTEM Any error as detected by the licensing system	SERVER_TIME_STAMP CLIENT_TIME_STAMP LICENSE_SERVER_ID LICENSE_SYSTEM_INFO SYSTEM_ERROR_CODE SYSTEM_ERROR_MESSAGE

Chapter 10

Data Elements

This Chapter contains a detailed description of all the data elements defined in XSLM.

In the following tables, for each data element are indicated only the semantic tag and the type. The tables contain a concise listing of the data elements, with the indentation showing how the data elements are nested one within each other. For data elements of type STRUCT which appear in the table more than once, the indication of the components is done only the first time that they appear. For data elements of type LIST, the component can be repeated any number of times. The column labeled "Required" indicates if a data element is a required or an optional component of the data element in which it appears (the data elements where this column has been left blank are optional). The column labeled "Basic set" indicates if a data element belongs to the set of elements which must be supported by all the XSLM compliant Licensing System. All the data elements not in the Basic set are in the only functional tower presently defined, and their entry has been left blank.

For convenience, in the summary tables the data elements have been divided in three groups:

- The data elements that can appear on the physical certificate (that is, the one created by the application publisher)
- The data elements that are internally maintained by the Licensing System (which are referred to as *state information*)
- The data elements which are logged by the Licensing System and which become *historic data*.

Note that some data elements are repeated in the different tables. The final table provides a detailed description of all the data elements, arranged in alphabetical order.

10.1 Certificate Data Elements

Group certificate - Major data elements	Type	Required	Basic Set
GROUP_CERTIFICATE	STRUCT		
GROUP_TYPE	FIXED	YES	
CERTIFICATE_LIST	LIST	YES	
CERTIFICATE	STRUCT	YES	YES
BASE_SECTION	STRUCT	YES	YES
PUBLISHER_SECTION	STRUCT		
LICENSING_SYSTEM_SECTION_LIST	LIST		
LICENSING_SYSTEM_SECTION	STRUCT	YES	
AUTHENTICATION_SECTION	STRUCT		
GROUP_AUTHENTICATION_SECTION	STRUCT		
AUTHENTICATION_TYPE	FIXED	YES	
AUTHENTICATION_KEY	STRUCT	YES	
SIGNATURE	STRUCT	YES	

Certificate - Data elements by section	Type	Required	Basic Set
PUBLISHER_SECTION	STRUCT		
Application publisher defined	YES		
LICENSING_SYSTEM_SECTION	STRUCT		
PUBLISHER	STRUCT	YES	
PUBLISHER_ID	UUID	YES	
PUBLISHER_NAME	TEXT	YES	
LICENSING_SYSTEM_SPECIFIC_INFO	STRUCT	YES	
Licensing System publisher defined			
AUTHENTICATION_SECTION	STRUCT		
AUTHENTICATION_TYPE	FIXED	YES	
AUTHENTICATION_KEY	STRUCT	YES	
SIGNATURE	STRUCT	YES	
SIGNATURE_DIGEST_ALGORITHM	FIXED	YES	
SIGNATURE_ENCRYPTION_ALGORITHM	FIXED	YES	
SIGNATURE_ENCRYPTED_DIGEST	BSTR	YES	
BASE_SECTION	STRUCT	YES	YES
See Table below			

Certificate - Data elements of BASE_SECTION	Type	Required	Basic Set
FUNCTIONAL_LEVEL	STRUCT	YES	YES
FUNCTIONAL_SPECIFICATION_LEVEL	FIXED	YES	YES
FUNCTIONAL_TOWER_LIST	LIST	YES	YES
FUNCTIONAL_TOWER	FIXED	YES	YES
CERTIFICATE_CREATED	TIME	YES	YES
CERTIFICATE_ID	STRUCT	YES	YES
PUBLISHER_ID	UUID	YES	YES
PRODUCT_ID	FIXED	YES	YES
VERSION_ID	FIXED	YES	YES
FEATURE_ID	FIXED	YES	YES

Certificate - Data elements of BASE_SECTION	Type	Required	Basic Set
CERTIFICATE_SERIAL_NUMBER	FIXED	YES	YES
CERTIFICATE_DESCRIPTION	STRUCT	YES	YES
PUBLISHER_NAME	TEXT	YES	YES
PRODUCT_NAME	TEXT	YES	YES
VERSION_NAME	TEXT	YES	YES
FEATURE_NAME	TEXT	YES	YES
PUBLISHER_USE	TEXT		YES
REPLACE_CERTIFICATE	LIST		
CERTIFICATE_ID	STRUCT	YES	YES
LIFE	STRUCT		YES
LIFE_START	TIME		YES
LIFE_END	TIME		YES
DURATION	STRUCT		YES
DURATION_PERIOD	INTVL	YES	YES
DURATION_START_TYPE	FIXED	YES	YES
DURATION_ADDITIONAL	INTVL		YES
LICENSED_UNITS	STRUCT		YES
LICENSED_UNIT_TYPE	FIXED	YES	YES
LICENSED_UNIT_NUMBER	FIXED	YES	YES
LICENSED_ADDITIONAL_UNITS	FIXED		YES
PUBLISHER_CAPACITY_LIMITS_LIST	LIST		
CAPACITY	STRUCT	YES	
CAPACITY_TYPE	FIXED	YES	
CAPACITY_UNITS	FLOAT	YES	
CAPACITY_ADDITIONAL	FLOAT		
PUBLISHER_ASSIGNMENTS_LIST	LIST		
NODE_USERS_ASSOCIATION	STRUCT	YES	
NODE	STRUCT		YES
NODE_TYPE	FIXED	YES	YES
NODE_ID	BSTR	YES	YES
SUBNODE	STRUCT		
SUBNODE_TYPE	FIXED	YES	
SUBNODE_ID	BSTR	YES	
USER_LIST	LIST		
USER	STRUCT	YES	YES
USER_TYPE	FIXED	YES	YES
USER_ID	BSTR	YES	YES
CERTIFICATE_TARGET_NODES	LIST		YES
NODE	STRUCT	YES	YES
CUSTOMER_ASSIGNABLE_LIMITS	STRUCT		
ASSIGNABLE_UNITS	STRUCT		
LICENSED_UNITS	STRUCT	YES	YES
NOT_REASSIGNABLE	FIXED		
ASSIGNABLE_NODES	STRUCT		
NUMBER_OF_NODES	FIXED	YES	
NOT_REASSIGNABLE	FIXED		
ASSIGNABLE_USERS	STRUCT		
NUMBER_OF_USERS	FIXED	YES	
LINKED_TO_NODE	FIXED		
NOT_REASSIGNABLE	FIXED		
ASSIGNABLE_CAPACITY_LIST	LIST		
ASSIGNABLE_CAPACITY	STRUCT	YES	
CAPACITY	STRUCT	YES	

Certificate - Data elements of BASE_SECTION	Type	Required	Basic Set
NOT_REASSIGNABLE	FIXED		
ASSIGNABLE_CONSUMPTIVE_COUNTERS	LIST		
COUNTER	STRUCT	YES	
COUNTER_ID	FIXED	YES	
COUNTER_NAME	TEXT	YES	
COUNTER_VALUE	FLOAT	YES	
COUNTER_RESETABLE	FIXED		
COUNTER_ADDITIONAL_VALUE	FLOAT		
NOT_REASSIGNABLE	FIXED		
COUNTERS_CONSUMPTIVE	LIST		
COUNTER	STRUCT	YES	
COUNTERS_CUMULATIVE	LIST		
COUNTER	STRUCT	YES	
CONFIRM_INTERVAL	STRUCT	YES	
CONFIRM_INTERVAL_VALUE	INTVL	YES	YES
CONFIRM_INTERVAL_RANGE	STRUCT		YES
CONFIRM_INTERVAL_MIN	INTVL		YES
CONFIRM_INTERVAL_MAX	INTVL		YES
NON_MASKABLE_EVENTS	LIST		YES
EVENT	STRUCT	YES	YES
EVENT_CLASS	FIXED	YES	YES
EVENT_TYPE	FIXED	YES	YES
EVENT_SUBTYPE	FIXED	YES	YES
RESETTING_FREQUENCY	STRUCT		
PUBLISHER_HIGH_WATER_MARK	STRUCT		
RESET_MODE	FIXED	YES	
RESET_INTERVAL	INTVL		
RESETABLE_COUNTERS_LIST	LIST		
RESETABLE_COUNTER	STRUCT	YES	
COUNTER_ID	FIXED	YES	
RESET_MODE	FIXED	YES	
RESET_INTERVAL	INTVL		
LOCALLY_AVAILABLE	FIXED		
DEFAULT_UNITS_TO_GRANT	FIXED		
FORCE_RELEASE_OK	FIXED		
ADVANCE_EXPIRATION_NOTIFICATION	INTVL		
DISASTER_RECOVERY	INTVL		
MULTIUSE_ALLOWED	FIXED		

10.2 State Information

Certificate related information data elements	Type	Required	Basic Set
CERTIFICATE_RELATED_INFORMATION	STRUCT	YES	YES
CERTIFICATE_UPDATE_SEQUENCE	FIXED	YES	YES
CERTIFICATE_INSTALLED	STRUCT	YES	YES
INSTALL_DATE	TIME	YES	YES
INSTALL_ANNOTATION	TEXT		YES
ADMINISTRATOR	STRUCT	YES	YES
ADMINISTRATOR_TYPE	FIXED	YES	YES
ADMINISTRATOR_ID	BSTR	YES	YES
ADMINISTRATOR_NODE	STRUCT	YES	YES
NODE_TYPE	FIXED	YES	YES
NODE_ID	BSTR	YES	YES
SUBNODE	STRUCT		
SUBNODE_TYPE	FIXED	YES	
SUBNODE_ID	BSTR	YES	
CERTIFICATE_REMOVED	STRUCT		YES
REMOVED_DATE	TIME	YES	YES
REMOVE_ANNOTATION	TEXT		YES
ADMINISTRATOR	STRUCT	YES	YES
ADMINISTRATOR_NODE	STRUCT	YES	YES
CERTIFICATE_REPLACEMENT	STRUCT		
REPLACEMENT_DATE	TIME	YES	
CERTIFICATE_ID	STRUCT	YES	YES
PUBLISHER_ID	UUID	YES	YES
PRODUCT_ID	FIXED	YES	YES
VERSION_ID	FIXED	YES	YES
FEATURE_ID	FIXED	YES	YES
CERTIFICATE_SERIAL_NUMBER	FIXED	YES	YES
DURATION_IN_USE	STRUCT	YES	YES
DURATION_START_IN_USE	TIME	YES	YES
DURATION_END_IN_USE	TIME	YES	YES
ADVANCE_EXPIRATION_NOTIF_IN_USE	STRUCT		
PUBLISHER_ASSIGNED_NOTIF_DUE	TIME	YES	
PUBLISHER_ASSIGNED_NOTIF_ISSUED	TIME		
CUSTOMER_ASSIGNED_NOTIF_DUE	TIME		
CUSTOMER_ASSIGNED_NOTIF_ISSUED	TIME		
CONFIRM_CERTIFICATE_INTERVAL_IN_USE	INTVL	YES	YES
MASKED_EVENTS	LIST		YES
EVENT	STRUCT	YES	YES
EVENT_CLASS	FIXED	YES	YES
EVENT_TYPE	FIXED	YES	YES
EVENT_SUBTYPE	FIXED	YES	YES
HARD_SOFT_STOP_INDICATOR	FIXED	YES	
DISASTER_RECOVERY_INDICATOR	STRUCT		
DISASTER_RECOVERY_START	TIME		
DISASTER_RECOVERY_END	TIME		
ASSIGNED_CONFIRM_INTERVAL	INTVL		YES
ASSIGNED_ADVANCE_NOTIFICATION	INTVL		
ASSIGNED_LICENSED_UNITS	FIXED		
ASSIGNED_NODE_LIST	LIST		
NODE	STRUCT	YES	YES
NODE_TYPE	FIXED	YES	YES

Certificate related information data elements	Type	Required	Basic Set
NODE_ID	BSTR	YES	YES
SUBNODE	STRUCT		
SUBNODE_TYPE	FIXED	YES	
SUBNODE_ID	BSTR	YES	
ASSIGNED_NODE_USERS_LIST	LIST		
NODE_USERS_ASSOCIATION	STRUCT	YES	
NODE	STRUCT	YES	YES
USER_LIST	LIST	YES	
USER	STRUCT	YES	YES
USER_TYPE	FIXED	YES	YES
USER_ID	BSTR	YES	YES
ASSIGNED_CAPACITY_LIST	LIST		
CAPACITY	STRUCT	YES	
CAPACITY_TYPE	FIXED	YES	
CAPACITY_UNITS	FLOAT	YES	
ASSIGNED_CONSUMPTIVE_COUNTERS	LIST		
COUNTER_UNITS	STRUCT	YES	
COUNTER_ID	FIXED	YES	
COUNTER_VALUE	FLOAT	YES	
LICENSED_UNITS_CERTIFICATE_IN_USE	FIXED	YES	YES
ADMINISTRATOR_HWM_VALUE	FIXED		
PUBLISHER_HWM_VALUE	FIXED		
CAPACITY_IN_USE	LIST		
CAPACITY	STRUCT	YES	
COUNTERS_CONSUMPTIVE_IN_USE	LIST		
COUNTER_UNITS	STRUCT	YES	
COUNTERS_CUMULATIVE_IN_USE	LIST		
COUNTER_UNITS	STRUCT	YES	
CUSTOMER_ASSIGNED_LIC_SYS_INFO	TEXT		

License instances information data elements	Type	Required	Basic Set
LICENSE_INSTANCES_INFORMATION_LIST	LIST	YES	YES
LICENSE_INSTANCE_INFORMATION	STRUCT	YES	YES
LICENSE_INSTANCE_ID	STRUCT	YES	YES
CERTIFICATE_ID	STRUCT	YES	YES
INSTANCE_NUMBER	FIXED	YES	YES
SESSION_HANDLE	STRUCT		
Licensing System defined			
TRANSACTION_HANDLE	STRUCT	YES	YES
Licensing System defined			
LICENSED_UNITS_INSTANCE_IN_USE	FIXED	YES	YES
CONFIRM_INSTANCE_INTERV. IN_USE	STRUCT	YES	YES
CONFIRM_INTERVAL_VALUE	INTVL	YES	YES
NEXT_CONFIRM_TIME	TIME	YES	YES
REQUESTOR_ID	STRUCT	YES	YES
NODE	STRUCT	YES	YES
USER	STRUCT	YES	YES
USER_ID_FROM_API	BSTR		

License server information data elements	Type	Required	Basic Set
LICENSE_SERVER_INFORMATION	STRUCT	YES	YES
LICENSE_SERVER_ID	STRUCT	YES	
LICENSING_SYSTEM_PUBLISHER	STRUCT	YES	
PUBLISHER_ID	UUID	YES	
PUBLISHER_NAME	TEXT	YES	
LICENSE_SERVER_INSTANCE_ID	UUID	YES	
FUNCTIONAL_LEVEL	STRUCT	YES	YES
NODE	STRUCT	YES	YES
LICENSE_SERVER_DATA_ELEMENTS	LIST	YES	
DATA_ELEMENT_ID	FIXED	YES	
SERVER_START	TIME	YES	YES
SERVER_LAST_STOP	TIME	YES	
DISASTER_RECOVERY_SERVER_INDICATOR	STRUCT	YES	
DISASTER_RECOVERY_START	TIME	YES	
ADMINISTRATOR	STRUCT	YES	YES

10.3 Logged Data Elements

Data elements of LOGGED_EVENT	Type	Required	Basic Set
LOGGED_EVENT	STRUCT	YES	YES
EVENT_CLASS	FIXED	YES	YES
EVENT_TYPE	FIXED	YES	YES
EVENT_SUBTYPE	FIXED	YES	YES
LICENSE_SERVER_ID	STRUCT		
LICENSE_SYSTEM_INFO	STRUCT		YES
LIC_SYS_INFO_DESCRIPTOR	UUID	YES	YES
LIC_SYS_INFO_DATA	BSTR	YES	YES
SERVER_TIME_STAMP	TIME	YES	YES
CLIENT_TIME_STAMP	TIME	YES	YES
LOGGED_DATA	STRUCT	YES	YES
See table below			

Data elements of LOGGED_DATA	Type	Required	Basic Set
LOGGED_DATA	STRUCT	YES	YES
ADMINISTRATOR	STRUCT		YES
ADMINISTRATOR_NODE	STRUCT		YES
CERTIFICATE_ID	STRUCT		YES
INSTALL_ANNOTATION	TEXT		YES
REMOVE_ANNOTATION	TEXT		YES
REPLACE_CERTIFICATE	LIST		
CERTIFICATE_ID	STRUCT	YES	YES
ASSIGNED_LICENSED_UNITS	FIXED		
ASSIGNED_NODE_LIST	LIST		
NODE	STRUCT	YES	YES
ASSIGNED_NODE_USERS_LIST	LIST		
NODE_USERS_ASSOCIATION	STRUCT	YES	
ASSIGNED_CAPACITY_LIST	LIST		
CAPACITY	STRUCT	YES	
ASSIGNED_CONSUMPTIVE_COUNTERS	LIST		
COUNTER_UNITS	STRUCT	YES	
ASSIGNED_CONFIRM_INTERVAL	INTVL		
HARD_SOFT_STOP_POLICY	FIXED		
ASSIGNED_ADVANCE_NOTIFICATION	INTVL		
ADMINISTRATOR_HWM_VALUE	FIXED		
ADMIN_RESET_COUNTER_LIST	LIST		
COUNTER_UNITS	STRUCT	YES	
MASKED_EVENTS	LIST		YES
EVENT	STRUCT	YES	YES
FORCED_RELEASE_UNITS	FIXED		
DISASTER_RECOVERY_MODE	FIXED		
REQUESTOR_ID	STRUCT		
USER_ID_FROM_API	BSTR		
SESSION_HANDLE	STRUCT		
TRANSACTION_HANDLE	STRUCT		
LOGGED_MESSAGE	TEXT		
REQUESTED_UNITS	FIXED		

Data elements of LOGGED_DATA	Type	Required	Basic Set
GRANTED_UNITS	FIXED		
RETURNED_UNITS	FIXED		
RETURN_STATUS	STRUCT		
RETURN_FUNCTION_CODE	FIXED		
RETURN_STATUS_CODE	FIXED		
COUNTER_UNITS	STRUCT		
COUNTER_ID	FIXED	YES	
COUNTER_VALUE	FLOAT	YES	
COUNTER_UPDATE_UNITS	STRUCT		
COUNTER_ID	FIXED	YES	
COUNTER_VALUE	FLOAT	YES	
PUBLISHER_HWM_VALUE	FIXED		
SYSTEM_RESET_COUNTER_LIST	LIST		
COUNTER_UNITS	STRUCT		
DURATION_END_IN_USE	TIME		
APPLICATION_ERROR_CODE	FIXED		
SYSTEM_ERROR_CODE	FIXED		
SYSTEM_ERROR_MESSAGE	TEXT		

10.4 Detailed Data Element Descriptions

Each *Data Element* is described in alphabetic order below, starting with the tabular format:

Data Element Name	Type	Basic Set	ID number

and followed by a description and a list of components.

ADMIN_RESET_COUNTER_LIST	LIST	no	1

This data element contains the list of counters (and their values) when the administrator resets them to their initial value.

Components:

COUNTER_UNITS	One required

ADMINISTRATOR	STRUCT	yes	2

This data element identifies the customer's license administrator performing an action. This information is provided to the License Server by the management agent client library.

Components:

ADMINISTRATOR_TYPE	Required
ADMINISTRATOR_ID	Required

ADMINISTRATOR_HWM_VALUE	FIXED	no	3

This data element contains the value of the Administrator High Water Mark (AHWM). The AHWM is the maximum number of reusable units simultaneously in use since the last time this value was reset to zero, and is logically the same quantity as counted by the Publisher HWM, the only difference being when they are reset, and by whom. At any time the AHWM can be reset to zero by the administrator with an *xslm_set_admin_policy*( ) call.

ADMINISTRATOR_ID	BSTR	no	4

This data element contains a byte string which uniquely identifies the customer's license administrator performing the action which led to the logging of this event. The exact meaning and format of the byte string is determined by the ADMINISTRATOR_TYPE data element. If the value of ADMINISTRATOR_TYPE data element is 1 (login name), the content of this data element must be parsed as if it was of type TEXT. This information is provided to the License Server by the management agent client library.

ADMINISTRATOR_NODE	STRUCT	yes	5

This data element specifies the node from which the administrator made a request to the Licensing System.

Components:

NODE_TYPE	Required
NODE_ID	Required
SUBNODE	Optional

ADMINISTRATOR_TYPE	FIXED	no	6

This data element indicates the format and the meaning of the ADMINISTRATOR_ID data element associated with it. The only instance type presently defined is:

Value = 1	Login name

ADVANCE_EXPIRATION_NOTIFICATION	INTVL	no	7

The presence of this data element in the certificate instructs the Licensing System to log a warning before the expiration of the actual license period. The lead time of the warning is the time period here specified, or a shorter period specified by the customer. If the Licensing System is capable of generating (implementation dependent) alerts, an alert should also be generated.

