Sun Java Communications Suite 5 Event Notification Service Guide

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

Part No: 819–4435–10
March 2007

070317@17039

Contents

Preface 13

1 Introduction to Event Notification Service 19
Event Notification Service Overview 19
ENS in Calendar Server 20
ENS in Messaging Server 20
Event References 21
ENS Connection Pooling 22
Event Notification Service Architecture 23
Notify 23
Subscribe 24
Unsubscribe 24
How Calendar Server Interacts with ENS 24
How Messaging Server Interacts with ENS 28
Event Notification Service API Overview 29
ENS C API Overview 30
ENS Java API Overview 31
Building and Running Custom Applications 31

2 Event Notification Service C API Reference 37
Publisher API Functions List 37
Unreliable Publisher API 38
Subscriber API Functions List 38
Publish and Subscribe Dispatcher Functions List 39
Publisher API 39
publisher_t 40
publisher_cb_t 40

publisher_new_a 40
publisher_new_s 41
publish_a 42
publish_s 43
publisher_delete 44
publisher_get_subscriber 44
renl_create_publisher 45
renl_cancel_publisher 46
Unreliable Publisher API 46
upub_t 46
upub_publish 47
upub_init 47
upub_shutdown 48
Subscriber API 49
subscriber_t 49
subscription_t 49
subscriber_cb_t 50
subscriber_notify_cb_t 50
subscriber_new_a 51
subscriber_new_s 52
subscribe_a 53
unsubscribe_a 54
subscriber_delete 54
subscriber_get_publisher 55
renl_create_subscriber 55
renl_cancel_subscriber 56
Publish and Subscribe Dispatcher API 57
pas_dispatcher_t 57
pas_dispatcher_new 57
pas_dispatcher_delete 58
pas_dispatch 58
pas_shutdown 59

3 Event Notification Service Java (JMS) API Reference 61
Event Notification Service Java (JMS) API Implementation 61

Prerequisites to Use the Java API 61
Sample Java Programs 62
Instructions for Sample Programs 62
Java (JMS) API Overview 64
New Proprietary Methods 64
com.iplanet.ens.jms.EnsTopicConnFactory 65
com.iplanet.ens.jms.EnsTopic 65
Implementation Notes 66
Shortcomings of the Current Implementation 66
Notification Delivery 66
JMS Headers 66
Miscellaneous 67

4 Messaging Server Specific Information 69
Event Notification Types and Parameters 69
Parameters 71
Payload 73
Examples 74
Sample Code 75
▼ To use the sample code 75
Sample Publisher 75
Sample Subscriber 77
Implementation Notes 80

5 Calendar Server Specific Information 81
Calendar Server Notifications 81
Alarm Notifications 81
Calendar Update Notifications 82
Advanced Topics 84
WCAP appid parameter and X-Tokens 85
ENS Sample Code for Calendar Server 86
Sample Publisher and Subscriber 86
Reliable Publisher and Subscriber 91

A Debugging ENS 97
Environment Variables 97
GAP_DEBUG 98
GAP_LOG_MODULES 98
GAP_LOGFILE 99
XENP_TRACE 99
ENS_DEBUG 99
ENS_LOG_MODULES 100
ENS_LOGFILE 100
ENS_STATS 101
SERVICEBUS_DEBUG 101
How to Enable Debug Tracing 101
▼ To start tracing 101
Sample Debugging Sessions 102
Example 1: For Messaging Server 102
Example 2: For Messaging Server 104

Index 107

Figures

FIGURE 1–1 ENS in Calendar Server Overview 25

FIGURE 1–2 Example Event Notification Service Publish and Subscribe Cycle for Calendar Server 27

FIGURE 1–3 ENS in Messaging Server Overview 29

Tables

TABLE 1–1 Sample ENS Publish and Subscribe Cycle 27
TABLE 2–1 ENS Publisher API Functions List 37
TABLE 2–2 ENS Unreliable Publisher API Functions List 38
TABLE 2–3 ENS Subscriber API Functions List 38
TABLE 2–4 ENS Publish and Subscribe Dispatcher Functions List 39
TABLE 4–1 Event Types 69
TABLE 4–2 Mandatory Event Reference Parameters 71
TABLE 4–3 Optional Event Reference Parameters 72
TABLE 4–4 Available Parameters for Each Event Type 72
TABLE 4–5 Payload Configuration Parameters 73
TABLE 5–1 Alarm Notifications 82
TABLE 5–2 Calendar Update Notifications 83
TABLE 5–3 Advanced Topics Parameter 84
TABLE 5–4 Presence of appid and Value of X-Token X-NSCP-COMPONENT-SOURCE 85
TABLE A–1 Trace Level Values 98
TABLE A–2 GAP_LOG_MODULES Values 99
TABLE A–3 ENS_DEBUG Trace Level Values 99
TABLE A–4 ENS_LOG_MODULES Values 100

Examples

EXAMPLE 1–1 `Makefile.sample` File 33

Preface

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.

