Sun Java™ System

Communications Services 6 Event Notification Service Guide

2004Q2

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 817-5700-10

Copyright © 2004 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at http://www.sun.com/patents and one or more additional patents or pending patent applications in the U.S. and in other countries.

THIS PRODUCT CONTAINS CONFIDENTIAL INFORMATION AND TRADE SECRETS OF SUN MICROSYSTEMS, INC. USE, DISCLOSURE OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF SUN MICROSYSTEMS, INC.

U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

This distribution may include materials developed by third parties.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and in other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, Java, Solaris, JDK, Java Naming and Directory Interface, JavaMail, JavaHelp, J2SE, iPlanet, the Duke logo, the Java Coffee Cup logo, the Solaris logo, the SunTone Certified logo and the Sun ONE logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon architecture developed by Sun Microsystems, Inc.

Legato and the Legato logo are registered trademarks, and Legato NetWorker, are trademarks or registered trademarks of Legato Systems, Inc. The Netscape Communications Corp logo is a trademark or registered trademark of Netscape Communications Corporation.

The OPEN LOOK and Sun(TM) Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Products covered by and information contained in this service manual are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright © 2004 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. détient les droits de propriété intellectuels relatifs à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et ce sans limitation, ces droits de propriété intellectuelle peuvent inclure un ou plus des brevets américains listés à l'adresse http://www.sun.com/patents et un ou les brevets supplémentaires ou les applications de brevet en attente aux Etats - Unis et dans les autres pays.

CE PRODUIT CONTIENT DES INFORMATIONS CONFIDENTIELLES ET DES SECRETS COMMERCIAUX DE SUN MICROSYSTEMS, INC. SON UTILISATION, SA DIVULGATION ET SA REPRODUCTION SONT INTERDITES SANS L AUTORISATION EXPRESSE, ECRITE ET PREALABLE DE SUN MICROSYSTEMS, INC.

Cette distribution peut comprendre des composants développés par des tierces parties.

Des parties de ce produit pourront être dérivées des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, Java, Solaris, JDK, Java Naming and Directory Interface, JavaMail, JavaHelp, J2SE, iPlanet, le logo Duke, le logo Java Coffee Cup, le logo Solaris, le logo SunTone Certified et le logo Sun[tm] ONE sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

Legato, le logo Legato, et Legato NetWorker sont des marques de fabrique ou des marques déposées de Legato Systems, Inc. Le logo Netscape Communications Corp est une marque de fabrique ou une marque déposée de Netscape Communications Corporation.

L'interface d'utilisation graphique OPEN LOOK et Sun(TM) a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une license non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui, en outre, se conforment aux licences écrites de Sun.

Les produits qui font l'objet de ce manuel d'entretien et les informations qu'il contient sont regis par la legislation americaine en matiere de controle des exportations et peuvent etre soumis au droit d'autres pays dans le domaine des exportations et importations. Les utilisations finales, ou utilisateurs finaux, pour des armes nucleaires, des missiles, des armes biologiques et chimiques ou du nucleaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou reexportations vers des pays sous embargo des Etats-Unis, ou vers des entites figurant sur les listes d'exclusion d'exportation americaines, y compris, mais de maniere non exclusive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une facon directe ou indirecte, aux exportations des produits ou des services qui sont regi par la legislation americaine en matiere de controle des exportations et la liste de ressortissants specifiquement designes, sont rigoureusement interdites.

LA DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFACON.

Contents

List of Tables 7

About This Manual 9

Who Should Read This Manual 9
What You Need to Know 9
How This Manual is Organized 10
Document Conventions 10
  Monospaced Font 10
  Bold Monospaced Font 11
  Italicized Font 11
  Square or Straight Brackets 11
  Command Line Prompts 11
  Platform-specific Syntax 12
Where to Find Related Information 12
  Messaging Server Documents 12
  Calendar Server Documents 13
  Communications Services Documents 13
Where to Find This Manual Online 13
Related Third-Party Web Site References 14

Chapter 1 Introduction to Event Notification Service 15
Event Notification Service Overview 15
  ENS in Calendar Server 16
  ENS in Messaging Server 16
  Event References 17
  ENS Connection Pooling 18
Event Notification Service Architecture 19
  Notify 20
  Subscribe 21
  Unsubscribe 21

How Calendar Server Interacts with ENS 21
How Messaging Server Interacts with ENS 26
Event Notification Service API Overview 28
ENS C API Overview 28
ENS Java API Overview 29
Building and Running Custom Applications 30

Chapter 2 Event Notification Service C API Reference 35
Publisher API Functions List 35
Subscriber API Functions List 36
Publish and Subscribe Dispatcher Functions List 36
Publisher API 37
publisher_t 37
publisher_cb_t 38
publisher_new_a 38
publisher_new_s 39
publish_a 40
publish_s 40
publisher_delete 41
publisher_get_subscriber 41
renl_create_publisher 42
renl_cancel_publisher 43
Subscriber API 43
subscriber_t 44
subscription_t 44
subscriber_cb_t 44
subscriber_notify_cb_t 45
subscriber_new_a 45
subscriber_new_s 46
subscribe_a 47
unsubscribe_a 48
subscriber_delete 48
subscriber_get_publisher 49
renl_create_subscriber 49
renl_cancel_subscriber 50
Publish and Subscribe Dispatcher API 50
pas_dispatcher_t 50
pas_dispatcher_new 51
pas_dispatcher_delete 51
pas_dispatch 52
pas_shutdown 52

Chapter 3 Event Notification Service Java (JMS) API Reference . . . 53
Event Notification Service Java (JMS) API Implementation . . . 53
Prerequisites to Use the Java API . . . 53
Sample Java Programs . . . 54
Instructions for Sample Programs . . . 54
Java (JMS) API Overview . . . 57
New Proprietary Methods . . . 57
com.iplanet.ens.jms.EnsTopicConnFactory . . . 57
com.iplanet.ens.jms.EnsTopic . . . 58
Implementation Notes . . . 58
Shortcomings of the Current Implementation . . . 58
Notification Delivery . . . 59
JMS Headers . . . 59
Miscellaneous . . . 60

Chapter 4 Messaging Server Specific Information . . . 61
Event Notification Types and Parameters . . . 61
Parameters . . . 63
Payload . . . 66
Examples . . . 67
Sample Code . . . 67
Sample Publisher . . . 68
Sample Subscriber . . . 71
Implementation Notes . . . 73

Chapter 5 Calendar Server Specific Information . . . 75
Calendar Server Notifications . . . 75
Alarm Notifications . . . 76
Calendar Update Notifications . . . 77
Advanced Topics . . . 78
WCAP appid parameter and X-Tokens . . . 78
ENS Sample Code for Calendar Server . . . 79
Sample Publisher and Subscriber . . . 79
Reliable Publisher and Subscriber . . . 85

Appendix A Debugging ENS . . . 93
Environment Variables . . . 93
GAP_DEBUG . . . 94
GAP_LOG_MODULES . . . 94
GAP_LOGFILE . . . 95
XENP_TRACE . . . 95
ENS_DEBUG . . . 95

ENS_LOG_MODULES 96
ENS_LOGFILE 97
ENS_STATS 97
SERVICEBUS_DEBUG 97
How to Enable Debug Tracing 97
Sample Debugging Sessions 98
Example 1: For Messaging Server 99
Example 2: For Messaging Server 101

Glossary 103

Index 105

List of Tables

Table 1-1 Sample ENS Publish and Subscribe Cycle 25
Table 2-1 ENS Publisher API Functions List 35
Table 2-2 ENS Subscriber API Functions List 36
Table 2-3 ENS Publish and Subscribe Dispatcher Functions List 36
Table 4-1 Event Tyes 61
Table 4-2 Mandatory Event Reference Parameters 63
Table 4-3 Optional Event Reference Parameters 64
Table 4-4 Available Parameters for Each Event Type 65
Table 4-5 Payload Configuration Parameters 66
Table 5-1 Alarm Notifications 76
Table 5-2 Calendar Update Notifications 77
Table 5-3 Advanced Topics Parameter 78
Table 5-4 Presence of appid and Value of X-Token X-NSCP-COMPONENT-SOURCE 79
Table A-1 Trace Level Values 94
Table A-2 GAP_LOG_MODULES Values 95
Table A-3 ENS_DEBUG Trace Level Values 96
Table A-4 ENS_LOG_MODULES Values 96

About This Manual

This manual describes the Event Notification Service (ENS) architecture and APIs for Sun Java™ System Messaging Server and Sun Java™ System Calendar Server. It gives detailed instructions on the ENS APIs that you can use to customize your server installation.

Topics covered in this chapter include:

- Who Should Read This Manual
- Who Should Read This Manual
- How This Manual is Organized
- Document Conventions
- Where to Find Related Information
- Where to Find This Manual Online
- Related Third-Party Web Site References

Who Should Read This Manual

This manual is for programmers who want to customize applications in order to implement Messaging Server and Calendar Server.

What You Need to Know

This book assumes that you are a programmer with a knowledge of C/C++ and Java Messaging Service, and that you have a general understanding of the following:

- The Internet and the World Wide Web
- Messaging and calendaring concepts

How This Manual is Organized

This manual contains the following chapters and appendix:

- About This Manual (this chapter)
- Chapter 1, "Introduction to Event Notification Service"

  This chapter describes the Event Notification Service (ENS) components, architecture, and Application Programming Interfaces (APIs).
- Chapter 2, "Event Notification Service C API Reference"

  This chapter describes the ENS C API.
- Chapter 3, "Event Notification Service Java (JMS) API Reference"

  This chapter describes the ENS Java API and provides sample code.
- Chapter 5, "Calendar Server Specific Information"

  This chapter describes the Calendar Server event notifications and provides sample Calendar Server code.
- Chapter 4, "Messaging Server Specific Information"

  This chapter describes the Messaging Server event references and provides sample Messaging Server code.
- "Glossary"

Document Conventions

Monospaced Font

`Monospaced font` is used for any text that appears on the computer screen or text that you should type. It is also used for filenames, distinguished names, functions, and examples.

Bold Monospaced Font

`Bold monospaced font` is used to represent text within a code example that you should type. For example, you might see something like this:

`./installer`

In this example, **`./installer`** is what you would type at the command line.

Italicized Font

Italicized font is used to represent text that you enter using information that is unique to your installation (for example, variables). It is used for server paths and names.

For example, throughout this document you will see path references of the form:

msg_svr_base`/...`

The Messaging Server Base (*msg_svr_base*) represents the directory path in which you install the server. The default value of the *msg_svr_base* is `/opt/SUNWmsgsr`.

The Calendar Server Base (*cal_svr_base*) represents the directory path in which you install the server. The default value of the *cal_svr_base* is `/opt/SUNWics5`.

Square or Straight Brackets

Square (or straight) brackets `[]` are used to enclose optional parameters.

Command Line Prompts

Command line prompts (for example, `%` for a C-Shell, or `$` for a Korn or Bourne shell) are not displayed in the examples. Depending on which operating system you are using, you will see a variety of different command line prompts. However, you should enter the command as it appears in the document unless specifically noted otherwise.

Platform-specific Syntax

All paths specified in this manual are in UNIX format. If you are using Windows 2000-based Communications Services, you should assume the Windows 2000 equivalent file paths whenever UNIX file paths are shown in this book.

Where to Find Related Information

In addition to this manual, Communications Services products come with supplementary information for administrators, end users, and developers.

Messaging Server Documents

Use the following URL to see all the Messaging Server documentation:

`http://docs.sun.com/coll/MessagingServer_04q2`

The following documents are available:

- *Sun Java™ System Messaging Server Release Notes*
- *Sun Java™ System Messaging Server Deployment Planning Guide*
- *Sun Java™ System Messaging Server Administration Guide*
- *Sun Java™ System Messaging Server Administration Reference*
- *Sun Java™ System Messaging Server Developer's Reference*
- *Sun Java™ System Messaging Server Messenger Express Customization Guide*

The Messaging Server product suite contains other products such as Sun Java™ System Console, Directory Server, and Administration Server. Documentation for these and other products can be found at the following URL:

`http://docs.sun.com/db/prod/sunone`

In addition to the software documentation, see the Messaging Server Software Forum for technical help on specific Messaging Server product questions. The forum can be found at the following URL:

`http://swforum.sun.com/jive/forum.jsp?forum=15`

Calendar Server Documents

Use the following URL to see all the Calendar Server documentation:

http://docs.sun.com/coll/CalendarServer_04q2

The following documents are available:

- *Sun Java™ System Calendar Server Release Notes*
- *Sun Java™ System Calendar Server Administration Guide*
- *Sun Java™ System Calendar Server Developer's Guide*

Communications Services Documents

Use either one of the following URLs to see the documentation that applies to all Communications Services products:

http://docs.sun.com/coll/MessagingServer_04q2

or

http://docs.sun.com/coll/CalendarServer_04q2

The following documents are available:

- *Sun Java™ System Communications Services User Management Utility Administration Guide*
- *Sun Java System Communications Services Enterprise Deployment Planning Guide*
- *Sun Java™ System Communications Services Schema Migration Guide*
- *Sun Java™ System Communications Services Schema Reference*
- *Sun Java™ System Communications Services Event Notification Service Guide*
- *Sun Java™ System Communications Express Administration Guide*
- *Sun Java™ System Communications Express Customization Guide*

Where to Find This Manual Online

You can find the *Communications Services Event Notification Service Guide* online in PDF and HTML formats. This book can be found at the following URLs:

http://docs.sun.com/doc/817-5700

Related Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

NOTE Sun is not responsible for the availability of third-party Web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Chapter 1

Introduction to Event Notification Service

This chapter provides an overview of the Event Notification Service (ENS) components, architecture, and Application Programming Interfaces (APIs).

This chapter contains these sections:

- Event Notification Service Overview
- Event Notification Service Architecture
- Event Notification Service API Overview

Event Notification Service Overview

The Event Notification Service (ENS) is the underlying publish-and-subscribe service available in the following Sun Java™ System communications products:

- Calendar Server
- Messaging Server

NOTE	See Appendix C in the *Messaging Server Administration Guide* for instructions on enabling and administering ENS in Messaging Server.

ENS acts as a dispatcher used by Sun Java™ System applications as a central point of collection for certain types of *events* that are of interest to them. Events are changes to the value of one or more properties of a resource. In this structure, a URI (Uniform Resource Identifier) represents an event. Any application that wants to know when these types of events occur registers with ENS, which identifies events in order and matches notifications with subscriptions.

Event examples include:

- Arrival of new mail to a user's inbox
- User's mailbox has exceeded its quota
- Calendar reminders

Specifically, ENS accepts reports of events that can be categorized, and notifies other applications that have registered an interest in certain categories of events.

ENS provides a server and APIs for publishers and subscribers. A publisher makes an event available to the notification service; and a subscriber tells the notification service that it wants to receive notifications of a specific event. See "Event Notification Service API Overview" on page 28 for more information on the ENS APIs.

ENS in Calendar Server

By default, ENS is enabled in Calendar Server. For Calendar Server you do not need to do anything else to use ENS.

A user who wants to subscribe to notifications other than the alarms generated by Calendar Server needs to write a subscriber.

Sample ENS C publisher and subscriber code is bundled with Calendar Server. (See "ENS Sample Code for Calendar Server" on page 79.) Once Calendar Server is installed, the code can be found in the following directory:

```
/opt/SUNWics5/cal/csapi/samples/ens
```

ENS in Messaging Server

ENS and iBiff (the ENS publisher for Messaging Server, also referred to as the notification plug-in to Messaging Server) are bundled in Messaging Server and ENS is enabled. However, the iBiff plug-in file, `libibiff`, is not automatically loaded at installation.

