イベント通知サービスマニュアル

Sun™ ONE Messaging and Collaboration

Sun ONE Calendar Server 5.1.1;iPlanet™ Messaging Server 5.2

2002 年 8 月
816-7832-10

目次

本書について

このマニュアルでは、iPlanet™ Messaging Server および Sun™ ONE Calendar Server で利用されるイベント通知サービス (ENS) のアーキテクチャと API について説明します。サーバシステムのカスタマイズに使用できる ENS API についても、詳しく解説します。Sun ONE の以前の商標は iPlanet でした。Sun ONE および iPlanet はともに Sun Microsystems, Inc の商標です。移行中のため、このマニュアルでは両方とも使用されています。

ここでは、以下について説明します。

- 対象読者
- お読みになる前に
- このマニュアルの内容
- 表記上の規則
- 関連情報

対象読者

このマニュアルは、iPlanet Messaging Server と Sun ONE Calendar Server を実装するために、アプリケーションをカスタマイズする必要があるプログラマを対象にしています。

お読みになる前に

このマニュアルは、C/C++ および Java Messaging Service の知識や、次のことに関する一般的な知識をお持ちのプログラマの方を想定して書かれています。

- インターネットと WWW
- メッセージングとカレンダー処理の概念

このマニュアルの内容

このマニュアルは、次の章と付録で構成されています。

- 本書について (この章)
- 第 1 章「イベント通知サービスの概要」

  イベント通知サービス (ENS) のコンポーネント、アーキテクチャ、およびアプリケーションプログラミングインタフェース (API) について説明します。
- 第 2 章「イベント通知サービス C API リファレンス」

  ENS C API について説明します。
- 第 3 章「イベント通知サービス Java (JMS) API リファレンス」

  ENS Java API について説明し、コーディング例を掲載しています。
- 第 4 章「Sun ONE Calendar Server 固有の情報」

  Sun ONE Calendar Server のイベント通知について説明し、Sun ONE Calendar Server のコーディング例を掲載しています。
- 第 5 章「iPlanet Messaging Server 固有の情報」

  iPlanet Messaging Server のイベントリファレンスについて説明し、iPlanet Messaging Server のコーディング例を掲載しています。
- 用語集

表記上の規則

モノスペースフォント ー 画面上のコンピュータ出力のあらゆるテキストの表記に使用します。また、ファイル名、識別名、関数、コーディング例にも使用します。

イタリックフォント ー システムに固有の情報についてユーザが入力するテキスト (変数など) を表しています。サーバのパスや名前、アカウント ID などに使用します。

このマニュアルで使用されているパスはすべて UNIX 形式です。Windows NT ベースの iPlanet Messaging Server または Sun ONE Calendar Server をご使用の場合、このマニュアルに表記されている UNIX 形式のファイルパスを Windows NT の表現と読み替えてください。

関連情報

ほかにも、次の場所で関連マニュアルを参照できます。

- iPlanet Messaging Server マニュアル

  `http://docs.sun.com/db/prod/s1msgsrv?=ja#hic`

- Sun ONE Calendar Server マニュアル

  `http://docs.sun.com/db/prod/s1.s1calsrvl=ja#hic`

第 1 章

イベント通知サービスの概要

この章では、イベント通知サービス (ENS) のコンポーネント、アーキテクチャ、およびアプリケーションプログラミングインタフェース (API) の概要について説明します。

この章は、次の節で構成されています。

- イベント通知サービスの概要
- イベント通知サービスのアーキテクチャ
- イベント通知サービス API の概要

イベント通知サービスの概要

イベント通知サービス (ENS) は、Sun ONE の基礎となる公開および購読サービスで、次の Sun ONE 製品で利用できます。

- iPlanet Calendar Server 5.0、5.1、および Sun ONE Calendar Server 5.1.1
- iPlanet Messaging Server 5.1 以降 (統合版では使用不可になっている)

注 iPlanet Messaging Server で ENS を使用可能にする手順や管理する手順については、『iPlanet Messaging Server 5.2 管理者ガイド』の付録 C を参照してください。

ENS は、Sun ONE アプリケーションが使用するディスパッチャとして、配信対象の特定の種類のイベントを収集する中心的な役割を果たします。イベントとは、リソースの 1 つまたは複数のプロパティの値に対する変更のことです。ここでは、URI (Uniform Resource Identifier) がイベントを表しています。この種のイベントが発生した場合に通知されるアプリケーションはすべて ENS に登録します。ENS ではイベントを順番に識別し、通知を購読状況に照らし合わせます。イベントの例には次のものがあります。

- ユーザの受信箱への新規メールの着信
- ユーザのメールボックスの割り当てを超過
- カレンダーの通知

特に、ENS は分類できるイベントのレポートを受け付け、イベントの特定のカテゴリを配信対象として登録しているほかのアプリケーションに通知します。

イベント通知サービスは、パブリッシャおよびサブスクライバにサーバと API を提供します。パブリッシャは、通知サービスに対してイベントを使用可能にします。サブスクライバは、特定のイベントの通知の受信希望を通知サービスに伝えます。ENS API の詳細については、24 ページの 「イベント通知サービス API の概要」を参照してください。

Sun ONE Calendar Server での ENS

Sun ONE Calendar Server では、ENS がデフォルトで使用可能になっています。Sun ONE Calendar Server で ENS を使用するための設定は特に必要ありません。

Sun ONE Calendar Server が生成する、アラーム以外の通知を購読するユーザは、サブスクライバを作成する必要があります。

Sun ONE Calendar Server には、ENS C パブリッシャとサブスクライバのコーディング例が含まれています。コードについては、64 ページの 「Sun ONE Calendar Server コーディング例」を参照してください。

Sun ONE Calendar Server のコーディング例は、製品の次のディレクトリにあります。

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

iPlanet Messaging Server での ENS

ENS および iBiff (iPlanet Messaging Server の ENS パブリッシャ、iPlanet Messaging Server の通知プラグインとも呼ばれる) は、iPlanet Messaging Server に組み込まれていますが、デフォルトでは使用可能になっていません。

iPlanet Messaging Server で通知を購読するには、まず iPlanet Messaging Server ホスト上で次の 2 項目を実行する必要があります。

- iBiff 通知プラグインを読み込みます。
- Messaging Server を停止してから再起動します。

iPlanet Messaging Server で、ENS を使用可能にする手順については、『iPlanet Messaging Server 5.2 管理者ガイド』の付録 C を参照してください。

iPlanet Messaging Server 通知を購読したいユーザは、ENS API に対するサブスクライバを作成する必要があります。そのために、サブスクライバはさまざまな iPlanet Messaging Server 通知を理解する必要があります。詳細については、第 5 章、「iPlanet Messaging Server 固有の情報」を参照してください。

iPlanet Messaging Server には、ENS C パブリッシャとサブスクライバのコーディング例が組み込まれています。詳細については、83 ページの 「iPlanet Messaging Server のコーディング例 」を参照してください。

iPlanet Messaging Server のコーディング例は、 製品の次のディレクトリにあります。

server-root`/bin/msg/enssdk/examples`

イベント参照

イベント参照は、ENS が処理するイベントを識別します。イベント参照は、次の URI 構文 (RFC 2396 で指定) を使用します。

scheme`://`*authority resource*`/[?`*param1*`=`*value1*`&`*param2*`=`*value2*`&`*param3*`=`*value3*`]`

この構文の変数は次のとおりです。

- *scheme* は、`http`、`imap`、`ftp`、`wcap` などのアクセス方式

  Sun ONE Calendar Server および iPlanet Messaging Server では、ENS スキーマは、`enp` です。

- *authority* は、リソースへのアクセスを制御する DNS ドメインまたはホスト名

- *resource* は authority コンテキスト内のリソースへのパス。複数のパスコンポーネントで構成するには、スラッシュ (/) で区切る
- *param* はリソースの状態を表すパラメータの名前
- *value* はパラメータの値。パラメータと値の組み合わせは複数を指定しても、1つも指定しなくてもよい

通常、すべての Sun ONE Calendar Server イベントは次で始まります。

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

デフォルトでは、iPlanet Messaging Server 通知プラグイン iBiff は、次のスキーマとリソースを使用します。

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

注 イベント参照には URI 構文がありますが、scheme、authority、および resource には特別な意味はありません。ENS では単なる文字列としてしか解釈されません。

Sun ONE Calendar Server イベント参照の例

次に示すのは、`jac` というカレンダー ID によって、すべてのイベントアラームを購読するイベント参照 URI の例です。

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

注 エンドユーザが使用するためのものではありません。

iPlanet Messaging Server イベント参照の例

次に示すのは、ユーザ ID が `blim` というユーザのために、すべての `NewMsg` イベントの購読を要求するイベント参照の例です。

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

ENS を iPlanet Messaging Server とともに使用する場合は、指定するユーザ ID は大文字小文字が区別されます。

注 エンドユーザが使用するためのものではありません。

ENS 接続プール

ENS の接続プール機能を使用すると、サブスクライバのプールは単一のイベント参照から通知を受信できます。各イベントについて、ENS はプールからサブスクライバを 1 つ選択して、通知を送信します。つまり、プール内の 1 つのサブスクライバだけが通知を受け取ります。ENS サーバは、複数のサブスクライバ間で通知の送信のバランスをとります。この機能によって、単一のイベント参照からの全通知を協同で受信する、サブスクライバのプールをクライアントに設定することができます。

たとえば、イベント参照 `enp://127.0.0.1/store` に対して通知が公開されているとすると、サブスクライバは通常、このイベント参照を購読して通知を受信します。このイベント参照へのすべての通知をサブスクライバのプールで受信するには、プール内の各サブスクライバはこのイベント参照ではなく、イベント参照 `enp+pool://127.0.0.1/store` を購読するだけで十分です。ENS サーバは、プールからサブスクライバを 1 つ選択して通知を送信します。

注　パブリッシャは単純なイベント参照に通知を送信し続けます (この例では `enp://127.0.0.1/store`) 。つまり、パブリッシャはサブスクライバプールを認識していません。

複数プール拡張

接続プールは、複数のサブスクライバプールをサポートできます。つまり、サブスクライバプールを 2 つ設定し、各プールがイベント参照からのすべての通知を受信することができます。サブスクライバのイベント参照の構文は次のとおりです。

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

この *poolid* は base64 のアルファベットのみを使用する文字列です (base64 アルファベットについては、RFC1521 の Table 1 を参照してください)。たとえば、イベント参照 `enp://127.0.0.1/store` に対してサブスクライバプールが 2 つある例では、各プールが次のイベント参照を購読します。

`enp+pool.1://127.0.0.1/store` --> 最初のサブスクライバプール用
`enp+pool.2://127.0.0.1/store` --> 2 番目のサブスクライバプール用