Who Should Use This Book

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

Before You Read This Book

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 Book Is Organized

This manual contains the following chapters and appendix:

TABLE P–1 How This Book Is Organized

Chapter	Description
Chapter 1	Describes the Event Notification Service (ENS) components, architecture, and Application Programming Interfaces (APIs).
Chapter 2	Describes the ENS C API.
Chapter 3	Describes the ENS Java API and provides sample code.

TABLE P–1 How This Book Is Organized *(Continued)*

Chapter	Description
Chapter 4	Describes the Messaging Server event references and provides sample Messaging Server code.
Chapter 5	Describes the Calendar Server event notifications and provides sample Calendar Server code

Related Books

The `http://docs.sun.com`SM web site enables you to access Sun technical documentation online. You can browse the archive or search for a specific book title or subject.

Messaging Server Documents

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

`http://docs.sun.com/coll/1312.1`

The following documents are available:

- *Sun Java System Messaging ServerAdministration Guide*
- *Sun Java System Messaging Server Administration Reference*
- *Sun Java System Messaging Server MTA Developer's Reference*
- *Sun Java System 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/1313.1`

The following documents are available:

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

Communications Suite Documents

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

`http://docs.sun.com/coll/1312.1`

or

`http://docs.sun.com/coll/1313.1`

The following documents are available:

- *Sun Java Communications Suite Release Notes*
- *Sun Java Communications Suite Installation Guide*
- *Sun Java Communications Suite Upgrade Guide*
- *Sun Java Communications Suite Delegated Administrator Guide*
- *Sun Java Communications Suite Deployment Planning Guide*
- *Sun Java Communications Suite Schema Migration Guide*
- *Sun Java Communications Suite Schema Reference*
- *Sun Java Communications Suite 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 Suite Event Notification Service Guide* online in PDF and HTML formats. This book can be found at the following location:

Sun Java Communications Suite 5 Event Notification Service Guide

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 or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Documentation, Support, and Training

Sun Function	URL	Description
Documentation	http://www.sun.com/documentation/	Download PDF and HTML documents, and order printed documents
Support and Training	http://www.sun.com/supportraining/	Obtain technical support, download patches, and learn about Sun courses

Accessing Sun Resources Online

For product downloads, professional services, patches and support, and additional developer information, go to the following:

- Download Center http://wwws.sun.com/software/download/
- Professional Services http://www.sun.com/service/sunps/sunone/index.html
- Sun Enterprise Services, Solaris Patches, and Support http://sunsolve.sun.com/
- Developer Information http://developers.sun.com/prodtech/index.html

Contacting Sun Technical Support

If you have technical questions about this product that are not answered in the product documentation, go to http://www.sun.com/service/contacting.

Typographic Conventions

The following table describes the typographic changes that are used in this book.

TABLE P–2 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`** `Password:`
aabbcc123	Placeholder: replace with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. Perform a *patch analysis*. Do *not* save the file. [Note that some emphasized items appear bold online.]

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–3 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	`#`

Default Paths and File Names

The following table describes the default paths and file names used in this book.

TABLE P–4 Default Paths and File Names

Term	Description
msg-svr-base	Represents the base installation directory for Messaging Server. The default value of the *msg-svr-base* installation is as follows: Solaris™ systems: `/opt/SUNWmsgsr` Linux systems: `/opt/sun/messaging`
cal-svr-base	Represents the base installation directory for Calendar Server. The default value of the *cal-svr-base* installation is as follows: Solaris™ systems: `/opt/SUNWics5` Linux systems: `/opt/sun/calendar`

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions.

To share your comments, go to http://docs.sun.com and click Send Comments. In the online form, provide the document title and part number. The part number is a seven-digit or nine-digit number that can be found on the title page of the book or at the top of the document. For example, the title of this book is *Sun Java Communications Suite 5 Event Notification Service Guide*, and the part number is 819-4435.