To subscribe to notifications, you need to first perform the following two actions on the Messaging Server host:

- Load the iBiff notification plug-in
- Stop and restart the messaging server

See Appendix C in the *Messaging Server Administration Guide* for further instructions.

A user who wants to subscribe to Messaging Server notifications needs to write a subscriber to the ENS API. To do so, the subscriber needs to know what the various Messaging Server notifications are. See Chapter 4, "Messaging Server Specific Information" for that information.

Messaging Server comes bundled with sample ENS C publisher and subscriber code. See "Sample Code" on page 67 for more information.

Sample Messaging Server code is provided with the product in the following directory:

msg_server_base/`examples`

Event References

Event references identify an event handled by ENS. Event references use the following URI syntax (as specified by RFC 2396):

scheme`://`*authority resource*`/[?`*param1*`=`*value*`1&`*param2*`=`*value2*`&`*param3*`=`*value3*`]`

where:

- *scheme* is the access method, such as `http`, `imap`, `ftp`, or `wcap`.

  For Calendar Server and Messaging Server, the ENS scheme is `enp`.
- *authority* is the DNS domain or host name that controls access to the resource.
- *resource* is the path leading to the resource in the context of the authority. It can be composed of several path components separated by a slash ("`/`").
- *param* is the name of a parameter describing the state of a resource.
- *value* is its value. There can be zero or more parameter/value pairs.

In general, all Calendar Server events start with the following:

```
enp:///ics
```

The Messaging Server notification plug-in iBiff uses the following scheme and resource by default:

```
enp://127.0.0.1/store
```

NOTE Although the event reference has a URI syntax, the scheme, authority, and resource have no special significance. They are merely used as strings with no further interpretation in ENS.

Calendar Server Event Reference Example

The following is an example event reference URI to subscribe to all event alarms with a calendar ID of `jac`:

```
enp:///ics/alarm?calid=jac
```

NOTE This URI is not meant to be used by end users.

Messaging Server Event Reference Example

The following is an example event reference that requests a subscription to all `NewMsg` events for a user whose user ID is `blim`:

```
enp://127.0.0.1/store?evtType=NewMsg&mailboxName=blim
```

When using ENS with Messaging Server, the user ID you specify is case sensitive.

NOTE This URI is not meant to be used by end users.

ENS Connection Pooling

The connection pooling feature of ENS enables a pool of subscribers to receive notifications from a single event reference. For every event, ENS chooses one subscriber from the pool to send the notification to. Thus, only one subscriber in the pool receives the notification. The ENS server balances sending of notifications among the subscribers. This enables the client to have a pool of subscribers that work together to receive all notifications from a single event reference.

For example, if notifications are being published to the event reference `enp://127.0.0.1/store`, a subscriber will normally subscribe to this event reference to receive notifications. To have a pool of subscribers receive all the notifications to this event reference, each subscriber in the pool only needs to subscribe to the event reference `enp+pool://127.0.0.1/store` instead. The ENS server chooses one subscriber from the pool to send the notification to.

NOTE The publisher still sends notifications to the simple event reference, in the example above `enp://127.0.0.1/store`, that is, the publisher has no knowledge of the subscriber pool.

Multiple Pool Extension

Connection pooling can support multiple pools of subscribers. That is, you can have two pools of subscribers, each pool receiving all the notifications from the event reference. The syntax of the event reference for the subscriber is:

enp+pool[.*poolid*]://domain/event

where *poolid* is a string using only base64 alphabet. (See RFC1521, Table 1, for what the base64 alphabet contains.) So, for example, to have two pools of subscribers to the event reference `enp://127.0.0.1/store`, each pool could subscribe to the following event references:

`enp+pool.1://127.0.0.1/store` – for first pool of subscribers
`enp+pool.2://127.0.0.1/store` – for second pool of subscribers

Event Notification Service Architecture

On the Solaris platform, ENS runs as a daemon, `enpd`, along with other daemons in various calendar or messaging server configurations, to collect and dispatch events that occur to properties of resources. On Windows platforms, ENS runs as a service, `enpd.exe`.

For ENS, an event is a change that happens to a resource, while a resource is an entity such as a calendar or inbox. For example, adding an entry to a calendar (the resource) generates an event, which is stored by ENS. This event can then be subscribed to, and a notification would then be sent to the subscriber.

The ENS architecture enables the following three things to occur:

- Notification - This is a message that describes an event occurrence. Sent by the event publisher, it contains a reference to the event, as well as any additional parameter/value pairs added to the URI, and optional data (the payload) used by the event consumers, but opaque to the notification service. Whoever is interested in the event can subscribe to it.
- Subscription - This is a message sent to subscribe to an event. It contains an event reference, a client-side request identifier, and optional parameter/value pairs added to the URI. The subscription applies to upcoming events (that is, a subscriber asks to be notified of upcoming events).
- Unsubscription - This message cancels (unsubscribes) an existing subscription. An event subscriber tells ENS to stop relaying notifications for the specified event.

Notify

ENS notifies its subscribers of an event by sending a notification. Notify is also referred to as "publish." A notification can contain the following items:

- An event reference (which, optionally, can contain parameter/value pairs)
- Optional application-specific data ("opaque" for ENS, but the publisher and subscriber agree apriori to the format of the data)

The optional application-specific data is referred to as the "payload."

There are two kinds of notifications:

- **Unreliable notification** - Notification sent from an event publisher to a notification server. If the publisher does not know nor care about whether there are any consumers, or whether they get the notification, this request does not absolutely need to be acknowledged. However, a publisher and a subscriber, who are mutually aware of each other, can agree to set up a reliable event notification link (RENL) between themselves. In this case, once the subscriber has processed the publisher's notification, it sends an acknowledgment notification back to the publisher.
- **Reliable notification** - Notification sent from a server to a subscriber as a result of a subscription. This type of notification should be acknowledged. A reliable notification contains the same attributes as an unreliable notification.

See "Publisher API" on page 37 for more information.

Subscribe

ENS receives a request to be notified of events. The request sent by the event subscriber is a subscription. The subscription is valid during the life of the session, or until it is cancelled (unsubscribed).

A subscription can contain the following items:

- An event reference (which, optionally, can contain parameter/value pairs)
- A request identifier

See "Subscriber API" on page 43 for more information.

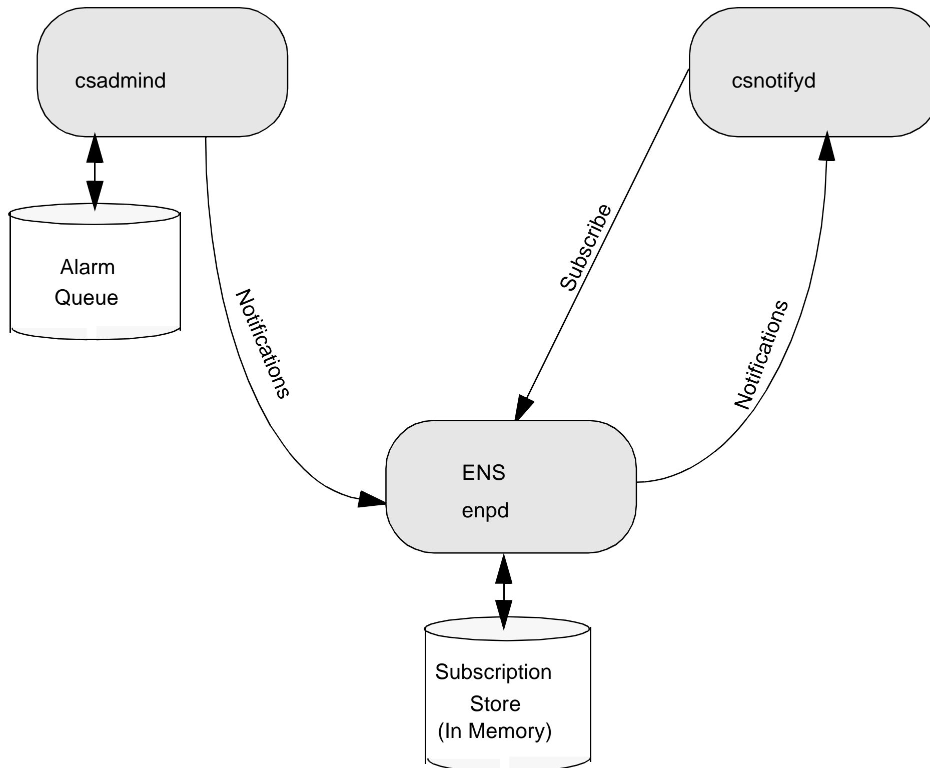
Unsubscribe

ENS receives a request to cancel an existing subscription. See "Subscriber API" on page 43 for more information.

How Calendar Server Interacts with ENS

Figure 1-1 on page 22 shows how ENS interacts with Calendar Server through the alarm queue and two daemons, `csadmind` and `csnotifyd`.

Figure 1-1 ENS in Calendar Server Overview



Calendar Server Alarm Queue

ENS is an alarm dispatcher. This decouples alarm delivery from alarm generation. It also enables the use of multiple delivery methods, such as email and wireless communication. The `csadmind` daemon detects events by sensing changes in the state of the alarm queue. The alarm queue's state changes every time an alarm is placed in the queue. An alarm is queued when a calendar event generates an alarm. The following URIs represent these kind of events:

for events:

`enp:///ics/eventalarm?calid=`***calid***`&uid=`***uid***`&rid=`***rid***`&aid=`***aid***

for todos (tasks):

`enp:///ics/todoalarm?calid=`***calid***`&uid=`***uid***`&rid=`***rid***`&aid=`***aid***

where:

- *calid* is the calendar ID.
- *uid* is the event/todo (task) ID within the calendar.
- *rid* is the recurrence id for a recurring event/todo (task).
- *aid* is the alarm ID within the event/todo (task). In case there are multiple alarms, the *aid* identifies the correct alarm.

The publisher `csadmind` dequeues the alarms and sends notifications to `enpd`. The `enpd` daemon then checks to see if anyone is subscribed to this kind of event and sends notifications to the subscriber, `csnotifyd`, for any subscriptions it finds. Other subscribers to alarm notifications (reminders) can be created and deployed within an Calendar Server installation. These three daemons interacting together implement event notification for Calendar Server.

Calendar Server Daemons

Calendar Server includes two daemons that communicate to the ENS daemon, `enpd`:

- `csadmind`

  The `csadmind` daemon contains a publisher that submits notifications to the notification service by sending alarm events to ENS. It manages the Calendar Server alarm queue. It implements a scheduler, which lets it know when an alarm has to be generated. At such a point, `csadmind` publishes an event. ENS receives and dispatches the event notification.

  To ensure alarm transfer reliability, `csadmind` requires acknowledgment for certain events or event types. (See "Alarm Transfer Reliability" on page 24.) The `csadmind` daemon uses Reliable Event Notification Links (RENLs) to accomplish acknowledgment.

- `csnotifyd`

  The `csnotifyd` daemon is the subscriber that expresses interest in particular events (subscribes), and receives notifications about these subscribed-to events from ENS, and sends notice of these events and todos (tasks) to its clients by email.

Though the ability to unsubscribe is part of the ENS architecture, `csnotifyd` does not bother to unsubscribe to events for the following two reasons: there is no need to unsubscribe or resubscribe during normal runtime; and due to the temporary nature of the subscriptions store (it is held in memory), all subscriptions are implicitly unsubscribed when the connection to ENS is shutdown.

The `csnotifyd` daemon subscribes to `enp:///ics/alarm/`. The todo (task) or event is specified in a parameter.

Alarm Transfer Reliability

To ensure that no alarm ever gets lost, `csadmind` and `csnotifyd` use the RENL feature of ENS for certain types of alarms. For these alarms, `csadmind` requests an end-to-end acknowledgment for each notification it sends, while `csnotifyd`, after successfully processing it, generates a notification acknowledgment for each RENL alarm notifications it receives.

For these RENL alarms, should the network, the ENS daemon, or `csnotifyd` fail to handle a notification, `csadmind` will not receive any acknowledgment, and will not remove the alarm from the alarm queue. The alarm will, therefore, be published again after a timeout.

Calendar Server Example

A typical ENS publish and subscribe cycle for Calendar Server resembles the following:

1. The event subscriber, `csnotifyd`, expresses interest in an event (subscribes).
2. The event publisher, `csadmind`, detects events and sends notification (publishes).
3. ENS publishes the event to the subscriber.
4. The event subscriber cancels interest in the event (unsubscribes). This step happens implicitly when the connection to ENS is shutdown.

Figure 1-2 on page 25 illustrates this cycle and Table 1-1 on page 25 provides the narrative for the figure.

Figure 1-2 Example Event Notification Service Publish and Subscribe Cycle for Calendar Server

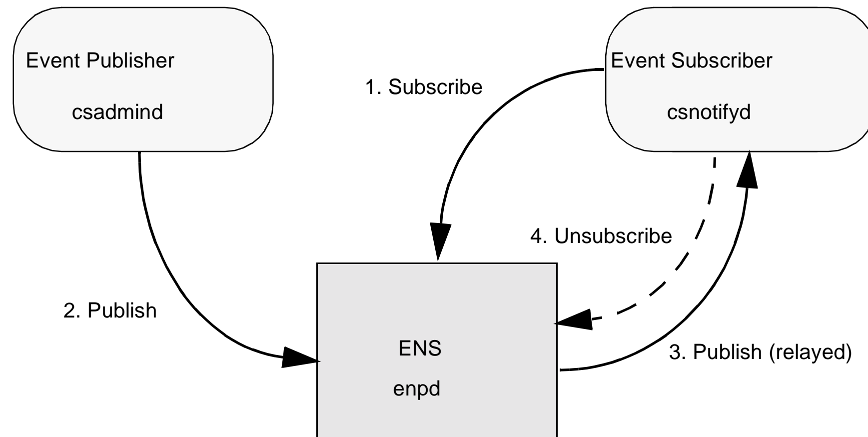


Table 1-1 Sample ENS Publish and Subscribe Cycle

Action	ENS Response
1. The `csnotifyd` daemon sends a subscription request to ENS.	ENS stores the subscription in the subscriptions database.
2. The `csadmind` daemon sends a notification request to ENS.	ENS queries the subscriptions database for subscriptions matching the notification.
3. The `csnotifyd` daemon receives a notification from ENS.	When ENS receives a notification from a publisher, it looks up its internal subscription table to find subscriptions matching the event reference of the notification. Then for each subscription, it relays a copy of the notification to the subscriber who owns this subscription.
4. Currently, `csnotifyd` does not bother sending cancellation requests to ENS.	Because the subscriptions store is in memory only (not in a database), all subscriptions are implicitly unsubscribed when the connection to ENS is shutdown.

How Messaging Server Interacts with ENS

Figure 1-3 on page 27 shows how ENS interacts with Messaging Server. In this figure, each oval represents a process, and each rectangle represents a host computer running the enclosed processes.

The ENS server delivers notifications from the Messaging Server notification plug-in to ENS clients (that is, iBiff subscribers). There is no guarantee of the order of notification prior to the ENS server because the events are coming from different processes (`MTA`, `stored`, and `imapd`).

