イベント通知サービス マニュアル

iPlanet™ Messaging and Collaboration

iPlanet Calendar Server 5.1 / iPlanet Messaging Server 5.2

816-4942-01
2002 年 1 月

目次

本書について

このマニュアルでは、iPlanet™ Messaging Server および iPlanet™ Calendar Server で利用される iPlanet™ 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 については、詳細に解説します。

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 iPlanet 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としています。

お読みになる前に

このマニュアルは、読者が C/C++ および Java メッセージングサービスの知識を持ち、以下の事柄について一般的に理解していることを前提として書かれています。

- インターネットと World Wide Web
- メッセージングとカレンダー処理の考え方

このマニュアルの内容

このマニュアルは、以下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」

  iPlanet 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 について説明します。サンプルコードも掲載します。
- 第 4 章 「iPlanet Calendar Server 固有の情報」

  iPlanet Calendar Server のイベント通知について説明します。iPlanet Calendar Server のサンプルコードも掲載します。
- 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 のサンプルコードも示します。
- 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、コード例にも使用します。

イタリックフォント (***Italic***) ー ユーザが入力するテキストをシステムに固有の情報を使って表現するときに使用します (変数など)。 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 iPlanet 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

このマニュアルのほかにも、以下のドキュメントが用意されています。

- iPlanet Messaging Server マニュアル

  `http://docs.iplanet.com/docs/manuals/messaging.html`

- iPlanet Calendar Server マニュアル

  `http://docs.iplanet.com/docs/manuals/calendar.html`

第 1 章

イベント通知サービスの概要

この章では、iPlanet 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、iPlanet 基礎を成す公開購読サービスで、次の iPlanet 製品で利用できます。

- iPlanet Calendar Server 5.0 以降
- iPlanet Messaging Server 5.1 以降 (統合されていますが、有効にはなっていません)

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、関係する特定の種類のイベントを収集する中心点として、iPlanet アプリケーションが使用するディスパッチャの役割を果たします。 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です。 この構成では、URI (Uniform Resource Identifier) はイベントを表しています。 この種のイベントがいつ発生するかを知りたい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、通知と購読を照合します。 イベントの例は次のとおりです。

- 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が割り当てを超過している
- 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に関係すると登録のある他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、通知サービスがイベントを有効にします。サブスクライバは、通知サービスに対して特定のイベントの通知を受信したいと知らせます。 ENS API の詳細については、25 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。

iPlanet Calendar Server における ENS

iPlanet Calendar Server では、デフォルトで ENS が使用できます。 iPlanet Calendar Server で ENS を使用するための設定は特に必要ありません。

iPlanet 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。

iPlanet Calendar Server には、ENS C パブリッシャとサブスクライバのコードサンプルが付属しています。 そのコードについては、68 ページの「iPlanet Calendar Server サンプルコード」 を参照してください。

iPlanet 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 における ENS

ENS および iBiff (iPlanet Messaging Server の ENS パブリッシャ、また、iPlanet Messaging Server の通知プラグインとも呼ばれています) は、iPlanet Messaging Server に付属しています。 ただし、デフォルトでは、有効になっていません。

iPlanet Messaging Server で通知を購読するには、最初に iPlanet Messaging Server ホスト上で次の 2 つの項目を実行します。

- iBiff 通知プラグインを読み込みます。
- メッセージングサーバを停止してから、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付属しています。 詳細については、87 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*=*value*1&*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です。

  iPlanet 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です。
- *resource* は、authority コンテキスト内のリソースへのパスです。 複数のパスコンポーネントで構成するには、スラッシュ (“/”) で区切ります。
- *param* は、リソースの状態を表すパラメータの名前です。
- *value* は、パラメータの値です。 パラメータとその値の組み合わせは複数個指定できます。指定しなくてもかまいません。