◆ ◆ ◆ 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" on page 19
- "Event Notification Service Architecture" on page 23
- "Event Notification Service API Overview" on page 29

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 B, "Administering Event Notification Service in Messaging Server," in *Sun Java System Messaging Server 6.3 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 29 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 86.) 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 B, "Administering Event Notification Service in Messaging Server," in *Sun Java System Messaging Server 6.3 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 for that information.

Messaging Server comes bundled with sample ENS C publisher and subscriber code. See "Sample Code" on page 75 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 39 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 49 for more information.

Unsubscribe

ENS receives a request to cancel an existing subscription. See "Subscriber API" on page 49 for more information.

How Calendar Server Interacts with ENS

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

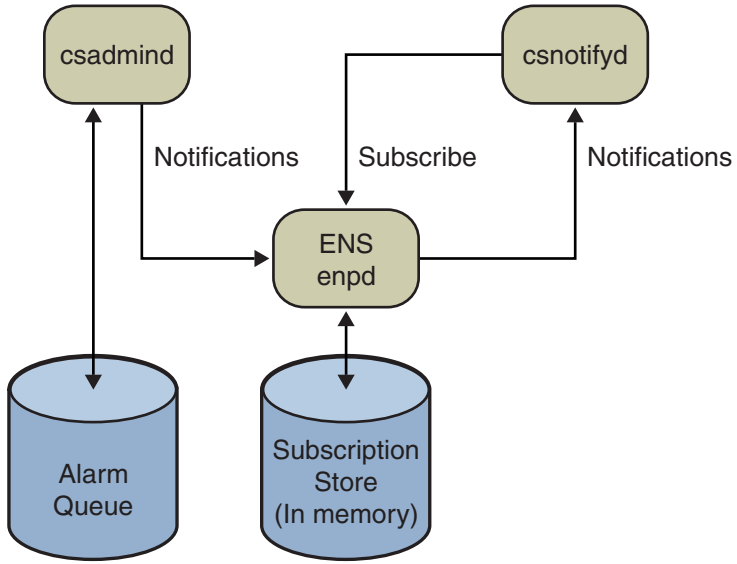

FIGURE 1–1 ENS in Calendar Server Overview

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 26.) 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 shut down.

Figure 1–2 illustrates this cycle and Table 1–1 provides the narrative for the figure.

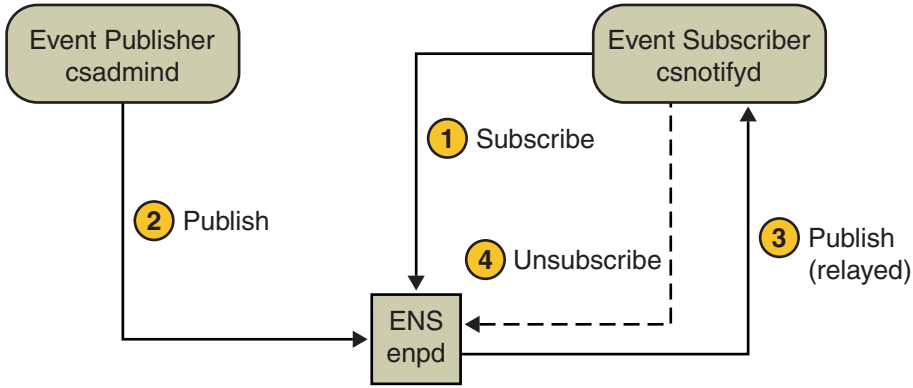

FIGURE 1–2 Example Event Notification Service Publish and Subscribe Cycle for Calendar Server

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 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.