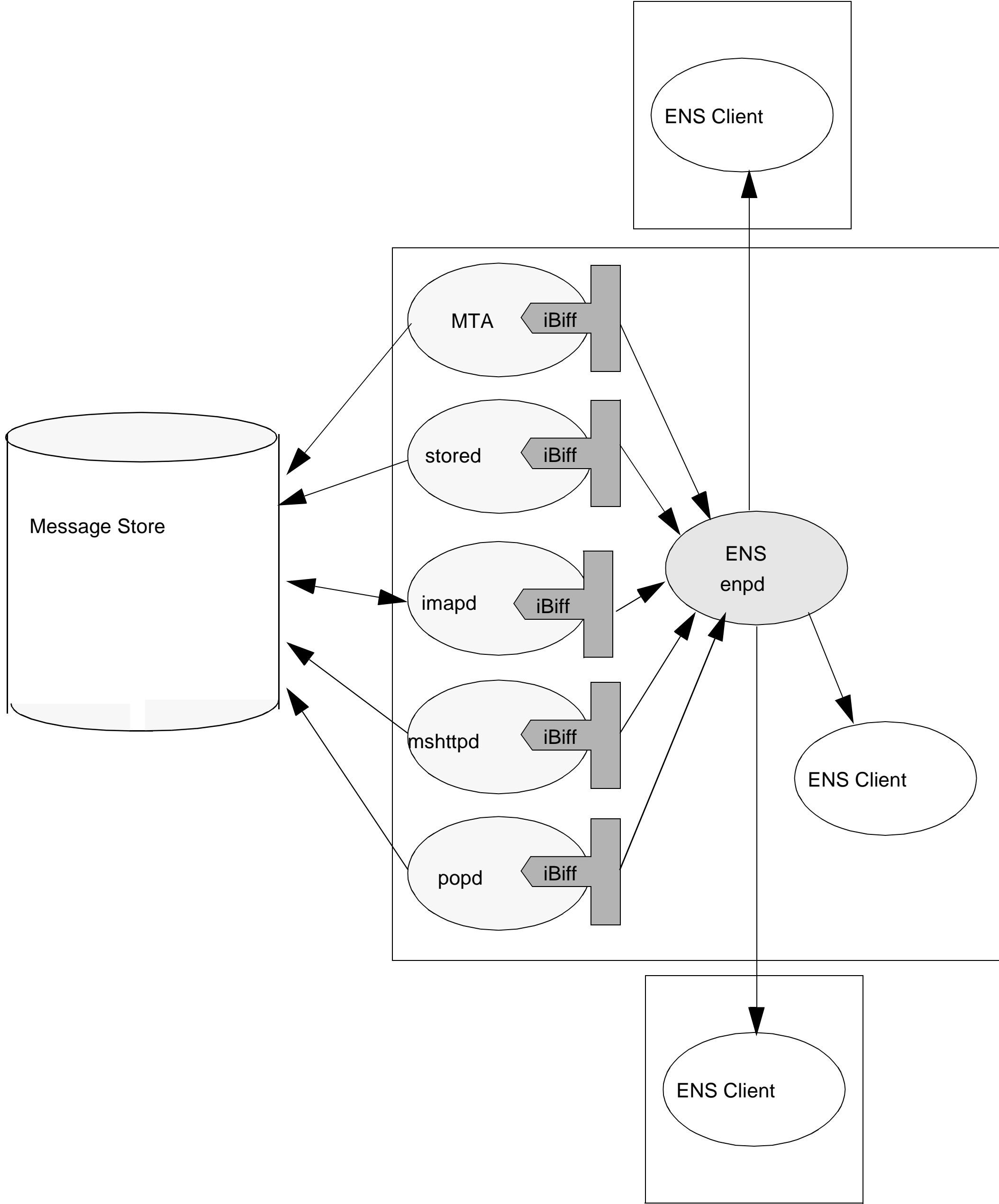
Notifications flow from the iBiff plug-in in the `MTA`, `stored`, and `imap` processes to ENS `enpd`. The ENS client subscribes to the ENS, and receives notifications. When iBiff is enabled, Messaging Server publishes the notifications with the iBiff plug-in, but no Messaging Server services subscribe to these notifications. A customer-provided ENS subscriber or client should be written to consume the notifications and do whatever is necessary. That is, Messaging Server itself does not depend on or use the notifications for its functions, and this is why ENS and iBiff are not enabled by default when you install Messaging Server.

The Messaging Server architecture enforces that a given set of mailboxes is served by a given host computer. A given mailbox is not served by multiple host computers. There are several processes manipulating a given mailbox but only one computer host serving a given mailbox. Thus, to receive notifications, end-users only need to subscribe to the ENS daemon that serves the mailbox they are interested in.

Messaging Server enables you to have either one ENS server for all mailboxes—that is, one ENS server for all the computer hosts servicing the message store—or multiple ENS servers, perhaps one ENS server per computer host. The second scenario is more scalable. Also, in this scenario, end users must subscribe to multiple ENS servers to get the events for mailboxes they are interested in.

Thus, the architecture requires an ENS server per computer host. The ENS servers and the client processes do not have to be co-located with each other or with messaging servers.

Figure 1-3 ENS in Messaging Server Overview



Event Notification Service API Overview

This section provides an overview of the two APIs for ENS, a C API and a Java API, which is a subset of the Java Messaging Service (JMS) API. Two sample Java subscribers are provided using the JMS API.

For detailed information on the Java (JMS) API, see Chapter 3, "Event Notification Service Java (JMS) API Reference." For JMS documentation, use the following URL:

`http://java.sun.com/products/jms/docs.html`

For detailed information on the ENS C API, see Chapter 2, "Event Notification Service C API Reference."

ENS C API Overview

ENS implements the following three APIs:

- Publisher API

  A publisher sends notification of a subscribed-to event to ENS, which then distributes it to the subscribers. Optionally, in Calendar Server, the application can request acknowledgment of receipt of the notification. To do this, a Reliable Event Notification Link (RENL) is necessary. An RENL has a publisher, a subscriber, and a unique ID, which identify notifications that are subject to acknowledgment. The publisher informs the application of the receipt of an acknowledgment by invoking the `end2end_ack` callback passed to `publish_a`. Currently, only Calendar Server supports RENL.

- Subscriber API

  A subscriber is a client to the notification service which expresses interest in particular events. When the notification service receives a notification about one of these events from a publisher, it relays the notification to the subscriber.

  A subscriber may also unsubscribe, which cancels an active subscription.

  In Calendar Server, to enable an RENL, the subscriber declares its existence to ENS, which then transparently generates notification acknowledgment on behalf of the subscriber application. The subscriber can revoke the RENL at any time.

- Publish and Subscribe Dispatcher API

When an asynchronous publisher is used, ENS needs to borrow threads from a thread pool in order to invoke callbacks. The application can either choose to create its own thread pool and pass it to ENS, or it can let ENS create and manage its own thread pool. In either case, ENS creates and uses a dispatcher object to instantiate the dispatcher used (`pas_dispatcher_t`).

GDisp (`libasync`) is the dispatcher supported.

ENS Java API Overview

The Java API for ENS uses a subset of the standard JMS API, with the addition of two new proprietary methods:

- `com.iplanet.ens.jms.EnsTopicConnFactory`
- `com.iplanet.ens.jms.EnsTopic`

The following list of JMS object classes is used in the Java API for ENS:

- `javax.jms.TopicSubscriber`
- `javax.jms.TopicSession`
- `javax.jms.TopicPublisher`
- `javax.jms.TopicConnection`
- `javax.jms.TextMessage`
- `javax.jms.Session`
- `javax.jms.MessageProducer`
- `javax.jms.MessageConsumer`
- `javax.jms.Message`
- `javax.jms.ConnectionMetaData`
- `javax.jms.Connection`

NOTE The Java API for ENS does not implement all the JMS object classes. When customizing, use only the object classes found on this list.

Building and Running Custom Applications

To assist you in building your own custom publisher and subscriber applications, Messaging Server and Calendar Server include sample code. This section tells you where to find the sample code, where the APIs' include (header) files are located, and where the libraries are that you need to build and run your custom programs.

NOTE This section applies to the C API only.

Location of Sample Code

Calendar Server

Calendar Server includes four simple sample programs to help you get started. The code for these samples resides in the following directory:

cal_server_base`/cal/csapi/samples/ens`

Messaging Server

Messaging Server 5.1 and higher contains sample programs to help you learn how to receive notifications. These sample programs are located in the following directory:

msg_server_base`/examples`

Location of Include Files

Calendar Server

The include (header) files for the publisher and subscriber APIs are: `publisher.h`, `suscriber.h`, and `pasdisp.h` (publish and subscribe dispatcher). They are located in the CSAPI `include` directory. The default `include` path is:

cal_server_base`/cal/csapi/include`

Messaging Server

The default `include` path for Messaging Server is:

msg_server_base`/bin/msg/enssdk/include`

Dynamically Linked/Shared Libraries

Calendar Server

Your custom code must be linked with the dynamically linked library `libens`, which implements the publisher and subscriber APIs. On some platforms all the dependencies of `libens` must be provided as part of the link directive. These dependencies, in order, are:

1. `libgap`
2. `libcyrus`
3. `libyasr`
4. `libasync`
5. `libnspr3`
6. `libplsd4`
7. `libplc3`

Calendar Server uses these libraries; therefore, they are located in the server's `bin` directory. The default `libens` path is:

`/opt/`***cal_server_base***`/cal/bin`

NOTE For Windows, in order to build publisher and subscriber applications, you also need the archive files (`.lib` files) corresponding to all the earlier mentioned libraries. These are located in the CSAPI library directory, `lib`. The default `lib` path is:

drive`:\ProgramFiles\iPlanet\CalendarServer5\cal\`
`csapi\lib`

Messaging Server

The libraries for Messaging Server are located in the following directory:

msg_server_base`/bin/msg/lib`

Refer to `msg_server_base/bin/msg/enssdk/examples/Makefile.sample` to help determine what libraries are needed.This makefile contains instructions on how to compile and run the `apub` and `asub` programs. This file also describes what libraries are needed, and what the `LD_LIBRARY_PATH` should be. Figure 1-4 shows a sample makefile.sample file.

Figure 1-4 `Makefile.sample` File

```
#
# Sample makefile
#
# your C compiler
CC = gcc

# LIBS
# Your library path should include <msg_server_base>/bin/msg/lib
LIBS = -lens -lgap -lxenp -lcyrus -lchartable -lyasr -lasync

all: apub asub

apub: apub.c
	$(CC) -o apub -I ../include apub.c $(LIBS)

asub: asub.c
	$(CC) -o asub -I ../include asub.c $(LIBS)

run:
	@echo 'run <msg_server_base>/start-ens'
	@echo run asub localhost 7997
	@echo run apub localhost 7997
```

NOTE The Windows distribution includes the following additional files:

msg_server_base\bin\msg\enssdk\examples

bin\msg\enssdk\examples\libens.lib

bin\msg\enssdk\examples\libgap.lib

bin\msg\enssdk\examples\libxenp.lib

bin\msg\enssdk\examples\libcyrus.lib

bin\msg\enssdk\examples\libchartable.lib

bin\msg\enssdk\examples\libyasr.lib

bin\msg\enssdk\examples\libasync.lib

bin\msg\enssdk\examples\asub.dsw

bin\msg\enssdk\examples\apub.dsp

bin\msg\enssdk\examples\asub.dsp

To build on Windows platforms:

1. A sample VC++ workspace is provided in `asub.dsw`. It has two projects in it: `asub.dsp` and `apub.dsp`.

   The required `.lib` files to link is in the same directory as `asub.c` and `apub.c`.

2. To run, it requires that the following DLLs are in your path.

   ```
   libens.dll
   libgap.dll
   libxenp.dll
   libcyrus.dll
   libchartable.dll
   libyasr.dll
   libasync.dll
   ```

   The simplest way to accomplish this is to include `msg_server_base` in`\msg\lib` in your `PATH`.

Runtime Library Path Variable

Calendar Server

In order for your custom programs to find the necessary runtime libraries, which are located in the `/opt/SUNWics5/cal/bin` directory, make sure your environment's runtime library path variable includes this directory. The name of the variable is platform dependent:

- SunOS and Linux: `LD_LIBRARY_PATH`
- Windows: `PATH`
- HPUX: `SHLIB_PATH`

Messaging Server

For Messaging Server, you need to set your `LD_LIBRARY_PATH` to `msg_server_base/bin/msg/lib`.

Chapter 2

Event Notification Service C API Reference

This chapter details the ENS C API; it is divided into three main sections:

- Publisher API
- Subscriber API
- Publish and Subscribe Dispatcher API

Publisher API Functions List

This chapter includes a description of the following Publisher functions, listed in Table 2-1:

Table 2-1 ENS Publisher API Functions List

Function	Description
publisher_t	Definition for a publisher.
publisher_cb_t	Generic callback function acknowledging an asynchronous call.
publisher_new_a	Creates a new asynchronous publisher.
publisher_new_s	Creates a new synchronous publisher.
publish_a	Sends an asynchronous notification to the notification service.
publish_s	Sends a synchronous notification to the notification service.
publisher_delete	Terminates a publish session.
publisher_get_subscriber	Creates a subscriber using the publisher's credentials.
renl_create_publisher	Creates an RENL, which enables the invocation of end2end_ack.

Table 2-1 ENS Publisher API Functions List *(Continued)*

renl_cancel_publisher	Cancels an RENL.

Subscriber API Functions List

This chapter includes a description of following Subscriber functions, listed in Table 2-2:

Table 2-2 ENS Subscriber API Functions List

Function	Description
subscriber_t	Definition of a subscriber.
subscription_t	Definition of a subscription.
subscriber_cb_t	Generic callback function acknowledging an asynchronous call.
subscriber_notify_cb_t	Synchronous callback; called upon receipt of a notification.
subscriber_new_a	Creates a new asynchronous subscriber.
subscriber_new_s	Creates a new synchronous subscriber.
subscribe_a	Establishes an asynchronous subscription.
unsubscribe_a	Cancels an asynchronous subscription.
subscriber_delete	Terminates a subscriber.
subscriber_get_publisher	Creates a publisher using the subscriber’s credentials.
renl_create_subscriber	Creates the subscription part of the RENL.
renl_cancel_subscriber	Cancels an RENL.

Publish and Subscribe Dispatcher Functions List

This chapter includes a description of the following Publish and Subscribe Dispatcher functions, listed in Table 2-3:

Table 2-3 ENS Publish and Subscribe Dispatcher Functions List

Function	Description
pas_dispatcher_t	Definition of a publish and subscribe dispatcher.
pas_dispatcher_new	Creates a dispatcher.

Table 2-3 ENS Publish and Subscribe Dispatcher Functions List *(Continued)*

pas_dispatcher_delete	Destroys a dispatcher created with pas_dispatcher_new.
pas_dispatch	Starts the dispatch loop of an event notification environment.
pas_shutdown	Stops the dispatch loop on an event notification environment started with pas_dispatch.

Publisher API

The Publisher API consists of one definition and nine functions:

- publisher_t
- publisher_cb_t
- publisher_new_a
- publisher_new_s
- publish_a
- publish_s
- publisher_delete
- publisher_get_subscriber
- renl_create_publisher
- renl_cancel_publisher

publisher_t

Purpose.

A publisher.

Syntax

```
typedef struct enc_struct publisher_t;
```

Parameters

None.

Returns

Nothing.

publisher_cb_t

Purpose.

Generic callback function invoked by ENS to acknowledge an asynchronous call.

Syntax

```
typedef void (*publisher_cb_t) (void *arg, int rc, void *data);
```

Parameters

arg	Context variable passed by the caller.
rc	The return code.
data	For an open, contains a newly created context.