イベント通知サービスのアーキテクチャ

Solaris プラットフォームでは、ENS はデーモン `enpd` として、さまざまな Calendar Server や Messaging Server 構成でほかのデーモンとともに実行され、リソースのプロパティに発生するイベントの収集およびディスパッチを行います。Windows NT プラットフォームでは、ENS は `enpd.exe` サービスとして実行されます。

ENS では、イベントとはリソースに発生した変更であり、リソースとはカレンダーや受信箱のようなエンティティです。たとえば、カレンダー (リソース) にエントリを追加するとイベントが発生し、ENS によって格納されます。そして、このイベントが購読されると、サブスクライバに通知が送信されます。

ENS アーキテクチャを使用すると、次の 3 つを実行できます。

- 通知 - イベントの発生を記述するメッセージ。イベントパブリッシャにより送信され、イベントへの参照のほか、URI に追加されるパラメータと値の組み合わせ、そしてイベントコンシューマが使用する通知サービスからは不透明な任意のデータ (ペイロード) を含む。イベントを配信対象にすればだれでも購読できる
- 購読 - イベントを購読するために送信されるメッセージ。イベント参照、クライアント側要求識別子、および URI に追加される任意のパラメータと値の組み合わせを含む。購読は次のイベントから適用される (サブスクライバは次のイベントから通知するよう要求)
- 購読の解除 - このメッセージは、既存の購読をキャンセル (購読を解除) する。イベントサブスクライバが ENS に、指定したイベントの通知の中継を停止するよう伝える

通知

ENS は通知を送信して、イベントのサブスクライバに通知します。通知することを「公開する」ともいいます。通知には次の項目を含めることができます。

- イベント参照 (オプションでパラメータと値の組み合わせを含める)
- オプションのアプリケーション固有のデータ (ENS には「不透明」だが、パブリッシャとサブスクライバはデータの形式についてあらかじめ合意している)

オプションのアプリケーション固有のデータは、「ペイロード」とも呼ばれます。

通知には、次の 2 種類があります。

- **低信頼通知** - イベントパブリッシャから通知サーバに送信される通知。コンシューマが存在するか、またはそれらが通知を受けるかについて、パブリッシャが認識していないか関心がない場合は、この要求に確実に肯定応答を返す必要はない。ただし、パブリッシャとサブスクライバとが相互に認識している場合は、パブリッシャとサブスクライバ間への高信頼イベント通知リンク (RENL) の設定に合意できる。この場合、サブスクライバがパブリッシャの通知を処理すると、肯定応答の通知がパブリッシャに返される
- **高信頼通知** - 購読の結果として、サーバからサブスクライバへ送信される通知。このタイプの通知に対しては、肯定応答を返す必要がある。高信頼通知には、低信頼通知と同じ属性が含まれる

詳細については、33 ページの 「パブリッシャ API 」を参照してください。

購読

ENS はイベントサブスクライバが送信するイベントの通知要求を受信します。この要求が購読です。購読は、セッションの存続期間中、またはそれがキャンセル (購読の解除) されるまで有効となります。

購読には、次の項目を含めることができます。

- イベント参照 (オプションでパラメータ / 値の組み合わせを含める)
- 要求識別子

詳細については、39 ページの 「サブスクライバ API 」を参照してください。

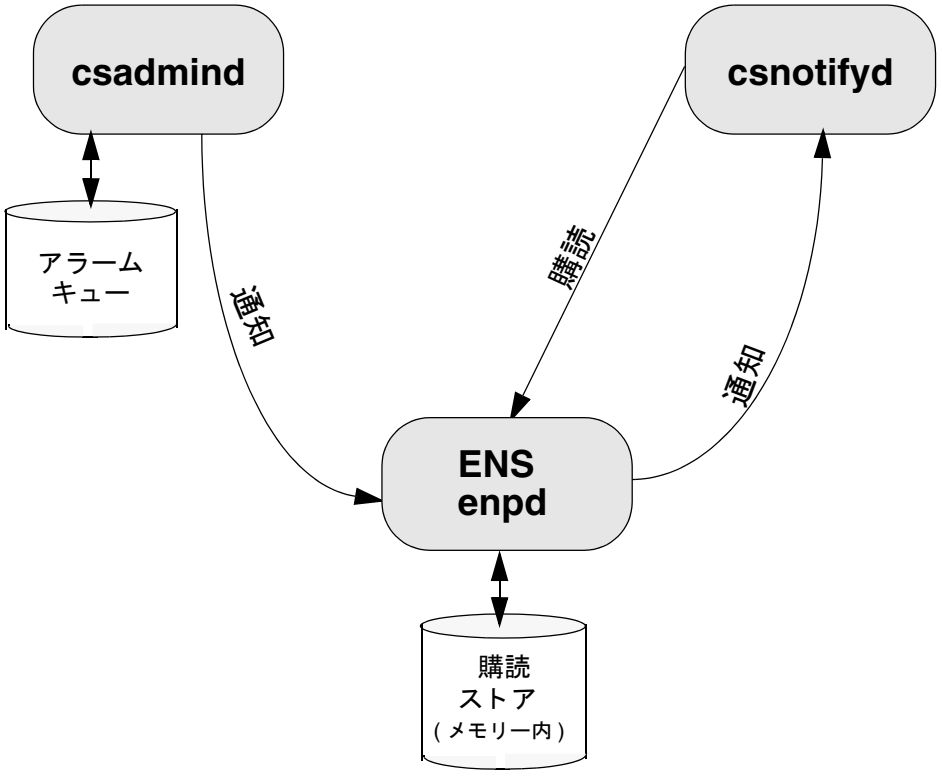
購読の解除

ENS は既存の購読をキャンセルする要求を受信します。詳細については、39 ページの 「サブスクライバ API 」を参照してください。

Sun ONE Calendar Server と ENS との対話

18 ページの図 1-1 で、ENS がアラームキューおよび `csadmind` と `csnotifyd` の 2 つのデーモンを介して Sun ONE Calendar Server と対話する方法について示します。

図 1-1 Sun ONE Calendar Server 内の ENS の概要



Sun ONE Calendar Server アラームキュー

ENS はアラームディスパッチャであり、アラームの生成と配信を分離します。これにより、電子メールや無線通信など複数の配信方式が可能となります。csadmind デーモンは、アラームキューの状態の変化を感知して、イベントを検出します。アラームキューの状態は、アラームがキューに格納されるたびに変化します。アラームは、カレンダーイベントがアラームを生成したときに、キューに格納されます。次の URI はこれらのイベントの種類を表しています。

イベントの場合

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

todo (仕事) の場合

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

この構文の各変数は次のとおりです。

- *calid* はカレンダー ID
- *uid* はカレンダー内のイベント / todo (仕事) ID
- *rid* は再帰イベント / todo (仕事) の再帰 ID
- *aid* はイベント / todo (仕事) 内のアラーム ID。アラームが複数ある場合は、*aid* が正しいアラームを指定

パブリッシャの `csadmind` は、アラームをキューから削除し、通知を `enpd` へ送信します。次に、`enpd` デーモンは、この種類のイベントの購読者がいるかどうかを調べ、検出した購読に対する通知をサブスクライバ `csnotifyd` に送信します。アラーム通知 (リマインダ) に対する、それ以外のサブスクライバは、Sun ONE Calendar Server のインストール中に作成し、配備できます。これらの 3 つのデーモン間の対話により、Sun ONE Calendar Server のイベント通知が実装されます。

Sun ONE Calendar Server デーモン

Sun ONE Calendar Server には、ENS デーモンの `enpd` と対話する、次の 2 つのデーモンが含まれています。

- `csadmind`

  `csadmind` デーモンには、アラームイベントを ENS へ送信して、通知サービスに通知を送信するパブリッシャが含まれています。`csadmind` デーモンは、Sun ONE Calendar Server のアラームキューを管理します。また、スケジューラを実装することにより、アラームをいつ生成するかを知ることができます。その時点で、`csadmind` はイベントを公開します。ENS はイベント通知を受信してディスパッチします。

  アラームを確実に送信するために、`csadmind` は特定のイベントやイベントの種類に対して肯定応答を要求します (20 ページの「アラームの送信の信頼性」を参照) `csadmind` デーモンは高信頼イベント通知リンク (RENL) を使用して、肯定応答を返します。

- `csnotifyd`

  `csnotifyd` デーモンは、特定のイベントを配信対象にする (購読する) サブスクライバです。購読しているイベントに関する通知を ENS から受け取り、これらのイベントや todo (仕事) の通知を電子メールでクライアントに送信します。

ENS アーキテクチャには購読を解除する機能がありますが、csnotifyd では次の 2 つの理由によりイベントの購読が解除されません。正常な実行時には購読の解除または購読の再開をする必要がないからです。また、購読は一時的に保存される (メモリーに保存される) だけなので、ENS への接続が切断されるとすべての購読が自動的に解除されるからです。

csnotifyd デーモンは、enp:///ics/alarm/ を購読します。todo (仕事) またはイベントは、パラメータで指定されます。

アラームの送信の信頼性

アラームイベントの紛失を防ぐために、csadmind と csnotifyd は、ENS の RENL 機能を特定の種類のアラームに対して使用します。これらのアラーム通知に対して、csadmind は送信したそれぞれの通知に対し終端間 (end to end) の肯定応答を要求します。また、csnotifyd は、処理が正常に終了すると、受け取った各 RENL アラーム通知に対する肯定応答を生成します。

RENL アラームの場合、ネットワーク、ENS デーモン、または csnotifyd が通知の処理に失敗すると、csadmind が肯定応答を受け取らないため、アラームキューからアラームを削除しません。したがって、タイムアウト後にアラームがまた公開されます。

Sun ONE Calendar Server の例

Sun ONE Calendar Server における ENS の公開および購読の一般的な流れを示します。

1. イベントサブスクライバの csnotifyd が、あるイベントを配信対象にすることを表明します (購読)。
2. イベントパブリッシャの csadmind は、イベントを検出して通知を送信します (公開)。
3. ENS がイベントをサブスクライバに公開します。
4. イベントサブスクライバが、イベントの配信対象をキャンセルします (購読の解除)。この手順は、ENS への接続が切断されると、自動的に行われます。