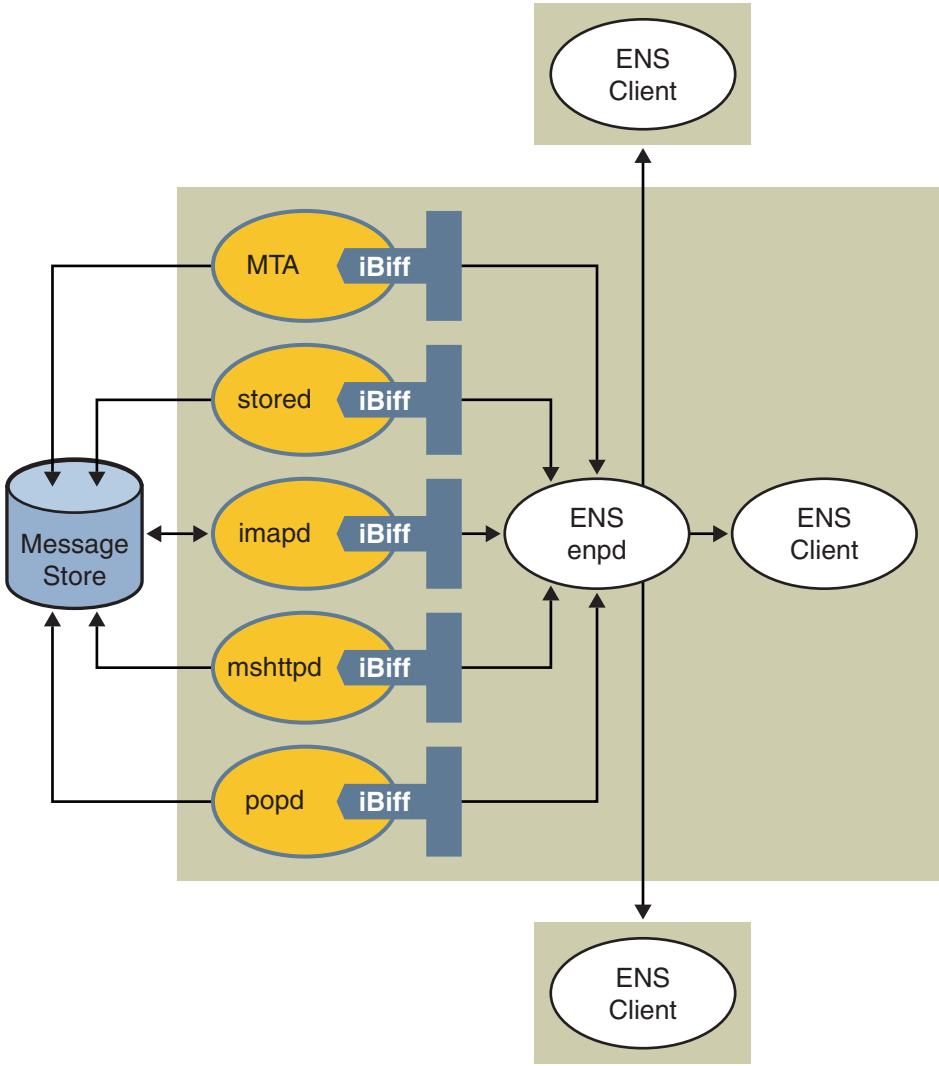

FIGURE 1–3 ENS in Messaging Server Overview

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. 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.

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.

- Unreliable Publisher API

  A publisher sends notification of an event to ENS, which distributes it to the subscribers. The term "unreliable" means that the API does not indicate whether a published event was successfully sent or received by a subscriber.

  This communication model is suitable for some applications, but not for others. If you are concerned about transfer reliability, use the full-blown publisher API (`publisher.h`) instead. However, the Unreliable Publisher API has the merit of being easier to use if transfer reliability is not a concern.

- 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/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`/lib`

Refer to `msg-server-base/examples/enssdk/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. The following listing shows a sample `makefile.sample` file.

EXAMPLE 1–1 `Makefile.sample` File

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

# LIBS
# Your library path should include <msg-server-base>/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/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" on page 39
- "Unreliable Publisher API" on page 46
- "Subscriber API" on page 49
- "Publish and Subscribe Dispatcher API" on page 57

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" on page 40	Definition for a publisher.
"publisher_cb_t" on page 40	Generic callback function acknowledging an asynchronous call.
"publisher_new_a" on page 40	Creates a new asynchronous publisher.
"publisher_new_s" on page 41	Creates a new synchronous publisher.
"publish_a" on page 42	Sends an asynchronous notification to the notification service.
"publish_s" on page 43	Sends a synchronous notification to the notification service.
"publisher_delete" on page 44	Terminates a publish session.
"publisher_get_subscriber" on page 44	Creates a subscriber using the publisher's credentials.

TABLE 2–1 ENS Publisher API Functions List *(Continued)*

"renl_create_publisher" on page	Creates an RENL, which enables the invocation of `end2end_ack`.
"renl_cancel_publisher" on page 46	Cancels an RENL.