ADVANCE_EXPIRATION_NOTIFICATION_IN_USE	STRUCT	no	8

This data element consists of data elements which provide information about the publisher and/or customer requested advance notifications associated with the last date for which the application has been licensed.

Components:

PUBLISHER_ASSIGNED	
NOTIFICATION_DUE	Required
PUBLISHER_ASSIGNED	
NOTIFICATION_ISSUED	Optional
CUSTOMER_ASSIGNED	
NOTIFICATION_DUE	Optional
CUSTOMER_ASSIGNED	
NOTIFICATION_ISSUED	Optional

APPLICATION_ERROR_CODE	FIXED	no	9

This data element specifies the type of error, caused by the application, which was detected by the Licensing System.

ASSIGNABLE_CAPACITY	STRUCT	no	10

This data element specifies a capacity which must be assigned by the customer before the Licensing System can grant a license. The customer can assign any value up to the maximum value indicated in this data element.

Components:

CAPACITY	Required
NOT_REASSIGNABLE	Optional

ASSIGNABLE_CAPACITY_LIST	LIST	no	11

The presence of this data element in the certificate indicates that the customer is required to assign a value for all the capacities here specified before the Licensing System can grant a license to the application. The assignment will be done by the customer's license administrator by providing to the Licensing System a list of CAPACITY data elements.

Components:

ASSIGNABLE_CAPACITY	One required

ASSIGNABLE_CONSUMPTIVE_COUNTERS	LIST	no	12

The presence of this data element in the certificate indicates that the customer is required to assign a value to all the consumptive counters here specified before the Licensing System can grant a license to the application. The assignment will be done by the customer's license administrator by providing to the Licensing System a list of COUNTER_UNITS data elements. For each counter, the customer can assign any value up to the maximum number indicated in this data element.

Components:

COUNTER	One required

ASSIGNABLE_NODES	STRUCT	no	13

The presence of this data element in the certificate indicates that the customer is required to assign one or more nodes to this certificate before the Licensing System can grant a license to the application. The assignment will be done by the customer's license administrator by providing to the Licensing System a list of NODE data elements. The customer can assign any number of nodes up to the maximum number indicated in this data element. The publisher can specify also if an assigned NODE can be deleted (or substituted) from the list.

Components:

NUMBER_OF_NODES	Required
NOT_REASSIGNABLE	Optional

ASSIGNABLE_UNITS	STRUCT	no	14

The presence of this data element in the certificate indicates that the customer is required to specify a value for the "units of use" (which then will become available to the application) before the Licensing System can grant a license to the application. The assignment will be done by the customer's license administrator by providing to the Licensing System any value up to the maximum number indicated in this data element.

Components:

LICENSED_UNITS	Required
NOT_REASSIGNABLE	Optional

ASSIGNABLE_USERS	STRUCT	no	15

The presence of this data element in the certificate indicates that the customer is required to assign one or more users to this certificate before the Licensing System can grant a license to the application. The assignment will be done by the customer's license administrator by providing to the Licensing System a list of USER data elements. The customer can assign any number of users up to the maximum number indicated in this data element. The publisher can specify also if an assigned USER can be deleted (or substituted) from the list. All the assignments (and possibly deletions and substitutions) will be logged by the Licensing System.

Components:

NUMBER_OF_USERS	Required
LINKED_TO_NODE	Optional
NOT_REASSIGNABLE	Optional

ASSIGNED_ADVANCE_NOTIFICATION	INTVL	no	16

This data element is a customer assigned time interval which represents the lead time by which the Licensing System must log (issue) a license expiration warning. The value set by the administrator must be shorter than the one specified in the ADVANCE_EXPIRATION_NOTIFICATION data element present in the certificate.

ASSIGNED_CAPACITY_LIST	LIST	no	17

This data element contains the value of the capacities which have been assigned by the customer's license administrator as a consequence of the presence of the ASSIGNABLE_CAPACITY_LIST data element in the certificate. The value of the CAPACITY_UNITS contained in this data element is set by the *xslm_set_admin_policy* ( ) call.

Components:

CAPACITY	Required

ASSIGNED_CONFIRM_INTERVAL	INTVL	no.	18

This data element specifies the confirm interval set by the license administrator for this certificate, when the CONFIRM_INTERVAL data element is present and the license administrator chooses a value within the CONFIRM_INTERVAL_RANGE. The value of this data element is set by the *xslm_set_admin_policy*( ) call.

ASSIGNED_CONSUMPTIVE_COUNTERS	LIST	no	19

This data element contains the list of consumptive counters (and their values) assigned by the administrator, when an assignment is required by the certificate. The value of the COUNTER_VALUE associated with this data element is set by the *xslm_set_admin_policy*( ) call.

Components:

COUNTER_UNITS	One required

ASSIGNED_LICENSED_UNITS	FIXED	no	20

This data element specifies the number of licensed units which have been assigned by the customers license administrator as a consequence of the presence of the ASSIGNABLE_UNITS data element in the certificate. The value of this data element is set by the *xslm*( )_set_admin_policy call.

ASSIGNED_NODE_LIST	LIST	no	21

This data element contains the list of the nodes assigned by the administrator, when an assignment is required by the certificate.

Components:

NODE	One required

ASSIGNED_NODE_USER_LIST	LIST	no	22

This data element contains the list of users assigned by the administrator, when an assignment is required by the certificate. If the certificate further specifies that the users must be "linked to a node", this data element contains also the nodes to which the users have been linked.

Components:

NODE_USERS_ASSOCIATION	One required

AUTHENTICATION_KEY	STRUCT	no	23

This data element of the certificate contains the publisher's public key to be used by the Licensing System in order to verify the integrity of the certificate. The key can be provided just as a byte string (value of the AUTHENTICATION_TYPE data element equal to 1) or as part of a

X.509 certificate (value of the AUTHENTICATION_TYPE data element equal to 2). In the latter case the Licensing System must support the X.509 certificate format provided in this data element.

Components:

X.509 defined	Required

AUTHENTICATION_SECTION	STRUCT	no	24

This data element of the certificate contains the information needed for the Licensing System to verify the integrity of the certificate, and for the application to verify its authenticity. The verification of the integrity is based on a message digest encrypted with a public private key encryption scheme.

Components:

AUTHENTICATION_TYPE	Required
AUTHNTICATION_KEY	Required
SIGNATURE	Required

AUTHENTICATION_TYPE	FIXED	no	25

This data element indicates which type of authentication scheme must be used by the Licensing System in order to verify the integrity of this certificate, and how the publisher's public key is being provided by this certificate. The only type presently specified is the validation based on a message digest encrypted with the publisher's private key, and verified by the Licensing System at certificate installation time by means of the publisher's public key provided on the certificate. If the value of this data element is equal to 1, the AUTHENTICATION_KEY data element contains just the key represented as a byte string; if the value of this data element is equal to 2, the AUTHENTICATION_KEY data element contains the complete X.509 certificate of the publisher.

Value = 1	Bare key
Value = 2	X.509 certificate

BASE_SECTION	STRUCT	yes	26

This data element is required on every certificate, and contains information about the application publisher, the licensed product and all other information used to grant licenses.

Components:

FUNCTIONAL_LEVEL	Required
CERTIFICATE_CREATED	Required
CERTIFICATE_ID	Required
CERTIFICATE_DESCRIPTION	Required
PUBLISHER_USE	Optional
REPLACE_CERTIFICATE	Optional
LIFE	Optional
DURATION	Optional
LICENSED_UNITS	Optional

PUBLISHER_CAPACITY_LIMITS_LIST	Optional
PUBLISHER_ASSIGNMENTS_LIST	Optional
CERTIFICATE_TARGET_NODES	Optional
CUSTOMER_ASSIGNABLE_LIMITS	Optional
COUNTERS_CONSUMPTIVE	Optional
COUNTERS_CUMULATIVE	Optional
CONFIRM_INTERVAL	Optional
NON_MASKABLE_EVENTS	Optional
RESETTING_FREQUENCY	Optional
LOCALLY_AVAILABLE	Optional
DEFAULT_UNITS_TO_GRANT	Optional
FORCE_RELEASE_OK	Optional
ADVANCE_EXPIRATION_NOTIFICATION	Optional
DISASTER_RECOVERY	Optional
MULTI_USE_ALLOWED	Optional

CAPACITY	STRUCT	no	27

This data element specifies the type of capacity and the value of the capacity which should not be exceeded in the node where the application is running.

Components:

CAPACITY_TYPE	Required
CAPACITY_UNITS	Required
CAPACITY_ADDITIONAL	Optional

CAPACITY_ADDITIONAL	FLOAT	no	28

This data element specifies an additional capacity value which can be used by the Licensing System to possibly grant a license (in "soft stop" mode) when the application is running on a node with a capacity greater than the one specified in the CAPACITY_UNITS data element (or in the customer assigned capacity value).

CAPACITY_IN_USE	LIST	no	29

This data element provides a list of data elements each of which specifies the total license capacity in use for this certificate at the current time.

Components:

CAPACITY	Required

CAPACITY_TYPE	FIXED	no	30

This data element indicates the format and the meaning of the CAPACITY_UNIT data element associated with it. The capacity types presently defined are:

Value = 1	MIPS

Value = 2	DASD
Value = 3	Memory

CAPACITY_UNITS	FLOAT	no	31

This data element specifies the capacity value which should not be exceed by the node where the application is running. This information is provided to the Licensing System by the application agent.

CERTIFICATE	STRUCT	yes	32

This is the major data element which encodes the terms and conditions under which the product identified in the certificate is licensed to a customer by the software publisher.

Components:

BASE_SECTION	Required
PUBLISHER_SECTION	Optional
LICENSING_SYSTEM_SECTION_LIST	Optional
AUTHENTICATION_SECTION	Optional

CERTIFICATE_CREATED	TIME	yes	33

This data element of the certificate contains the date and the time when this certificate was created by the application publisher

CERTIFICATE_DESCRIPTION	STRUCT	yes	34

This data element contains the textual representation of the components present in the CERTIFICATE_ID data element, to be used by the Licensing System or a management tool to display information about this certificate in a form more understandable for a human reader.

Components:

PUBLISHER_NAME	Required
PRODUCT_NAME	Required
VERSION_NAME	Required
FEATURE_NAME	Required

CERTIFICATE_ID	STRUCT	yes	35

This data element contains information which identify the publisher who created the certificate, the product this certificate refers to, and a certificate serial number. This information uniquely identify a certificate instance.

Components:

PUBLISHER_ID	Required
PRODUCT_ID	Required

VERSION_ID	Required
FEATURE_ID	Required
CERTIFICATE_SERIAL_NUMBER	Required

CERTIFICATE_INSTALLED	STRUCT	yes	36

This data element contains all the information related to the installation of this certificate.

Components:

INSTALL_DATE	Required
INSTALL_ANNOTATION	Required
ADMINISTRATOR	Required
ADMINISTRATOR_NODE	Required

CERTIFICATE_LIST	LIST	no	37

This data element consists of a list of certificates, related one to each other. If the list has only one certificate, this data element (and the GROUP_CERTIFICATE data element where it is contained) can be omitted.

Components:

CERTIFICATE	One required

CERTIFICATE_RELATED_INFORMATION	STRUCT	yes	38

This data element consists of those certificate related data elements that are not already present on the physical certificate. These data elements are returned by the *get certificate* call and provide the following types of certificate information: installation, replacement, license life, advanced notifications, confirm interval, status of optional log events, hard-stop status, disaster recovery status, customer assignments, *in use* values for licenses granted, capacity, counters and high water marks.

Components:

CERTIFICATE_UPDATE_SEQ	Required
CERTIFICATE_INSTALLED	Required
CERTIFICATE_REMOVED	Optional
DURATION_IN_USE	Optional
ADVANCE_EXPIRATION_NOTIF_IN_USE	Optional
CONFIRM_CERTIFICATE_INTERV.	Optional
MASKED_EVENTS	Optional
HARD_SOFT_STOP_INDICATOR	Optional
DISASTER_RECOVERY_INDICATOR	Optional
ASSIGNED_CONFIRM_INTERVAL	Optional
ASSIGNED_ADVANCE_NOTIFICATION	Optional
ASSIGNED_LICENSED_UNITS	Optional
ASSIGNED_NODE_LIST	Optional
ASSIGNED_NODE_USERS_LIST	Optional
ASSIGNED_CAPACITY_LIST	Optional

ASSIGNED_CONSUMPTIVE_COUNTERS	Optional
LICENSED_UNITS_CERTIFICATE_IN_USE	Required
ADMINISTRATOR_HWM_VALUE	Optional
PUBLISHER_HWM_VALUE	Optional
CAPACITY_IN_USE	Optional
COUNTERS_CONSUMPTIVE_IN_USE	Optional
COUNTERS_CUMULATIVE_IN_USE	Optional

CERTIFICATE_REMOVED	STRUCT	yes	39

This data element contains the information related to the removal or replacement of this certificate. To prevent reinstallation, a certificate can be deleted only at the end of the license duration. A certificate can be replaced at any time, which is equivalent to the deletion of the certificate being replaced, and the installation of the replacement certificate. The Licensing System must maintain information about the replaced certificate, to prevent its reinstallation.

Components:

REMOVED_DATE	Required
REMOVE_ANNOTATION	Required
ADMINISTRATOR	Required
ADMINISTRATOR_NODE	Required
CERTIFICATE_REPLACEMENT	Optional

CERTIFICATE_REPLACEMENT	STRUCT	no	40

This data element contains all the data elements related to the replacement of this certificate, including the replacement date and the certificate id of the replacement certificate.

Components:

REPLACEMENT_DATE	Required
CERTIFICATE_ID	Required

CERTIFICATE_SERIAL_NUMBER	FIXED	yes	41

The value of this data element contains a publisher specified certificate number, which must be unique within all the certificates for this same product, version and feature.

CERTIFICATE_TARGET_NODES	LIST	yes	42

The presence of this data element in the certificate indicates that this certificate can only be installed on one of the nodes specified by this data element. It is expected that the publisher will receive this information from the customer, before creating the certificate. If the data element is not present, the Licensing System will assume that the certificate can be installed on any node.

Components:

NODE	One required

CERTIFICATE_UPDATE_SEQUENCE	FIXED	yes	43

This data element contains the certificate's current update value. This value is an input parameter to the *xslm_set_admin_policy*( ) call and is updated every time this license certificate's values are updated by a *xslm_set_admin_policy*( ) call. The Licensing System will only update the certificate if the value passed by the *xslm_set_admin_policy*( ) call matches the value of this data element.

CLIENT_TIME_STAMP	TIME	yes	44

This data element contains the date and time at which the client library sent to the Licensing System a request which triggered the logging of an event. This information is given to the Licensing System by the client agent library (application or management). For those events where this information is missing or meaningless, it must be set to NULL.

CONFIRM_CERTIFICATE_INTERVAL_IN_USE	INTVL	yes	45

This data element specifies the value of the confirm interval that the Licensing System will use for this certificate. This value is dependent on the values of the ASSIGNED_CONFIRM_INTERVAL and the CONFIRM_INTERVAL data elements. The certificate's confirm interval is established in the following priority order: When present, the value of the ASSIGNED_CONFIRM_INTERVAL determines the confirm interval. If only the CONFIRM_INTERVAL data element is present, that value determines the confirm interval. If none of these data elements are present, the interval value is considered to be infinite.

CONFIRM_INSTANCE_INTERVAL_IN_USE	STRUCT	yes	46

This data element specifies the value of the confirm interval that the Licensing System will use for this license instance. The Licensing System will use the value of the CONFIRM_CERTIFICATE_INTERVAL_IN_USE data element for the license instances confirm interval value, whenever the *request license* and *confirm* calls do not provide a confirm interval value.

Components:

CONFIRM_INTERVAL_VALUE	Required
NEXT_CONFIRM_TIME	Required

CONFIRM_INTERVAL	STRUCT	yes	47

The presence of this data element in the certificate specifies the maximum elapsed time within which the license server should receive the next "confirm" call. If a "confirm" call is not received within this time, the Licensing System may assume that the application ended abnormally and may reclaim the reusable granted units (consumptive units are never reclaimed, as they are "consumed" when they are granted). The value of the confirm interval can be changed by the application (when issuing a "request license" or "confirm" call) or by the customer. This data element also specifies the range (minimum and maximum values) within which the customer may change the value of the confirm interval. In the absence of this data element, the Licensing

System will assume that the confirm interval is infinite, unless a value is set by the application or by the customer. In this last case, the customer can set the confirm interval to any value.

If an application uses the basic API set, the certificate should contain a CONFIRM_INTERVAL_RANGE with CONFIRM_INTERVAL_MIN and CONFIRM_INTERVAL_MAX both set to the confirmation interval assumed by the application.

Components:

CONFIRM_INTERVAL_VALUE	Required
CONFIRM_INTERVAL_RANGE	Optional

CONFIRM_INTERVAL_MAX	INTVL	yes	48

This data element specifies the maximum value of the confirm interval which can be set by the customer. When the value of the confirm interval is set by the application, it is not subject to this maximum. If this data element is NULL or not present, the Licensing System will assume that the maximum value is infinite.

CONFIRM_INTERVAL_MIN	INTVL	yes	49

This data element specifies the minimum value of the confirm interval which can be set by the customer. When the value of the confirm interval is set by the application, it is not subject to this minimum. If this data element is NULL or not present, the Licensing System will assume that the minimum value is zero.

CONFIRM_INTERVAL_RANGE	STRUCT	yes	50

The presence of this data element indicates that the customer is allowed to set the confirm interval value only between a minimum and maximum value specified by this data element. In the absence of this data element, the Licensing System will assume that the customer can set the confirm interval at any value. An application may set any confirmation interval, no matter what value is set by the customer.

Components:

CONFIRM_INTERVAL_MIN	Optional
CONFIRM_INTERVAL_MAX	Optional

CONFIRM_INTERVAL_VALUE	INTVL	yes	51

The value of this data element specifies the maximum elapsed time within which the license server must receive the next "confirm" call. If a "confirm" call is not received within this time, the Licensing System may assume that the application ended abnormally and may reclaim the reusable granted units (consumptive units are never reclaimed, as they are "consumed" when they are granted). The value of the confirm interval can be changed by the application, when issuing a "request license" or "confirm" call, or by the customer.

COUNTER	STRUCT	no	52

This data element identifies a counter (either cumulative or consumptive) that the Licensing System will maintain. The value of a counter can be changed only by the application, via a "record" call. A counter can be reset to its initial value by the Licensing System (in so instructed in the certificate) or by the administrator (if so allowed in the certificate). A counter is identified by its COUNTER_ID, which is a publisher assigned integer, unique within the set of counters specified on the same certificate.

Components:

COUNTER_ID	Required
COUNTER_NAME	Required
COUNTER_VALUE	Required
COUNTER_RESETABLE	Optional
COUNTER_ADDITIONAL_VALUE	Optional
NOT_REASSIGNABLE	Optional

COUNTER_ADDITIONAL_VALUE	FLOAT	no	53

This data element specifies an additional consumptive counter value which can be used by the Licensing System to possibly grant a license (in "soft stop" mode) when the application is requesting a license and all the consumptive counters are negative by a value not greater than the value here specified.

COUNTER_ID	FIXED	no	54

The value of this data element is a publisher assigned integer which uniquely identifies this counter within the set of counters specified in this certificate.

COUNTER_NAME	TEXT	no	55

The value of this data element is a character string which can be used by Licensing System as an external representation of a counter.

COUNTER_RESETABLE	FIXED	no	56

If this data element is present, the customer will be able to reset the actual value of the counter to its initial value that is, the value specified in the certificate. In the case of consumptive counters appearing in the ASSIGNED_CONSUMPTIVE_COUNTERS data element, the resetting of the counter will satisfy the assignment required condition. When present, the value of this data element must be set to zero. If this data element is not present, the Licensing System will assume that the counter is not resettable.

COUNTER_UNITS	STRUCT	no	57

This data element specifies a counter and a value associated with it. Depending on the "parent" data element, this value may be either the actual value of the counter or the quantity by which the counter is being incremented or decremented.

Components:

COUNTER_ID	Required
COUNTER_VALUE	Required

COUNTER_UPDATE_UNITS	STRUCT	no	58

This data element specifies a counter and an update value associated with it. Depending on the type of counter, this value will be added (subtracted) to (from) the counter actual value.

Components:

COUNTER_ID	Required
COUNTER_VALUE	Required

COUNTER_VALUE	FLOAT	no	59

The value of this data element is a number which can represent the current value of a counter, or its initial value, or the maximum value which can be assigned to the counter by the customer, depending on the data element where COUNTER_VALUE appears.

COUNTERS_CONSUMPTIVE	LIST	no	60

The presence of this data element in the certificate indicates that the application will make use of consumptive counters, which will be maintained by the Licensing system. All the counters must have an initial value greater than zero. After an application has been granted a valid license, it will issue "record" calls to instruct the Licensing System to decrement a counter by a given amount (both the COUNTER_ID and the amount are specified in the "record" call). "Record" calls received when the counter is negative will be honored and the counter will be updated. If all the consumptive counters specified in this certificate are less than or equal to zero when a "request license" call is received, the license will not be granted.