Returns

Nothing.

publisher_new_a

Purpose

Creates a new asynchronous publisher.

Syntax

```
void publisher_new_a (pas_dispatcher_t *disp,
                      void *worker,
                      const char *host,
                      unsigned short port,
                      publisher_cb_t cbdone,
                      void *cbarg);
```

Parameters

disp	P&S thread pool context returned by pas_dispatcher_new.
worker	Application worker. If not NULL, grouped with existing workers created by ENS to service this publisher session. Used to prevent multiple threads from accessing the publisher data at the same time.
host	Notification server host name.
port	Notification server port.

cbdone	The callback invoked when the publisher has been successfully created, or could not be created. There are three Parameters to cbdone: • cbarg The first argument. • A status code. If non-zero, the publisher could not be created; value specifies cause of the failure. • The new active publisher.
cbarg	First argument of cbdone.

Returns

Nothing. It passes the new active publisher as third argument of `cbdone` callback.

publisher_new_s

Purpose

Creates a new synchronous publisher.

Syntax

```
publisher_t *publisher_new_s (pas_dispatcher_t *disp,
                              void *worker,
                              const char *host,
                              unsigned short port);
```

Parameters

disp	P&S thread pool context returned by `pas_dispatcher_new`.
worker	Application worker. If not NULL, grouped with existing workers created by ENS to service this publisher session. Used to prevent multiple threads from accessing the publisher data at the same time.
host	Notification server host name.
port	Notification server port.

Returns

A new active publisher (`publisher_t`).

publish_a

Purpose

Sends an asynchronous notification to the notification service.

Syntax

```
void publish_a (publisher_t *publisher,
                const char *event_ref,
                const char *data,
                unsigned int datalen,
                publisher_cb_t cbdone,
                publisher_cb_t end2end_ack,
                void *cbarg,
                unsigned long timeout);
```

Parameters

`publisher_t`	The active publisher.
`event_ref`	The event reference. This is a URI identifying the modified resource.
`data`	The event data. The body of the notification message. It is opaque to the notification service, which merely relays it to the events' subscriber.
`datalen`	The length in bytes of the data.
`cbdone`	The callback invoked when the data has been accepted or deemed unacceptable by the notification service. What makes a notification acceptable depends on the protocol used. The protocol may choose to use the transport acknowledgment (TCP) or use its own acknowledgment response mechanism.
`end2end_ack`	The callback function invoked after acknowledgment from the consumer peer (in an RENL) has been received. Used only in the context of an RENL.
`cbarg`	The first argument of `cbdone` or `end2end_ack` when invoked.
`timeout`	The length of time to wait for an RENL to complete.

Returns

Nothing.

publish_s

Purpose

Sends a synchronous notification to the notification service.

Syntax

```
int publish_s (publisher_t *publisher,
               const char *event_ref,
               const char *data,
               unsigned int datalen);
```

Parameters

publisher	The active publisher.
event_ref	The event reference. This is a URI identifying the modified resource.
data	The event data. The body of the notification message. It is opaque to the notification service, which relays it to the events' subscriber.
datalen	The length in bytes of the data.

Returns

Zero if successful; a failure code if unsuccessful. If an RENL, the call does not return until the consumer has completely processed the notification and has successfully acknowledged it.

publisher_delete

Purpose

Terminates a publish session.

Syntax

```
void publisher_delete (publisher_t *publisher);
```

Parameters

publisher	The publisher to delete.

Returns

Nothing.

publisher_get_subscriber

Purpose

Creates a subscriber using the credentials of the publisher.

Syntax

```
struct subscriber_struct * publisher_get_subscriber(publisher_t
*publisher);
```

Parameters

`publisher`	The publisher whose credentials are used to create the subscriber.

Returns

The subscriber, or `NULL` if the creation failed. If the creation failed, use the subscriber_new to create the subscriber.

renl_create_publisher

Purpose

Declares an RENL, which enables the `end2end_ack` invocation. After this call returns, the `end2end_ack` argument is invoked when an acknowledgment notification matching the specified publisher and subscriber is received.

Syntax

```
void renl_create_publisher (publisher_t *publisher,
                            const char *renl_id,
                            const char *subscriber,
                            publisher_cb_t cbdone,
                            void *cbarg);
```

Parameters

`publisher`	The active publisher.
`renl_id`	The unique RENL identifier. This allows two peers to be able to set up multiple RENLs between them.
`subscriber`	The authenticated identity of the peer.
`cbdone`	The callback invoked when the RENL is established.
`cbarg`	The first argument of cbdone, when invoked.

Returns

Nothing.

renl_cancel_publisher

Purpose

This cancels an RENL. This does not prevent more notifications being sent, but should a client acknowledgment be received, the `end2end_ack` argument of publish will no longer be invoked. All RENLs are automatically destroyed when the publisher is deleted. Therefore, this function does not need to be called to free RENL-related memory before deleting a publisher.

Syntax

```
void renl_cancel_publisher (renl_t *renl);
```

Parameters

`renl`	The RENL to cancel.

Returns

Nothing.

Subscriber API

The Subscriber API includes two definitions and ten functions:

- subscriber_t
- subscription_t
- subscriber_cb_t
- subscriber_notify_cb_t
- subscriber_new_a
- subscriber_new_s
- subscribe_a
- unsubscribe_a
- subscriber_delete
- subscriber_get_publisher
- renl_create_subscriber
- renl_cancel_subscriber

subscriber_t

Purpose

A subscriber.

Syntax

```
typedef struct enc_struct subscriber_t;
```

Parameters

None.

Returns

Nothing.

subscription_t

Purpose

A subscription.

Syntax

```
typedef struct subscription_struct subscription_t;
```

Parameters

None.

Returns

Nothing.

subscriber_cb_t

Purpose

Generic callback function invoked by ENS to acknowledge an asynchronous call.

Syntax

```
typedef void (*subscriber_cb_t) (void *arg,
                                 int rc,
                                 void *data);
```

Parameters

arg	Context variable passed by the caller.

rc	The return code.
data	For an open, contains a newly created context.

Returns

Nothing

subscriber_notify_cb_t

Purpose

Subscriber callback; called upon receipt of a notification.

Syntax

```
typedef void (*subscriber_notify_cb_t) (void *arg,
                                        char *event,
                                        char *data,
                                        int datalen);
```

Parameters

arg	Context pointer passed to subscribe (notify_arg).
event	The event reference (URI). The notification event reference matches the subscription, but may contain additional information called event attributes, such as a uid.
data	The body of the notification. A MIME object.
datalen	Length of the data.

Returns

Zero if successful, non-zero otherwise.

subscriber_new_a

Purpose

Creates a new asynchronous subscriber.

Syntax

```
void subscriber_new_a (pas_dispatcher_t *disp,
                       void *worker,
                       const char *host,
                       unsigned short port,
                       subscriber_cb_t cbdone,
                       void *cbarg);
```

Parameters

disp	Thread dispatcher context returned by pas_dispatcher_new.
worker	Application worker. If not NULL, grouped with existing workers created by ENS to service this subscriber session. Used to prevent multiple threads from accessing the subscriber data at the same time. Only usable if the caller creates and dispatches the GDisp context.
host	Notification server host name or IP address.
port	Subscription service port number.
cbdone	The callback invoked when the subscriber session becomes active and subscriptions can be issued. There are three parameters to cbdone: • cbarg The first argument. • A status code. If non-zero, the subscriber could not be created; value specifies cause of the failure. • The new active subscriber (subscriber_t).
cbarg	First argument of cbdone.

Returns

Nothing. It passes the new active subscriber as third argument of cbdone callback.

subscriber_new_s

Purpose

Creates a new synchronous subscriber.

Syntax

```
subscriber_t *subscriber_new_s (pas_dispatcher_t *disp,
                                const char *host,
                                unsigned short port);
```

Parameters

disp	Publish and subscribe dispatcher returned by pas_dispatcher_new.
worker	Application worker. If not NULL, grouped with existing workers created by ENS to service this publisher session. Used to prevent multiple threads from accessing the publisher data at the same time. Only usable if the caller creates and dispatches the GDisp context.
host	Notification server host name or IP address.
port	Subscription service port number.

Returns

A new active subscriber (subscriber_t).

subscribe_a

Purpose

Establishes an asynchronous subscription.

Syntax

```
void subscribe_a (subscriber_t *subscriber,
                  const char *event_ref,
                  subscriber_notify_cb_t notify_cb,
                  void *notify_arg,
                  subscriber_cb_t cbdone,
                  void *cbarg):
```

Parameters

subscriber	The subscriber.
event_ref	The event reference. This is a URI identifying the event's source.
notify_cb	The callback invoked upon receipt of a notification matching this subscription.
notify_arg	The first argument of notify_arg. May be called at any time, by any thread, while the subscription is still active.
cbdone	Called when an unsubscribe completes. It has three Parameters: • cbarg (see below). • Status code. • A pointer to an opaque subscription object.
cbarg	The first argument of cbdone.

Returns

Nothing.

unsubscribe_a

Purpose

Cancels an asynchronous subscription.

Syntax

```
void unsubscribe_a (subscriber_t *subscriber,
                    subscription_t *subscription,
                    subscriber_cb_t cbdone,
                    void *cbarg);
```

Parameters

subscriber	The disappearing subscriber.
subscription	The subscription to cancel.
cbdone	Called when an unsubscribe completes. It has three parameters: • cbarg (see below). • Status code. • A pointer to an opaque subscription object.
cbarg	The first argument of cbdone.

Returns

Nothing.

subscriber_delete

Purpose

Terminates a subscriber.

Syntax

```
void subscriber_delete (subscriber_t *subscriber);
```

Parameters

subscriber	The subscriber to delete.

Returns.
Nothing

subscriber_get_publisher

Purpose
Creates a publisher, using the credentials of the subscriber.

Syntax
```
struct publisher_struct *subscriber_get_publisher (subscriber_t
*subscriber);
```

Parameters

subscriber	The subscriber whose credentials are used to create the publisher.

Returns
The publisher, or NULL if creation failed. In case the creation fails, use the publisher_new.

renl_create_subscriber

Purpose
Creates the subscription part of an RENL.

Syntax
```
renl_t *renl_create_subscriber (subscription_t *subscription,
                                const char *renl_id,
                                const char *publisher);
```

Parameters

subscription	The subscription.
renl_id	The unique RENL identifier. This allows two peers to be able to set up multiple RENLs between them.
publisher	The authenticated identity of the peer.

Returns
The opaque RENL object.

renl_cancel_subscriber

Purpose

This cancels an RENL. It does not cancel a subscription. It tells ENS not to acknowledge any more notifications received for this subscription. It destroys the RENL object, the application may no longer use this RENL. All RENLs are automatically destroyed when the subscription is canceled. Therefore, this function does not need to be called to free RENL-related memory before deleting a subscriber.

Syntax

```
void renl_cancel_subscriber (renl_t *renl);
```

Parameters

`renl`	The RENL to cancel.

Returns

Nothing.

Publish and Subscribe Dispatcher API

The Publish and Subscribe Dispatcher API includes one definition and four functions:

- `pas_dispatcher_t`
- `pas_dispatcher_new`
- `pas_dispatcher_delete`
- `pas_dispatch`
- `pas_shutdown`

NOTE	The only thread dispatcher supported is GDisp (libasync).

pas_dispatcher_t

Purpose

A publish and subscribe dispatcher.

Syntax

```
typedef struct pas_dispatcher_struct pas_dispatcher_t;
```

Parameters

None.

Returns

Nothing.

pas_dispatcher_new

Purpose

Creates or advertises a dispatcher.

Syntax

```
pas_dispatcher_t *pas_dispatcher_new (void *disp);
```

Parameters

dispcx	The dispatcher context. If NULL, to start dispatching notifications, the application must call pas_dispatch. If not NULL, the dispatcher is a libasync dispatcher.

Returns

The dispatcher to use when creating publishers or subscribers (pas_dispatcher_t).

pas_dispatcher_delete

Purpose

Destroys a dispatcher created with pas_dispatcher_new.

Syntax

```
void pas_dispatcher_delete (pas_dispatcher_t *disp);
```

Parameters

disp	The event notification client environment.

Returns
Nothing.

pas_dispatch

Purpose
Starts the dispatch loop of an event notification environment. It has no effect if the application uses its own thread pool.

Syntax
```
void pas_dispatch (pas_dispatcher_t *disp);
```

Parameters

disp	The new dispatcher.

Returns
Nothing.

pas_shutdown

Purpose
Stops the dispatch loop of an event notification environment started with `pas_dispatch`. It has no effect if an application-provided dispatcher was passed to `pas_dispatcher_new`.

Syntax
```
void pas_shutdown (pas_dispatcher_t *disp);
```

Parameters

disp	The dispatcher context to shutdown.

Returns
Nothing.

Chapter 3

Event Notification Service Java (JMS) API Reference

This chapter describes the implementation of the Java (JMS) API in ENS and the Java API itself.

This chapter contains these sections:

- Event Notification Service Java (JMS) API Implementation
- Java (JMS) API Overview
- Implementation Notes

Event Notification Service Java (JMS) API Implementation

The ENS Java API is included with Messaging Server and Calendar Server. The Java API conforms to the Java Message Service specification (JMS).

ENS acts as a provider to Java Message Service. Thus, it provides a Java API to ENS. The software consists of the base library plus a demo program.

Prerequisites to Use the Java API

To use the Java API, you need to load the ENS plug-in. For instructions on loading the ENS plug-in, see Appendix C in the *Messaging Server Administration Guide*. By default, ENS is already enabled.

In addition, you need to install the following software, which is not provided with either Messaging Server or Calendar Server:

- Java Development Kit (JDK) 1.2 or later
- Java Message Service 1.0.2a or later (tested with 1.0.2a)

You can download this software from:

`http://java.sun.com`.

Sample Java Programs

The Messaging Server sample programs, `JmsSample` and `JBiff`, are stored in the following directory:

```
msg_server_base/bin/msg/enssdk/java/com/iplanet/ens/samples
```

directory. `JmsSample` is a generic ENS sample program. `JBiff` is Messaging Server specific.

For `JBiff`, you will need the following additional items:

- Java Mail jar file (tested with JavaMail 1.2)
- Java Activation Framework (required by JavaMail, tested with JAF1.0.1)

You can download these items from:

`http://java.sun.com`.

Instructions for Sample Programs

This section contains instructions for compiling and running the two sample programs:

- JmsSample Program
- JBiff Sample Program

JmsSample Program

To compile the JmsSample program:

1. Set your `CLASSPATH` to include the following:

   ens.jar file - `ens.jar`

   (For Messaging Server, the `ens.jar` is located in the *msg_server_base*`/java/jars/` directory.)

   Java Message Service - `full-path/jms1.0.2/jms.jar`

2. Change to the *msg_server_base*`/bin/msg/enssdk/java` directory.
3. Run the following command:

```
javac com/iplanet/ens/samples/JmsSample.java
```

To run the JmsSample program:

1. Change to the *msg_server_base*`/bin/msg/enssdk/java` directory.
2. Run the following command:

```
java com.iplanet.ens.samples.JmsSample
```

3. You are prompted for three items:
   - ENS event reference (for example, for Messaging Server: `enp://127.0.0.1/store`)
   - ENS hostname
   - ENS port (typically 7997)
4. Publish events.

   For Messaging Server, the two ways to publish events are:

   - You can use the `apub` C sample program for ENS. See "Sample Code" on page 67 for more information.
   - If you have enabled ENS, configure iBiff to publish Messaging Server related events.