Unreliable Publisher API

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

TABLE 2–2 ENS Unreliable Publisher API Functions List

Function	Description
"upub_t" on page 46	Definition for an unreliable publisher.
"upub_publish" on page 47	Sends a notification.
"upub_init" on page 47	Creates the publisher.
"upub_shutdown" on page 48	Shuts down and frees the publisher.

Subscriber API Functions List

This chapter includes a description of following Subscriber functions, listed in Table 2–3:

TABLE 2–3 ENS Subscriber API Functions List

Function	Description
"subscriber_t" on page 49	Definition of a subscriber.
"subscription_t" on page 49	Definition of a subscription.
"subscriber_cb_t" on page 50	Generic callback function acknowledging an asynchronous call.
"subscriber_notify_cb_t" on page 50	Synchronous callback; called upon receipt of a notification.
"subscriber_new_a" on page 51	Creates a new asynchronous subscriber.
"subscriber_new_s" on page 52	Creates a new synchronous subscriber.
"subscribe_a" on page 53	Establishes an asynchronous subscription.
"unsubscribe_a" on page 54	Cancels an asynchronous subscription.

TABLE 2–3 ENS Subscriber API Functions List *(Continued)*

"subscriber_delete" on page	Terminates a subscriber.
"subscriber_get_publisher" on page 55	Creates a publisher using the subscriber's credentials.
"renl_create_subscriber" on page 55	Creates the subscription part of the RENL.
"renl_cancel_subscriber" on page 56	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 2–4:

TABLE 2–4 ENS Publish and Subscribe Dispatcher Functions List

Function	Description
"pas_dispatcher_t" on page 57	Definition of a publish and subscribe dispatcher.
"pas_dispatcher_new" on page 57	Creates a dispatcher.
"pas_dispatcher_delete" on page 58	Destroys a dispatcher created with `pas_dispatcher_new`.
"pas_dispatch" on page 58	Starts the dispatch loop of an event notification environment.
"pas_shutdown" on page 59	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" on page 40
- "publisher_cb_t" on page 40
- "publisher_new_a" on page 40
- "publisher_new_s" on page 41
- "publish_a" on page 42
- "publish_s" on page 43
- "publisher_delete" on page 44
- "publisher_get_subscriber" on page 44
- "renl_create_publisher" on page 45
- "renl_cancel_publisher" on page 46

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.

Unreliable Publisher API

The Unreliable Publisher (`upub`) API consists of the following functions:

- "upub_t" on page 46
- "upub_publish" on page 47
- "upub_init" on page 47
- "upub_shutdown" on page 48

upub_t

Purpose

Defines an unreliable publisher type.

Syntax

```
typedef struct upub_struct upub_t;
```

Parameters

None.

Returns

Nothing.

upub_publish

Purpose

Sends a notification.

Syntax

```
void upub_publish (upub_t *upub,
                               const char *event_ref,
                               void *payload,
                               unsigned int payload_len,
                               void (*delete_payload) (void *));
```

Parameters

`upub`	Publisher.
`event_ref`	Event reference. This is a URI identifying the resource that is modifed.
`payload`	Notification payload.
`payload_len`	Length in bytes of the payload.
`delete_payload`	Callback to free the payload when it is no longer needed.

Returns

Nothing.

upub_init

Purpose

Creates the publisher.

This function allocates the `upub` object and sets up the thread dispatcher. The thread dispatcher can be provided by the application if the application already has one. This will have the advantage of not creating a thread dedicated to this API.

Note: If a connection-based transport is used, the connection to the notification service is not set up here. This is deferred to the first time a notification is actually published.

Syntax

```
upub_t *upub_init(void *gdc,
                            const char *host, unisgned short port,
                            int num_workers);
```

Parameters

`gdc`	GDisp context (thread dispatcher).
`host`	Notification server host name.
`port`	Notification server port.
`num_workers`	Number of parallel sessions.

Returns

The `upub` ready to be used.

upub_shutdown

Purpose

Shuts down and frees the publisher.

Syntax

```
void upub_shutdown(upub_t *upub);
```

Parameters

`upub` Publisher.

Returns

Nothing.