通常、すべての iPlanet Calendar Server イベントは次のように始まります。

```
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、スキーマ、権限、およびリソース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使用され、それ以上には解釈されません。

iPlanet 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 で、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を均衡させます。 この機能によって、複数のサブスクライバがまとまって単一のイベント参照からの通知をすべて受信するような、サブスクライバのプールをクライアントに持たせることができます。

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け取ります。 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け取るには、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、依然として単純なイベント参照 (この例では `enp://127.0.0.1/store`) に通知を送信します。つまり、パブリッシャはサブスクライバプールを知らないということです。

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 つ持たせ、各プールがイベント参照からのすべての通知を受け取ることができます。 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 アルファベットだけを使用した文字列です (base64 アルファベットの内容については 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を 2 つ持つ例では、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、さまざまなカレンダーサーバやメッセージングサーバの構成で他の iPlanet 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を収集およびディスパッチします。 Windows NT プラットフォームでは、ENS はサービス enpd.exe として実行されます。

ENS では、イベントはリソースに発生した変更で、リソース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。 そこで、このイベントが購読され、通知がサブスクライバに送信されます。

ENS アーキテクチャを使用すると、次の 3 つの動作が実行できます。

- 通知 - イベントの発生を記述するメッセージです。イベントパブリッシャにより送信され、それはイベントへの参照のほか、付加的な URI に追加されるパラメータ / 値のペア、そしてイベント・コンシューマにより使われ通知サービスからは非透過な任意のデータ (ペイロード) を含んでいます。イベントに関心があるものはどんなものでも購読できます。
- 購読 - イベントを購読するために送信されるメッセージです。 イベント参照、クライアント側要求識別情報、および URI に追加される任意のパラメータ / 値のペアを含んでいます。購読は、「次のイベント」に適用されます (サブスクライバが「次のイベント」を通知するよう要求します)。
- 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) します。 イベントサブスクライバは、指定したイベントの通知の中継を停止するよう ENS に知らせます。

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」は、「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ことができます)
- オプションのアプリケーション固有のデータ (ENS に対しては「非透過」ですが、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ます)