21 ページの図 1-2 でこの流れを図示しています。また、21 ページの表 1-1 は図の説明です。

図 1-2 Sun ONE Calendar Server のイベント通知サービスの公開および購読の流れの例

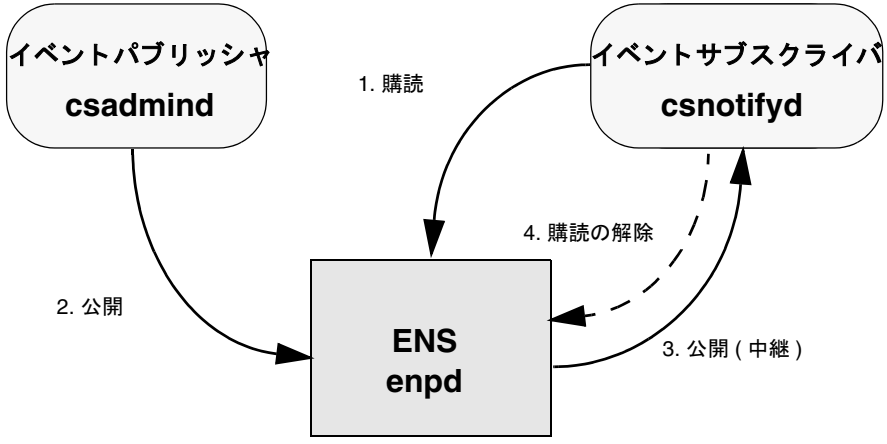


表 1-1 イベント通知サービスの公開および購読の流れの例

動作	ENS の応答
1. csnotifyd デーモンが ENS に購読要求を送信する	ENS が購読を購読データベースに格納する
2. csadmind デーモンが ENS に通知要求を送信する	ENS は通知に対応する購読を購読データベースで検索する
3. csnotifyd デーモンが ENS から通知を受信する	ENS がパブリッシャから通知を受信すると、内部購読テーブルを検索して、通知のイベント参照に一致する購読を検出する。次に、この購読を所有するサブスクライバへ通知のコピーを購読ごとに中継する
4. このバージョンの csnotifyd は、ENS にキャンセル要求を送信しない	購読は、データベースではなくメモリーだけに保存されるので、ENS への接続が切断されるとすべての購読が自動的に解除される

iPlanet Messaging Server と ENS との対話

23 ページの図 1-3 は、ENS が iPlanet Messaging Server と対話する方法を示したものです。この図ではそれぞれ、楕円形がプロセス、四角形は楕円で囲まれたプロセスを実行するホストコンピュータを表しています。

ENS サーバは、通知を iPlanet Messaging Server 通知プラグインから ENS クライアント (iBiff サブスクライバ) に配信します。ENS サーバより前の通知については、順序に保証はありません。イベントは異なるプロセス (`MTA`、`stored`、および `imapd`) から送信されるためです。

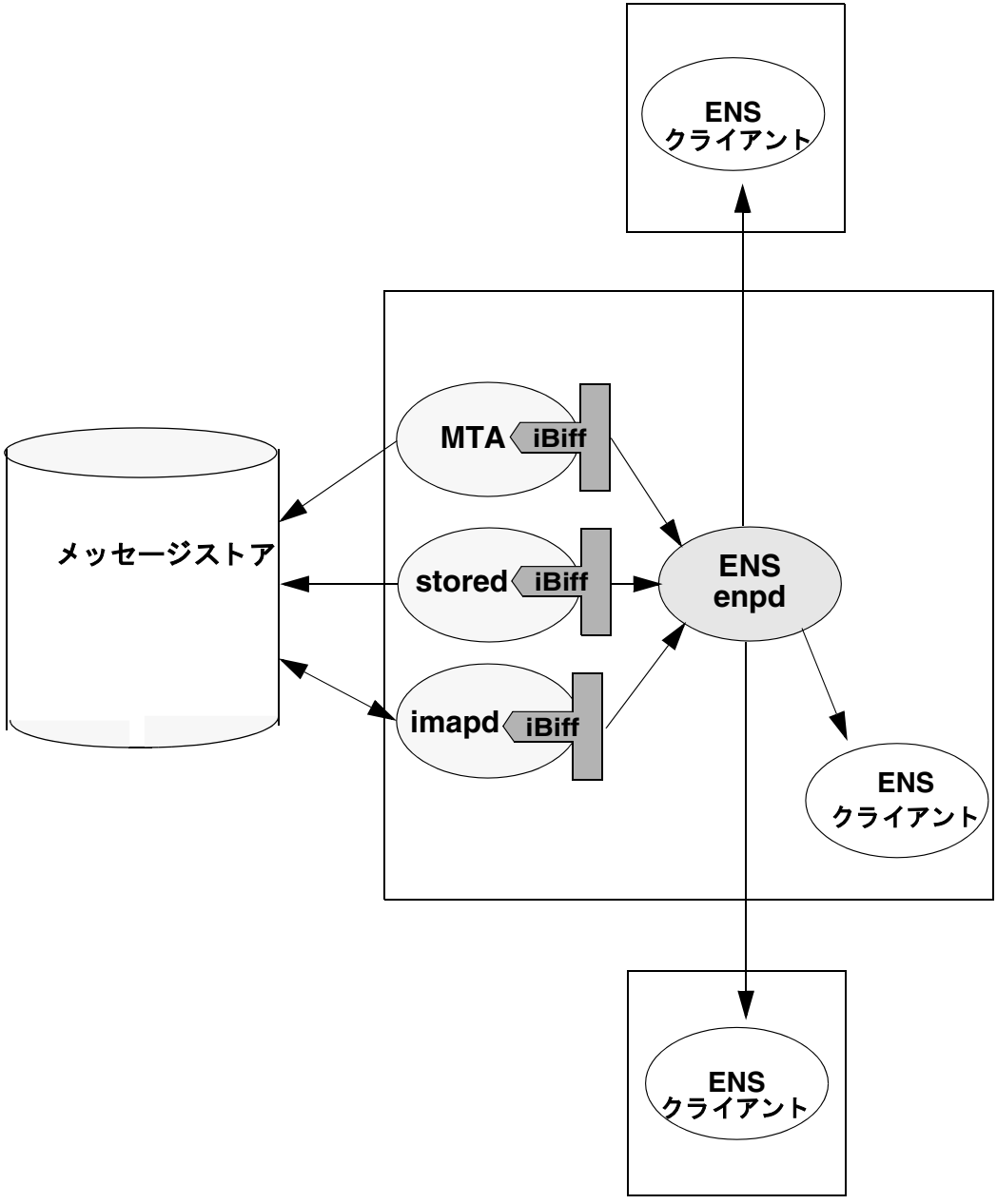
通知は、`MTA` プロセス、`stored` プロセス、および `imap` プロセスの iBiff プラグインから ENS `enpd` に転送されます。ENS クライアントは、ENS を購読して通知を受信します。iBiff が使用可能な場合、iPlanet Messaging Server は iBiff プラグインを使用して通知を公開しますが、これらの通知を購読する iPlanet Messaging Server サービスはありません。顧客作成の ENS サブスクライバまたはクライアントは、通知をコンシュームし、必要な処理を実行するように記述されている必要があります。つまり、iPlanet Messaging Server 自体は機能を実現するために、通知に依存することも通知を使用することもありません。そのため、iPlanet Messaging Server をインストールしたとき、デフォルトでは ENS と iBiff は使用できなくなっています。

iPlanet Messaging Server のアーキテクチャでは、指定した一連のメールボックスに対して、指定したホストコンピュータがサービスを提供します。指定したメールボックスに、複数のホストコンピュータがサービスを提供することはありません。指定したメールボックスを操作するプロセスはいくつかありますが、指定したメールボックスにサービスを提供するコンピュータホストは 1 つだけです。したがって、通知を受信するためには、エンドユーザは、配信対象のメールボックスにサービスを提供している ENS デーモンを購読するだけで十分です。

iPlanet Messaging Server を使用すると、すべてのメールボックス用に 1 台の ENS サーバ (すなわち、メッセージストアにサービスを提供するすべてのコンピュータホストに対して 1 台の ENS サーバ) を持つこと、あるいは複数の ENS サーバ、おおむねコンピュータホストごとに 1 台の ENS サーバを持つことができます。この場合は、2 番目の方がスケーラブルです。また、エンドユーザが配信対象のメールボックスのイベントを取得するには、複数の ENS サーバを購読しなければなりません。

したがって、このアーキテクチャでは、コンピュータホストごとに ENS サーバが必要です。ENS のサーバプロセスとクライアントプロセスは、2 つを同じ場所に配置したり、Messaging Server と同じ場所に配置したりする必要はありません。

図 1-3 iPlanet Messaging Server における ENS の概要



イベント通知サービス API の概要

この節では、C API および Java Messaging Service (JMS) API のサブセットである Java API の、ENS の 2 つの API について説明します。iPlanet Messaging Server 5.2 および Sun ONE Calendar Server 5.1 以降、ENS に Java API が追加されています。Java API は、Java Message Service 仕様 (JMS) に準拠しています。JMS API を使用した Java サブスクライバのサンプルが 2 つ用意されています。

ENS C API の詳細については、第 2 章、「イベント通知サービス C API リファレンス」を参照してください。Java (JMS) API の詳細については、第 3 章、「イベント通知サービス Java (JMS) API リファレンス」を参照してください。JMS のマニュアルについては、次の URL にアクセスしてください。

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

ENS C API の概要

ENS は、次の 3 つの API を実装しています。

- パブリッシャ API

  パブリッシャが購読対象イベントの通知を ENS に送信すると、ENS はこれをサブスクライバに配信します。Sun ONE Calendar Server では、アプリケーションが通知の受信に対する肯定応答を要求することもできます。これを行うには、RENL (高信頼イベント通知リンク) が必要です。RENL は、パブリッシャ、サブスクライバ、および肯定応答の対象になる通知を識別する一意の ID で構成されます。パブリッシャは、`publish_a` に渡される `end2end_ack` コールバックを呼び出して、アプリケーションに肯定応答の受信を知らせます。現在のところ、Sun ONE Calendar Server だけが RENL をサポートしています。

- サブスクライバ API

  サブスクライバは、特定のイベントを配信対象にする通知サービスに対するクライアントです。通知サービスは、これらのイベントのうちの 1 つに関する通知をパブリッシャから受信すると、サブスクライバにその通知を中継します。

  サブスクライバは購読を解除 (有効な購読をキャンセル) することもできます。

  Sun ONE Calendar Server で RENL を使用可能にするには、サブスクライバが ENS に対してその存在を宣言し、次に ENS がサブスクライバアプリケーションに代わって通知の肯定応答を透過的に作成します。サブスクライバは、いつでも RENL を廃棄できます。

- 公開および購読ディスパッチャ API

  非同期パブリッシャを使用する場合、ENS は、コールバックを呼び出すためにスレッドプールからスレッドを借りる必要があります。アプリケーションは、独自のスレッドプールを作成して ENS に渡すことも、あるいは ENS に独自のスレッドプールを作成して管理させることもできます。どちらの場合も、ENS はディスパッチャオブジェクトの作成して、使用するディスパッチャ (`pas_dispatcher_t`) をインスタンス化に利用できます。