   For Calendar Server, events are published by the calendar server.

JBiff Sample Program

To compile the JBiff program:

1. Set your `CLASSPATH` to include the following:

   ens.jar file - `ens.jar`

   (For Messaging Server, the `ens.jar` is located in the `msg_server_base/java/jars/` directory.)

   Java Message Service - `full-path/jms1.0.2/jms.jar`

   JavaMail - `full-path/javamail-1.2/mail.jar`

   Java Activation Framework - `full-path/jaf-1.0.1/activation.jar`

2. Change to the `msg_server_base/bin/msg/enssdk/java` directory.

3. Run the following command:

   ```
   javac com/iplanet/ens/samples/JBiff.java
   ```

To run the JBiff sample program:

Prerequisite: To run the `JBiff` sample program, you need to load the ENS (iBiff) plug-in. See Appendix C in the *Messaging Server Administrator's Guide* for instructions.

NOTE The demo is currently hardcoded to use the ENS event reference `enp://127.0.0.1/store`. This is the default event reference used by the iBiff notification plug-in.

---

1. Change to the `msg_server_base/bin/msg/enssdk/java` directory.

2. Run the following:

   ```
   java com.iplanet.ens.samples.JBiff
   ```

3. The program prompts for your userid, hostname, and password.

   The code assumes that the ENS server and the IMAP server are running on *hostname*. The userid and password are the IMAP username and password to access the IMAP account.

The two test programs are ENS subscribers. You receive events from iBiff when email messages flow through Messaging Server. Alternately you can use the `apub` C sample program to generate events. See "Sample Code" on page 67 for more information.

Java (JMS) API Overview

The Java API for ENS uses a subset of the standard Java Messaging Service (JMS) API, with the addition of two new proprietary methods:

- `com.iplanet.ens.jms.EnsTopicConnFactory`
- `com.iplanet.ens.jms.EnsTopic`

JMS requires the creation of a `TopicConnectionFactory` and a `Topic`, which is provided by the two ENS proprietary classes.

For more information on the standard JMS classes and methods, see the JMS documentation at:

http://java.sun.com/products/jms/docs.html

New Proprietary Methods

The two proprietary method classes are `EnsTopicConnFactory` and `EnsTopic`.

com.iplanet.ens.jms.EnsTopicConnFactory

About the method

The method is a constructor that returns a `javax.jms.TopicConnectionFactory`. Instead of using a JNDI-style lookup to obtain the `TopicConnectionFactory` object, this method is provided.

Syntax

```
public EnsTopicConnFactory (String name,
                            String hostname,
                            int port,
                            OutputStream logStream)

                  throws java.io.IOException
```

Arguments

Arguments	Type	Explanation
`name`	String	The client ID for the javax.jms.Connection
`hostname`	String	The hostname for the ENS server.
`port`	int	The TCP port for the ENS server.
`logStream`	OutputStream	Where messages are logged (cannot be null).

com.iplanet.ens.jms.EnsTopic

About this method

The method is a constructor that returns a `javax.jms.Topic`. Instead of using a JNDI-style lookup to obtain the `javax.jms.Topic`, this method is provided.

Syntax

```
public EnsTopic (String eventRef)
```

Arguments

Arguments	Type	Explanation
`eventRef`	String	The ENS event reference.

Implementation Notes

This section describes items to be aware of when implementing the ENS Java API.

Shortcomings of the Current Implementation

The current implementation of the Java API does not supply an initial provider interface.

JMS Topic Connection Factory and ENS Destination are called out explicitly. These are `com.iplanet.ens.jms.EnsTopicConnFactory` and `com.iplanet.ens.jms.EnsTopic`. ENS does not use JNDI to get the `TopicConnectionFactory` and `Topic` objects.

Notification Delivery

The notification is delivered as a `javax.jms.TextMessage`. The parameter/values of the ENS event reference are provided as property names to the TextMessage. The payload is provided as the data of the TextMessage.

JMS Headers

- JMSDeliveryMode is always set to `NON_PERSISTENT` (that is, no storing of message for future delivery).
- JMSRedelivered is always set to false.
- JMSMessageID is set to an internal id. Specifically it is not set to the SMTP MessageID in the header of the email message for Messaging Server.
- The payload is always a `javax.jms.TextMessage`. It corresponds to the ENS payload.
- JMSDestination is set to the full event reference (that is, it includes the parameter/values specific to this notification).
- JMSCorrelationID - Set to an internal sequence number.
- JMSTimestamp - Set to the time the message was sent.
    - For Messaging Server and iBiff, this corresponds to the `timestamp` parameter.
    - This is unused in Calendar Server.
- JMSType - The type of notification.
    - For Messaging Server and iBiff, this corresponds to the `evtType` parameter.
    - This is unused in Calendar Server.
- Additional properties:
    - Each parameter/value in the even reference becomes a property in the header. All property values are of type String.
- Unused headers are: JMSExpiration, JMSpriority, JMSReplyTo.

Miscellaneous

- MessageSelectors are not implemented.
- JMS uses the concept of durable and non-durable subscribers. A durable subscriber is a feature where notifications are guaranteed to be sent to subscribers even when they are offline, or if something catastrophic occurs, such as the ENS server going down after receiving the notification from the publisher but before delivering it to the subscriber.
    - Non-durable subscribers are implemented.
    - You can also use durable subscribers, however, the full functionality of being a durable subscriber is not implemented.
    - This aspect of being a durable subscriber is implemented: the publisher is acknowledged only after the subscriber receives a message.
    - This aspect of being a durable subscriber is not implemented: the message is not persistent, and delivery is not made to offline subscribers (after they come back online). In particular, JMSRedelivered is always set to false.

Chapter 4

Messaging Server Specific Information

This chapter describes the Messaging Server specific items you need to use the ENS APIs.

This chapter contains these sections:

- Event Notification Types and Parameters
- Sample Code
- Implementation Notes

Event Notification Types and Parameters

For Messaging Server, there is only one event reference, which can be composed of several parameters. There are various types of event notifications. Table 4-1 lists the event types supported by Messaging Server and gives a description of each:

Table 4-1 Event Tyes

Event Types	Description
`DeleteMsg`	Messages marked as "Deleted" are removed from the mailbox. This is the equivalent to IMAP expunge.
`Login`	User logged in from IMAP, HTTP, or POP.
`Logout`	User logged out from IMAP, HTTP, or POP.
`NewMsg`	New message was received by the system into the user's mailbox. Can have a payload of message headers and body.

Table 4-1 Event Tyes

Event Types	**Description**
`OverQuota`	Operation failed because the user's mailbox exceeded one of the quotas (diskquota, msgquota). The MTA channel holds the message until the quota changes or the user's mail box count goes below the quota. If the message expires while it is being held by the MTA, it will be expunged.
`PurgeMsg`	Message expunged (as a result of an expired date) from the mailbox by the server process imexpire. This is a server side expunge, whereas DeleteMsg is a client side expunge. This is not a purge in the true sense of the word.
`ReadMsg`	Message in the mailbox was read (in the IMAP protocol, the message was marked Seen).
`TrashMsg`	Message was marked for deletion by IMAP or HTTP. The user may still see the message in the folder, depending on the mail client's configuration. The messages are to be removed from the folder when an expunge is performed.
`UnderQuota`	Quota went back to normal from OverQuota state.
`UpdateMsg`	Message was appended to the mailbox (other than by NewMsg). for example, the user copied an email message to the mailbox. Can have a payload of message headers and body.

The following applies to the above supported event types:

- For `NewMsg` and `UpdateMsg`, message pay load is turned off by default to prevent overloading ENS. For information on how to enable the payload, see "Payload" on page 66. No other event types support a payload.
- Event notifications can be generated for changes to the `INBOX` alone, or to the `INBOX` and all other folders. The following configuration variable allows for `INBOX` only (value = `0`), or for both the `INBOX` and all other folders (value = `1`):

  ```
  local.store.notifyplugin.noneInbox.enable
  ```

  The default setting is for `INBOX` only (value = `0`).

NOTE There is no mechanism to select folders; all folders are included when the variable is enabled (value = `1`).

- The `NewMsg` notification is issued only after the message is deposited in the user mailbox (as opposed to “after it was accepted by the server and queued in the message queue”).
- Every notification carries several pieces of information (called parameters) depending on the event type, for example, `NewMsg` indicates the IMAP `uid` of the new message. For details on the parameters each event type takes, see “Available Parameters for Each Event Type” on page 65.
- Events are not generated for POP3 client access.
- All event types can be suppressed by issuing `XNOTNOTIFY`. For example, an IMAP script used for housekeeping only (the users are not meant to be notified) might issue it to suppress all events.

Parameters

iBiff uses the following format for the ENS event reference:

`enp://127.0.0.1/store`*?param=value¶m1=value1¶m2=value2*

The event key `enp://127.0.0.1/store` has no significance other than its uniqueness as a string. For example, the hostname portion of the event key has no significance as a hostname. It is simply a string that is part of the URI. However, the event key is user configurable. The list of iBiff event reference parameters is listed in Table 4-2 and Table 4-3 that follow.

The second part of the event reference consists of parameter-value pairs. This part of the event reference is separated from the event key by a question mark (`?`). The parameter and value are separated by an equals sign (`=`). The parameter-value pairs are separated by an ampersand (`&`). Note that there can be empty values, for which the value simply does not exist.

Table 4-2 on page 63 describes the mandatory event reference parameters that need to be included in every notification.

Table 4-2 Mandatory Event Reference Parameters

Parameter	Data Type	Description
`evtType`	string	Specifies the event type.
`hostname`	string	The hostname of the machine that generated the event.

Table 4-2 Mandatory Event Reference Parameters

Parameter	Data Type	Description
mailboxName	string	Specifies the mailbox name in the message store. The mailboxName has the format uid@domain, where uid is the user's unique identifier, and domain is the domain the user belongs to. The @domain portion is added only when the user does not belong to the default domain (i.e. the user is in a hosted domain).
pid	integer	ID of the process that generated the event.
process	string	Specifies the name of the process that generated the event.
timestamp	64-bit integer	Specifies the number of milliseconds since the epoch (midnight GMT, January 1, 1970).

Table 4-3 describes optional event reference parameters, which might be seen in the event depending on the event type (see Table 4-4).

Table 4-3 Optional Event Reference Parameters

Parameter	Data Type	Description
client	IP address	The IP address of the client logging in or out.
diskQuota	signed 32-bit integer	Specifies the disk space quota in kilobytes. The value is set to -1 to indicate no quotas.
diskUsed	signed 32-bit integer	Specifies the amount of disk space used in kilobytes.
hdrLen	unsigned 32-bit integer	Specifies the size of the message header. Note that this might not be the size of the header in the payload, because it might have been truncated.
imapUid	unsigned 32-bit integer	Specifies the IMAP uid parameter.
lastUid	unsigned 32-bit integer	Specifies the last IMAP uid value that was used.
numDel	unsigned 32-bit integer	Specifies the number of messages marked as deleted in the mailbox.
numMsgs	unsigned 32-bit integer	Specifies the number of total messages in the mailbox.
numMsgsMax	signed 32-bit integer	Specifies the quota for the maximum number of messages. The value is set to -1 to indicate no quotas.

Table 4-3 Optional Event Reference Parameters *(Continued)*

Parameter	Data Type	Description
numSeen	unsigned 32-bit integer	Specifies the number of messages in the mailbox marked as seen (read).
`size`	unsigned 32-bit integer	Specifies the size of the message. Note that this may not be the size of payload, since the payload is typically a truncated version of the message.
`uidValidity`	unsigned 32-bit integer	Specifies the IMAP uid validity parameter.

NOTE Subscribers should allow for undocumented parameters when parsing the event reference. This allows for future compatibility when new parameters are added.

Table 4-4 shows the parameters that are available for each event type. For example, to see which parameters apply to a `TrashMsg` event, look in the column header for "ReadMsg, TrashMsg" and then note that these events can use `numDel`, `numMsgs`, `numSeen`, and `userValidity`.

Table 4-4 Available Parameters for Each Event Type

Parameter	NewMsg, UpdateMsg	ReadMsg, TrashMsg	DeleteMsg, PurgeMsg	Login, Logout	OverQuota, UnderQuota
`client`	No	No	No	Yes	No
`diskQuota`	No	No	No	No	Yes
`diskUsed`	No	No	No	No	Yes
`hdrLen`	Yes	No	No	No	No
`imapUid`	Yes	No	Yes	No	No
`lastUid`	No	No	Yes	No	No
`numDel`	No	Yes	No	No	No
`numMsgs`	Yes	Yes	Yes	No	Yes
`numMsgsMax`	No	No	No	No	Yes
`numSeen`	No	Yes	No	No	No
`size`	Yes	No	No	No	No

Table 4-4 Available Parameters for Each Event Type *(Continued)*

Parameter	NewMsg, UpdateMsg	ReadMsg, TrashMsg	DeleteMsg, PurgeMsg	Login, Logout	OverQuota, UnderQuota
`uidValidity`	Yes	Yes	Yes	No	No
`userid`	No	No	No	Yes	No

Payload

ENS allows a payload for two event types: `NewMsg`, and `UpdateMsg`; the other event types do not carry a payload. The payload portion of these two notifications can contain any of the following data:

- No header or body data (default setting)
- Message header data only
- Message body data only
- Both message header and body data

The amount and type of data sent as the payload of the ENS event is determined by the confiuration parameters found in Table 4-5.

Table 4-5 Payload Configuration Parameters

Configuration Paramter	Description
`local.store.notifyplugin.maxBodySize`	Specifies the maximum size (in bytes) of the body that will be transmitted with the notification. Default setting is zero (0).
`local.store.notifyplugin.maxHeaderSize`	Specifies the maximum size (in bytes) of the header that will be transmitted with the notification. Default setting is zero (0).

Note that both parameters are set to zero as the default so that no header or body data is sent with ENS notifications.

Examples

The following example shows a `NewMsg` event reference (it is actually a single line that is broken up to several lines for readability):

```
enp://127.0.0.1/store?evtType=NewMsg&timestamp=1047488403000&
hostname=eman&process=imta&pid=476&mailboxName=testuser&numMsgs=16
&uidValidity=1046993605&imapUid=62&size=877&hdrLen=814
```

In this example, for the DeleteMsg event. Messages marked as deleted by IMAP or HTTP were expunged. The user would not see the message in the folder any more.

```
enp://127.0.0.1/store?evtType=DeleteMsg&timestamp=1047488588000&
hostname=eman&process=imapd&pid=419&mailboxName=testuser&
numMsgs=6&uidValidity=1046993605&imapUid=61&lastUid=62
```

And a third example shows a ReadMsg event. Message was marked as Seen by IMAP or HTTP.

```
enp://127.0.0.1/store?evtType=ReadMsg&timestamp=1047488477000&
hostname=eman&process=imapd&pid=419&mailboxName=testuser&
uidValidity=1046993605&numSeen=11&numDel=9&numMsgs=16
```

Sample Code

The following two code samples illustrate how to use the ENS API. The sample code is provided with the product in the following directory:

msg_server_base/examples

How to Use the Sample Code

1. Before running the makefile, set your library search path to include the directory:

   msg_server_base/lib

2. Compile the code using the Makefile.sample.

3. Run `apub` and `asub` as follows in separate windows:

```
apub localhost 7997
```

```
asub localhost 7997
```

   Whatever is typed into the `apub` window should appear on the `asub` window. If you use the default settings, all `iBiff` notifications should appear in the `asub` window.