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ということについてパブリッシャが知らないか関心がない場合は、この要求は確実にアクノリッジされる必要はありません。 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との間に RENL (Reliable Event Notification Link) を設定することに合意できます。 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、アクノリッジメント通知がパブリッシャに返されます。

- **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ります。 高信頼通知には、低信頼通知と同じ属性が含まれます。

詳細については、36 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通知要求を受け取ります。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 / 値のペアを含めることができます)
- 要求識別情報

詳細については、44 ページの「サブスクライバ API」を参照してください。

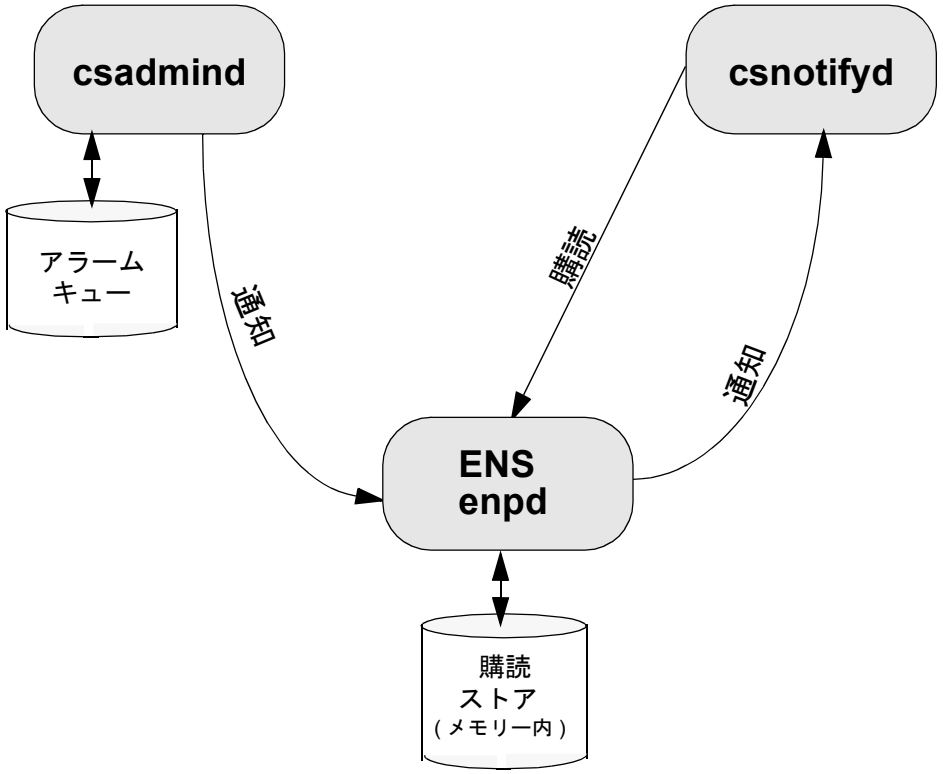
購読の解除

ENS が、既存の購読をキャンセルする要求を受け取ります。 詳細については、44 ページの「サブスクライバ API」を参照してください。

iPlanet Calendar Server が ENS と対話するしくみ

19 ページの図 1-1 に、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 iPlanet Calendar Server と対話するしくみを示します。

図 1-1 iPlanet Calendar Server における ENS の概要



iPlanet Calendar Server アラームキュー

ENS は、アラームディスパッチャで、アラームの配信と生成が分離されています。 また、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 です。
- ***uid*** は、カレンダー内のイベント / todo (仕事) ID です。
- ***rid*** は、再帰イベント / todo (仕事) の再帰 ID です。
- ***aid*** は、イベント / todo (仕事) 内のアラーム ID です。 アラームが複数ある場合は、***aid*** が正しいアラームを指定します。

パブリッシャ `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、iPlanet 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、iPlanet Calendar Server のイベント通知が実装されます。

iPlanet Calendar Server デーモン

iPlanet 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、通知を通知サービスに送信するパブリッシャが含まれています。これが iPlanet 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し、そうしてアラームがいつ生成されたかを知ることになります。この時点で、`csadmind` はイベントを公開します。ENS では、イベント通知の受信およびディスパッチが行われます。

  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 (21 ページの「アラームの送信の信頼性」参照)。`csadmind` デーモンは、RENL (Reliable Event Notification Links) を使用して、アクノリッジメントを達成します。

- `csnotifyd`

  `csnotifyd` デーモンは、特定のイベントへの関心を示す (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。正常に動作しているときに、購読の解除または再購読を行う必要はないからです。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暗黙に購読の解除をされるからです。