Subscriber API

The Subscriber API includes two definitions and ten functions:

- "subscriber_t" on page 49
- "subscription_t" on page 49
- "subscriber_cb_t" on page 50
- "subscriber_notify_cb_t" on page 50
- "subscriber_new_a" on page 51
- "subscriber_new_s" on page 52
- "subscribe_a" on page 53
- "unsubscribe_a" on page 54
- "subscriber_delete" on page 54
- "subscriber_get_publisher" on page 55
- "renl_create_subscriber" on page 55
- "renl_cancel_subscriber" on page 56

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" on page 57
- "pas_dispatcher_new" on page 57
- "pas_dispatcher_delete" on page 58
- "pas_dispatch" on page 58
- "pas_shutdown" on page 59

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" on page 61
- "Java (JMS) API Overview" on page 64
- "Implementation Notes" on page 66

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 B, "Administering Event Notification Service in Messaging Server," in *Sun Java System Messaging Server 6.3 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/examples/enssdk/java/com/iplanet/ens/samples
```

`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" on page 62
- "JBiff Sample Program" on page 63

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*`/examples/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*/examples/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 75
   - 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/examples/enssdk/java` directory.**
3. **Run the following command:**

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

▼ To run the JBiff sample program:

Before You Begin To run the `JBiff` sample program, you need to load the ENS (iBiff) plug-in.See Appendix B, "Administering Event Notification Service in Messaging Server," in *Sun Java System Messaging Server 6.3 Administration 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-svr-base/examples/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 75 for more information.

Java (JMS) API Overview

The Java API for ENS uses a subset of the standard Java Messaging Service (JMS) API (`http://java.sun.com/products/jms/docs.html`), 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" on page 69
- "Sample Code" on page 75
- "Implementation Notes" on page 80

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 Types

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.
`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.

TABLE 4–1 Event Types *(Continued)*

Event Types	Description
`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 73. 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 Table 4–4.
- 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&param1=value1&param2=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 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.
`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.
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

TABLE 4–4 Available Parameters for Each Event Type *(Continued)*

Parameter	NewMsg, UpdateMsg	ReadMsg, TrashMsg	DeleteMsg, PurgeMsg	Login, Logout	OverQuota, UnderQuota
`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
`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 configuration parameters found in Table 4–5.

TABLE 4–5 Payload Configuration Parameters

Configuration Parameter	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).

TABLE 4–5 Payload Configuration Parameters *(Continued)*

Configuration Parameter	Description
`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-svr-base/examples

▼ To use the sample code

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

```
msg-svr-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-svr-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 2006 by Sun Microsystems, Inc.
* All rights reserved
*
* Syntax:
*   apub host port
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "upub.h"
```

```
static upub_t *_publisher = NULL;

/* function prototypes */
static void _read_stdin(void);
static void _publish_ack(void *arg);

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

   while (1) {
      printf("apub> ");
      fflush(stdout);
      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) {
            break;
         }

         message = strdup(input);
         message_len = strlen(message);
         upub_publish(_publisher, "enp://yoyo.com/xyz",
                      message, message_len, _publish_ack);
      }
   }

   return;
}

/**
 * call back after publish is done
 **/
static void
_publish_ack(void *arg)
{
   free(arg);
   return;
}
```

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

   if (argc < 2) {
      printf("\nUsage:\n\tapub host port\n");
      exit(2);
   }
   if (*(argv[1]) == '0') {
      strcpy(host, "127.0.0.1");
   } else {
      strcpy(host, argv[1]);
   }
   if (argc > 2) {
      port = (unsigned short)atoi(argv[2]);
   }

   _publisher = upub_init(NULL, host, port, 1);
   if (_publisher == NULL) {
      printf("could not create publisher\n");
      exit(1);
   }

   _read_stdin();

   upub_shutdown(_publisher);
}
```