4. Remove the *msg_server_base*`/lib` path from your library search path.

NOTE If you do not remove this from the library search path, you will not be able to stop and start the directory server.

Sample Publisher

This sample code provides a simple interactive asynchronous publisher.

```
/*
 * Copyright 2000 by Sun Microsystems, Inc.
 * All rights reserved
 */
/*
 *
 *                            apub
 *                             --
 *         a simple interactive asynchronous publisher
 *                             --
 *
 * This simplistic program publishes events using the hard-coded
 * event reference
 *      enp://127.0.0.1/store
 * and the data entered at the prompt as notification payload.
 * Enter "." to end the program.
*
 * If you happen to run the corresponding subscriber, asub, on the
 * same notification server, you will notice the sent data printed
 * out in the asub window.
 * Syntax:
 *      $ apub <host> <port>
 * where
 *      <host> is the notification server hostname
 *      <port> is the notification server IP port number
 */
```

```
#include <stdlib.h>
#include <stdio.h>#include "pasdisp.h"
#include "publisher.h"

static pas_dispatcher_t *disp = NULL;
static publisher_t *_publisher = NULL;
static int _shutdown = 0;
static void _read_stdin();

static void _exit_usage()
{
   printf("\nUsage:\napub host port\n");
   exit(5);
}

static void _exit_error(const char *msg)
{
   printf("%s\n", msg);
   exit(1);
}

static void _call_shutdown()
{
   _shutdown = 1;
   pas_shutdown(disp);
}

static void _open_ack(void *arg, int rc, void *enc)
{
   _publisher = (publisher_t *)enc;
   (void *)arg;

if (!_publisher) {
      printf("Failed to create publisher with status %d\n", rc);
      _call_shutdown();
      return;
   }
   _read_stdin();
   return;
}

static void _publish_ack(void *arg, int rc, void *ignored)
{
   (void *)ignored;
   free(arg)
   if (rc != 0) {
      printf("Publish failed with status %d\n", rc);
      _call_shutdown();
      return;
   }
```

```
    _read_stdin();
    return;
}

static void _read_stdin()
{
    static char input[1024];
    printf("apub> ");
    fflush(stdout);
    while (!_shutdown) {
        if ( !fgets(input, sizeof(input), stdin) ) {
             continue;
         } else {
             char *message;
             unsigned int message_len;
             input[strlen(input) - 1] = 0; /* Strip off the \n */
            if (*input == '.' && input[1] == 0) {
                publisher_delete(_publisher);
                _call_shutdown();
                break;
             }
            message = strdup(input);
            message_len = strlen(message);
            publish(_publisher, "enp://127.0.0.1/store",
                    message, message_len,
                    _publish_ack, NULL, (void *)message, 0);
             return;
        }
    }
     return;
}

main(int argc, char **argv)
{
    unsigned short port = 7997;
    char host[256];
    if (argc < 2) _exit_usage();
    if (*(argv[1]) == '0') {
        strcpy(host, "127.0.0.1");
    }else {
        strcpy(host, argv[1]);
    }
    if (argc > 2) {
        port = (unsigned short)atoi(argv[2]);
    }
    disp = pas_dispatcher_new(NULL);
    if (disp == NULL) _exit_error("Can't create publisher");
    publisher_new_a(disp, NULL, host, port, _open_ack, disp);
```

```
    pas_dispatch(disp);
    _shutdown = 1;
    pas_dispatcher_delete(disp);
    exit(0);
}
```

Sample Subscriber

This sample code provides a simple subscriber.

```
/*
 * Copyright 1997 by Sun Microsystems, Inc.
 * All rights reserved
 *
 */

/*
 *
 *                              asub
 *                               --
 *                        a simple subscriber
 *                               --
 *
 * This simplistic program subscribes to events matching the
 * hard-coded event reference:
 *      enp://127.0.0.1/store
 * It subsequently received messages emitted by the apub processes
 * if any are being used, and prints the payload of each received
 * notification to stdout.
 *
 * Syntax
 *      $ asub <host> <port>
 * where
 *      <host> is the notification server hostname
 *      <port> is the notification server IP port number
 */

#include <stdlib.h>
#include <stdio.h>

#include "pasdisp.h"
#include "subscriber.h"

static pas_dispatcher_t *disp = NULL;
static subscriber_t *_subscriber = NULL;
static subscription_t *_subscription = NULL;
static renl_t *_renl = NULL;
```

```
static void _exit_usage()
{
   printf("\nUsage:\nasub host port\n");
   exit(5);
}

static void _exit_error(const char *msg)
{
   printf("%s\n", msg);
   exit(1);
}

static void _subscribe_ack(void *arg, int rc, void *subscription)
{
   (void)arg;
   if (!rc) {
      _subscription = subscription;
      printf("Subscription successful\n");
      subscriber_keepalive(_subscriber, 30000);
   }else {
      printf("Subscription failed - status %d\n", rc);
      pas_shutdown(disp);
   }
}

static void _unsubscribe_ack(void *arg, int rc, void *ignored)
{
(void *)ignored;
   (void *)arg;
   if (rc != 0) {
      printf("Unsubscribe failed - status %d\n", rc);
   }
   subscriber_delete(_subscriber);
   pas_shutdown(disp);
}

static int _handle_notify(void *arg, char *url, char *str, int len)
{
   (void *)arg;
   printf("[%s] %.*s\n", url, len, (str) ? str : "(null)");
   return 0;
}

static void _open_ack(void *arg, int rc, void *enc)
{
   _subscriber = (subscriber_t *)enc;
   (void *)arg;
   if (rc) {
```

```
        printf("Failed to create subscriber with status %d\n", rc);
        pas_shutdown(disp);
        return;
    }

    subscribe(_subscriber, "enp://127.0.0.1/store",
                _handle_notify, NULL,
                _subscribe_ack, NULL);
    return;
}

static void _unsubscribe(int sig)
{
    (int)sig;
     unsubscribe(_subscriber, _subscription, _unsubscribe_ack, NULL);
}

main(int argc, char **argv)
{
    unsigned short port = 7997;
    char host[256];
    if (argc < 2) _exit_usage();
    if (*(argv[1]) == '0') {
        strcpy(host, "127.0.0.1");
     }else {
        strcpy(host, argv[1]);
    }
    if (argc > 2) {
        port = (unsigned short)atoi(argv[2]);
    }   disp = pas_dispatcher_new(NULL);
    if (disp == NULL) _exit_error("Can't create publisher");
    subscriber_new_a(disp, NULL, host, port, _open_ack, NULL);
    pas_dispatch(disp);
    pas_dispatcher_delete(disp);
exit(0);
}
```

Implementation Notes

The current implementation does not provide security on events that can be subscribed to. Thus, a user could register for all events, and portions of all other users' mail. Because of this it is strongly recommended that the ENS subscriber be on the "safe" side of the firewall at the very least.

Chapter 5

Calendar Server Specific Information

This chapter describes the Calendar Server specific items you need to use the ENS APIs.

This chapter contains these sections:

- Calendar Server Notifications
    - Alarm Notifications
    - Calendar Update Notifications
    - Advanced Topics
    - WCAP appid parameter and X-Tokens
- ENS Sample Code for Calendar Server

Calendar Server Notifications

There are two parts to the format of an Calendar Server notification:

- The event reference – A URL identifying the event.
- The payload – The data describing the event. Three different payload formats are supported: binary, text/calendar, and text/XML.

There are two types of calendar notifications:

- Alarm Notifications – relay reminders
- Calendar Update Notifications – distribute changes to the calendar database

Alarm Notifications

Alarm notifications relay reminders. They are published by the `csadmind` daemon whenever it wants to send a reminder. The default subscriber for these alarms in Communications Services is the `csnotifyd` daemon. Notifications consumed by `csnotifyd` have a binary payload and are acknowledged (reliable).

Additionally, the server can be configured to generate one additional notification for each reminder, which can be consumed by a third party notification infrastructure.

Table 5-1 shows the configuration variables that enable these notifications.

Table 5-1 Alarm Notifications

ics.conf	Default Value	Descripton
`caldb.serveralarams.binary.url`	`enp:///ics/alarm`	Used by csadmind and csnotifyd to send SMTP reminders.
`caldb.serveralarms.binary.enable`	`yes`	Enable or disable the default alarm (binary) transport provided by the Calendar Server product.
`caldb.serveralarms.url`	`NULL`	ENS topic URL for custom implementation. If this is NULL, then no formatted messages will be published. The `ics.conf` value will be set to `enp:///ics/alarm`.
`caldb.serveralarms.contenttype`	`text/xml`	Content MIME type of formatted message.
`caldb.berkeleydb.alarmretrytime`	`300`	Retry interval in seconds for failed deliveries. Specify zero (`0`) to disable retry.

Event URL parameters are the same for either one:

- `calid` - Calendar ID
- `uid` - Component, either `event` or `todo` (task) ID
- `rid` - Recurrence ID
- `aid` - Alarm ID
- `comptype` - An event or a todo (task)
- URI

Calendar Update Notifications

Calendar update notifications distribute changes to the calendar database. They are published by the `cshttpd` or `csdwpd` daemons whenever a change is made to the database (if the notification is enabled for this type of change).

There are eleven types of notifications. Table 5-2 lists each type of calendar update notification, it's `ics.conf` parameters, and their default values.

Table 5-2 Calendar Update Notifications

Types	`ics.conf` Parameters	Default Value
Attendee refesh actions	caldb.berkeleydb.ensmsg.refreshevent	no
	caldb.berkeleydb.ensmsg.refreshevent.url	enp:///ics/caleventrefresh
	caldb.berkeleydb.ensmsg.refreshevent.contenttype	text/xml
Attendee reply action	caldb.berkeleydb.ensmsg.replyevent	no
	caldb.berkeleydb.ensmsg.replyevent.url	enp:///ics/caleventreply
	caldb.berkeleydb.ensmsg.replyevent.contenttype	text/xml
Calendar creation	`caldb.berkeleydb.ensmsg.createcal`	yes
	caldb.berkeleydb.ensmsg.createcal.url	enp:///ics/calendarcreate
Calendar deletion	`caldb.berkeleydb.ensmsg.deletecal`	yes
	caldb.berkeleydb.ensmsg.deletecal.url	enp:///ics/calendardelete
Calendar modification	`caldb.berkeleydb.ensmsg.modifycal`	yes
	caldb.berkeleydb.ensmsg.modifycal.url	enp:///ics/calendarmodify
Event creation	`caldb.berkeleydb.ensmsg.createevent`	yes
	caldb.berkeleydb.ensmsg.createevent.url	enp:///ics/caleventcreate
Event modification	`caldb.berkeleydb.ensmsg.modifyevent`	yes
	caldb.berkeleydb.ensmsg.modifyevent.url	enp:///ics/caleventmodify
Event deletion	`caldb.berkeleydb.ensmsg.deleteevent`	yes
	caldb.berkeleydb.ensmsg.deleteevent.url	enp:///ics/caleventdelete
Todo (task) creation	`caldb.berkeleydb.ensmsg.createtodo`	yes
	caldb.berkeleydb.ensmsg.createtodo.url	enp:///ics/caltodocreate
Todo (task) modification	`caldb.berkeleydb.ensmsg.modifytodo`	yes
	caldb.berkeleydb.ensmsg.modifytodo.url	enp:///ics/caltodomodify

Table 5-2 Calendar Update Notifications *(Continued)*

Types	ics.conf **Parameters**	**Default Value**
Todo (task) deletion	caldb.berkeleydb.ensmsg.deletetodo	yes
	caldb.berkeleydb.ensmsg.deletetodo.url	enp:///ics/caltododelete

Event URL parameters include:

- calid - Calendar ID
- uid - Component, either event or todo (task) ID
- rid - Recurrence ID

Advanced Topics

Normally, ENS notifications for attendee replies and organizer refreshes are published to the caldb.berkeleydb.ensmsg.modifyevent topic along with other modifications. By setting the ics.conf parameter caldb.berkeleydb.ensmsg.advancedtopics to "yes" (the default is "no"), the ENS notifications can be published to separate modify, reply and refresh topics. This allows the consumer of the notification to understand more precisely what type of transaction triggered the notification.

Table 5-3 shows the topics ENS publishes notifications to depending on the setting of the ics.conf parmeter caldb.berkeleydb.ensmsg.advancedtopics.

Table 5-3 Advanced Topics Parameter

Value of Advanced Topics Parameter	Topics to Which ENS Publishes Attendee Notifications
yes	caldb.berkeleydb.ensmsg.modifyevent caldb.berkeleydb.ensmsg.refreshevent caldb.berkeleydb.ensmsg.replyevent
no	caldb.berkeleydb.ensmsg.modifyevent

WCAP appid parameter and X-Tokens

When ENS sends out notifications of modifications made to existing events, it returns two X-Tokens with the notification, X-NSCP-COMPONENT-SOURCE and X-NSCP-TRIGGERED-BY.

The contents of the `X-NSCP-COMPONENT-SOURCE X-Token` varies depending on who originated the event and the absence or presence of the `appid` parameter in the original WCAP command that requested the event.

If the `appid` parameter is present in the original WCAP command, ENS returns its value in the `X-NSCP-COMPONENT-SOURCE X-Token`. (Only certain commands take the `appid` parameter. See the *Calendar Server Programmer's Manual* for further information on the `appid` parameter.) Using this mechanism, applications can "tag" ENS notifications in order to detect which ones it originated. The value of the `appid` command is a character string of the application's choosing. If the `appid` parameter is missing, standard values are assigned to the X-Token depending on the origin, see Table 5-4 that follows for the standard values).

The X-Token, `X-NSCP-TRIGGERED-BY` holds the name (`uid`) of the organizer or attendee that triggered the notification regardless of the absence or presence of the `appid` parameter.

Table 5-4 shows the effect of the presence of the `appid` parameter in WCAP commands on the value of the X-Token `X-NSCP-COMPONENT-SOURCE`.

Table 5-4 Presence of appid and Value of X-Token X-NSCP-COMPONENT-SOURCE

appid Present?	Value of X-Token X-NSCP-COMPONENT-SOURCE (with Request Origin)
no	`WCAP` (default) `CALENDAR EXPRESS` (from UI) `ADMIN` (from Admin tools)
yes	Value of `appid`

ENS Sample Code for Calendar Server

Calendar Server ships with a complete ENS implementation. If you wish to customize it, you may use the ENS APIs to do so. The following four code samples, a simple publisher and subscriber pair, and a reliable publisher and subscriber pair, illustrate how to use the ENS API. The sample code is provided with the product in the following directory:

`/opt/SUNWics5/cal/csapi/samples/ens`

Sample Publisher and Subscriber

This sample code pair establishes a simple interactive asynchronous publisher and subscriber.