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 はそれが送信したそれぞれの通知に対し終端間 (end to end) のアクノリッジメントを要求します。また、csnotifyd は、処理を正常に処理すると、受け取った各 RENL アラーム通知に対するアクノリッジメント通知を生成します。

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 がアクノリッジメント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、アラームが再公開されます。

iPlanet Calendar Server 例

iPlanet Calendar Server における ENS の公開と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 から、あるイベントに対する関心を表します。(購読)。
2. イベントパブリッシャの csadmind は、イベントを検出し、通知を送信します。(通知の公開)
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は、イベントへの関心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暗黙に実行されます。

22 ページの図 1-2 で、この流れを示します。また、22 ページの表 1-1 で、この図の説明をします。

図 1-2 iPlanet Calendar Server のイベント通知サービスの公開と購読の流れの例

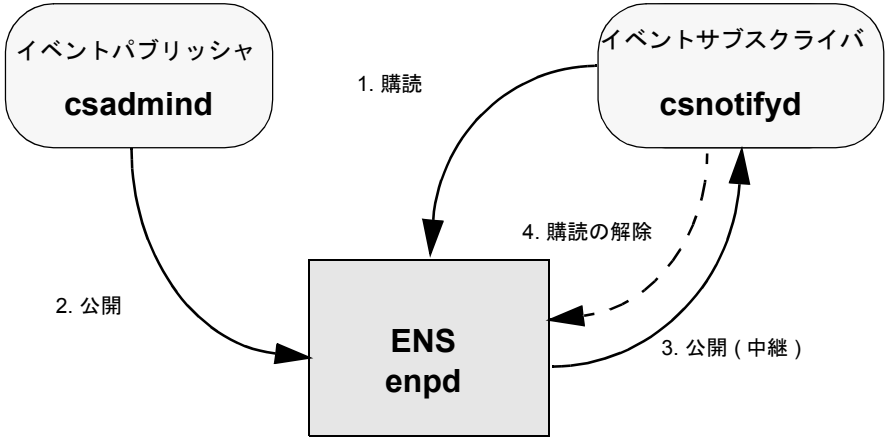


表 1-1 イベント通知サービスの公開と購読の流れの例

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 から通知を受け取る	ENS がパブリッシャから通知を受け取ると、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、現在のところ、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暗黙で解除される

iPlanet Messaging Server が ENS と対話するしくみ

24 ページの図 1-3 は、ENS が iPlanet Messaging Server と対話するしくみを示したものです。 この図では、各楕円形がプロセスを表し、各矩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ます。

iPlanet 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。

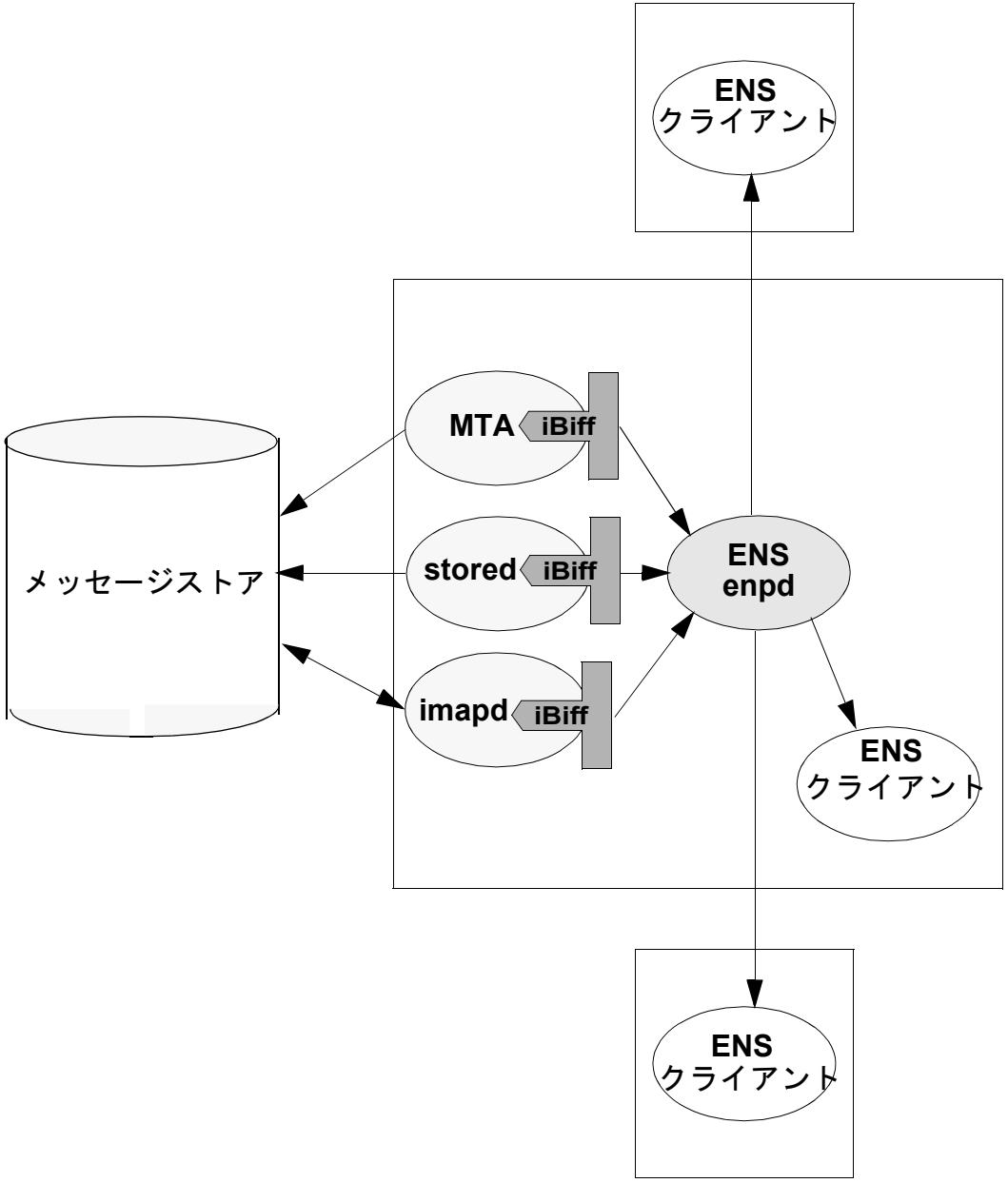
通知は、`MTA` プロセス、`stored` プロセス、および `imap` プロセスの iBiff プラグインから ENS `enpd` に転送されます。 ENS クライアントは、ENS を購読し、通知を受け取ります。 iBiff が使用可能な場合、iPlanet Messaging Server は iBiff プラグインを使用して通知を公開しますが、これらの通知を購読する iPlanet Messaging Server サービスはありません。 通知をコンシュームしたり、必要な処理を実行するためには、顧客提供の ENS サブスクライバまたはクライアントを作成するべきです。 つまり、iPlanet 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 は動作しないようになっています。

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、通知を受け取るためには、エンドユーザは、関心のある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) を持つことまたは、複数の ENS サーバ、おそらくコンピュータホストごとに 1 台の ENS サーバを持つことのどちらかが可能となります。 2 番目のシナリオの方がスケーラブルです。 また、このシナリオでは、エンドユーザが関心のある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、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を互いに同じ場所に配置したり、メッセージングサーバ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 iPlanet 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。