  GDisp (`libasync`) は、サポートされているディスパッチャです。

ENS Java API の概要

ENS 用 Java API では、標準 JMS API のサブセットとともに、次の 2 つの新しい専用メソッドを使用しています。

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

次の JMS オブジェクトクラスのリストは、 ENS 用 Java API で使用されるものです。

- `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`

注　ENS 用 Java API は、すべての JMS オブジェクトクラスを実装しているわけではありません。カスタマイズするときは、このリストにあるオブジェクトクラスだけを使用してください。

カスタムアプリケーションの構築と実行

ユーザ自身のカスタムパブリッシャとサブスクライバアプリケーションの構築を支援するために、iPlanet Messaging Server および Sun ONE Calendar Server にはコーディング例が含まれています。この節では、コーディング例、API のインクルード (ヘッダー) ファイルの場所、およびカスタムプログラムの構築と実行に必要なライブラリの場所を示します。

注　この節の説明は、C API にのみ適用されます。

コーディング例の場所

Sun ONE Calendar Server

Sun ONE Calendar Server には、初めての構築を支援するために、4 つの簡単なサンプルプログラムが含まれています。これらのコーディング例は、次のディレクトリにあります。

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

iPlanet Messaging Server

iPlanet Messaging Server 5.1 以降には、通知を受信する方法の理解を支援するためのサンプルプログラムが含まれています。これらのサンプルプログラムは、*server-root*/bin/msg/enssdk/examples ディレクトリにあります。

インクルードファイルの場所

Sun ONE Calendar Server

パブリッシャおよびサブスクライバ API には、publisher.h、suscriber.h および、pasdisp.h (公開および購読ディパッチャ) のインクルード (ヘッダー) ファイルがあります。これらのファイルは、CSAPI include ディレクトリにあります。デフォルトの include パスは、次のとおりです。

```
/opt/SUNWics5/cal/csapi/include
```

iPlanet Messaging Server

デフォルトの iPlanet Messaging Server の include パスは、次のとおりです。

server-root/bin/msg/enssdk/include

動的にリンクまたは共有されるライブラリ

Sun ONE Calendar Server

カスタムコードは、動的にリンクされる libens ライブラリとリンクする必要があります。このライブラリには、パブリッシャおよびサブスクライバの API が実装されています。一部のプラットフォームでは、libens に従属するすべてのライブラリを、リンク指示の一部に含める必要があります。それらの従属ライブラリを順番に示します。

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

上記のライブラリは、Sun ONE Calendar Server で使用されるため、サーバの bin ディレクトリに配置されています。デフォルトの libens パスは、次のとおりです。

```
/opt/SUNWics5/cal/bin
```

注 Windows NT の場合、パブリッシャおよびサブスクライバアプリケーションを構築するには、上記のすべてのライブラリに対応するアーカイブファイル (.lib ファイル) も必要となります。アーカイブファイルは、CSAPI ライブラリのディレクトリ lib にあります。デフォルトの lib パスは、次のとおりです。

drive:¥Program Files¥iPlanet¥cal¥csapi¥lib

iPlanet Messaging Server

iPlanet Messaging Server のライブラリは、次のディレクトリにあります。

server-root/bin/msg/lib

必要なライブラリを判断するには、*server-root*/bin/msg/enssdk/examples/Makefile.sample を参照してください。この Makefile には、apub プログラムと asub プログラムをコンパイルして実行する方法に関する指示が含まれています。また、このファイルは、必要なライブラリ、および LD_LIBRARY_PATH の形式についても説明しています。

図 1-4 Makefile.sample ファイル

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

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

all: apub asub

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

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

run:
        @echo 'run <server-root>/msg-<instance>/start-ens'
        @echo run asub localhost 7997
        @echo run apub localhost 7997
```

注

Windows NT には、次のファイルが追加されています。

server-root¥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

Windows NT 上で構築するには、次の手順に従います。

1. asub.dsw に、サンプル VC++ ワークスペースが用意されています。その中には、asub.dsp および apub.dsp という、2 つのプロジェクトがあります。

   リンクする必要のある .lib ファイルは、asub.c および apub.c と同じディレクトリにあります。

2. 実行するには、パスに次の DLL が必要になります。

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

   この作業を最も簡単に行う方法は、PATH の ¥msg¥lib に *server-root* を入れることです。

実行時ライブラリパス変数

Sun ONE Calendar Server

/opt/SUNWics5/cal/bin ディレクトリにある、カスタムプログラムで必要な実行時ライブラリを検出するためには、ユーザ環境の実行時ライブラリパスの変数に必ずこのディレクトリを入れます。この変数の名前は、プラットフォームによって異なります。

- SunOS および Linux: LD_LIBRARY_PATH
- Winidows NT: PATH
- HPUX: SHLIB_PATH

iPlanet Messaging Server

iPlanet Messaging Server では、LD_LIBRARY_PATH を *server-root*/bin/msg/lib に設定します。

第 2 章

イベント通知サービス C API リファレンス

この章では、ENS C API について詳しく説明します。次の 3 つの節から構成されています。

- パブリッシャ API
- サブスクライバ API
- 公開および購読ディスパッチャ API

パブリッシャ API 関数リスト

ここでは、表 2-1 のリストにあるパブリッシャ関数について説明します。

表 2-1 ENS パブリッシャ API 関数リスト

定義 / 関数	説明
publisher_t	パブリッシャの定義
publisher_cb_t	非同期呼び出しに肯定応答する汎用コールバック関数
publisher_new_a	新規非同期パブリッシャを作成する
publisher_new_s	新規同期パブリッシャを作成する
publish_a	非同期通知を通知サービスに送信する
publish_s	同期通知を通知サービスに送信する
publisher_delete	公開セッションを終了する
publisher_get_subscriber	パブリッシャの資格を使ってサブスクライバを作成する

表 2-1 ENS パブリッシャ API 関数リスト (続き)

renl_create_publisher	RENL を作成して、end2end_ack の呼び出しを使用可能にする
renl_cancel_publisher	RENL をキャンセルする

サブスクライバ API 関数リスト

ここでは、表 2-2 のリストにあるサブスクライバ関数について説明します。

表 2-2 ENS サブスクライバ API 関数リスト

定義 / 関数	説明
subscriber_t	サブスクライバの定義
subscription_t	購読の定義
subscriber_cb_t	非同期呼び出しに肯定応答する汎用コールバック関数
subscriber_notify_cb_t	通知の受信時に呼び出される同期コールバック
subscriber_new_a	新規非同期サブスクライバを作成する
subscriber_new_s	新規同期サブスクライバを作成する
subscribe_a	非同期購読を設定する
unsubscribe_a	非同期購読をキャンセルする
subscriber_delete	サブスクライバを終了する
subscriber_get_publisher	サブスクライバの資格を使ってパブリッシャを作成する
renl_create_subscriber	RENL の購読部分を作成する
renl_cancel_subscriber	RENL をキャンセルする

公開および購読ディスパッチャ関数リスト

ここでは、表 2-3 のリストにある公開および購読ディスパッチャ関数について説明します。

表 2-3 ENS 公開および購読ディスパッチャ関数リスト

定義 / 関数	説明
pas_dispatcher_t	公開および購読ディスパッチャの定義
pas_dispatcher_new	ディスパッチャを作成する
pas_dispatcher_delete	`pas_dispatcher_new` で作成したディスパッチャを破棄する
pas_dispatch	イベント通知環境のディスパッチループを開始する
pas_shutdown	`pas_dispatch` で開始したイベント通知環境のディスパッチループを中止する

パブリッシャ API

パブリッシャ API は、1 つの定義と 9 つの関数で構成されています。

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

publisher_t

目的
パブリッシャ

構文
```
typedef struct enc_struct publisher_t;
```

パラメータ
なし

戻り値
なし

publisher_cb_t

目的
非同期呼び出しに肯定応答するために ENS が呼び出す汎用コールバック関数

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

パラメータ

arg	呼び出し側が渡すコンテキスト変数
rc	リターンコード
data	汎用、新しく作成されたコンテキストを含む

戻り値
なし

publisher_new_a

目的
新規非同期パブリッシャを作成する

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

パラメータ

disp	pas_dispatcher_new が戻す P&S スレッドプールのコンテキスト

worker	アプリケーションワーカ。NULL 以外の場合、このパブリッシャのセッションを保守するために ENS が作成した既存のワーカとグループ化される。複数のスレッドをパブリッシャデータへ同時にアクセスさせないために使用する
host	通知サーバのホスト名
port	通知サーバのポート
cbdone	パブリッシャが正常に作成されたとき、または作成できなかったときに呼び出されるコールバック cbdone の 3 つのパラメータ • cbarg 最初の引数 • 状態コード ゼロ以外の場合は、パブリッシャが作成されず、値に失敗の原因が示される • 新規有効パブリッシャ
cbarg	cbdone の最初の引数

戻り値
なし。cbdone コールバックの 3 番目の引数として、新規有効パブリッシャを渡す

publisher_new_s

目的
新規同期パブリッシャを作成する

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

パラメータ

disp	pas_dispatcher_new が戻す P&S スレッドプールコンテキスト
worker	アプリケーションワーカ。NULL 以外の場合、このパブリッシャのセッションを保守するためにENS が作成した既存のワーカとグループ化される。複数のスレッドをパブリッシャデータへ同時にアクセスさせないために使用する

host	通知サーバのホスト名
port	通知サーバのポート

戻り値
新規有効パブリッシャ (publisher_t)

publish_a

目的
非同期通知を通知サービスに送信する

構文