Publisher Code Sample

```
/*
 * Copyright 2000 by Sun Microsystems, Inc.
 * All rights reserved
 *
 * apub : simple interactive asynchronous publisher using
 *
 * Syntax:
 *   apub host port
 */

#include <stdlib.h>
#include <stdio.h>

#include "pasdisp.h"
#include "publisher.h"

static pas_dispatcher_t *disp = NULL;
static publisher_t *_publisher = NULL;
static int _shutdown = 0;

static void _read_stdin();

static void _exit_usage()
{
    printf("\nUsage:\napub host port\n");
    exit(5);
}

static void _exit_error(const char *msg)
{
    printf("%s\n", msg);
    exit(1);
}

static void _call_shutdown()
{
    _shutdown = 1;
    pas_shutdown(disp);
}

static void _open_ack(void *arg, int rc, void *enc)
{
    _publisher = (publisher_t *)enc;
    (void *)arg;
```

```
    if (!_publisher)
    {
        printf("Failed to create publisher with status %d\n", rc);
        _call_shutdown();
        return;
    }

    _read_stdin();

    return;
}

static void _publish_ack(void *arg, int rc, void *ignored)
{
    (void *)ignored;

    free(arg);

    if (rc != 0)
    {

        printf("Publish failed with status %d\n", rc);

        _call_shutdown();

        return;
    }

    _read_stdin();

    return;
}

static void _read_stdin()
{

    static char input[1024];

    printf("apub> ");

    fflush(stdout);

    while (!_shutdown)
    {

        if ( !fgets(input, sizeof(input), stdin) )
        {
            continue;

        } else {
            char *message;
            unsigned int message_len;

            input[strlen(input) - 1] = 0; /* Strip off the \n */
```

```
            if (*input == '.' && input[1] == 0)
            {
                publisher_delete(_publisher);
                _call_shutdown();
                break;
            }
            message = strdup(input);
            message_len = strlen(message);
            publish(_publisher, "enp://siroe.com/xyz",message,
                    message_len,
                    _publish_ack, NULL, (void *)message, 0);
            return;
        }
    }
    return;
}
main(int argc, char **argv)
{
    unsigned short port = 7997;
    char host[256];
    if (argc < 2) _exit_usage();
    if (*(argv[1]) == '0')
    {
        strcpy(host, "127.0.0.1");
    } else {
        strcpy(host, argv[1]);
    }
    if (argc > 2)
    {
        port = (unsigned short)atoi(argv[2]);
    }
    disp = pas_dispatcher_new(NULL);
    if (disp == NULL) _exit_error("Can't create publisher");
    publisher_new_a(disp, NULL, host, port, _open_ack, disp);
    pas_dispatch(disp);
    _shutdown = 1;
    pas_dispatcher_delete(disp);
    exit(0);
}
```

Subscriber Code Sample

```
/*
 * Copyright 2000 by Sun Microsystems, Inc.
 * All rights reserved
 *
 * asub : example asynchronous subscriber
 *
 * Syntax:
 *   asub host port
 */

#include <stdlib.h>
#include <stdio.h>

#include "pasdisp.h"
#include "subscriber.h"

static pas_dispatcher_t *disp = NULL;
static subscriber_t *_subscriber = NULL;
static subscription_t *_subscription = NULL;
static renl_t *_renl = NULL;

static void _exit_usage()
{
    printf("\nUsage:\nasub host port\n");
    exit(5);
}

static void _exit_error(const char *msg)
{
    printf("%s\n", msg);
    exit(1);
}

static void _subscribe_ack(void *arg, int rc, void *subscription)
{
    (void)arg;

    if (!rc)
    {
        _subscription = subscription;
        printf("Subscription successful\n");
    } else {
        printf("Subscription failed - status %d\n", rc);
        pas_shutdown(disp);
    }
}
```

```
static void _unsubscribe_ack(void *arg, int rc, void *ignored)
{
    (void *)ignored;
    (void *)arg;

    if (rc != 0)
    {
        printf("Unsubscribe failed - status %d\n", rc);
    }

    subscriber_delete(_subscriber);
    pas_shutdown(disp);
}

static int _handle_notify(void *arg, char *url, char *str, int len)
{
    (void *)arg;
    printf("[%s] %.*s\n", url, len, (str) ? str : "(null)");
    return 0;
}

static void _open_ack(void *arg, int rc, void *enc)
{
    _subscriber = (subscriber_t *)enc;

    (void *)arg;
    if (rc)
    {
        printf("Failed to create subscriber with status %d\n", rc);
        pas_shutdown(disp);
        return;
    }

    subscribe(_subscriber, "enp://siroe.com/xyz",
              _handle_notify, NULL,
              _subscribe_ack, NULL);

    return;
}

static void _unsubscribe(int sig)
{
    (int)sig;
    unsubscribe(_subscriber, _subscription, _unsubscribe_ack, NULL);
}
```

```
main(int argc, char **argv)
{
    unsigned short port = 7997;
    char host[256];

    if (argc < 2) _exit_usage();
    if (*(argv[1]) == '0')
    {
        strcpy(host, "127.0.0.1");
    } else {
        strcpy(host, argv[1]);
    }
    if (argc > 2)
    {
        port = (unsigned short)atoi(argv[2]);
    }

    disp = pas_dispatcher_new(NULL);
    if (disp == NULL) _exit_error("Can't create publisher");

    subscriber_new_a(disp, NULL, host, port, _open_ack, NULL);

    pas_dispatch(disp);

    pas_dispatcher_delete(disp);

    exit(0);
}
```

Reliable Publisher and Subscriber

This sample code pair establishes a reliable asynchronous publisher and subscriber.

Reliable Publisher Sample

```
/*
 * Copyright 2000 by Sun Microsystems, Inc.
 * All rights reserved
 *
 * rpub : simple *reliable* interactive asynchronous publisher.
 *  It is designed to be used in combination with rsub,
 *  the reliable subscriber.
 *
 * Syntax:
 *   rpub host port
 */
```

```
#include <stdlib.h>
#include <stdio.h>

#include "pasdisp.h"
#include "publisher.h"

static pas_dispatcher_t *disp = NULL;
static publisher_t *_publisher = NULL;
static int _shutdown = 0;
static renl_t *_renl;

static void _read_stdin();

static void _exit_usage()
{

    printf("\nUsage:\nrpub host port\n");

    exit(5);
}

static void _exit_error(const char *msg)
{
    printf("%s\n", msg);
    exit(1);
}

static void _call_shutdown()
{
    _shutdown = 1;
    pas_shutdown(disp);
}

static void _renl_create_cb(void *arg, int rc, void *ignored)
{
    (void *)arg;
    (void *)ignored;

    if (!_publisher)
    {
        printf("Failed to create RENL - status %d\n", rc);
        _call_shutdown();
        return;
    }

    _read_stdin();

    return;
}
```

```
static void _publisher_new_cb(void *arg, int rc, void *enc)
{
    _publisher = (publisher_t *)enc;

    (void *)arg;

    if (!_publisher)
    {
        printf("Failed to create publisher - status %d\n", rc);
        _call_shutdown();
        return;
    }

    renl_create_publisher(_publisher, "renl_id", NULL,
                          _renl_create_cb,NULL);

    return;
}

static void _recv_ack(void *arg, int rc, void *ignored)
{
    (void *)ignored;

    if (rc < 0)
    {
        printf("Acknowledgment Timeout\n");
    } else if ( rc == 0) {
        printf("Acknowledgment Received\n");
    }
    fflush (stdout);

    _read_stdin();

    free(arg);

    return;
}

static void _read_stdin()
{
    static char input[1024];

    printf("rpub> ");
    fflush(stdout);
    while (!_shutdown)
    {
        if ( !fgets(input, sizeof(input), stdin) )
        {
```

```
            continue;
        } else {
            char *message;
            unsigned int message_len;

            input[strlen(input) - 1] = 0; /* Strip off the \n */

            if (*input == '.' && input[1] == 0)
            {
                publisher_delete(_publisher);
                _call_shutdown();
                break;
            }

            message = strdup(input);
            message_len = strlen(message);

            /* five seconds timeout */
            publish(_publisher, "enp://siroe.com/xyz",
                    message, message_len,
                    NULL, _recv_ack, message, 5000);

            return;
        }
    }
    return;
}

main(int argc, char **argv)
{
    unsigned short port = 7997;
    char host[256];

    if (argc < 2) _exit_usage();
    if (*(argv[1]) == '0')
    {
        strcpy(host, "127.0.0.1");
    } else {
        strcpy(host, argv[1]);
    }
    if (argc > 2)
    {
        port = (unsigned short)atoi(argv[2]);
    }

    disp = pas_dispatcher_new(NULL);
    if (disp == NULL) _exit_error("Can't create publisher");
```

```
    publisher_new_a(disp, NULL, host, port, _publisher_new_cb,
                    NULL);

    pas_dispatch(disp);

    _shutdown = 1;

    pas_dispatcher_delete(disp);

    exit(0);
}
```

Reliable Subscriber Sample

```
/*
 * Copyright 2000 by Sun Microsystems, Inc.
 * All rights reserved
 *
 * asub : example asynchronous subscriber
 *
 * Syntax:
 *   asub host port
 */

#include <stdlib.h>
#include <stdio.h>

#include "pasdisp.h"
#include "subscriber.h"

static pas_dispatcher_t *disp = NULL;
static subscriber_t *_subscriber = NULL;
static subscription_t *_subscription = NULL;
static renl_t *_renl = NULL;

static void _exit_usage()
{
    printf("\nUsage:\nasub host port\n");
    exit(5);
}

static void _exit_error(const char *msg)
{
    printf("%s\n", msg);
    exit(1);
}

static void _subscribe_ack(void *arg, int rc, void *subscription)
{
    (void)arg;
```

```
    if (!rc)
    {
        _subscription = subscription;
        printf("Subscription successful\n");
        _renl = renl_create_subscriber(_subscription, "renl_id", NULL);
    } else {
        printf("Subscription failed - status %d\n", rc)
        pas_shutdown(disp);
    }
}

static void _unsubscribe_ack(void *arg, int rc, void *ignored)
{
    (void *)ignored;
    (void *)arg;

    if (rc != 0)
    {
        printf("Unsubscribe failed - status %d\n", rc);
    }

    subscriber_delete(_subscriber);
    pas_shutdown(disp);
}

static int _handle_notify(void *arg, char *url, char *str, int len)
{
    (void *)arg;
    printf("[%s] %.*s\n", url, len, (str) ? str : "(null)");
    return 0;
}

static void _open_ack(void *arg, int rc, void *enc)
{
    _subscriber = (subscriber_t *)enc;

    (void *)arg;
    if (rc)
    {
        printf("Failed to create subscriber with status %d\n", rc);
        pas_shutdown(disp);
        return;
    }

    subscribe(_subscriber, "enp://siroe.com/xyz",_handle_notify,
                NULL,_subscribe_ack, NULL);

    return;
}
```

```
static void _unsubscribe(int sig)
{
    (int)sig;
    unsubscribe(_subscriber, _subscription, _unsubscribe_ack, NULL);
}

main(int argc, char **argv)
{
    unsigned short port = 7997;
    char host[256];

    if (argc < 2) _exit_usage();
    if (*(argv[1]) == '0')
    {
        strcpy(host, "127.0.0.1");
    } else {
        strcpy(host, argv[1]);
    }
    if (argc > 2)
    {
        port = (unsigned short)atoi(argv[2]);

    }

    disp = pas_dispatcher_new(NULL);

    if (disp == NULL) _exit_error("Can't create publisher");

    subscriber_new_a(disp, NULL, host, port, _open_ack, NULL);

    pas_dispatch(disp);

    pas_dispatcher_delete(disp);

    exit(0);

}
```

Debugging ENS

This appendix contains instructions for obtaining trace information that can be valuable for debugging problems with any program that uses the ENS API. This includes all servers that send notifications through `enpd`, `csadmind`, `csnotifyd`, the `iBiff` plug-in, `stored`, `imapd`. Trace information can be obtained by setting several environment variables.

This appendix is divided into the following topics:

- Environment Variables
- How to Enable Debug Tracing
- Sample Debugging Sessions

Environment Variables

Tracing can be done at both the GAP (generic request and reply protocol layer) and ENP (publish and subscribe protocol layer) levels. Also, service bus traces can be set. The default is for no logging or tracing.

The following environment variables can be set for GAP tracing:

- `GAP_DEBUG`
- `GAP_LOG_MODULES`
- `GAP_LOGFILE (Calendar Server only)`

The following environment variables can be set for ENP tracing:

- `XENP_TRACE`
- `ENS_DEBUG`

- ENS_LOG_MODULES
- ENS_LOGFILE (Calendar Server only)
- ENS_STATS

The following environment variable can be set for service bus tracing: SERVICEBUS_DEBUG.

GAP_DEBUG

The value is a positive integer which indicates the trace level. Each higher trace level includes the output from the levels below it. For example, if you set the trace level to 7, level 1-6 traces are also included. The default value for this variable is 4, but since GAP_LOG_MODULES defaults to zero (0), no logging is done.

While it is possible to set the variable to any integer value greater than 7 and less than 100, the effect will be the same as setting it to 7.

Table A-1lists the trace levels for the variable GAP_DEBUG:

Table A-1 Trace Level Values

Trace Level	Trace Level Name	Description
0	N/A	No output except emergency messages
1	NSLOG_ALERT	Alert messages
2	NSLOG_CRIT	Critical messages
3	NSLOG_ERR	Software error conditions
4	NSLOG_WARNING	Default; warning messages (user error conditions)
5	NSLOG_NOTICE	Normal but significant conditions
6	NSLOG_INFO	Informational messages
7	NSLOG_DEBUG	Debug messages
100	NSLOG_TRACE	Full trace

GAP_LOG_MODULES

Use this variable to obtain trace information specific to one or more functional modules in the GAP code. This variable is a bit map. That is, each bit set in the variable turns on tracing for a particular module.

More than one module can be specified at once. To specify multiple modules, add the individual values of the modules you want. For example, if you want to trace both the connection layer and the transaction modules, you set the value of this variable to 10; to get all modules, set the value to 15.

Table A-2 lists the values for the variable GAP_LOG_MODULES:

Table A-2 GAP_LOG_MODULES Values

Value	Value Name	Description
0	N/A	Default; no modules logged.
1	GAPLOG_CONNECTION	Connection layer – socket input output calls
2	GAPLOG_SESSION	Session layer – session setup and closing
4	GAPLOG_TRANSACTION	Transaction creation – continuation and termination
8	GAPLOG_DISPATCHER	Thread dispatcher code – GDisp tracing

GAP_LOGFILE

This variable is used for Calendar Server only. This variable tells the system where to output GAP tracing. To send the output to a log file, set the variable to a text file name. The default (variable set to zero) sends GAP tracing to standard out.

XENP_TRACE

Use this variable to generate encoded data traces. Any non-zero value activates the trace.

ENS_DEBUG

Use this variable to trace functional (unencoded) client or server request responses.

The value is a positive integer which indicates the trace level. Each higher trace level includes the output from the levels below it. For example, if you set the trace level to 4, level 1-3 traces are also included.

While it is also possible to set the variable to any integer between 7 and 100, the effect will be the same as setting it to 7. That is, anything less than 100 but greater than 6 is treated the same.

Table A-3 lists the trace level values for the ENS_DEBUG variable:

Table A-3 ENS_DEBUG Trace Level Values

Trace Level	Trace Level Name	Desciption
0	N/A	No output except emergency messages
1	NSLOG_ALERT	Alert messages
2	NSLOG_CRIT	Critical messages
3	NSLOG_ERR	Software error conditions
4	NSLOG_WARNING	Warning messages (user error conditions)
5	NSLOG_NOTICE	Normal but significant conditions
6	NSLOG_INFO	Informational messages
7	NSLOG_DEBUG	Debug messages
100	NSLOG_TRACE	Full trace

ENS_LOG_MODULES

Use this variable to obtain trace information specific to one or more functional modules in the ENS code. This variable is a bit map. That is, each bit set in the variable turns on tracing for a particular module.

More than one module can be specified at once. To specify multiple modules, add the individual values of the modules you want. For example, if you want to trace both the server and the RENL modules, you set the value of this variable to 10; to get all modules, set the value to 31.