iPlanet Calendar Server では、アプリケーションがオプションで通知の受信のアクノリッジメントを要求することもできます。これを行うには、RENL (Reliable Event Notification Link) が必要です。RENL は、パブリッシャ、サブスクライバ、およびアクノリッジメント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アクノリッジメントの受信を知らせます。現状では、iPlanet 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、特定のイベントへの関心を表す、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、パブリッシャからこれらのイベントのうちの 1 つに関する通知を受け取ると、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する (活性化している購読をキャンセルする) こともできます。

  iPlanet 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アクノリッジメント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、または、ENS に独自のスレッドプールを作成、管理させることもできます。いずれの場合も、ENS はディスパッチャオブジェクトを作成し、そのオブジェクトを使用して、使用するディスパッチャ (`pas_dispatcher_t`) をインスタンス化します。

  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 用の Java API は、標準 JMS API のサブセットを使い、次の 2 つの新しい専用メソッドを持っています。

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

次に示す 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作成と実行

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作成を支援するために、iPlanet Messaging Server および iPlanet Calendar Server にはコーディング例が付いています。 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、およびカスタムプログラムの作成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

iPlanet Calendar Server

iPlanet Calendar Server には、ユーザが始めるのを支援するために、4 つの簡単なサンプルプログラムが付いています。これらのサンプルのコードは、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付いています。 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

iPlanet 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スパッチャ) 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

iPlanet 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 に依存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、iPlanet 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作成するには、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 がどうなるべきかについても説明しています。

図 1-4 Makefile.sample ファイル