```
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);
```

パラメータ

publisher_t	有効パブリッシャ
event_ref	イベント参照。修正されたリソースを識別する URI
data	イベントデータ。通知メッセージの本体。通知サービスに対して不透明で、通知サービスはイベントサブスクライバへの中継だけを行う
datalen	バイト単位のデータの長さ
cbdone	通知サービスによってデータが受け入れられた場合、または受け入れられなかったと見なされた場合に呼び出されるコールバック。通知の受け入れは、使用するプロトコルに依存する。プロトコルには、トランスポート肯定応答 (TCP) または独自の肯定応答レスポンスメカニズムの使用を選択できる
end2end_ack	RENL 内のコンシューマピアから肯定応答を受信したあとに呼び出されるコールバック関数。RENL のコンテキスト内だけで使用される
cbarg	cbdone または end2end_ack を呼び出すときの最初の引数
timeout	RENL の完了を待機する時間の長さ

戻り値
なし

publish_s

目的
同期通知を通知サービスに送信する

構文

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

パラメータ

publisher	有効パブリッシャ
event_ref	イベント参照。修正されたリソースを識別する URI
data	イベントデータ。通知メッセージの本体。通知サービスに対して不透明で、イベントサブスクライバへの中継だけを行う
datalen	バイト単位のデータの長さ

戻り値
成功した場合はゼロ、失敗した場合は障害コード。RENL の場合、コンシューマが通知を完全に処理し、正常に肯定応答するまで、呼び出しは戻されない

publisher_delete

目的
公開セッションを終了する

構文

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

パラメータ

publisher	削除対象のパブリッシャ

戻り値
なし

publisher_get_subscriber

目的
パブリッシャの資格を使ってサブスクライバを作成する

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

パラメータ

publisher	サブスクライバの作成に使用される資格を持つパブリッシャ

戻り値
作成に成功した場合はそのサブスクライバ、失敗した場合は NULL。作成に失敗した場合は、subscriber_new を使用してサブスクライバを作成する

renl_create_publisher

目的
RENL を宣言し、end2end_ack の呼び出しを使用可能にする。この呼び出しが戻されたあと、指定されたパブリッシャおよびサブスクライバに一致する肯定応答の通知を受信したときに、end2end_ack 引数が呼び出される

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

パラメータ

publisher	有効パブリッシャ
renl_id	一意の RENL 識別子。これにより、2 つのピアの間に複数の RENL を設定できる

subscriber	認証されたピアの識別情報
cbdone	RENL が確立されたときに呼び出されるコールバック
cbarg	cbdone を呼び出すときの最初の引数

戻り値
なし

renl_cancel_publisher

目的
RENL をキャンセルする。通知は送信され続けるが、クライアントの肯定応答を受信しても、公開の end2end_ack 引数は呼び出されない。パブリッシャを削除すると、すべての RENL は自動的に破棄されるため、パブリッシャを削除する前に、この関数を呼び出して RENL 関連のメモリーを解放しなくてもよい

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

パラメータ

renl	キャンセル対象の RENL

戻り値
なし

サブスクライバ API

サブスクライバ API は、2 つの定義と 10 の関数で構成されています。

- subscriber_t
- subscription_t
- subscriber_cb_t
- subscriber_notify_cb_t
- subscriber_new_a
- subscriber_new_s

- subscribe_a
- unsubscribe_a
- subscriber_delete
- subscriber_get_publisher
- renl_create_subscriber
- renl_cancel_subscriber

subscriber_t

目的
サブスクライバ

構文
```
typedef struct enc_struct subscriber_t;
```

パラメータ
なし

戻り値
なし

subscription_t

目的
購読

構文
```
typedef struct subscription_struct subscription_t;
```

パラメータ
なし

戻り値
なし

subscriber_cb_t

目的
非同期呼び出しに肯定応答するために ENS が呼び出す汎用コールバック関数

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

パラメータ

arg	呼び出し側が渡すコンテキスト変数
rc	リターンコード
data	汎用、新しく作成されたコンテキストを含む

戻り値
なし

subscriber_notify_cb_t

目的
通知の受信時に呼び出されるサブスクライバのコールバック

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

パラメータ

arg	購読するために渡されるコンテキストポインタ (notify_arg)
event	イベント参照 (URI)。通知イベント参照は購読と一致するが、uid などのイベント属性と呼ばれる追加情報を含む場合がある
data	通知の本体。MIME オブジェクト
datalen	データの長さ

戻り値
成功した場合はゼロ、失敗した場合はゼロ以外の値

subscriber_new_a

目的
新規非同期サブスクライバを作成する

構文

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

パラメータ

disp	pas_dispatcher_new が戻すスレッドディスパッチャコンテキスト
worker	アプリケーションワーカ。NULL 以外の場合、このパブリッシャのセッションを保守するために ENS が作成した既存のワーカとグループ化される。また、複数のスレッドをパブリッシャデータへ同時にアクセスさせないために使用される。呼び出し側が GDisp コンテキストを作成してディスパッチする場合にだけ使用できる
host	通知サーバのホスト名または IP アドレス
port	購読サービスのポート番号
cbdone	サブスクライバセッションが有効になり、購読が発行されるときに呼び出されるコールバック cbdone の 3 つのパラメータ • cbarg 最初の引数 • 状態コード ゼロ以外の場合は、サブスクライバを作成できなかったことを示し、値に失敗の原因が示される • 新規有効サブスクライバ (subscriber_t)
cbarg	cbdone の最初の引数

戻り値
なし。cbdone コールバックの 3 番目の引数として、新規有効サブスクライバを渡す

subscriber_new_s

目的
新規同期サブスクライバを作成する

構文

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

パラメータ

disp	pas_dispatcher_new が返す公開および購読ディスパッチャ
worker	アプリケーションワーカ。NULL 以外の場合、このパブリッシャのセッションを保守するために ENS が作成した既存のワーカとグループ化される。複数のスレッドをパブリッシャデータへ同時にアクセスさせないために使用する。呼び出し側が GDisp コンテキストを作成してディスパッチする場合にだけ使用できる
host	通知サーバのホスト名または IP アドレス
port	購読サービスのポート番号

戻り値
新規有効サブスクライバ (subscriber_t)

subscribe_a

目的
非同期購読を設定する

構文

```
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):
```

パラメータ

subscriber	サブスクライバ
event_ref	イベント参照。イベント ソースを識別する URI
notify_cb	この購読に一致する通知を受信したときに呼び出されるコールバック
notify_arg	notify_arg の最初の引数。購読が有効な間、任意のスレッドから任意のタイミングで呼び出される
cbdone	購読の解除が完了したときに呼び出される。パラメータは次の 3 つ • cbarg (下記の説明を参照) • 状態コード • 不透明購読オブジェクトへのポインタ
cbarg	cbdone の最初の引数

戻り値
なし

unsubscribe_a

目的
非同期購読をキャンセルする

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

パラメータ

subscriber	消滅するサブスクライバ
subscription	キャンセル対象の購読
cbdone	購読の解除が完了したときに呼び出される。パラメータは次の 3 つ • cbarg (下記の説明を参照) • 状態コード • 不透明購読オブジェクトへのポインタ

cbarg	cbdone の最初の引数

戻り値
なし

subscriber_delete

目的
サブスクライバを削除する

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

パラメータ

subscriber	削除対象のサブスクライバ

戻り値
なし

subscriber_get_publisher

目的
サブスクライバの資格を使ってパブリッシャを作成する

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

パラメータ

subscriber	パブリッシャの作成に使用される資格を持つサブスクライバ

戻り値
作成に成功した場合はそのパブリッシャ、失敗した場合は NULL。作成に失敗した場合は、publisher_new を使用してサブスクライバを作成する

renl_create_subscriber

目的
RENL の購読部分を作成する

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

パラメータ

subscription	購読
renl_id	一意の RENL 識別子。これにより、2 つのピアの間に複数の RENL を設定できる
publisher	認証されたピアの識別情報

戻り値
不透明 RENL オブジェクト

renl_cancel_subscriber

目的
RENL をキャンセルするが、購読はキャンセルしない。この購読で受信した通知にこれ以上肯定応答しないように ENS に伝える。これにより、RENL オブジェクトが破棄されるため、アプリケーションはこの RENL を使用できなくなる。購読をキャンセルすると、すべての RENL は自動的に破棄される。サブスクライバを削除する前に、この関数を呼び出して RENL 関連のメモリーを解放しなくてよい

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

パラメータ

renl	キャンセル対象の RENL

戻り値
なし

公開および購読ディスパッチャ API

公開および購読ディスパッチャ API は、1 つの定義と 4 つの関数で構成されています。

- pas_dispatcher_t
- pas_dispatcher_new
- pas_dispatcher_delete
- pas_dispatch
- pas_shutdown

注	サポートされているスレッドディスパッチャは、GDisp (libasync) だけです。

pas_dispatcher_t

目的
公開および購読ディスパッチャ

構文
```
typedef struct pas_dispatcher_struct pas_dispatcher_t;
```

パラメータ
なし

戻り値
なし

pas_dispatcher_new

目的
ディスパッチャを作成または通知する

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

パラメータ

dispcx	ディスパッチャコンテキスト。NULL の場合、通知のディスパッチを開始するには、アプリケーションから pas_dispatch を呼び出す必要がある NULL 以外の場合、ディスパッチャは libasync になる

戻り値
パブリッシャまたはサブスクライバを作成するときに使用するディスパッチャ (pas_dispatcher_t)

pas_dispatcher_delete

目的
pas_dispatcher_new で作成したディスパッチャを削除する

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

パラメータ

disp	イベント通知のクライアント環境

戻り値
なし

pas_dispatch

目的
イベント通知環境のディスパッチループを開始する。アプリケーションが独自のスレッドプールを使用している場合は効果なし

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

パラメータ

disp	新規ディスパッチャ

戻り値
なし

pas_shutdown

目的
pas_dispatch で開始したイベント通知環境のディスパッチループを停止する。アプリケーション供給のディスパッチャが pas_dispatcher_new に渡された場合は効果なし

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

パラメータ

disp	停止対象のディスパッチャコンテキスト

戻り値
なし

第 3 章

イベント通知サービス Java (JMS) API リファレンス

この章では、ENS における Java (JMS) API の実装について、および Java API について説明します。

この章は、次の節で構成されています。

- イベント通知サービス Java (JMS) API の実装
- Java (JMS) API の概要
- 実装上の注意

イベント通知サービス Java (JMS) API の実装

ENS Java API は、iPlanet Messaging Server 5.2 および Sun ONE Calendar Server 5.1 に組み込まれています。Java API は、Java Message Service 仕様 (JMS) に準拠しています。

ENS は、Java Message Service に対するプロバイダの役割を果たします。つまり、JMS は ENS に Java API を提供します。このソフトウェアは、ベースライブラリとデモプログラムで構成されています。

Java API を使用するための前提条件

Java API を使用するには、ENS を使用可能にしておく必要があります。iPlanet Messaging Server で ENS を使用可能にする方法については、『iPlanet Messaging Server 5.2 管理者ガイド』の付録 C を参照してください。Sun ONE Calendar Server では、デフォルトで ENS を使用できます。

また、 iPlanet Messaging Server と Sun ONE Calendar Server には含まれていない次のソフトウェアをインストールする必要があります。

- Java 開発キット (JDK) 1.2 以降
- Java Message Service 1.0.2a 以降 (1.0.2a でテスト済み)

このソフトウェアは、`http://java.sun.com` からダウンロードできます。

サンプル Java プログラム

iPlanet Messaging Server 5.2 サンプルプログラムの `JmsSample` および `JBiff` は、*server-root*`/bin/msg/enssdk/java/com/iplanet/ens/samples` ディレクトリに格納されています。`JmsSample` は、汎用 ENS サンプルプログラムです。`JBiff` は、iPlanet Messaging Server 固有のサンプルプログラムです。

`JBiff` の場合、次の項目を追加する必要があります。

- Java Mail jar ファイル (JavaMail 1.2 でテスト済み)
- Java Activation Framework (JavaMail に必要、JAF1.0.1 でテスト済み)

これらも、`http://java.sun.com` からダウンロードできます。

環境の設定

この節では、サンプルプログラムをコンパイルして実行するために必要な手順について説明します。

JmsSample プログラムをコンパイルするには

1. `CLASSPATH` に次の項目の組み込みを設定します。

   ens.jar ファイル - `ens.jar`

   (iPlanet Messaging Server では、`ens.jar` は *server-root*`/java/jars/` ディレクトリにある)

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

2. *server-root*`/bin/msg/enssdk/java` ディレクトリに移動します。
3. 次のコマンドを実行します。

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

JBiff プログラムをコンパイルするには

1. CLASSPATH に次の項目の組み込みを設定します。

   ens.jar ファイル - `ens.jar`

   (iPlanet Messaging Server では、`ens.jar` は *`server-root`*`/java/jars/` ディレクトリにある)

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

   Java Mail - *`full-path`*`/javamail-1.2/mail.jar`

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

2. *`server-root`*`/bin/msg/enssdk/java` ディレクトリに移動します。
3. 次のコマンドを実行します。

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

JmsSample プログラムを実行するには

1. *`server-root`*`/bin/msg/enssdk/java` ディレクトリに移動します。
2. 次のコマンドを実行します。

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

3. 次の 3 つの項目の入力が要求されます。
   - ENS イベント参照 (iPlanet Messaging Server の場合の例 : `enp://127.0.0.1/store`)
   - ENS ホスト名
   - ENS ポート (通常は 7997)
4. イベントを公開します。

   iPlanet Messaging Server でイベントを公開するには、次の 2 つの方法があります。

   - ENS の `apub` C サンプルプログラムを使用できます。詳細については、83 ページの「iPlanet Messaging Server のコーディング例」を参照してください。

- ENS を使用可能にした場合は、iBiff が iPlanet Messaging Server 関連のイベントを公開するように設定します。

Sun ONE Calendar Server の場合、イベントは Calendar Server によって公開されます。

JBiff デモプログラムを実行するには

前提条件 :`JBiff` デモプログラムを実行するには、iPlanet Messaging Server で ENS を使用可能にしておく必要があります。その手順については、『iPlanet Messaging Server 5.2 管理者ガイド』の付録 C を参照してください。

注 デモは現在、ENS イベント参照 `enp://127.0.0.1/store` を使用するようハードコード化されています。これは、iBiff 通知プラグインが使用するデフォルトのイベント参照です。

1. *server-root*`/bin/msg/enssdk/java` ディレクトリに移動します。
2. 次のコマンドを実行します。

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

3. ユーザ ID (userid)、ホスト名 (hostname)、およびパスワード (password) の入力を求められます。

   コードは、ENS サーバおよび IMAP サーバがそのホスト上で稼動中であることを前提としています。userid および password は、IMAP アカウントにアクセスするための IMAP ユーザ名およびパスワードです。

2 つのテストプログラムは、ENS サブスクライバです。電子メールメッセージが iPlanet Messaging Server を介して転送されると、iBiff からイベントを受信します。または、`apub` C サンプルプログラムを使用して、イベントを生成することもできます。詳細については、83 ページの「iPlanet Messaging Server のコーディング例」を参照してください。

Java (JMS) API の概要

Java API for ENS は、標準 Java Messaging Service (JMS) API のサブセットを使用し、次の 2 つの新しい固有メソッドが追加されています。

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

- com.iplanet.ens.jms.EnsTopic

JMS では、2 つの ENS 固有クラスが提供する TopicConnectionFactory および Topic の作成が必要です。

標準 JMS のクラスとメソッドの詳細については、次のサイトにある JMS マニュアルを参照してください。

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

新しい固有メソッド

2 つの固有メソッドクラスとは、EnsTopicConnFactory および EnsTopic です。

com.iplanet.ens.jms.EnsTopicConnFactory

メソッドについて

このメソッドは、javax.jms.TopicConnectionFactory を返すコンストラクタです。JNDI 形式の検索で TopicConnectionFactory オブジェクトを取得する代わりに、このメソッドを使用できます。

構文

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

                throws java.io.IOException
```