Sample Subscriber

This sample code provides a simple subscriber.

```
/*
 * Copyright 1997, 2006 by Sun Microsystems, Inc.
 * All rights reserved
 *
 * asub : example asynchronous subscriber
 *
 * Syntax:
 *      asub host port
*/
*
#include <stdlib.h>
```

```
#include <stdio.h>
#include <string.h>
#include "pasdisp.h"
#include "subscriber.h"

#define DEFAULT_EVENT_REF "enp://yoyo.com/xyz"

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

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);
}

int
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");
    s}else {
        strcpy(host, argv[1]);
    }
    if (argc > 2) {
        port = (unsigned short)atoi(argv[2]);
    }
```

```
    if (argc > 3) {
        _event_ref = argv[3];
    }

    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" on page 81
  - "Alarm Notifications" on page 81
  - "Calendar Update Notifications" on page 82
  - "Advanced Topics" on page 84
  - "WCAP appid parameter and X-Tokens" on page 85
- "ENS Sample Code for Calendar Server" on page 86

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" on page 81– relay reminders
- "Calendar Update Notifications" on page 82– 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 Suite 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 refresh actions	`caldb.berkeleydb.ensmsg.refreshevent`
	`caldb.berkeleydb.ensmsg.refreshevent.url`
	`caldb.berkeleydb.ensmsg.refreshevent.contenttype`
		`no`
		`enp:///ics/caleventrefresh`
		`text/xml`

Attendee reply action	`caldb.berkeleydb.ensmsg.replyevent`
	`caldb.berkeleydb.ensmsg.replyevent.url`
	`caldb.berkeleydb.ensmsg.replyevent.contenttype`
		`no`
		`enp:///ics/caleventreply`
		`text/xml`

Calendar creation	`caldb.berkeleydb.ensmsg.createcal` `caldb.berkeleydb.ensmsg.createcal.url`	`yes` `enp:///ics/calendarcreate`
Calendar deletion	`caldb.berkeleydb.ensmsg.deletecal` `caldb.berkeleydb.ensmsg.deletecal.url`	`yes` `enp:///ics/calendardelete`
Calendar modification	`caldb.berkeleydb.ensmsg.modifycal` `caldb.berkeleydb.ensmsg.modifycal.url`	`yes` `enp:///ics/calendarmodify`
Event creation	`caldb.berkeleydb.ensmsg.createevent` `caldb.berkeleydb.ensmsg.createevent.url`	`yes` `enp:///ics/caleventcreate`
Event modification	`caldb.berkeleydb.ensmsg.modifyevent` `caldb.berkeleydb.ensmsg.modifyevent.url`	`yes` `enp:///ics/caleventmodify`
Event deletion	`caldb.berkeleydb.ensmsg.deleteevent` `caldb.berkeleydb.ensmsg.deleteevent.url`	`yes` `enp:///ics/caleventdelete`

TABLE 5–2 Calendar Update Notifications *(Continued)*

Types	ics.conf Parameters	Default Value
Todo (task) creation	caldb.berkeleydb.ensmsg.createtodo caldb.berkeleydb.ensmsg.createtodo.url	yes enp:///ics/caltodocreate
Todo (task) modification	caldb.berkeleydb.ensmsg.modifytodo caldb.berkeleydb.ensmsg.modifytodo.url	yes enp:///ics/caltodomodify
Todo (task) deletion	caldb.berkeleydb.ensmsg.deletetodo caldb.berkeleydb.ensmsg.deletetodo.url	yes 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

TABLE 5–3 Advanced Topics Parameter *(Continued)*

Value of Advanced Topics Parameter	Topics to Which ENS Publishes Attendee Notifications
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);

}
```

◆ ◆ ◆ APPENDIX A

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" on page 97
- "How to Enable Debug Tracing" on page 101
- "Sample Debugging Sessions" on page 102

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" on page 98
- "GAP_LOG_MODULES" on page 98
- "GAP_LOGFILE" on page 99 (Calendar Server only)

The following environment variables can be set for ENP tracing:

- "XENP_TRACE" on page 99
- "ENS_DEBUG" on page 99
- "ENS_LOG_MODULES" on page 100
- "ENS_LOGFILE" on page 100 (Calendar Server only)
- "ENS_STATS" on page 101

The following environment variable can be set for service bus tracing:
"SERVICEBUS_DEBUG" on page 101.

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–1 lists 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

TABLE A–3 ENS_DEBUG Trace Level Values *(Continued)*

Trace Level	Trace Level Name	Desciption
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

The following procedure describes how to enable debug tracing.

▼ To start tracing

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

Index

A

alarm transfer reliability, 26
APIs
  ENS
    publish and subscribe dispatcher, 57-59
    publisher, 39-46
    subscriber, 49-56
    unreliable publisher, 46-48

C

Communications Suite, documentation, 15
configuration parameters, general, 72
custom applications, building and running, 31-35

D

debugging ENS, 101-102
debugging tips, 97-101
documentation
  overview, 14-15
  where to find Communications Suite documentation, 15
  where to find Messaging Server documentation, 14