Components:

COUNTER	One required

COUNTERS_CONSUMPTIVE_IN_USE	LIST	no	61

This data element provides a list of data elements each of which specifies current value of one of the consumptive counters in use for this certificate.

Components:

COUNTER_UNITS	Required

COUNTERS_CUMULATIVE	LIST	no	62

The presence of this data element indicates that the application will make use of cumulative counters, which will be maintained by the Licensing system. All the counters will have an initial positive or negative value. After an application has been granted a valid license, it will issue "record" calls to instruct the Licensing System to increment a counter by a given amount (both the COUNTER_ID and the amount are specified in the "record" call).

Components:

COUNTER	One required

COUNTERS_CUMULATIVE_IN_USE	LIST	no	63

This data element provides a list of data elements each of which specifies current value of one of the cumulative counters in use for this certificate.

Components:

COUNTER_UNITS	Required

CUSTOMER_ASSIGNABLE_LIMITS	STRUCT	no	64

The presence of this data element in the certificate indicates that the customer must assign a value to the data elements here indicated before the Licensing System can grant a license to the application. The customer can assign any value, up to the limit indicated in the data element.

Components:

ASSIGNABLE_UNITS	Optional
ASSIGNABLE_NODES	Optional
ASSIGNABLE_USERS	Optional
ASSIGNABLE_CAPACITY_LIST	Optional
ASSIGNABLE_CONSUMPTIVE_COUNTERS	Optional

CUSTOMER_ASSIGNED_NOTIFICATION_DUE	TIME	no	65

This data element specifies the date and time at which the customer has requested that the Licensing System log a warning indicating when the license will expire. If the Licensing System is capable of generating (implementation dependent) alerts, an alert should also be generated. The value of this data element is determined by the administrator through a *xslm_set_admin_policy*( ) call which specifies a time interval shorter than the one specified by the ADVANCE_EXPIRATION_NOTIFICATION data element. If the ADVANCE_EXPIRATION_NOTIFICATION data element is not present in the certificate, the administrator is not allowed to set any value, and no notification will be logged.

CUSTOMER_ASSIGNED_NOTIFICATION_ISSUED	TIME	no	66

This data element specifies the date and time at which the Licensing System logged a customer requested warning indicating when the license will expire. This data element is only present when the notification has been issued.

CUSTOMER_ASSIGNED_APPL_INFO	TEXT	no	67

Customer-defined application data. If defined within the certificate, this field is always customer-assignable

CUSTOMER_ASSIGNED_LIC_SYS_INFO	TEXT	no	68

Customer-defined licensing-system data. If defined within the certificate, this field is always customer-assignable

DATA_ELEMENT_ID	FIXED	yes	69

This data element specifies a Data Element ID value.

DEFAULT_UNITS_TO_GRANT	FIXED	no	70

The presence of this data element in the certificate specifies the number of units which will be granted when the application issues a "request license" call where the number of units requested is equal to "default units". If this data element is not present, the Licensing System will assume that the default number of units to grant is 1.

DISASTER_RECOVERY	INTVL	no	71

The presence of this data element in the certificate indicates that all restrictions specified on the certificate will be waived when the customer declares to the Licensing System (in an implementation dependent way) that the environment is now entering Disaster Recovery Mode. In this mode also the restrictions imposed by the CERTIFICATE_TARGET_NODES data element are waived. The Licensing System will resume enforcement of the restrictions specified on this certificate when the maximum duration here specified is elapsed or when Disaster Recovery Mode has ended, whichever occurs first.

DISASTER_RECOVERY_END	TIME	no	72

This data element specifies the date and time on which this certificate will no longer honor disaster recovery mode. This data element value is dependent on the values of the DISASTER_RECOVERY_START and DISASTER_RECOVERY data elements.

DISASTER_RECOVERY_INDICATOR	STRUCT	no	73

This data element contains data elements which track the start of disaster recovery mode for a certificate and the date on which disaster recovery mode is no longer honored by this certificate.

Components:

DISASTER_RECOVERY_START	Required
DISASTER_RECOVERY_END	Required

DISASTER_RECOVERY_MODE	FIXED	no	74

This data element indicates how the administrator set the Disaster Recovery Mode.

Value = 1	Disaster recovery start
Value = 2	Disaster recovery end

DISASTER_RECOVERY_SERVER_INDICATOR	STRUCT	no	75

The presence of this data element indicates that this server is currently in "disaster recovery" mode and will ignore all restrictions for those certificates which contain the DISASTER_RECOVERY data element for the interval specified in this certificate data element.

Components:

DISASTER_RECOVERY_START	Required
ADMINISTRATOR	Required

DISASTER_RECOVERY_START	TIME	no	76

This data element specifies the date and time on which the customer's license administrator declared "disaster recovery mode" to the Licensing System

DURATION	STRUCT	yes	77

The presence of this data element in the certificate specifies the period (as a time interval) during which the application is licensed to the customer. The beginning of this period will be either at the time the certificate is installed or the first time that the application will run and will issue a "request license" call. If this data element is not present, the Licensing System will assume that the duration is the same as the certificate life, and it will start at certificate installation.

Components:

DURATION_PERIOD	Required
DURATION_START_TYPE	Required
DURATION_ADDITIONAL	Optional

DURATION_ADDITIONAL	INTVL	yes	78

This data element specifies an additional duration of the certificate which can be used by the Licensing System to possibly grant a license (in "soft stop" mode) if the application requests a license and the duration period has not expired by more than the time interval here specified. This can be used by the application publisher to provide the customer with a "grace period".

DURATION_END_IN_USE	TIME	yes	79

This data element specifies the actual end date and time for which this certificate is valid. The value of this element is dependent on the presence and values of the DURATION data element.

DURATION_IN_USE	STRUCT	yes	80

This data element consists of the data elements which provide information about the certificate's actual duration.

Components:

DURATION_START_IN_USE	Required
DURATION_END_IN_USE	Required

DURATION_PERIOD	INTVL	yes	81

This data element specifies the period (as a time interval) during which the application is licensed to the customer. The beginning of this period will be either at the time the certificate is installed or the first time that the application will run and will issue a "request license" call, depending on the DURATION_START_TYPE data element.

DURATION_START_IN_USE	TIME	yes	82

This data element specifies the actual start date and time for which this certificate is valid. The value of this data element may be either the date on which the certificate was installed or the date on which the certificate was first used, depending on the value of DURATION_START_TYPE data element present on the certificate.

DURATION_START_TYPE	FIXED	yes	83

This data element specifies if the period during which the application is licensed (DURATION_PERIOD data element) will start at certificate installation or at certificate first use (the first time that the application issues a "request license" call). In both cases the life of the certificate will be extended, if needed, until the end of the duration period.

Value = 1	Start date at certificate installation
Value = 2	Start date at first certificate use

EVENT	STRUCT	yes	84

This data element uniquely identifies a category of events by specifying their class, type and subtype. The EVENT_CLASS data element is required. If one or both of the other data elements are omitted (or specified as NULL) the Licensing System will assume that this data element specifies all the events of the specified class, or all the events of the specified class and type.

Components:

EVENT_CLASS	Required
EVENT_TYPE	Optional
EVENT_SUBTYPE	Optional

EVENT_CLASS	FIXED	yes	85

This data element specifies a class of events that can be logged by the Licensing System

Value = 1	Administration
Value = 2	Application
Value = 3	Licensing System
Value = 9 (XSLM_LOGCLASS_ANY)	Any event class

EVENT_SUBTYPE	FIXED	yes	86

This data element specifies a subtype of events (within a given class and a given type), that can be logged by the Licensing System.

Value = 0	NULL
Value = 10	New
Value = 11	Replace
Value = 20	Units
Value = 21	Nodes
Value = 22	Users
Value = 23	Capacity
Value = 24	Consumptive counters
Value = 30	Hard soft stop
Value = 31	Confirm interval
Value = 32	Advance notification
Value = 33	Reset administrator HWM
Value = 34	Reset counters
Value = 35	Mask events
Value = 36	Release units
Value = 37	Disaster recovery
Value = 40	Granted
Value = 41	Denied
Value = 42	Consumptive
Value = 43	Consumptive zero crossing
Value = 44	Consumptive underflow
Value = 45	Cumulative
Value = 46	Cumulative overflow
Value = 47	Cumulative zero crossing

Value = 50	Publisher high water mark
Value = 51	Counters
Value = 53	System
Value =999 (XSLM_LOGSUBTYPE_ANY)	Any event subtype

EVENT_TYPE	FIXED	yes	87

This data element specifies a type of events (within a given class), that can be logged by the Licensing System.

Value = 1	Install
Value = 2	Delete
Value = 3	Assign
Value = 4	Set policy
Value = 11	Log message
Value = 12	Request license
Value = 13	Release license
Value = 14	Confirm
Value = 15	Record
Value = 16	Begin session
Value = 17	End session
Value = 21	License server start
Value = 22	License server stop
Value = 23	Reset
Value = 24	Expiration warning
Value = 25	Errors
Value =99 (XSLM_LOGTYPE_ANY)	Any event type

FEATURE	STRUCT	yes	88

This data element describes a feature by ID and name.

Components:

FEATURE-ID	Required
FEATURE_NAME	Required

FEATURE_ID	FIXED	yes	89

The value of this data element is a publisher defined integer which uniquely identifies the feature of the product among all the other features of this product.

FEATURE_NAME	TEXT	yes	90

The value of this data element is the textual representation of the version of the product for which this certificate was created, normally used by the Licensing System or a management tool to provide external information about the product in a human readable form.

FORCE_RELEASE_OK	FIXED	no	91

The presence of this data element in the certificate indicates that the administrator can instruct the Licensing System to release the reusable units presently in use by the application. This capability can be useful when the customer may have knowledge that the application is no longer running, while the Licensing System is not aware of this fact. If this data element is not present, the units in use can only be released by the Licensing System when a "confirm" call has failed. The value of this data element must be set to zero.

FORCED_RELEASE_UNITS	FIXED	no	92

This data element contains the number of reusable units which have been released by the administrator.

FUNCTIONAL_LEVEL	STRUCT	yes	93

When used within a license certificate, this data element specifies the minimum level of functionality that a license server must have in order to support this license certificate. If the license server does not have this level of functionality, the certificate will not be installed. This data element also specifies the functional level supported by a license server, and is part of the information maintained by the Licensing System for each license server. This information is returned to a management application in response to the *xslm_query_api_level*( ) and *xslm_query_servers*( ) calls.

Components:

FUNCTIONAL_SPECIFICATION_LEVEL	Required
FUNCTIONAL_TOWER_LIST	Required

FUNCTIONAL_SPECIFICATION_LEVEL	FIXED	yes	94

This data element specifies the specification revision level supported by a license server. A certificate can be installed only by license servers which support at least the functional level specified in the certificate.

The current specification is defined as Value = 1.

FUNCTIONAL_TOWER	FIXED	yes	95

This data element identifies a functional tower supported by a license server. The functional capabilities of the functional towers are defined in the specifications with the revision level indicated in the SPECIFICATION_LEVEL data element. A certificate can be installed only by license servers which support all the functional towers specified in the certificate.

Note that:

1. All license servers must support Basic (Legacy) XMAPI and Basic XLCA.
2. The Advanced XAAPI includes the Basic XAAPI.

Value = 1	Basic XAAPI

Value = 2	Advanced XAAPI
Value = 3	Advanced XMAPI
Value = 4	Advanced XLCA

FUNCTIONAL_TOWER_LIST	LIST	yes	96

This data element specifies a list of functional towers supported by a license server. A certificate can be installed only by license servers which support all the functional towers specified on the certificate

Components:

FUNCTIONAL_TOWER	One required

GRANTED_UNITS	FIXED	yes	97

This data element contains the number of licensed units which have been granted by the Licensing System after receiving a "request license" call from an application. In some cases this number may be different from the number of units requested.

GROUP_AUTHENTICATION_SECTION	STRUCT	no	98

This data element contains the authentication information for all the certificates contained in the GROUP_CERTIFICATE data element. This information will allow the Licensing System to verify the integrity of the certificate, and will allow the application to verify its authenticity. The verification is based on a public private key encryption scheme.

Components:

AUTHENTICATION_TYPE	Required
AUTHENTICATION_KEY	Required
SIGNATURE	Required

GROUP_CERTIFICATE	STRUCT	no	99

This data element consists of a number of complete certificates related one to each other, and an optional global authentication section with the signature of the group of certificates. If only one certificate is present, this data element may be omitted.

Components:

CERTIFICATE_LIST	Required
GROUP_AUTHENTICATION_SECTION	Optional

GROUP_TYPE	FIXED	no	100

This data element identifies the type of relationship that exists between the individual certificates in a group certificate.

Value = 0	Bundle
Value = 1	Suite

HARD_SOFT_STOP_INDICATOR	FIXED	yes	101

This data element specifies whether or not the administrator has placed this certificate in "hard stop" mode. When in "hard stop" mode the Licensing System ignores the values in the following data elements: LICENSED_ADDITIONAL_UNITS, CAPACITY_ADDITIONAL, COUNTER_ADDITIONAL_VALUE and DURATION_ADDITIONAL. The value of this data element is set by the *xslm_set_admin_policy* ( ) call.

Value = 0	Soft stop mode
Value = 1	Hard stop mode

HARD_SOFT_STOP_POLICY	FIXED	yes	102

This data element indicates how the administrator set the hard soft stop policy (at the time this event was logged). In administrator set "hard stop mode" the Licensing System will not make use of any "additional" quantities in order to return a "license granted in soft stop mode". The administrator can change this policy any number of times.

Value = 1	Soft stop mode
Value = 2	Hard stop mode

INSTALL_ANNOTATION	TEXT	yes	103

This data element contains any descriptive text which the administrator chooses to provide when installing this certificate. The value of this data element is set by the *xslm_set_admin_policy* ( ) call.

INSTALL_DATE	TIME	yes	104

This data element specifies the date and time at which this certificate was installed by the Licensing System.

LICENSE_ADMINISTRATOR	STRUCT	yes	105

This data element describes each license administrator defined to this license server. The values for the ADMINISTRATOR_TYPE and ADMINISTRATOR_ID_STRING associated with this data element are set by the *xslm_set_admin_policy* ( ) call.

Components:

ADMINISTRATOR	Required
ADMINISTRATOR_CAPABILITIES_LIST	Required
ADMINISTRATOR_CREATION_DATE	Required

LICENSE_ADMINISTRATOR_LIST	LIST	yes	106

This data element lists the license administrators defined to this license server.

Components:

LICENSE_ADMINISTRATOR	Required

LICENSE_INSTANCE_ID	STRUCT	yes	107

This data element consists of the data elements which uniquely identify a license instance.

Components:

CERTIFICATE_ID	Required
INSTANCE_NUMBER	Required

LICENSE_INSTANCE_INFORMATION	STRUCT	yes	108

This data element consists of the set of data elements containing information about license instances. These data elements are returned by the "get license instances" call and provide the following types of instance information: instance id, handles, license units in use, confirm interval, user id and node id of license requester.

Components:

LICENSE_INSTANCE_ID	Required
SESSION_HANDLE	Required
TRANSACTION_HANDLE	Required
LICENSED_UNITS_INSTANCE_IN_USE	Required
CONFIRM_INSTANCE_INTERVAL_IN_USE	Required
REQUESTOR_ID	Required
USERID_FROM_API	Optional

LICENSE_INSTANCE_INFORMATION_LIST	LIST	yes	109

This data element contains a list of license instances, and is returned as answer to a "get license instances" call.

Components:

LICENSE_INSTANCE_INFORMATION	One required

LICENSE_INSTANCE_NUMBER	FIXED	yes	110

This data element specifies a value provided by the Licensing System which uniquely differentiates this license instance from all other license instances associated with a specific certificate.

LICENSE_SERVER_DATA_ELEMENTS	LIST	yes	111

This data element contains a list of all the data element IDs supported by this licensed server. This data element is returned from the *xslm_query_server_info*( ) call.

Components:

DATA_ELEMENT_ID	One required

LICENSE_SERVER_ID	STRUCT	no	112

This data element uniquely identifies the License Server responsible for the logging of this event.

Components:

LICENSING_SYSTEM_PUBLISHER_ID	Required
LICENSE_SERVER_INSTANCE_ID	Required
FUNCTIONAL_LEVEL	Required
NODE	Required

LICENSE_SERVER_INFORMATION	STRUCT	yes	113

This data element consists of the data elements that describe the license server and its level, the license server's disaster recovery indicator setting, and the license administrators registered to this license server to use the management API. These data elements are returned by the 'query servers' call.

Components:

LICENSE_SERVER_ID	Required
LICENSE_SERVER_DATA_ELEMENTS	Required
SERVER_START	Required
SERVER_LAST_STOP	Required
DISASTER_RECOVERY_SERVER_INDICATOR	Required
LICENSE_ADMINISTRATOR_LIST	Required

LICENSE_SERVER_INSTANCE_ID	UUID	yes	114

This data element identifies the instance of a License Server running on a node. This information is usually provided to the License Server by the Operating System.

LIC_SYS_INFO_DATA	BSTR	yes	115

The data described by the associated LIC_SYS_INFO_DESCRIPTOR element.

LIC_SYS_INFO_DESCRIPTOR	UUID	yes	116

A unique identification, describing the format of data contained in the associated LIC_SYS_INFO_DATA element.

LICENSE_SYSTEM_INFO	STRUCT	yes	117

This data element contains optional information recorded by the licensing system as part of a log record. The contents and format of the information is entirely up to the licensing system.

Components:

LIC_SYS_INFO_DESCRIPTOR	Required
LIC_SYS_INFO_DATA	Required

LICENSED_ADDITIONAL_UNITS	FIXED	yes	118

The value of this data element is a number of additional "units of use" which can be used by the Licensing System to grant a license (in "soft stop" mode) when the number of units requested by the application (with a "request license" call) is greater than the number of available units.

LICENSED_UNIT_NUMBER	FIXED	yes	119

The value of this data element is the number of "licensed units" which are available to the application. The Licensing System will grant a license to an application issuing a "request license" call if the number of units requested is smaller or equal to the number of available units, and will decrease the number of available units by the granted amount. If the units are reusable the number of available units is increased by the granted amount when the application issues a "release license" call. If the units are non-reusable the number of available units is not increased when the application issues a "release license call", as the units have been "consumed" by the application when they have been granted.

LICENSED_UNIT_TYPE	FIXED	yes	120

If the value of this data element is equal to 1 the "licensed units" are reusable, that is, the number of available units is increased by the granted amount when the application issues a "release license" call. If the value of this data element is equal to 2 the "licensed units" are non-reusable, that is, the number of available units is not increased when the application issues a "release license call", as the units have been "consumed" by the application when they have been granted.

Value = 1	Reusable units
Value = 2	Non-reusable units

LICENSED_UNITS	STRUCT	yes	121

This data element of the certificate provides the number of "licensed units" which are available to the licensed application. The units can be reusable, in which case the number of available units is increased by the granted amount when the application issues a "release license" call, or can be non-reusable, in which case the number of available units is not increased when the application issues a "release license call". This data element may contain an additional number of "licensed units", to be used by the Licensing System to grant a license when the number of units requested by the application with a "request license" call is greater than the number of available units. In this case a status will be set to indicate that the license was granted in a "soft stop" condition. If this data element is not present, the Licensing System will assume that the number of "licensed units" is infinite.

Components:

LICENSED_UNIT_TYPE	Required
LICENSED_UNIT_NUMBER	Required
LICENSED_ADDITIONAL_UNITS	Optional

LICENSED_UNITS_CERTIFICATE_IN_USE	FIXED	yes	122

This data element specifies the total number of license units granted for this certificate. When the LICENSED_UNIT_TYPE data element specifies "reusable" units, this data element represents the current number of licensed units in use at this time. When the LICENSED_UNIT_TYPE data element specifies "non-reusable" units, this data element represents the total number of licensed units requested and consumed as this point in time.

LICENSED_UNITS_INSTANCE_IN_USE	FIXED	yes	123

This data element specifies the total number of license units granted for this license instance.

LICENSING_SYSTEM_PUBLISHER	STRUCT	no	124

This data element uniquely identifies a publisher of a Licensing System. It can appear in the certificate (as a component of the LICENSING_SYSTEM_SECTION data element), and is also maintained by the Licensing System as a component of the LICENSE_SERVER_ID data element.

Components:

PUBLISHER_ID	Required
PUBLISHER_NAME	Required

LICENSING_SYSTEM_SECTION	STRUCT	no	125

This data element of the certificate contains information defined for a particular Licensing System, and in general their meaning is understood only by the Licensing System which defines them. Required components of this data element are the ID and the name of the Licensing System publisher, in addition to the Licensing System specific information.

Components:

LICENSING_SYSTEM_PUBLISHER	Required
LICENSING_SYSTEM_SPECIFIC_INFO	Required

LICENSING_SYSTEM_SECTION_LIST	LIST	no	126

This data element of the certificate contains a list of sections, each one specific to a particular Licensing System. If this data element is present, the certificate can only be installed by a Licensing System which supports at least one of the sections.