引数

表 3-1 EnsTopicConnFactory の引数

引数	型	説明
name	String	javax.jms.Connection のクライアント ID
hostname	String	ENS サーバのホスト名
port	int	ENS サーバの TCP ポート
logStream	OutputStream	メッセージが記録される場所 (NULL にはできない)

com.iplanet.ens.jms.EnsTopic

このメソッドについて
このメソッドは、javax.jms.Topic を返すコンストラクタです。JNDI 形式の検索で javax.jms.Topic オブジェクトを取得する代わりに、このメソッドを使用できます。

構文
public EnsTopic (String eventRef)

引数

表 3-2 EnsTopic の引数

引数	型	説明
eventRef	String	ENS イベント参照

実装上の注意

この節では、ENS Java API を実装するときに注意すべき点について説明します。

現在の実装における欠点

現行の Java API の実装では、初期プロバイダインタフェースを備えていません。

JMS Topic Connection Factory および ENS Destination が明示的に呼び出されます。これらは、com.iplanet.ens.jms.EnsTopicConnFactory および com.iplanet.ens.jms.EnsTopic です。ENS では、TopicConnectionFactory オブジェクトや Topic オブジェクトの取得には JNDI を使用しません。

通知の配信

通知は、javax.jms.TextMessage として配信されます。ENS イベント参照のパラメータとその値は TextMessage に対するプロパティ名として提供されます。ペイロードは、TextMessage のデータとして提供されます。

JMS ヘッダー

- JMSDeliveryMode は常に NON_PERSISTENT に設定されている (次の配信のためにメッセージを保存しない)
- JMSRedelivered は常に false に設定されている
- JMSMessageID は内部 ID に設定されており、iPlanet Messaging Server の電子メールメッセージのヘッダーにある SMTP MessageID には設定されていない
- ペイロードは常に javax.jms.TextMessage。ENS のペイロードに相当する
- JMSDestination は、完全イベント参照に設定されている (この通知に固有のパラメータ / 値を含む)
- JMSCorrelationID - 内部シーケンス番号に設定される
- JMSTimestamp - メッセージが送信された時刻に設定される
  - iPlanet Messaging Server および iBiff では、timestamp パラメータに相当
  - Sun ONE Calendar Server では未使用
- JMSType - 通知の種類
  - iPlanet Messaging Server および iBiff では evtType パラメータに相当
  - Sun ONE Calendar Server では未使用
- 追加のプロパティ :
  - イベント参照の各パラメータと値は、ヘッダーではプロパティになる。プロパティ値はすべて String 型
- 未使用ヘッダーは、JMSExpiration、JMSpriority、JMSReplyTo

その他

- MessageSelectors は実装されていない
- JMS は、永続サブスクライバおよび非永続サブスクライバという概念を使用している。永続サブスクライバは、オフライン、または突然の障害が発生した場合でも、サブスクライバに通知が送信されることを保証する機能。突然の障害には、ENS サーバがパブリッシャから通知を受信したが、サブスクライバへの配信を行う前にダウンした、などがある
  - 非永続サブスクライバは実装されている
  - 永続サブスクライバも使用できるが、すべての機能は実装できない

- 永続サブスクライバが実装されているという解釈は、サブスクライバがメッセージを受信して初めてパブリッシャが肯定応答を戻すこと
- 永続サブスクライバが実装されていないという解釈は、メッセージが持続的ではなく、オフラインサブスクライバに対しては (オンラインになった後も) 配信が行われないこと。特に、JMSRedelivered は常に false に設定されている

第 4 章

Sun ONE Calendar Server 固有の情報

この章では、ENS API を使用するために必要となる Sun™ ONE Calendar Server に固有の項目について説明します。

この章は、次の節で構成されています。

- Sun ONE Calendar Server 通知
  - アラーム通知
  - カレンダー更新通知
  - 拡張機能トピック
  - WCAP appid パラメータおよび X-Token
- Sun ONE Calendar Server コーディング例

Sun ONE Calendar Server 通知

Sun ONE Calendar Server 通知の形式は、次の 2 つの部分に分かれています。

- イベント参照 - イベントを識別する URL
- ペイロード - イベントを記述するデータ。バイナリ、テキスト / カレンダー、およびテキスト /XML の 3 種類のペイロード形式をサポート

カレンダー通知には、リマインダを中継するアラーム通知、およびカレンダーデータベースに変更を配信するカレンダー更新通知の 2 種類があります。

アラーム通知

アラーム通知は、リマインダを中継します。csadmind デーモンがリマインダを送信しようとするたびに公開されます。Sun ONE では、これらのアラームのデフォルトのサブスクライバは、csnotifyd デーモンです。csnotifyd がコンシュームする通知は、バイナリペイロードを持ち、肯定応答を受けます (高信頼)。

また、追加の通知を各リマインダに 1 つ生成するようにサーバを構成して、サードパーティの通知インフラストラクチャがコンシュームできるようにすることも可能です。

表 4-1 に、2 種類のアラーム通知を使用可能にする方法、そのベースイベント URL、および各アラームのイベントペイロード形式を示します (63 ページの「カレンダー通知の形式」を参照)。

表 4-1 アラーム通知

種類	使用可能にする方法	ベースイベント URL	イベントペイロード形式
デフォルトのアラーム通知	デフォルト	enp:///ics/alarms	バイナリ
オプションのアラーム通知	ics.conf に caldb.serveralarms.contenttype の NULL 以外の値が存在すると、オプションのアラーム通知が使用可能になる	ics.conf ファイルのパラメータ caldb.serveralarms.url の値を enp:///ics/alarms に設定する	ics.conf ファイルのパラメータ caldb.serveralarms.contenttype の値を text/calendar または text/xml に設定する

イベント URL のパラメータは、次のいずれかと同じです。

- calid - カレンダー ID
- uid - event または todo (仕事) のコンポーネント ID
- rid - 再帰 ID
- aid - アラーム ID
- comptype - イベントまたは todo (仕事)

カレンダー更新通知

カレンダー更新通知 は、カレンダーデータベースに変更を配信します。データベースに変更が加えられるたびに、cshttpd デーモンまたは csdwpd デーモンによって公開されます (このタイプの変更に対する通知が使用可能になっている場合)。