Table A-4 lists the values for the variable ENS_LOG_MODULES:

Table A-4 ENS_LOG_MODULES Values

Values	Value Names	Description
0	N/A	Default; no modules logged.
1	ENSLOG_CLIENT_API	Client API generated transactions
2	ENSLOG_SERVER	Server generated transactions
4	ENSLOG_UPUB	Publisher transactions
8	ENSLOG_RENL	Reliable event notifications
16	ENSLOG_STORE	ENS message store transactions

ENS_LOGFILE

This variable is used for Calendar Server only. This variable tells the system where to output ENS tracing. To send the output to a log file, set the variable to a text file name. The default (variable set to zero) sends ENS tracing to standard out.

ENS_STATS

To have statistics printed periodically, set this variable to a non-zero value.

SERVICEBUS_DEBUG

Service Bus is a process monitoring system based on ENS, and is used in ENS. Any non-zero value causes service bus traces to be sent to standard out. There is no logfile variable for service bus. To send the traces to a log file, temporarily redefine standard out to a text file name. During this time, all standard out messages will appear in the text file you create.

How to Enable Debug Tracing

In order to start tracing, follow these steps:

1. If ENS is running, `stop enpd`.

   To start and stop `enpd`, you must be in the `bin` directory.

   For example:

   - For Calendar Server on Unix, `/opt/SUNWics/cal/bin`.
   - For Calendar Server on Windows, `C:\Program Files\Sun ONE Calendar Server\..\.cal\bin`.

NOTE You can enable debugging for specific services by stopping only that service, for example `stop csnotifyd`, instead of the entire ENS server.

---

2. Set all variables to the desired value.

   For Unix:

   ❍ Bourne shell

     variable_name=*value*; export *variable_name*

     For example:

     ```
     GAP_DEBUG=2; export GAP_DEBUG
     ```

   ❍ C shell

     setenv *variable_name value*

     For example:

     ```
     setenv GAP_DEBUG 2
     ```

   For Windows:

   set `variable_name=value`

   For example,

   ```
   set GAP_DEBUG=2
   ```

3. If you want the traces to print to a log file, set the appropriate logfile variables (for `END_LOGFILE`, or `GAP_LOGFILE`) or temporarily redefine standard out to a text file.

4. Restart ENS – `start enpd`

   If you only disabled one service rather than the whole ENS server, you start that service only, for example `start csnotifyd`.

Sample Debugging Sessions

The following are sample debugging sessions on the Messaging Server and Calendar Server.

Each example has three parts:

- Set Environment Variables
- Sample Trace Output
- Short Commentary

Example 1: For Messaging Server

Set Environment Variables

```
setenv LD_LIBRARY_PATH msg_svr_base/lib/
stop-ens
setenv SERVICEBUS_DEBUG 1
setenv ENS_DEBUG 1
setenv ENS_LOG_MODULES 1
setenv GAP_DEBUG 1
setenv GAP_LOG_MODULES 1
setenv XENP_TRACE 1
setenv ENS_STATS 1
msg_svr_base/bin/enpd
```

Sample Trace Output

```
1  | servbus 3451633705  [26321]: Starting Service Bus
2  | servbus 3451636227  [26321]: Service Bus subscriber created
successfully
3  | servbus 3451636286  [26321]: Service Bus Ready
4  |         XENP -> len=36  servbus:///monitor/ens|subs|00010000
5  |         XENP -> len=60  servbus:///service/ens&pid=26321
&state=running|ntfy|00000000
6  |         XENP <- len=36  servbus:///monitor/ens|subs|00010000
7  |         XENP <- len=4   PACK
8  |         XENP <- len=60  servbus:///service/ens&pid=26321
&state=running|ntfy|00000000
9  |secs: pub: pub/s: pub/s(i): ntfy: ntfy/s :ntfy/s(i):
10 |  5 :   1:    0  : 0 :   0 :      0 : 0 :
11 |10 :   1:    0  : 0 :   0 :      0 : 0 :
12 |         XENP <-
len=232enp://127.0.0.1/store?evtType=NewMs&mailboxName=ServiceAdmin&
timestamp=1027623669000&process=2637&hostname=ketu&numMsgs=14&size=621
&uidValidity=1025118712&imapUid=14&hdrLen=547&qUsed=16&qMax=-1&
qMsgUsed=15&qMsgMax=-1|ntfy|00000000
13 | 15 :   2:   0  :       0 :   0 :      0 :       0 :
14 | 20 :   2:   0  :       0 :   0 :      0 :       0 :
15 | 25 :   2:   0  :       0 :   0 :      0 :       0 :
16 | 30 :   2:   0  :       0 :   0 :      0 :       0 :
17 | 35 :   2:   0  :       0 :   0 :      0 :       0 :
18 | 40 :   2:   0  :       0 :   0 :      0 :       0 :
19 | 45 :   2:   0  :       0 :   0 :      0 :       0 :
20 | 51 :   2:   0  : 0 :   0 :      0 :       0 :
21 | 56 :   2:   0  :       0 :   0 :      0 :       0 :
```

```
22 | 61 :   2:   0  :       0 :   0 :      0 :       0 :
23 | 66 :   2:   0  :       0 :   0 :      0 :       0 :
24 | 71 :   2:   0  :       0 :   0 :      0 :       0 :
25 | 76 :   2:   0  :       0 :   0 :      0 :       0 :
26 |secs: pub: pub/s: pub/s(i): ntfy: ntfy/s :ntfy/s(i):
27 | 81 :   2:   0  : 0 :   0 :      0 :       0 :
28 | 86 :   2:   0  :       0 :   0 :      0 :       0 :
29 | 91 :   2:   0  :       0 :   0 :      0 :       0 :
30 | 96 :   2:   0  :       0 :   0 :      0 :       0 :
31 |101:   2:   0  :       0 :   0 :      0 :       0 :
32 |106:   2:   0  :       0 :   0 :      0 :       0 :
33 |111:   2:   0  :       0 :   0 :      0 :       0 :
34 |116:   2:   0  :       0 :   0 :      0 :       0 :
35 |121:   2:   0  :       0 :   0 :      0 :       0 :
36 |126:   2:   0  :       0 :   0 :      0 :       0 :
37 |131:   2:   0  :       0 :   0 :      0 :       0 :
38 |136:   2:   0  :       0 :   0 :      0 :       0 :
39 |141:   2:   0  :       0 :   0 :      0 :       0 :
40 |146:   2:   0  :       0 :   0 :      0 :       0 :
41 |151:   2:   0  :       0 :   0 :      0 :       0 :
42 |^C
43 | XENP -> len=60  servbus:///service/ens&pid=26321
&state=stopped|ntfy|00000000
44 |servbus 3466881202  [26321]: Service Bus going away
45 |servbus 3466881542  [26321]: Failed to create subscriber-     error-1
```

Short Commentary

The following comments apply to the lines of the preceding trace output:

Line Number	**Comment**
1 - 8	Printed upon startup
9 - 11 and 13 - 41	Periodic statistics print out
12	A message is sent
42	Control-c stopped operation. This was done to end the sample only. Not recommended for stopping processes normally.

Example 2: For Messaging Server

Set Environment Variables

```
1  | (293 root) setenv ENS_DEBUG 99
2  | (294 root) setenv ENS_LOG_MODULES 63
3  | (295 root) msg_svr_base/bin/enpd
```

Sample Trace Output

```
4  | ENS 3588422667 [26400]: LOGIN   2
5  | ENS 3588423361 [26400]: _enp_session_open_cb : new session id=2 created
6  | ENS 3588423380 [26400]: recorded new subscription : 0001;
servbus:///monitor/ens
7  | ENS 3588423395 [26400]: subscribe
(event=servbus:///monitor/ens, sid=2) = 0
8  | ENS 3588423403 [26400]:publish
(event=servbus:///service/ens&pid=26400&state=running, sid=2)
9  | ENS 3588423414 [26400]:publish
(event=servbus:///service/ens&pid=26400&state=running, sid=2) = 0
10  | ENS 3588423825 [26400]: _ens_recv_request_cb: sid=2
op=1 id=00010000
11  | ENS 3588423842 [26400]: simple|store_req
(servbus:///monitor/ens#2) =2,servbus:///monitor/ens
12  | ENS 3588423848 [26400]: simple|store_evt
(servbus:///monitor/ens#2) = 2,servbus:///monitor/ens
13  | ENS 3588423853 [26400]: SUBS 2 servbus:///monitor/ens
00010000
14  | ENS 3588424389 [26400]: _ens_recv_request_cb: sid=2
op=2 id=00000000
15  | ENS 3588424395 [26400]: NTFY 2 servbus:///service/ens
&pid=26400&state=running
16  | ENS 3588424409 [26400]:ens_notify
(event=servbus:///service/ens&pid=26400&state=running,
id=00000000,sid=2):no match
17  | ENS 3588503451 [26400]: LOGIN   3
18  | ENS 3588504099 [26400]: LOGIN   4
19  | ENS 3588504938 [26400]: LOGIN   5
20  | ENS 3588505284 [26400]: LOGIN   6
21
22  | ENS 3591631839 [26400]: LOGIN   7
23  | ENS 3591637445 [26400]: _ens_recv_request_cb: sid=7
op=2 id=00000000
24  | ENS 3591637452 [26400]: NTFY    7 enp://127.0.0.1/store?evtType=NewMsg
&mailboxName=ServiceAdmin&timestamp=1027625056000&process=2646
&hostname=ketu&numMsgs=19&size=621&uidValidity=1025118712
```

```
&imapUid=19&hdrLen=547&qUsed=19&qMax=-1&qMsgUsed=20&qMsgMax=-1
25  | ENS 3591637467 [26400]:ens_notify
(event=enp://127.0.0.1/store?evtType=NewMsg
&mailboxName=ServiceAdmin&timestamp=1027625056000&process=2646
&hostname=ketu&numMsgs=19&size=621&uidValidity=1025118712
&imapUid=19&hdrLen=547&qUsed=19&qMax=-1&qMsgUsed=20
&qMsgMax=-1, id=00000000, sid=7): no match
26  |
27  | ENS 3595049771 [26400]: session closing 7
28  | ^CENS 3596193757 [26400]:publish
(event=servbus:///service/ens&pid=26400&state=stopped, sid=2)
29  | ENS 3596193782 [26400]:publish
(event=servbus:///service/ens&pid=26400&state=stopped, sid=2) = 0
30  | ENS 3596193987 [26400]: pas_dispatcher_delete : clean up
starting
31  | ENS 3596194018 [26400]: _enp_session_closing_cb : closing
session id=2
32  | ENS 3596194024 [26400]: destroying subscription :0001;
servbus:///monitor/ens
33  | ENS 3596194041 [26400]: pas_dispatcher_delete : 0 client(s) have been
bumped
34  | ENS 3596194065 [26400]: session closing 2
35  | ENS 3596194075 [26400]: simple|remov_evt
(2, servbus:///monitor/ens)
36  | ENS 3596194107 [26400]: session closing 3
37  | ENS 3596194216 [26400]: session closing 4
38  | ENS 3596194281 [26400]: session closing 5
39  | ENS 3596195039 [26400]: session closing 6
```

Short Commentary

The following comments apply to the lines of the preceding trace output:

Line Number	**Comment**
1 - 20	Initialization
22-26	Sent email message
27	Printed asynchronously
28	Control-c stopped operation. This was done to end the sample only. Not recommended for stopping processes normally.
29-39	enpd exiting

Glossary

Refer to the *Java Enterprise System Glossary* (`http://docs.sun.com/doc/816-6873`) for a complete list of terms that are used in this documentation set.

Index

A

alarm transfer reliability 24
APIs
  ENS
    publish and subscribe dispatcher 50
    publisher 37
    subscriber 43

C

Communications Services
  documentation 13
configuration parameters
  general 63, 64
custom applications
  building and running 30

D

debugging ENS 97
debugging tips 93
documentation
  where to find Communications Services documentation 13
  where to find Messaging Server documentation 12

E

enabling ENS (iBiff) 16
enabling traces 97
ENS
  code samples
    publisher 79
  daemons
    csadmind 35
    csnotifyd 35
  debugging tips 93
  enabling traces 97
  environment variables
    ENS_DEBUG 95
    ENS_LOG_MODULES 96
    ENS_LOGFILE 97
    GAP_DEBUG 94
    GAP_LOG_MODULES 94
    GAP_LOGFILE 95
    SERVICEBUS_DEBUG 97
    XENP_TRACE 95
  publish and subscribe dispatcher API 50
  publisher API 37
  RENL definition 37
  subscriber API 43
  subscriber_new_a function 45
ENS APIs
  functions list
    publish and subscribe dispatcher 50
    publisher 37
    subscriber 43
  publish and subscribe dispatcher functions
    pas_dispatch 52
    pas_dispatcher_delete 51

pas_dispatcher_new 51
pas_dispatcher_t definition 50
pas_shutdown 52
publisher functions
publish_a 40
publish_s 40
publisher_cb_t 38
publisher_delete 41
publisher_new_a 38
publisher_new_s 39
publisher_t 37
renl_cancel_publisher 43
renl_create_publisher 42
subscriber functions
renl_cancel_subscriber 50
renl_create_subscriber 49
subscribe_a 47
subscriber_cb_t 44
subscriber_delete 48
subscriber_new_a 45
subscriber_new_s 46
subscriber_notify_cb_t 45
subscriber_t 44
subscription_t 44
unsubscribe_a 48
ENS C API overview 28
ENS connection pooling 18
ENS Java API
overview 29
environment variables, for ENS tracing 93
Event Notification Service
API overview 28
architecture 19
enabling in Messaging Server 17
how Calendar Server interacts with 21
how Messaging Serer interacts with 26
in Calendar Server 16
in Messaging Server 16
overview 15
event references
Calendar Server example 18
Messaging Server example 18
overview 17

F

formatting conventions used in this document 10

I

iBiff notification plug-in 16, 18
include files
location of 30

L

libibiff 16

M

Messaging Server
and ENS 16
documentation 12
enabling ENS 17

N

notification
overview 20
reliable 20
unreliable 20

P

pas_dispatch function (ENS) 52
pas_dispatcher_delete function (ENS) 51
pas_dispatcher_new function (ENS) 51
pas_dispatcher_t definition (ENS) 50
pas_shutdown function (ENS) 52

publish and subscribe dispatcher functions (ENS)
  list 50
  pas_dispatch 52
  pas_dispatcher_delete 51
  pas_dispatcher_new 51
  pas_dispatcher_t definition t 50
  pas_shutdown 52
publish_a function (ENS) 40
publish_s function (ENS) 40
publisher_cb_t function (ENS) 38
publisher_delete function (ENS) 41
publisher_new_a function (ENS) 38
publisher_new_s function (ENS) 39
publisher_t function (ENS) 37

R

Reliable Event Notification Link (RENL) (ENS) 28, 37
renl_cancel_publisher function (ENS) 43
renl_cancel_subscriber function (ENS) 50
renl_create_publisher function (ENS) 42
renl_create_subscriber function (ENS) 49
runtime library path variable 34

S

sample code
  location of 30
shared libraries
  Calendar Server 31
  Messaging Server 31
subscribe_a function (ENS) 47
subscriber_cb_t function (ENS) 44
subscriber_delete function (ENS) 48
subscriber_new_a function (ENS) 45
subscriber_new_s function (ENS) 46
subscriber_t function (ENS) 44
subscription
  overview 20
subscription_t function (ENS) 44
Sun Java™ System Calendar Server
  alarm queue 22
  and ENS 16
  daemons 23
  ENS example 24

U

unsubscribe_a function (ENS) 48
unsubscription
  overview 20