Components:

LICENSING_SYSTEM_SECTION	One required

LICENSING_SYSTEM_SPECIFIC_INFO	STRUCT	no	127

This data element of the certificate contains information whose exact format and meaning are defined by the Licensing System publisher.

Components:

Licensing System publisher defined	Required

LIFE	STRUCT	yes	128

The presence of this data element in the certificate specifies the time span during which this certificate is valid, that is, it can be used. The certificate can be installed at any time, but no licenses will be granted before the certificate life start date or after the certificate life end date. In the absence of this data element the Licensing System will assume that the certificate is always valid.

Components:

LIFE_START	Optional
LIFE_END	Optional

LIFE_END	TIME	yes	129

This data element specifies the date when the validity of this certificate will end. After this date no licenses will be granted and the certificate can be deleted from the Licensing System. The actual end date of the certificate life can possibly be extended by the time interval specified in the DURATION data element, if the activation of the license was done just before the certificate life end date. If this data element is not present or is specified as NULL, the Licensing System will assume that the certificate will never expire.

LIFE_START	TIME	yes	130

This data element specifies the date when the validity of this certificate will start. The certificate can be installed at any time, but before this date no licenses will be granted. If this data element is not present or is specified as NULL, licenses can be granted immediately after installation.

LINKED_TO_NODE	FIXED	no	131

The presence of this data element in the certificate (as an optional component of the ASSIGNABLE_USERS data element) indicates that the customer must also specify a node to which the named users are assigned. The value of this data element must be set to zero.

LOCALLY_AVAILABLE	FIXED	no	132

The presence of this data element in the certificate indicates that this certificate must be "locally" available to the application, that is, the application should be able to request and release licenses without the need of accessing a network. How this is accomplished is implementation dependent. The value of this data element must be set to zero.

LOGGED_DATA	STRUCT	yes	133

This data element contains data specific to the event being logged. The components of this data element depend on the Class, Type and Subtype of the event being logged. The chapter on Recording and Logging specifies which data elements are required for each combination of Class, Type and Subtype.

Components:

ADMINISTRATOR	Optional
ADMINISTRATOR_NODE	Optional
CERTIFICATE_ID	Optional
INSTALL_ANNOTATION	Optional
REMOVE_ANNOTATION	Optional
REPLACE_CERTIFICATE	Optional
ASSIGNED_LICENSE_UNITS	Optional
ASSIGNED_NODE_LIST	Optional
ASSIGNED_NODE_USERS_LIST	Optional
ASSIGNED_CAPACITY_LIST	Optional
ASSIGNED_CONSUMPTIVE_COUNTERS	Optional
HARD_SOFT_STOP_POLICY	Optional
ASSIGNED_ADVANCE_NOTIFICATION	Optional
CONFIRM_INTERVAL_VALUE	Optional
ADMINISTRATOR_HWM_VALUE	Optional
ADMIN_RESET_COUNTER_LIST	Optional
MASKED_EVENTS	Optional
FORCED_RELEASE_UNITS	Optional
DISASTER_RECOVERY_MODE	Optional
REQUESTOR_ID	Optional
USER_ID_FROM_API	Optional

SESSION_HANDLE	Optional
TRANSACTION_HANDLE	Optional
LOGGED_MESSAGE	Optional
REQUESTED_UNITS	Optional
GRANTED_UNITS	Optional
RETURNED_UNITS	Optional
RETURN_STATUS	Optional
COUNTER_UNITS	Optional
COUNTER_UPDATE_UNITS	Optional
PUBLISHER_HWM_VALUE	Optional
SYSTEM_RESET_COUNTER_LIST	Optional
DURATION_END_IN_USE	Optional
APPLICATION_ERROR_CODE	Optional
SYSTEM_ERROR_CODE	Optional

LOGGED_EVENT	STRUCT	yes	134

This data element contains all the information for an event which is being logged by a specific License Server of a Licensing System. This information is comprised of a set of components which are the same for all the events (the first six components listed below) and a component (namely LOGGED_DATA) whose contents depends on the Class, Type and Subtype of the event being logged.

Components:

EVENT_CLASS	Required
EVENT_TYPE	Required
EVENT_SUBTYPE	Required
LICENSE_SERVER_ID	Required
SERVER_TIME_STAMP	Required
CLIENT_TIME_STAMP	Required
LOGGED_DATA	Required

LOGGED_MESSAGE	TEXT	yes	135

This data element contains the text string which has been passed to the Licensing System by the application, in order for it to be logged.

MASKED_EVENTS	LIST	yes	136

This data element consists of a list of the optional events for this certificate which the administrator requested not to log.

Components:

EVENT	One required

MULTI_USE_ALLOWED	FIXED	no	137

The presence of this data element in the certificate indicates that, for reusable licensed units, the Licensing System must grant a license without increasing the number of units in use (that is, without using any of the available units) when the requesting application satisfies the conditions indicated by the value of this data element.

Value = 1	The request comes from the same node as a previous request
Value = 2	The request comes from the same user as a previous request
Value = 3	The request comes from the same user and the same node as a previous request

NEXT_CONFIRM_TIME	TIME	yes	138

This data element specifies the next date and time by which the application associated with this license instance must issue a "confirm" call, otherwise the Licensing System may assume that the application ended abnormally and may reclaim any reusable licenses in use by this license instance.

NODE	STRUCT	yes	139

This data element specifies a node, and its meaning depends on the data element where it appears. If it appears in the certificate (as a component of CERTIFICATE_TARGET_NODES or NODE_USERS_ASSOCIATION), it specifies a node where the application must run in order to be granted a license. At run time, the information about the node where the application is running is provided to the Licensing System by the application agent, in addition to the information provide by the application through the parameters of the API calls. This information is usually platform dependent, and the publisher may have to receive this information from the customer before creating the certificate. Otherwise NODE, for information purposes, will indicate the node where the application is running, or a license server is running, or a management application is running.

Components:

NODE_TYPE	Required
NODE_ID	Required
SUBNODE	Optional

NODE_ID	BSTR	yes	140

This data element contains a byte string which uniquely identifies a node where the application is licensed to run. The exact meaning and format of the byte string is determined by the NODE_TYPE data element. At run time, this information is provided to the Licensing System by the application agent, or the management agent, or the Operating System.

NODE_TYPE	FIXED	yes	141

This data element indicates the format and the meaning of the NODE_ID data element associated with it. The node types presently defined are:

Value = 1	System board serial number
Value = 2	Network interface MAC address
Value = 3	External Hardware Key
Value = 4	Operating System provided
Value = 5	Licensing System provided

NODE_USERS_ASSOCIATION	STRUCT	no	142

This data element indicates a node where the application must (will) run in order to be granted a license, and a list of users which are licensed to use this application. When the application issues a "request license" call, the Licensing System will grant a license (provided that all other conditions are met) only if the application is running the specified node, and is being used by one of the specified users. The information about the node where the application is running is provided to the Licensing System by the application agent (not the application itself). The information about the user of the application can be provided by the application itself (as one of the parameter of the "request license" call) or by the application agent. This data element can be specified in the certificate by the application publisher, or can be defined by the administrator, if so requested in the certificate (ASSIGNABLE_NODES and ASSIGNABLE_USERS).

Components:

NODE	Required
USER_LIST	Required

NON_MASKABLE_EVENTS	LIST	yes	143

If this data element is present in the certificate, the Licensing System will not allow a license administrator to mask the logging of the optional events listed in this data element. If this data element is not present, the logging of all the events defined as optional will be under the control of the license administrator.

Components:

EVENT	One required

NOT_REASSIGNABLE	FIXED	no	144

The presence of this data element in the certificate indicates that the assignment specified in the parent data element, once done by the customer's license administrator, cannot be "undone", that is, the assignment become permanent and its value cannot be changed. If this data element is not present, the Licensing System will assume that the administrator is allowed to perform the assignment operation any number of times. The value of this data element must be set to one.

NUMBER_OF_NODES	FIXED	no	145

This data element indicates the maximum number of nodes that the customer can assign to this certificate. If this data element is present in the certificate, the customer is required to assign at least one node.

NUMBER_OF_USERS	FIXED	no	146

This data element indicates the maximum number of named users that the customer can assign to this certificate. If this data element is present in the certificate, the customer is required to assign at least one named user.

PRODUCT	STRUCT	yes	147

This data element describes a product by ID and name.

Components:

PRODUCT_ID	Required
PRODUCT_NAME	Required

PRODUCT_ID	FIXED	yes	148

The value of this data element is a publisher defined integer which uniquely identifies this product among all the other products of this publisher.

PRODUCT_NAME	TEXT	yes	149

The value of this data element is the textual representation of the name of the product for which this certificate was created, normally used by the Licensing System or a management tool to provide external information about the product in a human readable form.

PUBLISHER_ASSIGNED_NOTIFICATION_DUE	TIME	no	150

This data element specifies the date and time at which the publisher has requested that the Licensing System log a warning indicating when the license will expire. If the Licensing System is capable of generating (implementation dependent) alerts, an alert should also be generated.

PUBLISHER_ASSIGNED_NOTIFICATION_ISSUED	TIME	no	151

This data element specifies the date and time at which the Licensing System logged a publisher requested warning indicating when this certificate will expire. This data element is only present when the notification has been issued.

PUBLISHER_ASSIGNMENTS_LIST	LIST	no	152

The presence of this data element in the certificate indicates that the use of the application has been restricted to the set of users here specified, and can only run on the nodes here specified. Each data element of the list associates a node with a list of users. The application can only be used by those users identified in the user list, when the application is running on the specified node. The node can be indicated as NULL, and the meaning is that the application can only be used by those users identified in the user list, regardless of where the application is running. The list of users can be indicated as NULL, and the meaning is that the application can be used by any user when it is running on the specified node. The restrictions specified in each node-users association are independent one from the other.

If this data element is not present, the Licensing System will assume that the application can run on any node and can be used by any user.

Components:

NODE_USERS_ASSOCIATION	One required

PUBLISHER_CAPACITY_LIMITS_LIST	LIST	no	153

The presence of this data element in the certificate indicates that the application is licensed to run on nodes which do not exceed the capacity limits specified by this data element. The capacity can be based on CPU power such as MIPS, or memory size, or megabytes of storage and the like. The information about the capacity of the node where the application is running will be provided to the Licensing System by the application agent, and not by the application itself.

Components:

CAPACITY	One required

PUBLISHER_HIGH_WATER_MARK	STRUCT	no	154

The presence of this data element in the certificate specifies when the Licensing System must reset to zero the value of the Publisher High Water Mark (PHWM, see the description for the PUBLISHER_HWM_VALUE data element). If this data element is not present, the value of PHWM will always increase.

Components:

RESET_MODE	Required
RESET_INTERVAL	Optional

PUBLISHER_HWM_VALUE	FIXED	no	155

This data element contains the value of the Publisher High Water Mark (PHWM). The PHWM is the maximum number of reusable units simultaneously in use since the last time this value was reset to zero, and is logically the same quantity as counted by the Administrator HWM, the only difference being when they are reset, and by whom. The PHWM is reset to zero by the Licensing System with a frequency specified in the RESETTING_FREQUENCY data element.

PUBLISHER	STRUCT	yes	156

This data element describes a publisher by ID and name.

Components:

PUBLISHER_ID	Required
PUBLISHER_NAME	Required

PUBLISHER_ID	UUID	yes	157

This data element uniquely identifies (by means of a UUID)the publisher of the application software or the publisher of the Licensing System, depending on the "parent" data element where it appears.

PUBLISHER_NAME	TEXT	yes	158

This data element provides the textual representation of the name of the publisher, normally used by the Licensing System or a management tool to display external information about the publisher in a human readable form.

PUBLISHER_SECTION	STRUCT	no	159

This data element of the certificate contains publisher defined information, which will be passed to the application by the Licensing System in answer to a "query" call .The components of this data element will be publisher defined.

Components:

Publisher defined	Required

PUBLISHER_USE	TEXT	yes	160

This data element contains additional publisher defined information, which can be used either by the application (which can retrieve it via a "query" call) or by the Licensing System or a management tool for external display. This data element can be used as an alternative to the PUBLISHER_SECTION data element in those Licensing System which do not support the latter.

REMOVE_ANNOTATION	TEXT	yes	161

This data element contains any descriptive text which the administrator chooses to provide when requesting that this certificate be removed or replaced. The value of this data element is set by the *xslm_set_admin_policy* ( ) call.

REMOVED_DATE	TIME	yes	162

This data element specifies the date and time at which this certificate was removed from the Licensing System per the administrator's request or was replaced by a replacement certificate.

REPLACE_CERTIFICATE	LIST	no	163

The presence of this data element in the certificate indicates that this entire certificate is to replace one or more previously installed certificates. The certificates to be replaced are identified by a list of CERTIFICATE_ID data elements. If a match is found for all the CERTIFICATE_IDs, the certificates being replaced are "logically deleted" from the Licensing System and the present certificate is installed. However the Licensing System must maintain enough information to prevent the re-installation of the certificates being replaced. If a match is not found, no replacement will occur.

Components:

CERTIFICATE_ID	One required

REPLACEMENT_DATE	TIME	no	164

This data element specifies the date and time at which this certificate was replaced by another certificate.

REQUESTED_UNITS	FIXED	yes	165

This data element contains the number of licensed units which have been requested by the application with a "request license" call.

REQUESTOR_ID	STRUCT	no	166

This data identifies the requester of a license on the part of the Licensing System. The requester is the combination of a node and a user, and is provided to the Licensing System by the application client agent.

Components:

NODE	Required
USER	Required

RESET_INTERVAL	INTVL	no	167

This data element specifies the value of the time interval at which the Licensing System must reset to zero the value of the Publisher High Water Mark, or reset to their initial value the resettable counters. If this data element is not present the Licensing System will assume that the resetting interval is infinite, unless the RESET_MODE data element specifies a value other than 1.

RESET_MODE	FIXED	no	168

This data element specifies the frequency with which the Licensing System must reset to zero the value of the Publisher High Water Mark, or reset to their initial value the resettable counters.

Value = 1	Frequency in RESET_INTERVAL
Value = 2	Every hour
Value = 3	Every day
Value = 4	Every week
Value = 5	Every month
Value = 6	Every year

RESETABLE_COUNTER	STRUCT	no	169

This data element indicates a counter (consumptive or cumulative) whose value must be reset to the initial value by the Licensing System, and the frequency of the resetting action.

Components:

COUNTER_ID	Required
RESET_MODE	Optional
RESET_INTERVAL	Optional

RESETABLE_COUNTERS_LIST	LIST	no	170

This data element specifies the counters (consumptive or cumulative) whose value must be reset to the initial value by the Licensing System, and the frequency of the resetting action.

Components:

RESETABLE_COUNTER	One required

RESETTING_FREQUENCY	STRUCT	no	171

This data element specifies the data elements whose value is to be reset to an initial value by the Licensing System , and the frequency of the resetting action.

Components:

PUBLISHER_HIGH_WATER_MARK	Optional
RESETABLE_COUNTERS	Optional

RETURN_FUNCTION_CODE	FIXED	yes	172

This data element contains the code being returned by the Licensing System as the value of the function being invoked by the client library (application or management). The possible values and meaning of this data element are described for each API function.

RETURN_STATUS	STRUCT	yes	173

This data element contains the values being returned by the Licensing System at the completion of an API call.

Components:

RETURN_FUNCTION_CODE	Required
RETURN_STATUS_CODE	Optional

RETURN_STATUS_CODE	FIXED	yes	174

This data element contains the code being returned by the Licensing System as the output parameter value which appears in each function being invoked by the client library (application or management).

The possible values and meaning of this data element are described for each API function.

RETURNED_UNITS	FIXED	yes	175

This data element contains the number of reusable licensed units which have been returned by the application with a "release license" call. This number must match the number of units granted by the Licensing System in a previous "request license" call.

SERVER_LAST_STOP	TIME	yes	176

This data element specifies the date and time that this license server was last stopped.

SERVER_START	TIME	yes	177

This data element specifies the date and time that this license server was last started.

SERVER_TIME_STAMP	TIME	yes	178

This data element contains the date and time at which the License Server logged this event.

SESSION_HANDLE	STRUCT	yes	179

This data element contains the reference established by the Licensing System in order to uniquely identify a session of an application with the Licensing System. The components of this data element are implementation dependent and are defined by the Licensing System publisher.

Components:

Licensing System defined

SIGNATURE	STRUCT	no	180

This data element of the certificate contains the specification of the digest algorithm used to generate the message digest, the encryption algorithm used to encrypt the digest, and the encrypted message digest itself.

Components:

SIGNATURE_DIGEST_ALGORITHM	Required
SIGNATURE_ENCRYPTION_ALGORITHM	Required
SIGNATURE_ENCRYPTED_DIGEST	Required

SIGNATURE_DIGEST_ALGORITHM	FIXED	no	181

This data element specifies the algorithm used to generate the digest of the certificate. Presently the only algorithm defined is MD5.

Value = 1	MD5

SIGNATURE_ENCRYPTED_DIGEST	BSTR	no	182

This data element contains the digest of the certificate, encrypted with the publisher's private key. The input to the digest algorithm consists of the whole certificate, except this data element itself.

SIGNATURE_ENCRYPTION_ALGORITHM	FIXED	no	183

This data element specifies the algorithm used to encrypt (with the publisher's private key) the digest of the certificate. Presently the only algorithm defined is RSA.

Value = 1	RSA

SUBNODE	STRUCT	no	184

This data element contains further identification of a node, for those platforms where the notion of "sub-node" is relevant

Components:

SUBNODE_TYPE	Required
SUBNODE_ID	Required

SUBNODE_ID	BSTR	no	185

This data element contains a byte string which uniquely identifies a node where the application is licensed to run. The exact meaning and format of the byte string is determined by the NODE_TYPE data element. At run time, this information is provided to the Licensing System by the client library (application or management agent).

SUBNODE_TYPE	FIXED	no	186

This data element indicates the format and the meaning of the SUBNODE_ID data element associated with it. Only one sub-node type is presently defined.

Value = 1	LPAR

SYSTEM_ERROR_CODE	FIXED	no	187

This data element specifies the type of error which was detected by the Licensing System in the platform where the License Server was running. The error may have been caused by the Licensing system itself, by the Operating System, or by the hardware. System error codes are the implementation-defined values.

SYSTEM_ERROR_MESSAGE	TEXT	no	188

This data element contains the implementation-defined error messages.

SYSTEM_RESET_COUNTER_LIST	LIST	no	189

This data element contains the list of counters and their values when the Licensing System resets them to their initial value, with a frequency specified by the application publisher in the certificate.

Components:

COUNTER_UNITS	One required

TRANSACTION_HANDLE	STRUCT	yes	190

This data element contains the reference established by the Licensing System in order to uniquely identify a "request license" transaction with an application. The components of this data element are implementation dependent and are defined by the Licensing System publisher.

Components:

Licensing System defined

USER	STRUCT	yes	191

This data element identifies a user to which the application has been licensed. At run time the information about the user of the application is provided to the Licensing System by the application agent by means of information derived from the operating system. Additional information may be provided by the application itself, through a parameter of the "request license" call.

Components:

USER_TYPE	Required

USER_ID	Required

USER_ID	BSTR	yes	192

The value of this data element is a byte string which uniquely identifies a user of an application. The exact meaning and format of the byte string is determined by the USER_TYPE data element. If the value of USER_TYPE data element is 1 (login name), the content of this data element must be parsed as if it was of type TEXT. At run time this information is provided to the Licensing System by the application agent by means of information derived from the operating system. Additional information may be provided by the application itself through a parameter of the "request license" call (e.g. an application server which provides the identity of the application clients "logging on" to the application).

USER_ID_FROM_API	STRUCT	yes	193

This data element identifies the user to which the application is licensed and is passed by the "request" call. The user type is the same as that defined by the USER_TYPE data element.

USER_LIST	LIST	no	194

This data element is a list of users to which the application has been licensed. At run time the information about the user of the application is provided to the Licensing System by the application itself , through a parameter of the "request license" call, and also by the application agent by means of information derived from the operating system.

Components:

USER	One required

USER_TYPE	FIXED	yes	195

The value of this data element indicates the format and the meaning of the USER_ID data element associated with it. The user types presently defined are:

Value = 1	Login name

VERSION	STRUCT	yes	196

This data element describes a publisher by ID and name.

Components:

VERSION_ID	Required
VERSION_NAME	Required

VERSION_ID	FIXED	yes	197

The value of this data element is a publisher defined integer which uniquely identifies the version of the product among all the other versions of this product.

VERSION_NAME	TEXT	yes	198

The value of this data element is the textual representation of the version of the product for which this certificate was created, normally used by the Licensing System or a management tool to provide external information about the product in a human readable form.