E

enabling ENS (iBiff), 20
enabling traces, 101-102
ENS
  code samples
    publisher, 86-96
  daemons
    csadmind, 37
    csnotifyd, 37
  debugging tips, 97-101
  enabling traces, 101-102
  environment variables
    ENS_DEBUG, 99-100
    ENS_LOG_MODULES, 100
    ENS_LOGFILE, 100
    GAP_DEBUG, 98
    GAP_LOG_MODULES, 98-99
    GAP_LOGFILE, 99
    SERVICEBUS_DEBUG, 101
    XENP_TRACE, 99
  publish and subscribe dispatcher API, 57-59
  publisher API, 39-46
  RENL definition, 39-46
  subscriber API, 49-56
  subscriber_new_a function, 51-52
  unreliable publisher API, 46-48
ENS APIs
  functions list
    publish and subscribe dispatcher, 57
    publisher, 39
    subscriber, 49
    unreliable publisher, 46
  publish and subscribe dispatcher functions
    pas_dispatch, 58-59
    pas_dispatcher_delete, 58

ENS APIs, publish and subscribe dispatcher functions *(Continued)*
pas_dispatcher_new, 57-58
pas_dispatcher_t definition, 57
pas_shutdown, 59
publisher functions
publish_a, 42-43
publish_s, 43-44
publisher_cb_t, 40
publisher_delete, 44
publisher_new_a, 40-41
publisher_new_s, 41-42
publisher_t, 40
renl_cancel_publisher, 46
renl_create_publisher, 45
subscriber functions
renl_cancel_subscriber, 56
renl_create_subscriber, 55-56
subscribe_a, 53
subscriber_cb_t, 50
subscriber_delete, 54-55
subscriber_new_a, 51-52
subscriber_new_s, 52-53
subscriber_notify_cb_t, 50-51
subscriber_t, 49
subscription_t, 49-50
unsubscribe_a, 54
unreliable publisher functions
upub_init, 47-48
upub_publisher, 47
upub_shutdown, 48
upub_t, 46-47
ENS C API overview, 30
ENS connection pooling, 22
ENS Java API, overview, 31
environment variables, for ENS tracing, 97-101
Event Notification Service
API overview, 29-35
architecture, 23-28
enabling in Messaging Server, 20
how Calendar Server interacts with, 24-28
how Messaging Serer interacts with, 28
in Calendar Server, 20
in Messaging Server, 20-21
Event Notification Service *(Continued)*
overview, 19-22
event references
Calendar Server example, 21-22
Messaging Server example, 22
overview, 21-22

I

iBiff notification plug-in, 20, 21
include files, location of, 32

L

libibiff, 20
Linux, default base directory for, 18

M

Messaging Server
and ENS, 20-21
documentation, 14
enabling ENS, 20

N

notification
overview, 23
reliable, 24
unreliable, 23

P

pas_dispatch function (ENS), 58-59
pas_dispatcher_delete function (ENS), 58
pas_dispatcher_new function (ENS), 57-58
pas_dispatcher_t definition (ENS), 57
pas_shutdown function (ENS), 59
publish_a function (ENS), 42-43

publish and subscribe dispatcher functions (ENS)
list, 57
pas_dispatch, 58-59
pas_dispatcher_delete, 58
pas_dispatcher_new, 57-58
pas_dispatcher_t definition t, 57
pas_shutdown, 59
publish_s function (ENS), 43-44
publisher_cb_t function (ENS), 40
publisher_delete function (ENS), 44
publisher_new_a function (ENS), 40-41
publisher_new_s function (ENS), 41-42
publisher_t function (ENS), 40

R

Reliable Event Notification Link (RENL) (ENS), 30, 39-46
renl_cancel_publisher function (ENS), 46
renl_cancel_subscriber function (ENS), 56
renl_create_publisher function (ENS), 45
renl_create_subscriber function (ENS), 55-56
runtime library path variable, 35

S

sample code, location of, 31-32
shared libraries
Calendar Server, 32-35
Messaging Server, 33-35
Solaris
patches, 16
support, 16
subscribe_a function (ENS), 53
subscriber_cb_t function (ENS), 50
subscriber_delete function (ENS), 54-55
subscriber_new_a function (ENS), 50-51, 51-52
subscriber_new_s function (ENS), 52-53
subscriber_t function (ENS), 49
subscription, overview, 23
subscription_t function (ENS), 49-50
Sun Java™, 20, 25-26, 26, 27-28
support, Solaris, 16
System Calendar Server
alarm queue, 25-26
and ENS, 20
daemons, 26
ENS example, 27-28

U

unsubscribe_a function (ENS), 54
unsubscription, overview, 23
upub_init function (ENS), 47-48
upub_publish function (ENS), 47
upub_shutdown function (ENS), 48
upub_t function (ENS), 46-47