```
#
# サンプル makefile
#
#  C コンパイラ
CC = gcc

# LIBS
# ライブラリパスには <server-root>/bin/msg/lib を含める
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 のディストリビューションには、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 上で作成するには、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、次の実行パスに DLL が必要です。

   ```
   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 に \msg\lib の *server-root* を含めることです。

実行時ライブラリパス変数

iPlanet Calendar Server

`/opt/SUNWics5/cal/bin` ディレクトリからカスタムプログラムが必要とする実行時ライブラリを見つけられるように、ユーザ環境の実行時ライブラリのパス変数にこのディレクトリを含めます。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`
- Windows 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。この章は次の 3 つの節に分かれています。

- パブリッシャ API
- サブスクライバ API
- 公開と購読ディスパッチャ API

パブリッシャ API 関数リスト

この節では、表 2-1 に示すパブリッシャ関数を説明します。

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`	非同期呼び出しをアクノリッジ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
`renl_create_publisher`	RENL を作成する。これにより `end2end_ack` の呼び出しを有効にする
`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

この節では、表 2-2 に示すサブスクライバ関数を説明します。

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`	非同期呼び出しをアクノリッジ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

表 2-2 ENS サブスクライバ API 関数リスト（続き）

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

この節では、表 2-3 に示す公開と購読ディスパッチャ関数を説明します。

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	公開と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、次の 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

非同期呼び出しをアクノリッジするために 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	呼び出し元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します。

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 が返す 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ー。 NULL 以外の場合、ENS によって作成されたこのパブリッシャセッションをサービスする既存のワーカーとグループ化されます。複数のスレッドがパブリッシャデータに同時にアクセスしないようにするために使用します。
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 には次の 3 つのパラメータがあります • cbarg 最初の引数 • 状態コード ゼロ以外の場合は、パブリッシャを作成できなかったことを示し、値は失敗の原因を示します • 新規活性化パブリッシャ
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、新規活性化パブリッシャを渡します。

publisher_new_s

目的

新規同期パブリッシャを作成します。

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 が返す 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ー。 NULL 以外の場合、ENS によって作成されたこのパブリッシャセッションをサービスする既存のワーカーとグループ化されます。複数のスレッドがパブリッシャデータに同時にアクセスしないようにするために使用します。
host	通知サーバのホスト名
port	通知サーバのポート

戻り値

新規活性化パブリッシャ (publisher_t)

publish_a

目的

非同期通知を通知サービスに送信します。

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	活性化パブリッシャ
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非透過で、通知サービスはイベントのサブスクライバへの中継だけを行います
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します。プロトコルには、トランスポートアクノリッジメント (TCP) または独自のアクノリッジメント応答メカニズムのどちらを使用するかを選べます
end2end_ack	RENL 内のコンシューマピアからアクノリッジメントを受信した後に呼び出されるコールバック関数。 RENL のコンテキスト内でだけ使用されます
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します。

構文

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

パラメータ

publisher	活性化パブリッシャ
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非透過で、通知サービスはイベントのサブスクライバへの中継だけを行います
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、それを正常にアクノリッジするまで、呼び出しは戻ってきません。

publisher_delete

目的

公開セッションを終了します。

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します。

構文

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

パラメータ

publisher	サブスクライバを作成するときに使用す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します。

renl_create_publisher

目的

RENL を宣言します。これにより end2end_ack の呼び出しが有効になります。 この呼び出しが戻った後、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アクノリッジメント通知が受信されたときに、end2end_ack 引数が呼び出されます。

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	活性化パブリッシャ
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 を設定できます
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 をキャンセルします。 キャンセルしても、通知は送信されます。ただし、クライアントのアクノリッジメントを受信しても、公開の `end2end_ack` 引数は呼び出されません。 パブリッシャを削除すると、すべての RENL が自動的に破棄されます。 このため、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する必要はありません。

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、次の 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

非同期呼び出しをアクノリッジするために 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	呼び出し元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対応しています。ただし、uid などのイベント属性と呼ばれる情報を追加できます。
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します。

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 が返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ー。 NULL 以外の場合、ENS によって作成されたこのサブスクライバセッションをサービスする既存のワーカーとグループ化されます。また、複数のスレッドがパブリッシャデータに同時にアクセスしないようにするために使用します。 呼び出し元が 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ます
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活性化し、購読が発行されるときに呼び出されるコールバック cbdone には次の 3 つのパラメータがあります • 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、値は失敗の原因を示します • 新規活性化サブスクライバ (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、新規活性化サブスクライバを渡します。

subscriber_new_s

目的

新規同期サブスクライバを作成します。

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ー。 NULL 以外の場合、ENS によって作成されたこのパブリッシャセッションをサービスする既存のワーカーとグループ化されます。複数のスレッドがパブリッシャデータに同時にアクセスしないようにするために使用します。