10.5 Defined Symbols and their Assigned Values

Symbol	Assigned Value
XSLM_AUTH_ERROR	151
XSLM_BAD_BUFFER_LENGTH	101
XSLM_BAD_LICENSE_HANDLE	102
XSLM_BAD_PARM	103
XSLM_BAD_SERVER_ID	104
XSLM_BAD_SESSION_HANDLE	105
XSLM_CAPACITY_LIMIT	106
XSLM_CERT_AUTH_NONE	0
XSLM_CERT_AUTH_PUB_KEY	1
XSLM_CERT_AUTH_CA	2
XSLM_CERT_ERR	2
XSLM_CERT_EXP	107
XSLM_CERT_IN_USE	108
XSLM_CERT_NOT_FOUND	109
XSLM_CERT_NOT_REMOVABLE	110
XSLM_CERT_NOT_STARTED	111
XSLM_CERT_NOT_SUPPORTED	112
XSLM_CERT_VALIDITY_FAILURE	113
XSLM_COMM_ERR	1
XSLM_COMM_UNAVAIL	114
XSLM_COUNT_OVERFLOW	115
XSLM_COUNT_UNDERFLOW	116
XSLM_DEFAULT_UNITS	0
XSLM_DUPLICATE_CERT	117
XSLM_GRANT_FULL	1
XSLM_GRANT_PARTIAL	2
XSLM_INVALID_API_USE	118
XSLM_INVALID_PUBLIC_KEY	119
XSLM_INVALID_STRUCTURE	120
XSLM_INVALID_TOKEN	121
XSLM_INVALID_VALUE	122
XSLM_INVALID_VALUES	123
XSLM_INV_COUNTER_ID	124
XSLM_IN_RECOVERY_MODE	125
XSLM_IN_SOFT_STOP	126
XSLM_LIC_SYS_NOT_RESP	131
XSLM_LOGCLASS_ADMIN	1
XSLM_LOGCLASS_APPL	2
XSLM_LOGCLASS_SYSTEM	3
XSLM_LOGCLASS_ANY	9
XSLM_LOGTYPE_ANY	99
XSLM_LOGSUBTYPE_ANY	999
XSLM_LOGTOD_APPL	1
XSLM_LOGTOD_SERVER	2
XSLM_LOG_ERROR	128
XSLM_MASK_APPLIED	129
XSLM_MSG_TOO_LONG	130
XSLM_NOT_AUTHORIZED	152
XSLM_NOT_ENOUGH_CAPACITY	132
XSLM_NOT_ENOUGH_LICS	133
XSLM_NO_CERTIFICATES	134

Symbol	Assigned Value
XSLM_NO_LICS	135
XSLM_NO_LONGER_CHANGABLE	136
XSLM_NO_MATCHING_NODE	137
XSLM_NO_MATCHING_USERID	138
XSLM_NO_MATCHING_INSTANCE	139
XSLM_NO_RES	140
XSLM_OK	0
XSLM_PARM_ERR	4
XSLM_PARTIAL_DATA	142
XSLM_QUERY_CERTIFICATE	3
XSLM_QUERY_CERT_RELATED_INFO	4
XSLM_QUERY_CUST_DEF_INFO	1
XSLM_QUERY_PUBLISHER_INFO	2
XSLM_RESRC_UNAVL	3
XSLM_SERVER_ERROR	143
XSLM_SET_POLICY_ADD	1
XSLM_SET_POLICY_DELETE	2
XSLM_SET_POLICY_REPLACE	3
XSLM_STATUS_OK	0
XSLM_TOO_MANY_UNITS	144
XSLM_TOO_SMALL	145
XSLM_UNCHANGABLE_POLICY	146
XSLM_UNRECOGNIZED_ID	147
XSLM_UNRECOGNIZED_SEQ	148
XSLM_UPDATE_ID_ERROR	149
XSLM_ZERO_REACHED	150

Appendix A

License Types

This section contains two lists:

- Commonly experienced licensing terms and conditions with a brief description of each
- Terms and conditions by license type.

This specification also makes an allowance for software-publisher-unique requirements within the license certificate.

A.1 License Types and Terms and Conditions

BASIC

The BASIC license type is the base line. It represents a license for which there are no restrictions, other than time. In contrast, all the other license types define restrictions within which the application is licensed and the customer is to abide.

BUNDLE

A pricing and packaging option for two or more products which are licensed individually, not collectively. For instance if a customer were licensed for a bundle of five products and 50 uses, when 40 of product A are in use and 10 of product B are in use then the customer has available for use ten on product A, 40 on product B, and 50 on each of the remaining three products by users either actively using one or more of the products or not currently using any of the products. This type of terms and conditions is derived from the CONCURRENT license type.

CAPACITY

The CAPACITY license types compares the capacity of the operating environment (as defined by the machine serial number) along with a predefined table, for instance, to assure the application is running in a machine whose computing capacity is not larger that that for which the product is licensed.

CHECKOUT

See DISCONNECTED.

COMPLEX-WIDE

A license assigned to a complex, which is comprised of one or more unique serial numbers. This type of terms and conditions is derived from the CAPACITY or NAMED license types.

COMPONENT

A license that governs the use of other licensed runtime software components with the application being developed. An example of this would be the inclusion of a spell checker from software publisher A within a word processor from software publisher B. This type of terms and conditions can be any of the license types.

CONCURRENT

A license type for which the charges are based on counting the number of simultaneous demands or uses of a product, independent of who or what user is using the application, quite the opposite from the NAMED concept. Further, these license uses are reusable: when the license is no longer required it is returned to the license system and becomes available for re-issuance against another license request. The number and defined unit of measure may include a minimum or maximum number permitted per request. For instance, a CONCURRENT license may require that whenever a license request is made, five units of the measure defined (users, for instance), must be requested as a minimum.

CONSUMPTIVE

A license type for which the charges are based on counting the defined units executed, perhaps over a specified period of time, against those licensed. Of principal importance with this license type is that a license count once used is not retrievable or reusable. As with CONCURRENT licensing, the number and defined unit of measure may include a minimum or maximum number permitted per request. For instance, a CONSUMPTIVE license may require that whenever a license request is made, five units of the measure defined (blocks of time or gigabytes of storage, for instance), must be requested as a minimum. This license type might be useful in a "peak" use or need situation.

CUMULATIVE

A license type for which the charges are based on counting a defined unit of measure against the number of units of that measure which were licensed. While CUMULATIVE licensing merely accumulates the defined units of measure, as with CONSUMPTIVE licensing, once used these units are not retrievable, or reusable. This license type might be useful in a post- pay term and condition.

DEMO

A license typically restricted to a certain time period, number of executions, or limited set of functions. These licenses may allow any of the other types of use. This license is also known as "Try and Buy" or "Supply before Buy". This type of terms and conditions can be an added restriction to any of the license types.

DEPLOYMENT

See RUN TIME.

DISASTER RECOVERY

A license granted by the vendor to allow a specified product to execute under conditions defined as "disaster recovery" for a specified period of time or for a specified number of occurrences. This type of terms and conditions can be an added restriction to any of the license types.

DISCONNECTED

A license that allows an end-user application use to be licensed while disconnected from the licensing system. (Also known as Laptop or Checkout). This circumstance can be associated with any of the license types.

ENTERPRISE

A license assigned to an enterprise, which may be comprised of multiple sites, complexes, nodes and or serial numbers. It is an all encompassing license to a single entity. This type of terms and conditions is derived from any of the license types.

FEATURES

A packaging and enablement option. An optional feature of a product can be packaged, licensed and enabled at the discretion of the software publisher. Features can be licensed in the same manner as software products and can, therefore, be of any license type.

FLOATING

A license for which the software product (including application client and server) is not tied to a specific ID, site, or user. This type of terms and conditions is derived from CONCURRENT, CUMULATIVE, or CONSUMPTIVE license types.

GROUP-BASED subsets

A license that allows the customer to subdivide uses within an organization. For instance, the customer might allocate 10 uses to engineering and 10 to accounting. This type of terms and conditions is derived from the Named license type.

LAPTOP

See DISCONNECTED.

LPAR

A license granted for use on less than a full machine. Some mainframe computers can be logically divided into smaller pieces (known as LPARs) and the licenses are for these smaller pieces. This is an interpretation of the CAPACITY or NAMED license types.

MEASURED USE

A license for which the charges are based on counting a defined unit of measure; it is a measurement based on the function of the product. For example a backup product would be based on the number of bytes backed up over the course of a specified period. This type of terms and conditions is derived from the CUMULATIVE, CONCURRENT, and CONSUMPTIVE license types.

MIPS

A license based upon the number of MIPS - either single processor or aggregated across several processors. This type of terms and conditions is based on the CAPACITY license type.

NAMED

A license type which compares name or serial number or ID or node address etc., against those licensed. The NAMED license type implies pre-definition of the name. However, to build the registered or named "authorization list," the NAMED license type can also allow for a "first come-first served" concept where license-requesting users (for instance) are registered (accepted / defined) until the number of users licensed is reached.

NODE

A license based upon a specific node(s); some examples of nodes are Network Nodes and JES Nodes. This type of terms and conditions is based on either the Capacity or the Named license type.

PARTITIONING

A license granted for use on less than a full machine. Some processors can be divided into smaller pieces (e.g. partitions) and the licenses are for these smaller pieces. This type of terms and conditions is derived from the Capacity license type. PEAK - A license for which the charges are based on the maximum number of defined units used during a specific time period. This type of terms and conditions can be any of the license types.

POTENTIAL USE

See RESOURCE.

PROCESSOR

A license for which charges are based on the size of a machine. This type of terms and conditions is based on the CAPACITY license type.

REGISTERED USE

This license relies upon the counting and comparing of the ID or node address (for example) against the pre-defined IDs or node addresses licensed. The REGISTERED USE license type implies pre-definition of the user. However, it can also allow for "first come-first served" concept where license-requesting users are registered (accepted / defined) until the entitled number of registered users licensed is reached. This type of terms and conditions is an implementation of the NAMED license type.

RESOURCE

A license for which charges are based on the size of specifically identified resources, such as amount of memory used, number of gigabytes managed, etc. This type of terms and conditions comes from the CAPACITY or NAMED license type.

RUN-TIME

A kind of software license that governs the use of run time software where run time software is defined as those modules of compilers, data base programs, and other development tools that are required in order to operate a program developed using the tools. This type of terms and conditions is a form or descriptor of the COMPONENT type of terms and conditions, and is sometimes referred to as a DEPLOYMENT license.

SERIAL NUMBER

A license assigned to a serial number of a specific hardware device, including those devices that can be carved into smaller pieces (for example an LPAR serial number). This type of terms and conditions is derived from the Named license type.

SITE

A license of a software product on all computers at a geographic location. This type of terms and conditions is implemented from the Capacity or Named license type.

SUITE

A pricing and packaging option for two or more products which are licensed collectively, not individually. For instance if a customer were licensed for a suite of five products and 50 uses, when 40 of product A are in use and 10 of product B are in use by 50 different users, then the maximum allowable uses are reached; there are no remaining uses for the other three products by users not represented in the current count of 50. One of the concepts with this option is that an individual using one of the products in the suite can use all of the other products in the suite without them being counted as additional uses. This type of terms and conditions option is derived from the CONCURRENT license type.

SUPPLY BEFORE BUY

See DEMO.

TARGET ID

See NAMED.

TIME

Each license type is modifiable by time (also known as TIME DELIMITED).

TIME DELIMITED

See TIME.

TIME-SHIFTED

A license which, because of time differences in various locations of the licensee, allows use in multiple geographic locations, but not more than once at a time. For example, a multinational corporation may acquire licenses allowing use within either the US office or the Japanese office, but not both at the same time. This type of terms and conditions is a variant of the CONCURRENT or CONSUMPTIVE or CUMULATIVE license type.

TRY AND BUY

See DEMO.

UNRESTRICTED

See BASIC.

USE-ONCE

See CONSUMPTIVE.

USER-SPECIFIED

A customer administrator can specify more restrictive conditions then stated in the type of terms and conditions of the license. This type of terms and conditions is derived from any of the license types.

YEAR 2000

A license granted by the vendor to allow a specified product to support Year 2000 testing for a specified period of time or for a specified number of occurrences. This type of terms and conditions is derived from the time-option of any of the license types. It is similar to the DEMO type of terms and conditions.

A.2 Terms and Conditions by License Type

BASIC
- CHECKOUT
- COMPONENT
- DEMO
- DEPLOYMENT
- DISASTER RECOVERY
- DISCONNECTED
- ENTERPRISE
- FEATURES
- LAPTOP
- PEAK
- RUN-TIME
- SUPPLY BEFORE BUY
- TIME
- TRY AND BUY
- UNRESTRICTED
- USER-SPECIFIED
- YEAR 2000

CAPACITY
- CHECKOUT
- COMPLEX-WIDE
- COMPONENT
- DEMO
- DEPLOYMENT
- DISASTER RECOVERY
- DISCONNECTED
- ENTERPRISE
- FEATURES
- LAPTOP
- LPAR
- MIPS
- NODE
- PARTITIONING
- PEAK
- POTENTIAL USE
- PROCESSOR
- RESOURCE
- RUN-TIME
- SITE
- SUPPLY BEFORE BUY
- TIME
- TRY AND BUY
- USER-SPECIFIED
- YEAR 2000

CONCURRENT
- BUNDLE
- CHECKOUT
- COMPONENT
- DEMO
- DEPLOYMENT
- DISASTER RECOVERY
- DISCONNECTED
- ENTERPRISE
- FEATURES
- FLOATING
- LAPTOP
- MEASURED USE

PEAK
RUN-TIME
SUITE
SUPPLY BEFORE BUY
TIME
TIME-SHIFTED
TRY AND BUY
USER-SPECIFIED
YEAR 2000

CONSUMPTIVE

CHECKOUT
COMPONENT
DEMO
DEPLOYMENT
DISASTER RECOVERY
DISCONNECTED
ENTERPRISE
FEATURES
FLOATING
LAPTOP
MEASURED USE
PEAK
RUN-TIME
SUPPLY BEFORE BUY
TIME
TIME-SHIFTED
TRY AND BUY
USE-ONCE
USER-SPECIFIED
YEAR 2000

CUMULATIVE

CHECKOUT
COMPONENT
DEMO
DEPLOYMENT
DISASTER RECOVERY
DISCONNECTED
ENTERPRISE
FEATURES
FLOATING
LAPTOP
MEASURED USE
PEAK
RUN-TIME
SUPPLY BEFORE BUY
TIME
TIME-SHIFTED
TRY AND BUY
USER-SPECIFIED
YEAR 2000

NAMED

CHECKOUT
COMPLEX-WIDE
COMPONENT
DEMO
DEPLOYMENT
DISASTER RECOVERY
DISCONNECTED
ENTERPRISE

FEATURES
GROUP-BASED
LAPTOP
LPAR
NODE
PEAK
POTENTIAL USE
REGISTERED USE
RESOURCE
RUN-TIME
SERIAL NUMBER
SITE
SUPPLY BEFORE BUY
TARGET ID
TIME
TRY AND BUY
USER-SPECIFIED
YEAR 2000

Appendix B

Implementation Guidelines

The following sections give examples of the interrelationships between a license management enabled application and a license server. The Named, Concurrent, and Consumptive license types and the Demo terms and conditions are used as examples to illustrate how the data, process and the license entitlement interrelate to affect the behavior of the application. The requests and responses are similar, but they are clearly not identical and their relationship depends largely on the license type. The logic to challenge the XSLM license server and to interpret its response is implementation specific. The application behaviors based on the possible responses are summarized in the last paragraph.

B.1 Named License

The following are assumed to exist:

- A license certificate specifying the unique identity of the licensed processor
- The application with (implicit) access to the identification of its run-time processor
- Logic both to challenge the XSLM license server and to interpret its response.

After loading, the application would determine the identification of its current host and issue a query to XSLM requesting authorization to execute. Possible responses would be:

- Both the license ID and the run-time processor's host identifications are the same.
- No license certificate could be found (the license server is not aware of this product or its entitlement).
- A license exists for the application but the host processor's ID does not match that of the certificate.

B.2 Concurrent License

The following are assumed to exist:

- A license certificate specifying the number of concurrent users in accordance with the entitlement
- The check-in time within which the application re-states its need to be in execution
- The application with logic to challenge the XSLM license server and to interpret its response.

After loading, the application program would issue a query to XSLM requesting authorization to execute. Possible responses would be:

- A concurrent use is available.(the count of use requests is less than the entitlement)
- No license certificate could be found (the license server is not aware of this product or its entitlement)
- A concurrent use is not available (the count of use requests has equaled or exceeded the entitlement).

Assuming the application goes into execution, from time to time the application confirms to the license server the fact that the application program is still in operation. When that condition is

no longer valid, the authorization at that time assigned to this instance of use of the application is returned to the license server, and becomes available as a positive response to another program-use request.

B.3 Consumptive License

The following are assumed to exist:

- A license certificate specifying the number of uses in accordance with the entitlement
- The application, with logic to challenge the XSLM license server and to interpret its response.

After loading, the application program would issue a query to XSLM, requesting authorization to execute. Possible responses would be:

- A license is available (the count of use requests is less than the entitlement)
- No license certificate could be found (the license server is not aware of this product or its entitlement)
- A license is not available (the count of use requests has equaled or exceeded the entitlement).

Continued checking of operation by the licensing system in conjunction with the application and the license client is not required because the authorization, once granted, cannot be returned for reuse.

B.4 Demo License

The following are assumed to exist:

- A license certificate specifying the duration of the license
- Licensed code with access to the duration (specified time period) of the license
- Logic to challenge the demo license and to interpret the response.

After loading, the program would periodically request authorization to start or to continue executing by issuing a query to XSLM. Possible responses could be:

- The program is within the specified duration of the license
- No license certificate could be found
- A license exists for the program but the time period allowed for running the period has ended.

B.5 Summary

In all of the above examples there is a relationship between the response and the application behavior. For the first response in each example, the application will execute its intended functions, the terms of the license agreement having been met. In the other two responses, the application will cause the appropriate messages and logs and initiate actions that support the license terms and conditions. For instance, the application will not go into execution or the application will go into execution but will inform the licensing system of that fact and cause a logging of the excess of entitlement execution.

Appendix C

Function Sets and Functional Towers

XSLM functionality is described in terms of a set of *Base Function* and one or more optional *Function Towers*. Function Towers provide for the optional extensions in licensing system functionality while retaining consistency across all licensing systems that implement the same function towers.

The Base Function set is a prerequisite for all function towers. That is, the Base Function set must be present before any optional function tower may be implemented. Each Function Tower offers a complete set of functions; that is, if one function of the Tower is implemented, then all functions defined as being part of that same Tower must also be implemented.

C.1 Currently Defined Functional Towers

The relationship between the Base Function set and the Functional Towers is depicted in Figure C-1. The current level of the XSLM specification defines four Functional Towers:

- Advanced Management API (Advanced XMAPI) tower
- Basic Application API (Basic XAAPI) tower
- Advanced Application API (Advanced XAAPI) tower
- Advanced License Certificate Architecture (Advanced XLCA) tower.

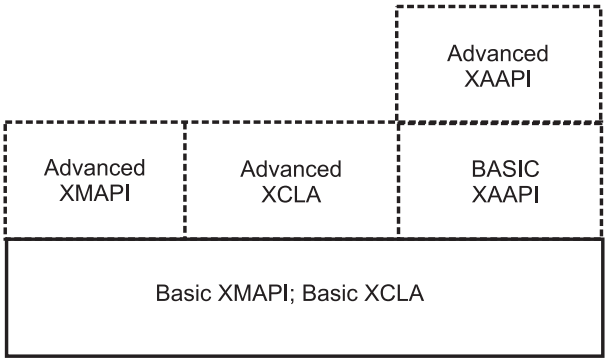


Figure C-1 Base Function Set and Functional Towers

C.2 Base Function Set

The *Base Function* set consists of two elements:

- A common License Certificate Architecture (XLCA)
- A standard set of management application programming interfaces (referred to as the XMAPI function set).

The common License Certificate (data) Architecture provides the capability to fully encode application license terms and conditions in the form of an architected (standardized) license certificate that may be installed into any XSLM-compliant licensing system that supports the embedded certificate data elements. These certificates may be subsequently managed and/or interpreted by any compliant management application. The XLCA is also used to define license certificate meta data (for example, usage measures) and optionally, licensing system specific certificate data elements.

The XMAPI defines the set of programming interfaces necessary to build licensing system management tools capable of managing (installing, removing, etc.) standardized license certificates and/or providing comprehensive application license usage reports.

The XAAPI defines the set of programming interfaces (API functions) necessary to enable for license management those applications that run in trusted environments. A trusted environment is, in this context, one which is by design physically secured, or through hardware and/or software services protects programs and data from unauthorized alteration. (The Base Function set does not include the XAAPI).

C.3 Advanced Function Towers

The *Advanced Application API* Function Tower adds a set of services intended to provide more comprehensive protection (with respect to licensing) and flexibility for application publishers, particularly for those applications that run in untrusted operating environments.

C.4 Function-Related Management API Functions

Management applications must have the ability to determine the set of XLCA defined data elements, as well as the APIs, supported by the license servers that comprise a customer's licensing system. The *xslm_query_server_info*( ) function is defined to satisfy this requirement.

This function facilitates processing of licensing data provided by Legacy Functional Level license servers; servers which may, at the discretion of the implementor, elect to limit support to a proper subset of XLCA-defined certificate data (and meta data) elements and logged data.