表 4-2 に、各タイプのカレンダー更新通知と、それぞれの ics.conf 設定、ベースイベント URL を示します。

表 4-2 カレンダー更新通知

種類	使用可能にする ics.conf のパラメータ (特に記載のないかぎり、パラメータのデフォルトはすべて「yes」)	ベースイベント URL および値 またはイベントペイロード
出席者の再表示処理	caldb.berkeleydb.ensmsg.refreshevent	caldb.berkeleydb.ensmsg.refreshevent.url
	default=no	デフォルト値: enp:///ics/caleventrefresh
出席者の再表示処理	caldb.berkeleydb.ensmsg.refreshevent.contenttype	イベントペイロードのデフォルト値: text/xml
出席者の応答処理	caldb.berkeleydb.ensmsg.replyevent	caldb.berkeleydb.ensmsg.replyevent.url
	default=no	デフォルト値: enp:///ics/caleventreply
出席者の応答処理	caldb.berkeleydb.ensmsg replyevent.contenttype	イベントペイロードのデフォルト値: text/xml
カレンダー作成	caldb.berkeleydb.ensmsg.createcal	caldb.berkeleydb.ensmsg.createcal.url
		デフォルト値: enp:///ics/calendarcreate
カレンダー削除	caldb.berkeleydb.ensmsg.deletecal	caldb.berkeleydb.ensmsg.deletecal.url
		デフォルト値: enp:///ics/calendardelete
カレンダー修正	caldb.berkeleydb.ensmsg.modifycal	caldb.berkeleydb.ensmsg.modifycal.url
		デフォルト値: enp:///ics/calendarmodify

表 4-2 カレンダー更新通知 (続き)

種類	使用可能にする ics.conf のパラメータ (特に記載のないかぎり、パラメータのデフォルトはすべて「yes」)	ベースイベント URL および値 またはイベントペイロード
イベント作成	caldb.berkeleydb.ensmsg.createeevent	caldb.berkeleydb.ensmsg.createeevent.url デフォルト値: enp:///ics/caleventcreate
イベント修正	caldb.berkeleydb.ensmsg.modifyevent	caldb.berkeleydb.ensmsg.modifyevent.url デフォルト値: enp:///ics/caleventmodify
イベント削除	caldb.berkeleydb.ensmsg.deleteevent	caldb.berkeleydb.ensmsg.deleteevent.url デフォルト値: enp:///ics/caleventdelete
todo (仕事) 作成	caldb.berkeleydb.ensmsg.createtodo	caldb.berkeleydb.ensmsg.createtodo.url デフォルト値: enp:///ics/caltodocreate
todo (仕事) 修正	caldb.berkeleydb.ensmsg.modifytodo	caldb.berkeleydb.ensmsg.modifytodo デフォルト値: enp:///ics/caltodomodify
todo (仕事) 削除	caldb.berkeleydb.ensmsg.deletetodo	caldb.berkeleydb.ensmsg.deletetodo.url デフォルト値: enp:///ics/caltododelete

イベント URL パラメータに含まれる項目は次のとおりです。

- calid - カレンダー ID
- uid - event または todo (仕事) のコンポーネント ID
- rid - 再帰 ID

カレンダー通知の形式

通知は、次の 2 つの部分に分かれています。

- イベント参照 - イベントを識別する URL
- ペイロード - イベントを記述するデータ。バイナリ、テキスト / カレンダー、テキスト /XML の 3 種類のデータ形式をサポート

拡張機能トピック

通常、 出席者の応答および開催者の再表示のための ENS 通知は、 他の修正と共に caldb.berkeleydb.ensmsg.modifyevent トピックに公開されます。ics.conf の変数 caldb.berkeleydb.ensmsg.advancedtopics を「yes」(デフォルトは「no」) に設定すると、ENS 通知を公開してトピックの修正、応答および再表示を分けることができます。これにより通知のコンシューマは、どの種類のトランザクションが通知をトリガーしたかをより正確に把握することができます。

表 4-3 は、ENS が ics.conf の変数 caldb.berkeleydb.ensmsg.advancedtopics の設定に応じて通知を公開するトピックを示します。

表 4-3 拡張機能トピックの変数

ics.conf 変数 caldb.berkeleydb.ensmsg. advancedtopics の値	ENS が出席者通知を公開するトピック
yes	caldb.berkeleydb.ensmsg.modifyevent caldb.berkeleydb.ensmsg.refreshevent caldb.berkeleydb.ensmsg.replyevent
no	caldb.berkeleydb.ensmsg.modifyevent

WCAP appid パラメータおよび X-Token

ENS が既存のイベントに行った修正の通知を送信する場合、通知と共に X-NSCP-COMPONENT-SOURCE および X-NSCP-TRIGGERED-BY の 2 つの X-Token を戻します。

X-NSCP-COMPONENT-SOURCE X-Token の内容は、イベントの送信者やそのイベントを要求したオリジナルの WCAP コマンドに appid パラメータが存在しているかどうかによって変わります。

appid パラメータがオリジナルの WCAP コマンドに存在する場合、ENS は X-NSCP-COMPONENT-SOURCE X-Token にその値を返します (特定のコマンドだけが appid パラメータを取得します。appid パラメータの詳細は、『Sun ONE Calendar Server プログラマーズマニュアル』を参照してください)。このメカニズムを使用して、アプリケーションは、送信した通知を検出するために ENS 通知に「タグ」を付けることができます。appid コマンドの値は、アプリケーション選択の文字列です。appid パラメータが存在しない場合は、標準値は送信元に応じて X-Token に割り当てられます 。これらの標準値については、表 4-4 を参照してください 。

X-Token (X-NSCP-TRIGGERED-BY) は、appid パラメータの有無に関係なく、通知をトリガーした開催者または出席者の名前 (uid) を保持します。

表 4-4 に、WCAP コマンドに appid パラメータが存在するかどうかによって、X-Token X-NSCP-COMPONENT-SOURCE の値がどのように異なるかを示します。

表 4-4 appid の存在および X-Token X-NSCP-COMPONENT-SOURCE の値

appid の存在	要求元による X-Token X-NSCP-COMPONENT-SOURCE の値
なし	標準値 : WCAP (デフォルト) CALENDAR EXPRESS (UI から) ADMIN (管理ツールから)
あり	appid の値

Sun ONE Calendar Server コーディング例

Sun ONE Calendar Server は、完全な ENS 実装を伴って出荷されています。ENS API を使用して Calendar Server をカスタマイズすることができます。次の 4 つのコーディング例 (簡単なパブリッシャとサブスクライバ、および信頼性の高いパブリッシャとサブスクライバ) で、ENS API の使い方を示します。コーディング例は、製品の次のディレクトリにあります。

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

パブリッシャおよびサブスクライバのコーディング例

次の 2 つのコーディング例では、簡単な対話形式の非同期パブリッシャとサブスクライバを確立します。

パブリッシャのコーディング例

```
/*
 * 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://yoyo.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);
}
```

サブスクライバのコーディング例

```
/*
 * Copyright 1997 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://yoyo.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);
}
```

信頼性の高いパブリッシャとサブスクライバ

次の 2 つのコーディング例では、信頼性の高い非同期パブリッシャとサブスクライバを確立します。

信頼性の高いパブリッシャのコーディング例

```
/*
 * 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://yoyo.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);
}
```

信頼性の高いサブスクライバのコーディング例

```
/*
 * Copyright 1997 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://yoyo.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);
}
```

第 5 章

iPlanet Messaging Server 固有の情報

この章では、ENS API を使用するために必要となる iPlanet™ Messaging Server に固有の項目について説明します。

この章は、次の節で構成されています。

- iPlanet Messaging Server のイベントおよびパラメータ
- iPlanet Messaging Server のコーディング例

iPlanet Messaging Server のイベントおよびパラメータ

iPlanet Messaging Server では、イベント参照は 1 つだけですが、そのイベント参照を複数のパラメータで構成することができます。各パラメータには、値とペイロードがあります。

iPlanet Messaging Server は、次の種類のイベントをサポートしています。

- NewMsg - ユーザのメールボックスに新しいメッセージを受信
- DeleteMsg - ユーザがメールボックスからメッセージを削除 (IMAP プロトコルでは「expunge (抹消)」)
- UpdateMsg - メッセージが (NewMsg 以外のイベントによって) メールボックスに追加。たとえば、ユーザが電子メールをメールボックスにコピー
- ReadMsg - メールボックス内のメッセージが既読になった (IMAP プロトコルではメッセージが 「Seen (既読)」のマークが付いた)
- PurgeMsg - メッセージがメールボックスからパージされた (IMAP プロトコルでは「expunge (抹消)」)

上記のサポートされるイベントに対しては、以下が適用されます。

- すべてのイベントは「受信箱」だけに関係する
- NewMsg 通知は、メッセージがユーザのメールボックスに入って初めて発行される (「サーバが受け付けて、メッセージキューに入ったところで発行される」とは反対)
- `DeleteMsg` イベントと `PurgeMsg` イベントはどちらも、メッセージがユーザのメールボックスから削除されたとき (IMAP プロトコルでは、メッセージに expunge が実行されたとき) に相当する。IMAP プロトコルで削除のマークが付いたときではない。2 つのイベントの唯一の違いは、誰がメッセージを削除したかという点。`DeleteMsg` は、ユーザがメッセージを削除したことを示し、`PurgeMsg` は、iPlanet Messaging Server がメッセージを削除したことを示す (たとえば、メッセージの期限が切れた場合)
- 通知では、イベントの種類に応じて、さまざまな情報を運ぶ。たとえば、`NewMsg` は、新しいメッセージの IMAP uid を示す
- POP3 クライアントのアクセスに対するイベントは生成されない

パラメータ

iBiff は、ENS イベント参照に次の形式を使用します。

`enp://127.0.0.1/store`*?param=value&param1=value1&param2=value2*

イベントキーの `enp://127.0.0.1/store` は、文字列として一意であるということ以外、大きな意味はありません。たとえば、イベントキーのホスト名の部分には、ホスト名としての意味はありません。URI の一部である文字列というだけです。ただし、イベントキーはユーザが構成することができます。iBiff 構成パラメータの一覧を、以下の別の節に示します。

イベント参照の 2 番目の部分は、パラメータとその値のペアで構成されています。この部分は、疑問符 (?) を使用してイベントキーと区切られています。パラメータと値は、等号 (=) で区切られています。パラメータと値の各ペアは、アンパサンド (&) で区切られています。値が空の場合もありますが、これはただ値が存在しないということです。

79 ページの表 5-1 では、すべての通知に含める必要がある、必須構成パラメータについて説明します。

表 5-1 必須構成パラメータ

パラメータ	データ型	説明
evtType	文字列	イベントの種類を指定する。NewMsg、UpdateMsg、ReadMsg、DeleteMsg、PurgeMsg のいずれか
mailboxName	文字列	メッセージストアのメールボックス名を指定する。mailboxName の書式は uid@domain (uid はユーザ ID で、domain はユーザが属するドメイン)。@domain 部分は、ユーザがデフォルトのドメインに属していない場合 (ユーザがホストドメイン内にいる場合) に限って追加される
timestamp	64-ビット整数	epoch (1970 年 1 月 1 日午前 0 時 GMT) から起算してミリ秒で指定する
process	文字列	イベントを生成したプロセスの名前を指定する。プロセス名がわからない場合は、プロセス ID が使用される (整数)
hostname	文字列	イベントを生成したマシンのホスト名