C.5 Legacy Functional Level (Level 0)

This functional level is intended to provide a transition path to the standard compliance level (or higher) for existing licensing systems; specifically, the ability to respond to a subset of XMAPI functions (the ones querying certificate and logged data), returning a semantically correct subset of XLCA-defined (the Basic XLCA set) certificate data elements and a semantically correct subset of logged data.

To support Legacy Functional Level a licensing system must implement the following XMAPI functions (the Basic XMAPI set):

xslm_get_certificate( )
xslm_query_cert_ids( )
xslm_query_next_level_cert_names( )
xslm_query_servers( )
xslm_get_license_instances( )
xslm_get_logged_data( )

Appendix D

Futures

The discussion is this section pertains to those items that have been reviewed, analyzed and evaluated by the XSLM specification team and have been deferred to the next release of the specification.

Note: The discussion in this Appendix represents an indication of current thinking only. It is not an expression of commitment to adopt and deliver this functionality in a future version of this Technical Standard.

D.1 Network Computing and Component Licensing

The world of network computing presents a number of new challenges to license use management.

As the software industry moves toward the adoption of distributed object and component technologies it becomes less and less clear as to what constitutes (from the customer's point of view) a product, and what exactly is being licensed. A customer consumable product may be comprised of objects and components that themselves are licensed (from other software publishers) by the product developer.

When objects or components are statically bound into a single customer product entity, or shipped in their entirety as a single customer product, there is no real problem. However, when these components/objects are dynamically acquired by a product from a network of distributed object/component servers the product packages containing the objects and components themselves become individually licensable products. As a result a customer consumable product may in fact be comprised of several sub-products, each requiring its own license. These sub-licenses are, in a well-defined way, ultimately tied to the customer product license.

There is high customer value in being able to maintain a single copy of a given distributed component or object that can be shared by multiple products, potentially from multiple software publishers, across a single logical computing network. This fact when considered in conjunction with Java's promise of true binary portability across disparate computing platforms, would tend to indicate there is a reasonably high probability that the aforementioned multiple-license-to-one-product problem will become a very real licensing issue in the not too distant future.

Performance is always a key concern in a distributed computing network. Network distributable objects and components (including Java applets and beans) represent a formidable performance challenge to those who would like to individually license those entities. Questions such as when and how often these entities should interact with a software license use management system quickly arise.

D.2 Mobile Computing

The mobile (also known as nomad or "disconnected") computer user represents a unique challenge with respect to license use management. This is due to the fact that these users are not continuously connected to the corporate/office network. While this specification is not intended to address the software licensing needs of unmanaged computers (that is, computers which are never connected to a network), the mobile computer users most definitely fall within the domain of interest.

When connected to the network a mobile user possesses all the attributes of a standard desktop computer user. However, when disconnected from the network a mobile user must be able to retain all non-network related application functionality in absence of a network connection. It is this basic difference that presents an interesting set of problems to software license use management systems.

D.3 Server-to-Server Interaction

This specification assumes that all license servers taking part in a licensing system are, somehow, interconnected so that certain actions taken on one server are reflected to other servers. For example, a license certificate should only be installable once within a licensing system (even though it may possibly be installed on more than one physical server to provide redundancy, if the particular implementation supports this). While this requirement is clearly stated, this version of the specification doesn't fully provide the definitions needed for such an interaction between different implementations.

Appendix E

Java Bindings for Application Program API

This appendix contains the description of the standard Java bindings for the APIs (both basic and advanced) of the application programs for the License Use Management (XSLM) Technical Standard.

package org.opengroup.xslm

Interface Index

AdvancedApplicationClient
BasicApplicationClient
XSLMConstants

Class Index

LicenseAgent
LicenseBroker
LicenseUseManagement
Publisher
Session

Exception Index

CommunicationException
InvalidParameterException
LicenseCertificateException
LicenseClientException
NoSuchAPILevelException
NoSuchPublisherException
ResourceUnavailableException

Interface org.opengroup.xslm.AdvancedApplicationClient

public interface **AdvancedApplicationClient** The AdvancedApplicationClient interface is the interface for the first functional tower of the XSLM specification the Advanced XAAPI.

See Also:

BasicApplicationClient, LicenseAgent, LicenseBroker

Method Index

Methods

confirm

```
public abstract void confirm(long confirmTime,
                             int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Confirms that a license is currently in use. Should the license system not receive any confirm from the application client within the default time period specified in the certificate, the license is no longer in use and will return it to the pool of free, available licenses.

Parameters:

confirmTime - elapsed time (in seconds) whithin which the liscense system may expect the next confirm.

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

getAuthenticationSignature

```
public abstract byte[] getAuthenticationSignature()
throws LicenseClientException
```

Returns the digital signature created by the licensing system from the input and output parameters and the authorization token.

Returns:
the digital signature created by the licensing system.

Throws: LicenseClientException
if a generic license client error occurs

logMessage

```
public abstract void logMessage(String message,
                                int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Logs an application-specified text message into license system's log

Parameters:
message - the text string of the message, it can be of any length; however, a licensing system is not required to accept more than 4096 bytes.

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

queryAPILevel

```
public abstract long queryAPILevel()
throws CommunicationException, LicenseClientException,
ResourceUnavailableException
```

Returns the maximum API specification level supported by all license servers in the licensing system.

Returns:
the maximum API specification level supported by all license servers in the licensing system

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

queryFunctionalTowers

```
public abstract long[] queryFunctionalTowers()
throws CommunicationException, LicenseClientException,
ResourceUnavailableException
```

Returns an array of functional tower identifiers supported by all license servers in the licensing system.

Returns:
an array of functional tower identifiers supported by all license servers in the licensing system

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■queryCertificateInfo

```
public abstract byte[] queryCertificateInfo(int queryType,
                                            int authenticationToken)
throws CommunicationException, LicenseCertificateException,
LicenseClientException, ResourceUnavailableException
```

Returns the various types of license certificate information associated with this license client object

Parameters:
queryType - a value which identifies the information to be returned

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Returns:
When queryType=XSLM_QUERY_PUBLISHER_INFO, the data element PUBLISHER_SECTION, if present, followed by the data element PUBLISHER_USE, if present.

When queryType=XSLM_QUERY_CUST_DEF_INFO, the data element CUSTOMER_ASSIGNED_APPL_INFO, if present.

When queryType=XSLM_QUERY_CERTIFICATE, the data element CERTIFICATE.

When queryType=XSLM_QUERY_CERT_RELATED_INFO, the data element CERTIFICATE_RELATED_INFORMATION

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■record

```
public abstract void record(long counterIdentifier,
                            float counter,
                            int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Informs the license system about a metering activity in progress

Parameters:

counterIdentifier - the identifier of one of the 8 different counters (defined within the license certificate)

counter - the value of the number of units the license system should add to or subtract from the total count

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

releaseLicense

```
public abstract void releaseLicense(long units,
                                    int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Releases some or all license units associated with this instance of the license client

Parameters:

units - number of license units to be released

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■requestLicense

```
public abstract long requestLicense(PublicKey publicKey,
                                    String namedUser,
                                    int authenticationToken)
 throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Requests the license system to grant default number of units licenses to the specified user.

Parameters:

publicKey - the software publisher public key. Used to confirm authenticity of a license certificate. The value passed must match the value of the publisher's public key contained in the license certificate.

namedUser - the named user for which license units are being requested

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Returns:

number of license units granted

Throws: CommunicationException

if a communication error occurs between the license client and the license server

Throws: InvalidParameterException

if a parameter passed to a method is invalid

Throws: LicenseCertificateException

if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException

if a generic license client error occurs

Throws: ResourceUnavailableException

if a resource required to execute the requested method is not available

■requestLicense

```
public abstract long requestLicense(PublicKey publicKey,
                                    long confirmTime,
                                    int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Requests the license system for default number of units licenses (as specified within a license certificate).

Parameters:

publicKey - the software publisher public key. Used to confirm authenticity of a license certificate. The value passed must match the value of the publisher's public key contained in the license certificate.

confirmTime - elapsed time (in seconds) whithin which the license system may expect the next confirm.

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Returns:
number of license units granted

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■**requestLicense**

```
public abstract long requestLicense(PublicKey publicKey,
                                    String namedUser,
                                    long confirmTime,
                                    long units,
                                    boolean forceNumber,
                                    int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Requests the license system for specified number of units licenses

Parameters:
publicKey - the software publisher public key. Used to confirm authenticity of a license certificate. The value passed must match the value of the publisher's public key contained in the license certificate.

namedUser - the named user for which license units are being requested

confirmTime - elapsed time (in seconds) whithin which the liscense system may expect the next confirm.

units - number of license units to be requested

forceNumber - whether or not a lower number of license units than that specified should be granted

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Returns:
number of license units granted

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

requestLicense

```
public abstract long requestLicense(PublicKey publicKey,
                                    long units,
                                    boolean forceNumber,
                                    int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Requests the license system for specified number of license units

Parameters:

publicKey - the software publisher public key. Used to confirm authenticity of a license certificate. The value passed must match the value of the publisher's public key contained in the license certificate.

units - number of license units to be requested

forceNumber - whether or not a lower number of license units than that specified should be granted

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Returns:
number of license units granted

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

---

Interface org.opengroup.xslm.BasicApplicationClient

public interface **BasicApplicationClient** The BasicApplicationClient interface is the interface for the Base Functional Level of the XSLM Application API.

See Also:

AdvancedApplicationClient, LicenseAgent, LicenseBroker

Method Index

Methods

confirm

```
public abstract void confirm()
throws CommunicationException, LicenseCertificateException,
LicenseClientException, ResourceUnavailableException
```

Confirms that a license is currently in use. Should the license system not receive any confirm from the application client within the default time period specified in the certificate, the license is no longer in use and will return it to the pool of free, available licenses.

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

queryAPILevel

```
public abstract long queryAPILevel()
throws CommunicationException, LicenseClientException,
ResourceUnavailableException
```

Returns the maximum API specification level supported by all license servers in the licensing system.

Returns:
the maximum API specification level supported by all license servers in the licensing system

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

queryFunctionalTowers

```
public abstract long[] queryFunctionalTowers()
throws CommunicationException, LicenseClientException,
ResourceUnavailableException
```

Returns an array of functional tower identifiers supported by all license servers in the licensing system.

Returns:
an array of functional tower identifiers supported by all license servers in the licensing system

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

releaseLicense

```
public abstract void releaseLicense()
throws CommunicationException, LicenseCertificateException,
LicenseClientException, ResourceUnavailableException
```

Releases the licenses associated with this instance of the license client

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

requestLicense

```
public abstract long requestLicense(PublicKey publicKey)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Requests the license system for default number of units licenses (as specified within a license certificate)

Parameters:
publicKey - the software publisher public key. Used to confirm authenticity of a license certificate. The value passed must match the value of the publisher's public key contained in the license certificate.

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

---

Interface org.opengroup.xslm.XSLMConstants

public interface **XSLMConstants** The XSLMConstants interface defines all the constants used thoughout the org.opengroup.xslm package

Variable Index

Variables

DEFAULT_CONFIRM_TIME
public static final long DEFAULT_CONFIRM_TIME
Confirm time value used to request default confirm time

DEFAULT_UNITS

public static final long DEFAULT_UNITS
Number of units value used to request default number of units licenses

`public static final int XSLM_QUERY_PUBLISHER_INFO`
Query the application publisher specific section

`public static final int XSLM_QUERY_CUST_DEF_INFO`
Query application-related information defined by the customer

`public static final int XSLM_QUERY_CERTIFICATE`
Query the certificate as it has been created by the application publisher

`public static final int XSLM_QUERY_CERT_RELATED_INFO`
Query the certificate related information such as certificate status

Class org.opengroup.xslm.LicenseAgent

```
java.lang.Object
   |
   +----org.opengroup.xslm.LicenseAgent
```

public abstract class **LicenseAgent**
extends Object
implements BasicApplicationClient, AdvancedApplicationClient, XSLMConstants The LicenseAgent class provides LicenseBroker and application clients with access to a specific publisher's implementation of the XSLM API. Whereas the LicenseBroker class provides implementation transparency to application clients, the LicenseAgent class provides server transparency.
Like other classes in org.opengroup.xslm, the LicenseAgent class has two major components:

LicenseAgent API (Application Program Interface)
This is the interface of methods called either by applications needing License Agent services to have their requests forwarded to that specifc publisher's licensing system implementation, or by license brokers routing application client's methods invocation to any of the licensing systems available in the environment. The API consists of all public methods.

LicenseAgent SPI (Service Provider Interface)
This is the interface implemented by licensing system publishers that supply specific license use management packages. It consists of all methods whose names are prefixed by `engine`. Each such method is called by a correspondingly-named public API method. For example, the `engineRequestLicense` method is called by the `requestLicense` method. The SPI methods are abstract; publishers must supply a concrete implementation.

LicenseAgent provides implementation-independent objects, a caller (license broker or application code) can request a particular implementation of a LicenseAgent object from a particular publisher. The system will determine if there is an implementation of the requested LicenseAgent object in the package requested, and throw an exception if there is not.

See Also:
BasicApplicationClient, AdvancedApplicationClient, LicenseBroker

Constructor Index

LicenseAgent()

Method Index

confirm()
Confirms that a license is currently in use.

confirm(long, int)
Confirms that a license is currently in use.

engineConfirm()
SPI: Confirms that a license is currently in use.

engineConfirm(long, int)
SPI: Confirms that a license is currently in use.

engineGetAuthenticationSignature()
SPI: Returns the digital signature created by the licensing system from the input and output parameters and the authorization token.

engineLogMessage(String, int)
SPI: Logs and application-specified text message into license system's log.

engineQueryAPILevel()
SPI: Returns the maximum API specification level supported by all license servers in the licensing system.

engineQueryCertificateInfo(int, int)
SPI: Returns various types of license certificate information

engineQueryFunctionalTowers()
SPI: Returns the maximum API specification level supported by all license servers in the licensing system.

engineRecord(long, float, int)
SPI: Informs the license system about a metering activity in progress

engineReleaseLicense()
SPI: Releases the licenses associated with this instance of the license client

engineReleaseLicense(long, int)
SPI: Releases some or all license units associated with this instance of the license client

engineRequestLicense(PublicKey)
SPI: Requests the license system for default number of units licenses (as specified within a license certificate).

engineRequestLicense(PublicKey, String, long, long, boolean, int)
SPI: Requests the license system for the specified number of units licenses.

getAdvancedApplicationClientInstance(String, Session, String, long, long, long)
Generates an AdvancedApplicationClient object as suppled from the specified publisher.

getAuthenticationSignature()
Returns the digital signature created by the licensing system from the input and output parameters and the authorization token.

getBasicApplicationClientInstance(String, String, long, long, long)
Generates a BasicApplicationClient object as supplied from the specified publisher.

init(Session, String, long, long, long)
Initializes the LicenseBroker object returned by either a `getBasicApplicationClientInstance` or `getAdvancedApplicationClientInstance` method invocation.

logMessage(String, int)
Logs and application-specified text message into license system's log

queryAPILevel()
Returns the maximum API specification level supported by all license servers in the licensing system.

queryCertificateInfo(int, int)
Returns various types of license certificate information.

queryFunctionalTowers()
Returns an array of functional tower identifiers supported by all license servers in the licensing system.

record(long, float, int)
Informs the license system about a metering activity in progress

releaseLicense()
Releases the licenses associated with this instance of the license client

releaseLicense(long, int)
Releases some or all license units associated with this instance of the license client

requestLicense(PublicKey)
Requests the license system for default number of units licenses (as specified within a license certificate).

requestLicense(PublicKey, long, boolean, int)
Requests the license system for specified number of license units

requestLicense(PublicKey, long, int)
Requests the license system for default number of units licenses (as specified within a license certificate).

requestLicense(PublicKey, String, int)
Requests the license system for default number of units licenses (as specified within a license certificate).

requestLicense(PublicKey, String, long, long, boolean, int)
Requests the license system for specified number of units licenses

Constructors

LicenseAgent

```
public LicenseAgent()
```

Methods

■getBasicApplicationClientInstance

```
 public static BasicApplicationClient
getBasicApplicationClientInstance(String licensingSystemPublisher,
                                  String publisherID,
                                  long productID,
                                  long versionID,
                                  long featureID)
throws NoSuchPublisherException, NoSuchAPILevelException
```

Generates a BasicApplicationClient object as supplied from the specified publisher.

Parameters:

licensingSystemPublisher - the name of the license system publisher.

publisherID - the identifier of the software publisher

productID - the identifier of the product

versionID - the identifier of the version

featureID - the identifier of the feature

Returns:

the new BasicApplicationClient object

Throws: NoSuchPublisherException

if there is no subclass of LicenseAgent provided by the specified publisher.

Throws: NoSuchAPILevelException

if the package provded by the specified publisher does not contain the requested API level

See Also:

LicenseBroker, Publisher

■getAdvancedApplicationClientInstance

```
 public static AdvancedApplicationClient
getAdvancedApplicationClientInstance(String licensingSystemPublisher,
                                     Session session,
                                     String publisherID,
                                     long productID,
                                     long versionID,
                                     long featureID)
throws NoSuchPublisherException, NoSuchAPILevelException
```

Generates an AdvancedApplicationClient object as suppled from the specified publisher.

Parameters:

licensingSystemPublisher - the name of the license system publisher.

session - the session this application client object is bound to

publisherID - the identifier of the software publisher

productID - the identifier of the product

versionID - the identifier of the version

featureID - the identifier of the feature

Returns:
the new AdvancedApplicationClient object

Throws: NoSuchPublisherException
if there is no subclass of LicenseAgent provided by the specified publisher.

Throws: NoSuchAPILevelException
if the package provded by the specified publisher does not contain the requested API level

See Also:
Publisher

■confirm

```
public void confirm()
throws CommunicationException, LicenseCertificateException,
LicenseClientException, ResourceUnavailableException
```

Confirms that a license is currently in use. Should the license system not receive any confirm fromthe application client within the default time period specified in the certificate, the license is no longer in use and will return it to the pool of free, available licenses.

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■releaseLicense

```
public void releaseLicense()
throws CommunicationException, LicenseCertificateException,
LicenseClientException, ResourceUnavailableException
```

Releases the licenses associated with this instance of the license client

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

requestLicense

```
public long requestLicense(PublicKey publicKey)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Requests the license system for default number of units licenses (as specified within a license certificate).

Parameters:

publicKey - the software publisher public key. Used to confirm authenticity of a license certificate. The value passed must match the value of the publisher's public key contained in the license certificate.

Returns:

number of license units granted

Throws: CommunicationException

if a communication error occurs between the license client and the license server

Throws: InvalidParameterException

if a parameter passed to a method is invalid

Throws: LicenseCertificateException

if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException

if a generic license client error occurs

Throws: ResourceUnavailableException

if a resource required to execute the requested method is not available

confirm

```
public void confirm(long confirmTime,
                    int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Confirms that a license is currently in use. Should the license system not receive any confirm from the application client within the default time period specified in the certificate, the license is no longer in use and will return it to the pool of free, available licenses.

Parameters:

confirmTime - elapsed time (in seconds) whithin which the liscense system may expect the next confirm.

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException

if a communication error occurs between the license client and the license server

Throws: InvalidParameterException

if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

getAuthenticationSignature

```
public byte[] getAuthenticationSignature()
throws LicenseClientException
```

Returns the digital signature created by the licensing system from the input and output parameters and the authorization token.

Returns:
the digital signature created by the licensing system.

Throws: LicenseClientException
if a generic license client error occurs

logMessage

```
public void logMessage(String message,
                       int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Logs and application-specified text message into license system's log

Parameters:
message - the text string of the message, it can be of any length; however, a licensing system is not required to accept more than 4096 bytes.

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■queryAPILevel

```
 public long queryAPILevel()
throws CommunicationException, LicenseClientException,
ResourceUnavailableException
```

Returns the maximum API specification level supported by all license servers in the licensing system.

Returns:
the API level supported by all license servers in the licensing system

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■queryFunctionalTowers

```
 public long[] queryFunctionalTowers()
throws CommunicationException, LicenseClientException,
ResourceUnavailableException
```

Returns an array of functional tower identifiers supported by all license servers in the licensing system.

Returns:
an array of functional tower identifiers supported by all license servers in the licensing system

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■queryCertificateInfo

```
 public byte[] queryCertificateInfo(int queryType, int
authenticationToken) throws CommunicationException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Returns the various types of license certificate information associated with this license client object.

Parameters:
queryType - a value which identifies the information to be returned

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Returns:

When queryType=XSLM_QUERY_PUBLISHER_INFO, the data element PUBLISHER_SECTION, if present, followed by the data element PUBLISHER_USE, if present.

When queryType=XSLM_QUERY_CUST_DEF_INFO, the data element CUSTOMER_ASSIGNED_APPL_INFO, if present.

When queryType=XSLM_QUERY_CERTIFICATE, the data element CERTIFICATE.

When queryType=XSLM_QUERY_CERT_RELATED_INFO, the data element CERTIFICATE_RELATED_INFORMATION

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■record

```
 public void record(long counterIdentifier,
                    float counter,
                    int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Informs the license system about a metering activity in progress

Parameters:

counterIdentifier - the identifier of one of the 8 different counters (defined within the license certificate)

counter - the value of the number of units the license system should add to or subtract from the total count

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

releaseLicense

```
public void releaseLicense(long units,
                           int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Releases some or all license units associated with this instance of the license client

Parameters:
units - number of license units to be released

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

requestLicense

```
public long requestLicense(PublicKey publicKey,
                           String namedUser,
                           int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Requests the license system for default number of units licenses (as specified within a license certificate).

Parameters:
publicKey - the software publisher public key. Used to confirm authenticity of a license certificate. The value passed must match the value of the publisher's public key contained in the license certificate.

namedUser - the named user for which license units are being requested

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Returns:
number of license units granted

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

requestLicense

```
public long requestLicense(PublicKey publicKey,
                           long confirmTime,
                           int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Requests the license system for default number of units licenses (as specified within a license certificate).

Parameters:

publicKey - the software publisher public key. Used to confirm authenticity of a license certificate. The value passed must match the value of the publisher's public key contained in the license certificate.

confirmTime - elapsed time (in seconds) whithin which the license system may expect the next confirm.

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Returns:
number of license units granted

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

requestLicense

```
public long requestLicense(PublicKey publicKey,
                           String namedUser,
                           long confirmTime,
                           long units,
                           boolean forceNumber,
                           int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Requests the license system for specified number of units licenses

Parameters:

publicKey - the software publisher public key. Used to confirm authenticity of a license certificate. The value passed must match the value of the publisher's public key contained in the license certificate.

namedUser - the named user for which license units are being requested

confirmTime - elapsed time (in seconds) whithin which the liscense system may expect the next confirm.

units - number of license units to be requested

forceNumber - whether or not a lower number of license units than that specified should be granted

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Returns:

number of license units granted

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■**requestLicense**

```
public long requestLicense(PublicKey publicKey,
                           long units,
                           boolean forceNumber,
                           int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Requests the license system for specified number of license units

Parameters:

publicKey - the software publisher public key. Used to confirm authenticity of a license certificate. The value passed must match the value of the publisher's public key contained in the license certificate.

units - number of license units to be requested

forceNumber - whether or not a lower number of license units than that specified should be granted

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Returns:

number of license units granted

Throws: CommunicationException

if a communication error occurs between the license client and the license server

Throws: InvalidParameterException

if a parameter passed to a method is invalid

Throws: LicenseCertificateException

if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException

if a generic license client error occurs

Throws: ResourceUnavailableException

if a resource required to execute the requested method is not available

■**init**

```
protected final void init(Session session,
                          String publisherID,
                          long productID,
                          long versionID,
                          long featureID)
```

Initializes the LicenseBroker object returned by either a `getBasicApplicationClientInstance` or `getAdvancedApplicationClientInstance` method invocation.

Parameters:

session - the session this LicenseBroker object is associated with

publisherID - the identifier of the software publisher

productID - the identifier of the product

versionID - the identifier of the version

featureID - the identifier of the feature

engineConfirm

```
protected abstract void engineConfirm()
throws CommunicationException, LicenseCertificateException,
LicenseClientException, ResourceUnavailableException
```

SPI: Confirms that a license is currently in use.

Parameters:

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException

if a communication error occurs between the license client and the license server

Throws: LicenseCertificateException

if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException

if a generic license client error occurs

Throws: ResourceUnavailableException

if a resource required to execute the requested method is not available

engineReleaseLicense

```
protected abstract void engineReleaseLicense()
throws CommunicationException, LicenseCertificateException,
LicenseClientException, ResourceUnavailableException
```

SPI: Releases the licenses associated with this instance of the license client

Parameters:

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException

if a communication error occurs between the license client and the license server

Throws: LicenseCertificateException

if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException

if a generic license client error occurs

Throws: ResourceUnavailableException

if a resource required to execute the requested method is not available

■engineRequestLicense

```
 protected abstract long engineRequestLicense(PublicKey publicKey)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

SPI: Requests the license system for default number of units licenses (as specified within a license certificate).

Parameters:

publicKey - the software publisher public key. Used to confirm authenticity of a license certificate. The value passed must match the value of the publisher's public key contained in the license certificate.

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Returns:

number of license units granted

Throws: CommunicationException

if a communication error occurs between the license client and the license server

Throws: InvalidParameterException

if a parameter passed to a method is invalid

Throws: LicenseCertificateException

if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException

if a generic license client error occurs

Throws: ResourceUnavailableException

if a resource required to execute the requested method is not available

■engineConfirm

```
 protected abstract void engineConfirm(long confirmTime,
                                       int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

SPI: Confirms that a license is currently in use.

Parameters:

confirmTime - elapsed time (in seconds) whithin which the liscense system may expect the next confirm.

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException

if a communication error occurs between the license client and the license server

Throws: InvalidParameterException

if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■engineGetAuthenticationSignature

```
protected abstract byte[] engineGetAuthenticationSignature()
throws LicenseClientException
```

SPI: Returns the digital signature created by the licensing system from the input and output parameters and the authorization token.

Returns:
the digital signature created by the licensing system.

Throws: LicenseClientException
if a generic license client error occurs

■engineLogMessage

```
protected abstract void engineLogMessage(String message,
                                         int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

SPI: Logs and application-specified text message into license system's log.

Parameters:
message - the text string of the message, it can be of any length; however, a licensing system is not required to accept more than 4096 bytes.

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■engineQueryAPILevel

```
 protected abstract long engineQueryAPILevel()
throws CommunicationException, LicenseClientException,
ResourceUnavailableException
```

SPI: Returns the maximum API specification level supported by all license servers in the licensing system.

Returns:
the maximum API specification level supported by all license servers in the licensing system

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■engineQueryFunctionalTowers

```
 protected abstract long[] engineQueryFunctionalTowers()
throws CommunicationException, LicenseClientException,
ResourceUnavailableException
```

SPI: Returns the maximum API specification level supported by all license servers in the licensing system.

Returns:
the maximum API specification level supported by all license servers in the licensing system

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■engineQueryCertificateInfo

```
 protected abstract byte[]
engineQueryCertificateInfo(int queryType,
                           int authenticationToken)
throws CommunicationException, LicenseCertificateException,
LicenseClientException, ResourceUnavailableException
```

SPI: Returns the various types of license certificate information associated with this license client object.

Parameters:
queryType - a value which identifies the information to be returned

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Returns:

When queryType=XSLM_QUERY_PUBLISHER_INFO, the data element PUBLISHER_SECTION, if present, followed by the data element PUBLISHER_USE, if present.

When queryType=XSLM_QUERY_CUST_DEF_INFO, the data element CUSTOMER_ASSIGNED_APPL_INFO, if present.

When queryType=XSLM_QUERY_CERTIFICATE, the data element CERTIFICATE.

When queryType=XSLM_QUERY_CERT_RELATED_INFO, the data element CERTIFICATE_RELATED_INFORMATION

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■**engineRecord**

```
protected abstract void engineRecord(long counterIdentifier,
                                     float counter,
                                     int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

SPI: Informs the license system about a metering activity in progress

Parameters:

counterIdentifier - the identifier of one of the 8 different counters (defined within the license certificate)

counter - the value of the number of units the license system should add to or subtract from the total count

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

engineReleaseLicense

```
protected abstract void engineReleaseLicense(long units,
                                             int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

SPI: Releases some or all license units associated with this instance of the license client

Parameters:
authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

units - number of license units to be released

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

engineRequestLicense

```
protected abstract long engineRequestLicense(PublicKey publicKey,
                                             String namedUser,
                                             long confirmTime,
                                             long units,
                                             boolean forceNumber,
                                             int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

SPI: Requests the license system for the specified number of units licenses.

Parameters:
publicKey - the software publisher public key. Used to confirm authenticity of a license certificate. The value passed must match the value of the publisher's public key contained in the license certificate.

namedUser - the named user for which license units are being requested

confirmTime - elapsed time (in seconds) whithin which the liscense system may expect the next confirm.

units - number of license units to be requested

forceNumber - whether or not a lower number of license units than that specified should be granted

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Returns:
number of license units granted

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

---

Class org.opengroup.xslm.LicenseBroker

```
java.lang.Object
   |
   +----org.opengroup.xslm.LicenseBroker
```

public abstract class **LicenseBroker**
extends Object
implements BasicApplicationClient, AdvancedApplicationClient, XSLMConstants The LicenseBroker class is used to render the presence of multiple licensing systems implementations transparent to application clients; that is to create the illusion of a single logical license management system, when two or more implementations are available in the environment.
Like other classes in org.opengroup.xslm, the LicenseBroker class has two major components:

LicenseBroker API (Application Program Interface)
This is the interface of methods called by applications needing License Broker services to have their requests routed to real implementations of licensing systems available in the environment. The API consists of all public methods.

LicenseBroker SPI (Service Provider Interface)
This is the interface implemented by licensing system publishers that supply specific license use management packages. It consists of all methods whose names are prefixed by `engine`. Each such method is called by a correspondingly-named public API method. For example, the `engineRequestLicense` method is called by the `requestLicense` method. The SPI methods are abstract; publishers must supply a concrete implementation.

LicenseBroker provides implementation-independent objects, a caller (application code) can request a LicenseBroker object, the system will determine if there is an implementation available in the environment, and if there is more than one, if there is a preferred one.

See Also:
BasicApplicationClient, AdvancedApplicationClient, LicenseAgent

Constructor Index

LicenseBroker()

Method Index

confirm()
Confirms that a license is currently in use.

confirm(long, int)
Confirms that a license is currently in use.

engineConfirm()
SPI: Confirms that a license is currently in use.

engineConfirm(long, int)

SPI: Confirms that a license is currently in use.

engineGetAuthenticationSignature()
SPI: Returns the digital signature created by the licensing system from the input and output parameters and the authorization token.

engineGetLicensingSystemPublisherID()
SPI: Returns the licensing system publisher identifier of this LicenseBroker object.

engineLogMessage(String, int)
SPI: Logs and application-specified text message into license system's log.

engineQueryAPILevel()
Returns the API level supported by all license servers in the licensing system.

engineQueryCertificateInfo(int, int)
Returns various types of license certificate information.

engineQueryFunctionalTowers()
SPI: Returns the maximum API specification level supported by all license servers in the licensing system.

engineRecord(long, float, int)
SPI: Informs the license system about a metering activity in progress

engineReleaseLicense()
SPI: Releases the licenses associated with this instance of the license client

engineReleaseLicense(long, int)
SPI: Releases some or all license units associated with this instance of the license client

engineRequestLicense(PublicKey)
SPI: Requests the license system for default number of units licenses (as specified within a license certificate).

engineRequestLicense(PublicKey, String, long, long, boolean, int)
SPI: Requests the license system for the specified number of units licenses.

getAdvancedApplicationClientInstance(Session, String, long, long, long)
Generates an AdvancedApplicationClient object of the default LicenseUseManagement package.

getAuthenticationSignature()
Returns the digital signature created by the licensing system from the input and output parameters and the authorization token.

getBasicApplicationClientInstance(String, long, long, long)
Generates a BasicApplicationClient object of the default LicenseUseManagement package.

getLicensingSystemPublisherID()
Returns the licensing system publisher identifier of this LicenseBroker object.

init(Session, String, long, long, long)
Initializes the LicenseBroker object returned by either a `getBasicApplicationClientInstance` or `getAdvancedApplicationClientInstance` method invocation.

logMessage(String, int)
Logs and application-specified text message into license system's log

queryAPILevel()
Returns the maximum API specification level supported by all license servers in the licensing system.

queryCertificateInfo(int, int)
Returns various types of license certificate information

queryFunctionalTowers()
Returns an array of functional tower identifiers supported by all license servers in the licensing system.

record(long, float, int)
Informs the license system about a metering activity in progress

releaseLicense()
Releases the licenses associated with this instance of the license client

releaseLicense(long, int)
Releases some or all license units associated with this instance of the license client

requestLicense(PublicKey)
Requests the license system for default number of units licenses (as specified within a license certificate).

requestLicense(PublicKey, long, boolean, int)
Requests the license system for the specified number of license units

requestLicense(PublicKey, long, int)
Requests the license system for default number of units licenses (as specified within a license certificate).

requestLicense(PublicKey, String, int)
Requests the license system for default number of units licenses (as specified within a license certificate).

requestLicense(PublicKey, String, long, long, boolean, int)
Requests the license system for specified number of units licenses

Constructors

LicenseBroker

```
public LicenseBroker()
```

Methods

■getBasicApplicationClientInstance

```
 public static BasicApplicationClient
getBasicApplicationClientInstance(String publisherID,
                                  long productID,
                                  long versionID,
                                  long featureID)
throws NoSuchPublisherException, NoSuchAPILevelException
```

Generates a BasicApplicationClient object of the default LicenseUseManagement package. The default LicenseUseManagement package is looked for in the xslm.properties file.

Parameters:

publisherID - the identifier of the software publisher

productID - the identifier of the product

versionID - the identifier of the version

featureID - the identifier of the feature

Returns:

the new BasicApplicationClient object.

Throws: NoSuchPublisherException

if there is no subclass of LicenseAgent provided by the specified publisher.

Throws: NoSuchAPILevelException

if the package provded by the specified publisher does not contain the requested API level

■getAdvancedApplicationClientInstance

```
 public static AdvancedApplicationClient
getAdvancedApplicationClientInstance(Session session,
                                     String publisherID,
                                     long productID,
                                     long versionID,
                                     long featureID)
throws NoSuchPublisherException, NoSuchAPILevelException
```

Generates an AdvancedApplicationClient object of the default LicenseUseManagement package. The default LicenseUseManagement package is looked for in the xslm.properties file.

Parameters:

session - the session this application client object is bound to

publisherID - the identifier of the software publisher

productID - the identifier of the product

versionID - the identifier of the version

featureID - the identifier of the feature

Returns:

the new AdvancedApplicationClient object.

Throws: NoSuchPublisherException
if there is no subclass of LicenseAgent provided by the specified publisher.

Throws: NoSuchAPILevelException
if the package provded by the specified publisher does not contain the requested API level

confirm

```
public void confirm()
throws CommunicationException, LicenseCertificateException,
LicenseClientException, ResourceUnavailableException
```

Confirms that a license is currently in use. Should the license system not receive any confirm fromthe application client within the default time period specified in the certificate, the license is no longer in use and will return it to the pool of free, available licenses.

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

releaseLicense

```
public void releaseLicense()
throws CommunicationException, LicenseCertificateException,
LicenseClientException, ResourceUnavailableException
```

Releases the licenses associated with this instance of the license client

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

requestLicense

```
public long requestLicense(PublicKey publicKey)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Requests the license system for default number of units licenses (as specified within a license certificate).

Parameters:
publicKey - the software publisher public key. Used to confirm authenticity of a license

certificate. The value passed must match the value of the publisher's public key contained in the license certificate.

Returns:
number of license units granted

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■confirm

```
public void confirm(long confirmTime,
                    int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Confirms that a license is currently in use. Should the license system not receive any confirm from the application client within the default time period specified in the certificate, the license is no longer in use and will return it to the pool of free, available licenses.

Parameters:
confirmTime - elapsed time (in seconds) whithin which the liscense system may expect the next confirm.

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■getAuthenticationSignature

```
 public byte[] getAuthenticationSignature()
throws LicenseClientException
```

Returns the digital signature created by the licensing system from the input and output parameters and the authorization token.

Returns:
the digital signature created by the licensing system.

Throws: LicenseClientException
if a generic license client error occurs

■getLicensingSystemPublisherID

```
 public String getLicensingSystemPublisherID()
throws LicenseClientException
```

Returns the licensing system publisher identifier of this LicenseBroker object. It may be used to determine how to calculate the digital signature required to authenticate the licensing system itself.

Returns:
the licensing system publisher identifier of this LicenseBrokerObject.

Throws: LicenseClientException
if a generic license client error occurs

■logMessage

```
 public void logMessage(String message,
                        int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Logs and application-specified text message into license system's log

Parameters:
message - the text string of the message, it can be of any length; however, a licensing system is not required to accept more than 4096 bytes.

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

queryAPILevel

```
 public long queryAPILevel()
throws CommunicationException, LicenseClientException,
ResourceUnavailableException
```

Returns the maximum API specification level supported by all license servers in the licensing system.

Returns:
the API level supported by all license servers in the licensing system

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

queryFunctionalTowers

```
 public long[] queryFunctionalTowers()
throws CommunicationException, LicenseClientException,
ResourceUnavailableException
```

Returns an array of functional tower identifiers supported by all license servers in the licensing system.

Returns:
an array of functional tower identifiers supported by all license servers in the licensing system

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

queryCertificateInfo

```
 public byte[] queryCertificateInfo(int queryType,
                                    int authenticationToken)
throws CommunicationException, LicenseCertificateException,
LicenseClientException, ResourceUnavailableException
```

Returns the various types of license certificate information associated with this license client object

Parameters:
queryType - a value which identifies the information to be returned

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Returns:

When queryType=XSLM_QUERY_PUBLISHER_INFO, the data element PUBLISHER_SECTION, if present, followed by the data element PUBLISHER_USE, if present.

When queryType=XSLM_QUERY_CUST_DEF_INFO, the data element CUSTOMER_ASSIGNED_APPL_INFO, if present.

When queryType=XSLM_QUERY_CERTIFICATE, the data element CERTIFICATE.

When queryType=XSLM_QUERY_CERT_RELATED_INFO, the data element CERTIFICATE_RELATED_INFORMATION

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■record

```
 public void record(long counterIdentifier,
                    float counter,
                    int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Informs the license system about a metering activity in progress

Parameters:

counterIdentifier - the identifier of one of the 8 different counters (defined within the license certificate)

counter - the value of the number of units the license system should add to or subtract from the total count

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException

if a resource required to execute the requested method is not available

releaseLicense

```
public void releaseLicense(long units,
                           int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Releases some or all license units associated with this instance of the license client

Parameters:

units - number of license units to be released

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException

if a communication error occurs between the license client and the license server

Throws: InvalidParameterException

if a parameter passed to a method is invalid

Throws: LicenseCertificateException

if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException

if a generic license client error occurs

Throws: ResourceUnavailableException

if a resource required to execute the requested method is not available

requestLicense

```
public long requestLicense(PublicKey publicKey,
                           String namedUser,
                           int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Requests the license system for default number of units licenses (as specified within a license certificate).

Parameters:

publicKey - the software publisher public key. Used to confirm authenticity of a license certificate. The value passed must match the value of the publisher's public key contained in the license certificate.

namedUser - the named user for which license units are being requested

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Returns:

number of license units granted

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

requestLicense

```
public long requestLicense(PublicKey publicKey,
                           long confirmTime,
                           int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Requests the license system for default number of units licenses (as specified within a license certificate).

Parameters:

publicKey - the software publisher public key. Used to confirm authenticity of a license certificate. The value passed must match the value of the publisher's public key contained in the license certificate.

confirmTime - elapsed time (in seconds) whithin which the license system may expect the next confirm.

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Returns:
number of license units granted

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■**requestLicense**

```
 public long requestLicense(PublicKey publicKey,
                            String namedUser,
                            long confirmTime,
                            long units,
                            boolean forceNumber,
                            int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Requests the license system for specified number of units licenses

Parameters:

publicKey - the software publisher public key. Used to confirm authenticity of a license certificate. The value passed must match the value of the publisher's public key contained in the license certificate.

namedUser - the named user for which license units are being requested

confirmTime - elapsed time (in seconds) whithin which the liscense system may expect the next confirm.

units - number of license units to be requested

forceNumber - whether or not a lower number of license units than that specified should be granted

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Returns:

number of license units granted

Throws: CommunicationException

if a communication error occurs between the license client and the license server

Throws: InvalidParameterException

if a parameter passed to a method is invalid

Throws: LicenseCertificateException

if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException

if a generic license client error occurs

Throws: ResourceUnavailableException

if a resource required to execute the requested method is not available

■**requestLicense**

```
public long requestLicense(PublicKey publicKey,
                           long units,
                           boolean forceNumber,
                           int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

Requests the license system for the specified number of license units

Parameters:

publicKey - the software publisher public key. Used to confirm authenticity of a license certificate. The value passed must match the value of the publisher's public key contained in the license certificate.

units - number of license units to be requested

forceNumber - whether or not a lower number of license units than that specified should be granted

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Returns:

number of license units granted

Throws: CommunicationException

if a communication error occurs between the license client and the license server

Throws: InvalidParameterException

if a parameter passed to a method is invalid

Throws: LicenseCertificateException

if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException

if a generic license client error occurs

Throws: ResourceUnavailableException

if a resource required to execute the requested method is not available

■**init**

```
protected final void init(Session session,
                          String publisherID,
                          long productID,
                          long versionID,
                          long featureID)
```

Initializes the LicenseBroker object returned by either a `getBasicApplicationClientInstance` or `getAdvancedApplicationClientInstance` method invocation.

Parameters:

session - the session this LicenseBroker object is associated with

publisherID - the identifier of the software publisher

productID - the identifier of the product

versionID - the identifier of the version

featureID - the identifier of the feature

engineConfirm

```
protected abstract void engineConfirm()
throws CommunicationException, LicenseCertificateException,
LicenseClientException, ResourceUnavailableException
```

SPI: Confirms that a license is currently in use.