表 5-2 で、イベントの種類に依存する、オプションの構成パラメータについて説明します。

表 5-2 オプションの構成パラメータ

パラメータ	データ型	説明
numMsgs	符号なし 32 ビット整数	既存のメッセージ数を指定する
size	符号なし 32 ビット整数	メッセージのサイズを指定する。ペイロードは通常、メッセージが短縮されたものであるため、これは、ペイロードのサイズとは一致しない
uidValidity	符号なし 32 ビット整数	IMAP uid 有効パラメータを指定する
imapUid	符号なし 32 ビット整数	IMAP uid パラメータを指定する
uidSeqSeen	文字列	「1:6」のように、IMAP 構文で 「seen (既読)」のマークが付いた uid のリストを指定する
lastUid	符号なし 32 ビット整数	最後に使用された IMAP uid を指定する
hdrLen	符号なし 32 ビット整数	メッセージヘッダーのサイズを指定する。ペイロードは短縮されているため、これは、ペイロードのヘッダーのサイズとは一致しない
qUsed	符号なし 32 ビット整数	割り当てで使用されるディスク容量を KB で指定する

表 5-2 オプションの構成パラメータ (続き)

パラメータ	データ型	説明
qMax	符号なし 32 ビット整数	ディスク容量割り当てを KB で指定する。値が -1 に設定されているときは、割り当てがないことを示す
qMsgUsed	符号なし 32 ビット整数	割り当てで使用されるメッセージの数を指定する。numMsgs と同じ値を指定すること
qMsgMax	符号なし 32 ビット整数	最大メッセージ数に対する割り当てを指定する。値が -1 に設定されているときは、割り当てがないことを示す

注 サブスクライバは、イベント参照を構文解析するとき、文書化されていないパラメータについても考慮する必要があります。将来、新しいパラメータが追加されたときに互換性を持たせることができます。

ペイロード

イベントに応じて、ENS 通知のペイロード部分には次のデータが含まれます。

- メッセージのヘッダー (文字列) - 長さは特定の (構成可能な) サイズに制限される。次の各節に示す構成パラメータ参照
- メッセージ本体の最初の数バイト (文字列) - 実際のバイト数は構成可能。次の各節に示す構成パラメータ参照

表 5-3 に、各イベントの種類で使用できるパラメータを示します。

表 5-3 各イベントタイプに使用できるパラメータ

フィールド名	NewMsg、UpdateMsg	ReadMsg	DeleteMsg、PurgeMsg
numMsgs	可	不可	可
size	可	不可	不可
uidValidity	可	可	可
imapUid	可	不可	可
uidSeqSeen	不可	可	不可
uidSeqDel	不可	可	不可
lastUid	不可	不可	可

表 5-3 各イベントタイプに使用できるパラメータ (続き)

フィールド名	NewMsg、UpdateMsg	ReadMsg	DeleteMsg、PurgeMsg
hdrLen	可	不可	不可
qUsed	可	不可	可
qMax	可	不可	可
qMsgUsed	可	不可	可
qMsgMax	可	不可	可
ペイロード (ヘッダー / 本文)	可	不可	不可

例

次に、NewMsg イベント参照の例を示します (読みやすくするために数行に分けてありますが、実際は 1 行です)。

```
enp://127.0.0.1/store?evtType=NewMsg&mailboxName=ketu310&timestamp=972423964000
&process=16233&hostname=ketu&numMsgs=1&size=3339&uidValidity=972423964&
imapUid=1&hdrLen=810
```

これは、関連付けられているペイロードです。本体の部分は短縮されています。

```
Return-path: <>
Received: from process-daemon.ketu.siroe.com by ketu.siroe.com
(iPlanet Messaging Server 5.0  (built Oct 17 2000))
id <0G2Y00C01F4SIY@ketu.siroe.com> for ketu310@ims-ms-daemon
(ORCPT ketu310@siroe.com); Tue, 24 Oct 2000 14:46:04 -0700 (PDT)
Received: from ketu.siroe.com
(iPlanet Messaging Server 5.0  (built Oct 17 2000))
id <0G2Y00C01F4RIX@ketu.siroe.com>; Tue, 24 Oct 2000 14:46:04 -0700 (PDT)
Date: Tue, 24 Oct 2000 14:46:04 -0700 (PDT)
From: Internet Mail Delivery
Subject: Delivery Notification: Delivery has failed
To: ketu310@siroe.com
Message-id: <0G2Y00C05F4SIX@ketu.siroe.com>
MIME-version: 1.0
Content-type: multipart/report; report-type=delivery-status;
boundary="Boundary_(ID_VlTrnuIgC5ferJnL2SCzhQ)"


--Boundary_(ID_VlTrnuIgC5ferJnL2SCzhQ)
Content-type: text/plain; charset=us-ascii
Content-language
```

次に別の例として、DeleteMsg イベントの例を示します (ここでも読みやすくするために 1 行を数行に分けています)。この例では、ホストドメイン `symult.com` のユーザ ID `blim` の mailboxName を示しています。

```
enp://127.0.0.1/store?evtType=DeleteMsg&mailboxName=blim@symult.com&
timestamp=972423953000&process=15354&hostname=ketu&numMsgs=0&
uidValidity=972423928&imapUid=2&lastUid=2
```

3 番目に、ReadMsg イベントの例を示します (ここでも読みやすくするために 1 行を数行に分けています)。この例では、uidSeqSeen パラメータに空の値が示されています。また、前述の例と同じユーザ ID を共有していますが、これは別のユーザ、つまりデフォルトのドメイン内にいるユーザと対応しています。

```
enp://127.0.0.1/store?evtType=ReadMsg&mailboxName=blim&timestamp=972423952000&
process=15354&hostname=ketu&uidValidity=972423928&uidSeqSeen=&uidSeqDel=1
```

iPlanet Messaging Server のコーディング例

iPlanet Messaging Server は、完全な ENS 実装を伴って出荷されていますが、デフォルトでは使用できません。ENS で iPlanet Messaging Server を使用可能にするには、『iPlanet Messaging Server 5.2 管理者ガイド』の付録 C を参照してください。

次の 2 つのコーディング例で、ENS API の使い方を説明します。コーディング例は、製品の次のディレクトリにあります。

server-root/bin/msg/enssdk/examples

パブリッシャのコーディング例

次のコーディング例は、簡単な対話形式の非同期パブリッシャを設定します。

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

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

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

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

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

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

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

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

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

    _read_stdin();
    return;
}
```

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

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

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

サブスクライバのコーディング例

次のコーディング例は、簡単なサブスクライバを設定します。

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

実装上の注意

現在の実装では、購読するイベントに対するセキュリティ保護を提供していません。したがって、ユーザは、すべてのイベントおよび他のすべてのユーザのメールの一部を登録することができます。このため、ENS サブスクライバは、できるだけファイアウォールの「安全」な側に配置することを強くお勧めします。

用語集

iBiff　iPlanet Messaging Server において、メッセージストア通知を公開するプラグインのこと。通知を購読する方法についての仕様を含む。

todo　サーバ側の Sun ONE Calendar Server では、実行すべきことを指定するカレンダーのコンポーネント。クライアント側の Calendar Express では、todo は仕事とも呼ばれる。

イベント (event)　イベント参照用データを生成すること。Sun ONE Calendar Server の場合、リソース (カレンダー) に変更があると、イベントが発生する。iPlanet Messaging Server の場合、発生するイベントのリストがある (`NewMsg`、`DeleteMsg`、など)。

イベント通知サービス (Event Notification Service)　パブリッシャからサブスクライバに送信された通知を中継するアプリケーションフレームワーク。

イベントコンシューマ (event consumer)　「イベントサブスクライバ」の同義語。

イベントサブスクライバ (event subscriber)　イベントをコンシュームするアプリケーション。

イベント参照 (event reference)　ENS によって処理されるイベントを識別する。RFC 2396 で定義されている URI 構文に準拠。

イベントパブリッシャ (event publisher)　ほかのアプリケーションにイベントを通知するアプリケーション。

イベントプロデューサ (event producer)　「イベントパブリッシャ」の同義語。

公開する (publish)　通知を送信すること。イベントパブリッシャは、通知サービスに対してイベントを使用可能にする。

高信頼イベント通知リンク (RENL) (Reliable Event Notification Link (RENL))

RENL は、パブリッシャ、サブスクライバ、および肯定応答の対象になる通知を識別する一意の ID で構成される。

購読 (subscription) イベントサブスクライバが送信するメッセージ。イベント参照、クライアント側の要求識別子、およびオプションのアクセス制御規則を含む。

購読する (subscribe) 購読を送信すること。イベントサブスクライバは、特定のイベントの通知の受信希望を通知サービスに知らせる。

購読の解除 (unsubscription) このメッセージは、既存の購読をキャンセル (購読を解除) する。イベントサブスクライバは、特定のイベントの通知の中継を停止するよう通知サービスに知らせる。

購読を解除する (unsubscribe) 購読をキャンセルすること。イベントサブスクライバは、特定のイベントの通知の中継を停止するよう通知サービスに知らせる。

コンシュームされた (consumed) 通知がサービスによって受信されたり、処理されたりすることを、プロセスによってコンシュームされるという。

仕事 (task) クライアント側の Calendar Express では、実行すべきことを指定するカレンダーのコンポーネント。サーバ側では、仕事は todo とも呼ばれる。

通知 (notification) イベント発生を記述するメッセージ。イベントパブリッシャによって送信され、イベントへの参照とイベントコンシューマによって使用される通知サービスに対しては、不透明であるオプションデータを含む。

通知サーバ (notification server) 通知サービスは、1 つまたは複数のサーバインスタンスで構成されています。各インスタンスは、異なるホスト上で実行される。

通知サービス (notification service) ほかのサービスから購読と通知を受け取る。通知をサブスクライバに中継する。

通知する (notify) 「公開する」の同義語。

ペイロード (payload) イベントを記述するデータ。バイナリ、テキスト / カレンダー、およびテキスト /XML の 3 種類のペイロード形式がサポートされている。

リソース (resource) IP ネットワークからアクセスされるデータ。たとえば、カレンダー。

リソースの状態 (resource state) リソースを表わす属性の値。たとえば、会議の時刻など。

索引

A

E

I

P

R

S

U

あ

い

か

き

こ

し

つ