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

engineReleaseLicense

```
protected abstract void engineReleaseLicense()
throws CommunicationException, LicenseCertificateException,
LicenseClientException, ResourceUnavailableException
```

SPI: Releases the licenses associated with this instance of the license client

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

engineRequestLicense

```
protected abstract long engineRequestLicense(PublicKey publicKey)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

SPI: Requests the license system for default number of units licenses (as specified within a license certificate).

Parameters:
publicKey - the software publisher public key. Used to confirm authenticity of a license certificate. The value passed must match the value of the publisher's public key contained in

the license certificate.

Returns:
number of license units granted

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■**engineConfirm**

```
protected abstract void engineConfirm(long confirmTime,
                                      int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

SPI: Confirms that a license is currently in use.

Parameters:
confirmTime - elapsed time (in seconds) whithin which the liscense system may expect the next confirm.

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■engineGetLicensingSystemPublisherID

```
 protected abstract String engineGetLicensingSystemPublisherID()
throws LicenseClientException
```

SPI: Returns the licensing system publisher identifier of this LicenseBroker object. It may be used to determine how to calculate the digital signature required to authenticate the licensing system itself.

Returns:
the licensing system publisher identifier of this LicenseBrokerObject.

Throws: LicenseClientException
if a generic license client error occurs

■engineGetAuthenticationSignature

```
 protected abstract byte[] engineGetAuthenticationSignature()
throws LicenseClientException
```

SPI: Returns the digital signature created by the licensing system from the input and output parameters and the authorization token.

Returns:
the digital signature created by the licensing system.

Throws: LicenseClientException
if a generic license client error occurs

■engineLogMessage

```
 protected abstract void engineLogMessage(String message,
                                          int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

SPI: Logs and application-specified text message into license system's log.

Parameters:
message - the text string of the message, it can be of any length; however, a licensing system is not required to accept more than 4096 bytes.

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■engineQueryAPILevel

```
 protected abstract long engineQueryAPILevel()
throws CommunicationException, LicenseClientException,
ResourceUnavailableException
```

Returns the API level supported by all license servers in the licensing system.

Returns:
the API level supported by all license servers in the licensing system

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■engineQueryFunctionalTowers

```
 protected abstract long[] engineQueryFunctionalTowers()
throws CommunicationException, LicenseClientException,
ResourceUnavailableException
```

SPI: Returns the maximum API specification level supported by all license servers in the licensing system.

Returns:
the maximum API specification level supported by all license servers in the licensing system

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■engineQueryCertificateInfo

```
 protected abstract byte[]
 engineQueryCertificateInfo(int queryType, int authenticationToken)
throws CommunicationException, LicenseCertificateException,
LicenseClientException, ResourceUnavailableException
```

SPI: Returns the various types of license certificate information associated with this license client object

Parameters:
queryType - a value which identifies the information to be returned

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Returns:

When queryType=XSLM_QUERY_PUBLISHER_INFO, the data element PUBLISHER_SECTION, if present, followed by the data element PUBLISHER_USE, if present.

When queryType=XSLM_QUERY_CUST_DEF_INFO, the data element CUSTOMER_ASSIGNED_APPL_INFO, if present.

When queryType=XSLM_QUERY_CERTIFICATE, the data element CERTIFICATE.

When queryType=XSLM_QUERY_CERT_RELATED_INFO, the data element CERTIFICATE_RELATED_INFORMATION

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

■**engineRecord**

```
protected abstract void engineRecord(long counterIdentifier,
                                     float counter,
                                     int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

SPI: Informs the license system about a metering activity in progress

Parameters:

counterIdentifier - the identifier of one of the 8 different counters (defined within the license certificate)

counter - the value of the number of units the license system should add to or subtract from the total count

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

engineReleaseLicense

```
protected abstract void engineReleaseLicense(long units,
                                             int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

SPI: Releases some or all license units associated with this instance of the license client

Parameters:
units - number of license units to be released

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

engineRequestLicense

```
protected abstract long engineRequestLicense(PublicKey publicKey,
                                             String namedUser,
                                             long confirmTime,
                                             long units,
                                             boolean forceNumber,
                                             int authenticationToken)
throws CommunicationException, InvalidParameterException,
LicenseCertificateException, LicenseClientException,
ResourceUnavailableException
```

SPI: Requests the license system for the specified number of units licenses.

Parameters:
publicKey - the software publisher public key. Used to confirm authenticity of a license certificate. The value passed must match the value of the publisher's public key contained in the license certificate.

namedUser - the named user for which license units are being requested

confirmTime - elapsed time (in seconds) whithin which the liscense system may expect the next confirm.

units - number of license units to be requested

forceNumber - whether or not a lower number of license units than that specified should be granted

authenticationToken - a 32-bit arbitrary value created by the application and used as part of the licensing system authentication process

Returns:
number of license units granted

Throws: CommunicationException
if a communication error occurs between the license client and the license server

Throws: InvalidParameterException
if a parameter passed to a method is invalid

Throws: LicenseCertificateException
if the requested method cannot be completed on any license crtificate

Throws: LicenseClientException
if a generic license client error occurs

Throws: ResourceUnavailableException
if a resource required to execute the requested method is not available

---

Class org.opengroup.xslm.LicenseUseManagement

```
java.lang.Object
   |
   +----org.opengroup.xslm.LicenseUseManagement
```

public final class **LicenseUseManagement**
extends Object
This class centralizes all License Use Management properties and common methods. One of its primary uses is to manage License System Publishers.

Method Index

Methods

■insertPublisherAt

```
public static int insertPublisherAt(Publisher publisher,
                                    int position)
```

Adds a new publisher, at a specified position. The position is the preference order in which publishers are chosen when no specific publisher is requested. Note that it is not guaranteed that this preference will be respected. The position is 1-based, that is, 1 is most preferred, followed by 2, and so on. Sometimes it will be legal to add a publisher, but only in the last position, in which case the `position` argument will be ignored.

A publisher cannot be added if it is already installed.

Parameters:
publisher - the publisher to be added.

position - the preference position that the caller would like for this publisher.

Returns:
the actual preference position in which the publisher was added, or -1 if the publisher was not added because it is already installed.

See Also:
getPublisher, removePublisher

addPublisher

```
public static int addPublisher(Publisher publisher)
```
Adds a publisher to the next position available.

Parameters:
publisher - the publisher to be added.

Returns:
the preference position in which the publisher was added, or -1 if the publisher was not added because it is already installed.

See Also:
getPublisher, removePublisher

removePublisher

```
public static void removePublisher(String name)
```
Removes the publisher with the specified name. This method returns silently if the publisher is not installed.

Parameters:
name - the name of the publisher to remove.

See Also:
getPublisher, addPublisher

getPublishers

```
public static Publisher[] getPublishers()
```
Returns all publishers currently installed.

Returns:
an array of all publishers currently installed.

getPublisher

```
public static Publisher getPublisher(String name)
```
Returns the publisher installed with the specified name, if any. Returns null if no publisher with the speicified name is installed.

Parameters:
name - the name of the publisher to get.

Returns:
the publisher of the specified name.

See Also:
removePublisher, addPublisher

■getProperty

```
public static String getProperty(String key)
```

Gets a licenseUseManagement property.

Parameters:
key - the key of the property being retrieved.

Returns:
the value of the licenseUseManagement property corresponding to key.

■setProperty

```
public static void setProperty(String key,
                               String datum)
```

Sets a licenseUseManagement property.

Parameters:
key - the name of the property to be set.

datum - the value of the property to be set.

Class org.opengroup.xslm.Publisher

```
java.lang.Object
   |
   +----java.util.Dictionary
           |
           +----java.util.Hashtable
                   |
                   +----java.util.Properties
                           |
                           +----org.opengroup.xslm.Publisher
```

public abstract class **Publisher**
extends Properties This class represents a "License System Publisher", publisher for short, for the Java XSLM API. A Publisher implements the Base functional set of Java XSLM and some or all functional towers.
Each Publisher has a name and a version number, and is configured in each runtime it is installed in.

Constructor Index

Publisher(String, double, String)
Constructs a Publisher with the specified name, version number, and information.

getInfo()
Returns a human-readable description of the publisher and its services.

getName()
Returns the name of this publisher.

getVersion()
Returns the version number for this publisher.

toString()
Returns a string with the name and the version number of this publisher.

Constructors

Publisher

```
protected Publisher(String name,
                    double version,
                    String info)
```

Constructs a Publisher with the specified name, version number, and information.

Parameters:

name - the publisher name.

version - the publisher version number.

info - a description of the publisher and its services.

Methods

getName

```
public String getName()
```
Returns the name of this publisher.

Returns:
the name of this publisher.

getVersion

```
public double getVersion()
```
Returns the version number for this publisher.

Returns:
the version number for this publisher.

getInfo

```
public String getInfo()
```
Returns a human-readable description of the publisher and its services. This may return an HTML page, with relevant links.

Returns:
a description of the publisher and its services.

toString

```
public String toString()
```
Returns a string with the name and the version number of this publisher.

Returns:
the string with the name and the version number for this publisher.

Overrides:
toString in class Hashtable

Class org.opengroup.xslm.Session

```
java.lang.Object
   |
   +----org.opengroup.xslm.Session
```

public abstract class **Session**
extends Object This Session class is used to establish a reference which is used to keep track of composite licensing activities as a whole. A session may include licenses for products from one or more publishers.
Like other classes in org.opengroup.xslm, the Session class has two major components:

Session API (Application Program Interface)
This is the interface of methods called by applications needing session objects to call the `AdvancedApplicationClient` methods. The API consists of all public methods.

Session SPI (Service Provider Interface)
This is the interface implemented by licensing system publishers that supply specific license use management packages. It consists of all methods whose names are prefixed by `engine`. Each such method is called by a correspondingly-named public API method. For example, the `engineEnd` method is called by the `end` method. The SPI methods are abstract; publishers must supply a concrete implementation.

Also Session provides implementation-independent objects, whereby a caller (application code) requests just a session object and is handed back a properly initialized Session object to be used with a `AdvancedApplicationClient` provided by a `LicenseBroker`. It is also possible, if desired, to request a particular implementation of a session object from a particular publisher to be used with a `AdvancedApplicationClient` provided by a `LicenseAgent`. See the `getInstance` methods.
Thus, there are two ways to request a Session object: by not specifying any parameter, or by specifying a package publisher.
If no parameter is specified, the system will determine if there is an implementation of the requested session object available in the environment, and if there is more than one, if there is a preferred one.
If a package publisher is specified, the system will determine if there is an implementation of the session object in the package requested, and throw an exception if there is not.

Variable Index

STARTED
Possible state value, signifying that this session object has been started.

state
Current state of this session object.

STOPPED
Possible state value, signifying that this session object has been stopped.

UNINITIALIZED
Possible state value, signifying that this session object has not yet been initialized.

Constructor Index

Session()
Creates a Session object.

Method Index

begin()
Starts a session to keep track of composite licensing activity

end()
Terminates a session established by a `begin` method invocation

engineBegin()
SPI: Starts a session to keep track of composite licensing activity

engineEnd()
SPI: Terminates a session established by a `engineBegin` method invocation

getHandle()
Returns the handle of this session object.

getInstance()
Generates a Session object.

getInstance(String)
Generates a Session object as supplied from the specified publisher, if a Session subclass is available from the provider.

Variables

UNINITIALIZED
```
protected static final int UNINITIALIZED
```
Possible state value, signifying that this session object has not yet been initialized.

STARTED
```
protected static final int STARTED
```
Possible state value, signifying that this session object has been started.

STOPPED
```
protected static final int STOPPED
```
Possible state value, signifying that this session object has been stopped.

state
```
protected int state
```
Current state of this session object.

Constructors

Session

```
protected Session()
```
Creates a Session object.

Methods

■getInstance

```
public static Session getInstance()
throws NoSuchAPILevelException
```
Generates a Session object. Packages are searched until one implementing a Session object is found and an instance of that subclass is returned.

Returns:
the new Session object.

Throws: NoSuchAPILevelException
if an implementation for the Session class is not available in the environment

■getInstance

```
public static Session getInstance(String publisher)
throws NoSuchPublisherException, NoSuchAPILevelException
```
Generates a Session object as supplied from the specified publisher, if a Session subclass is available from the provider.

Parameters:
publisher - the name of the publisher.

Returns:
the new Session object.

Throws: NoSuchAPILevelException
if the Session subclass is not available in the package supplied by the requested publisher.

Throws: NoSuchPublisherException
if the publisher is not available in the environment.

See Also:
Publisher

■begin

```
public final void begin()
```
Starts a session to keep track of composite licensing activity

■end

```
public final void end()
```
Terminates a session established by a `begin` method invocation

getHandle

```
public final String getHandle()
```
Returns the handle of this session object.

Returns:
the handle of this session object.

engineBegin

```
protected abstract void engineBegin()
```
SPI: Starts a session to keep track of composite licensing activity

engineEnd

```
protected abstract void engineEnd()
```
SPI: Terminates a session established by a `engineBegin` method invocation

---

Class org.opengroup.xslm.CommunicationException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----org.opengroup.xslm.CommunicationException
```

public class CommunicationException
extends Exception This exception is thrown when a communication error occurrs between the LicenseAgent and a license server.

Constructors Index

CommunicationException()
Constructs a CommunicationException with no detail message.

CommunicationException(String)
Constructs a CommunicationException with the specified detail message.

Constructors

■CommunicationException

`public CommunicationException()`
Constructs a CommunicationException with no detail message. A detail message is a String that describes this particular exception.

■CommunicationException

`public CommunicationException(String msg)`
Constructs a CommunicationException with the specified detail message. A detail message is a String that describes this particular exception.

Parameters:
msg - the detail message.

Class org.opengroup.xslm.InvalidParameterException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----java.lang.RuntimeException
                           |
                           +----java.lang.IllegalArgumentException
                                   |
                                   +----
org.opengroup.xslm.InvalidParameterException
```

public class **InvalidParameterException**
extends IllegalArgumentException This exception is thrown when an invalid parameter is passed to a method.

Constructor Index

Constructors

InvalidParameterException

```
public InvalidParameterException()
```

Constructs an InvalidParameterException with no detail message. A detail message is a String that describes this particular exception.

InvalidParameterException

```
public InvalidParameterException(String msg)
```

Constructs an InvalidParameterException with the specified detail message. A detail message is a String that describes this particular exception.

Parameters:
msg - the detail message.

Class org.opengroup.xslm.LicenseCertificateException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----org.opengroup.xslm.LicenseCertificateException
```

public class **LicenseCertificateException**
extends Exception This exception is thrown when the requested method cannot be completed on any of the license certificates available, or to a particular license certificate if the application client is already bound to one.

Constructor Index

Constructors

LicenseCertificateException

```
public LicenseCertificateException()
```

Constructs a LicenseCertificateException with no detail message. A detail message is a String that describes this particular exception.

LicenseCertificateException

```
public LicenseCertificateException(String msg)
```

Constructs a LicenseCertificateException with the specified detail message. A detail message is a String that describes this particular exception.

Parameters:
msg - the detail message.

Class org.opengroup.xslm.LicenseClientException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----org.opengroup.xslm.LicenseClientException
```

public class **LicenseClientException**
extends Exception This exception is thrown when a generic `AdvancedApplicationClient` or `BasicApplicationClient` error occurs, as when a releaseLicense method is called before calling a requestLicense method.

Constructor Index

Constructors

LicenseClientException

```
public LicenseClientException()
```

Constructs a LicenseClientException with no detail message. A detail message is a String that describes this particular exception.

LicenseClientException

```
public LicenseClientException(String msg)
```

Constructs a LicenseClientException with the specified detail message. A detail message is a String that describes this particular exception.

Parameters:
msg - the detail message.

Class org.opengroup.xslm.NoSuchAPILevelException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----org.opengroup.xslm.NoSuchAPILevelException
```

public class **NoSuchAPILevelException**
extends Exception This exception is thrown when the requested API level is not available in the environment.

Constructor Index

Constructors

■NoSuchAPILevelException

`public NoSuchAPILevelException()`
Constructs a NoSuchAPILevelException with no detail message. A detail message is a String that describes this particular exception.

■NoSuchAPILevelException

`public NoSuchAPILevelException(String msg)`
Constructs a NoSuchAPILevelException with the specified detail message. A detail message is a String that describes this particular exception.

Parameters:
msg - the detail message.

Class org.opengroup.xslm.NoSuchPublisherException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----org.opengroup.xslm.NoSuchPublisherException
```

public class **NoSuchPublisherException**
extends Exception This exception is thrown when a particular License System Publisher is requested but is not available in the environment.

Constructor Index

Constructors

■NoSuchPublisherException

```
public NoSuchPublisherException()
```

Constructs a NoSuchPublisherException with no detail message. A detail message is a String that describes this particular exception.

■NoSuchPublisherException

```
public NoSuchPublisherException(String msg)
```

Constructs a NoSuchPublisherException with the specified detail message. A detail message is a String that describes this particular exception.

Parameters:
msg - the detail message.

Class org.opengroup.xslm.ResourceUnavailableException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----org.opengroup.xslm.ResourceUnavailableException
```

public class **ResourceUnavailableException**
extends Exception This exception is thrown when a resource required to execute the requestedp method is not available in the environment.

Constructor Index

Constructors

■ResourceUnavailableException

```
public ResourceUnavailableException()
```

Constructs a ResourceUnavailableException with no detail message. A detail message is a String that describes this particular exception.

■ResourceUnavailableException

```
public ResourceUnavailableException(String msg)
```

Constructs a ResourceUnavailableException with the specified detail message. A detail message is a String that describes this particular exception.

Parameters:
msg - the detail message.

Glossary

Terminology

administrator
A person, or program, that manages software licenses; this includes receiving and acting upon messages from the licensing system.

application agent
A licensing-system provided component that provides an interface between an application program (usually via an XSLM Broker) and the licensing system's licensing server.

customer
See **licensee**.

end-user
A person using an instance of an application program.

grace period
A time period during which the defined license type terms are permitted by the software application publisher to be exceeded.

hard stop license policy
Specifies that the licensing system will not grant a license request when there are no available licenses using the non-Soft Stop data elements. See also Soft Stop License Policy.

license agreement
A contract providing a license to use a program, including all terms and conditions governing such use.

license certificate
A machine-readable representation of the terms and conditions contained within a license agreement, together with certain license-management-related information that is not directly included within the license agreement, such as encoded passwords.

license certificate issuer
The entity that creates a license certificate associated with a license agreement. Often, the license certificate issuer is the software publisher, but it can also be a software distributor, a software reseller, or a separate certificate issuing authority.

license policy
Collection of all terms and conditions, agreed upon between a license certificate issuer and a customer, that govern the customers rights to use a software product. The license policy, or just policy, includes actions the software publishers application will take when a license condition is not met, for example hard stop or soft stop. The policy also includes optional customer-specified conditions, if permitted by the software publisher, such as implement hard stop even if the software publisher normally implements soft stop.

license type
The scope of use of a specific product. It specifies a set of restrictions that are defined in the license policy.

license use management
The ability of the customer to manage the use of an organizations software assets within the limits defined by the license policies.

licensee
One of the two parties to a licensing agreement, obtaining the right to use a software product subject to certain terms and conditions.

licensing system
A software product that implements the Software License Use Management (XSLM) specification.

licensing system publisher
A provider of a licensing system.

platform
Combination of computing hardware and operating system. Also called environment.

policy
See license policy.

soft stop license policy
Specifies that the licensing system will also use the values in the following Soft Stop data elements when granting a license request, which can not be satisfied by the non-Soft Stop data elements alone — LICENSED_ADDITIONAL_UNITS or CAPACITY_ADDITIONAL or COUNTER_ADDITIONAL_VALUE and DURATION_ADDITIONAL. Licenses granted under the Soft Stop policy must be logged. See also Hard Stop License Policy.

software publisher
An entity that owns a software product and, generally, sets the licensing terms and conditions of the licensing of the software; the software publisher may also manufacture, distribute, and market the software.

technical license manager
A software product, used to restrict a licensee's ability to use a software product as specified within the governing license agreement, but not providing any additional significant management capabilities. A technical license manager does not provide an implementation of this specification.

Acronyms

API	Application Programming Interface
CAE	Common Applications Environment
CIM	Common Information Model
DLL	Dynamic Link Library
DLM	Dynamically Loaded Module
GUID	Globally Unique Identifier
HWM	High Water Mark
ISO	International Standards Organization
JES	Job Entry Subsystem
LPAR	Logically Partitioned (Mode)
LS	Licensing System
LSAPI	License Service Application Programming Interface
MIPS	Millions of Instructions Per Second
MNLU	Maximum Number of License Units
OSF	The Open Software Foundation, Inc.
Ts&Cs	Terms and Conditions
TLM	Technical License Manager
UCS	Unicode Character Set
UTC	Universal Time Coordinate
UTF	Unified Test System
UUID	Universal Unique Identifier (see also GUID)
XAAPI	XSLM Application API
XLCA	XSLM License Certificate Architecture
XMAPI	XSLM Management API
XSLM	X/Open Software License Use Management

Index