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

新規活性化サブスクライバ (subscriber_t)

subscribe_a

目的

非同期購読を設定します。

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。 イベントのソースを識別する 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_cb の最初の引数。 購読が活性な間、任意のスレッドから任意のタイミングで呼び出されます。
cbdone	購読の解除が完了したときに呼び出されます。 次の 3 つのパラメータがあります。 • cbarg (以下の説明を参照) • 状態コード • 非透過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します。

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ます。 次の 3 つのパラメータがあります。 • cbarg (以下の説明を参照) • 状態コード • 非透過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します。

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します。

構文

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

パラメータ

subscriber	パブリッシャを作成するときに使用す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します。

renl_create_subscriber

目的

RENL の購読部を作成します。

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 を設定できます
publisher	認証されたピアの識別情報

戻り値

非透過 RENL オブジェクト

renl_cancel_subscriber

目的

RENL をキャンセルします。 ただし、購読はキャンセルしません。 この購読に受信した通知をこれ以上アクノリッジしないように、ENS に伝えます。 この関数によって、RENL オブジェクトが破棄され、アプリケーションはこの RENL をそれ以降使用できなくなります。 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ます。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する必要はありません。

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

公開と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、次の 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アドバタイズします。

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ります。 NULL 以外の場合、ディスパッチャ は libasync ディスパッチャです。

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します。

構文

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

パラメータ

disp	イベント通知クライアント環境

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します。 アプリケーションで自身のスレッドプールが使用されている場合は効果はありません。

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します。アプリケーションから指定されたディスパッチャが pas_dispatcher_new に渡された場合は効果はありません。

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 の実装および Java API 自体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 iPlanet Calendar Server 5.1 に含まれています。 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。 このため、JMS は ENS に Java API を提供します。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 を使用できるよう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。 iPlanet Calendar Server では、デフォルトで ENS が使用できます。

さらに、次のソフトウェアもインストールする必要があります。このソフトウェアは、iPlanet Messaging Server も iPlanet Calendar Server もどちらも提供していません。

- Java Development Kit (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、次の追加項目が必要となります。

- 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)

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

環境を設定する

この節では、サンプルプログラムをコンパイルして実行できるようにするために必要となることがら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` が次の項目を含むよう設定します。

   ens.jar ファイル - `ens.jar`

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

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

2. *`server-root`*`/bin/msg/enssdk/java` ディレクトリに移ります。
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 が次の項目を含むよう設定します。

   ens.jar ファイル - `ens.jar`

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

   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. *server-root*`/bin/msg/enssdk/java` ディレクトリに移ります。
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` ディレクトリに移ります。
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 には、イベントを公開する次の 2 つの方法があります。

   - ENS の `apub` C サンプルプログラムが使用できます。 詳細については、87 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 が使用可能である場合は、iBiff が iPlanet Messaging Server 関連のイベントを公開するよう設定します。

iPlanet Calendar Server の場合、イベントはカレンダーサーバ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 を使用できるようにしておく必要があります。 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。

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` ディレクトリに移ります。
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 *hostname* 上で実行してい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、87 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加わっています。

- 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 形式の検索を使用して `TopicConnectionFactory` オブジェクトを取得する代わりに、このメソッドが提供されます。

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 は指定できません)

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 形式の検索を使用して javax.jms.Topic オブジェクトを取得する代わりに、このメソッドが提供されます。

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 を実装するときに理解しておくべき項目について説明します。

現在の実装における欠点

Java API の現在の実装では、初期プロバイダインタフェースが提供されません。

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 JNDI は使用しません。

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` に設定されています (つまり以降の配信のためにメッセージは格納されません)。
- JMSRedelivered は常に、false に設定されています。
- JMSMessageID は、内部 ID に設定されています。つまり、iPlanet Messaging Server の電子メールメッセージのヘッダーの SMTP MessageID には設定されていません。
- ペイロードは常に、`javax.jms.TextMessage` です。 ENS のペイロードに対応しています。
- JMSDestination は、完全イベント参照に設定されています (つまり、この通知に固有のパラメータ / 値を含んでいます)。
- JMSCorrelationID - 内部シーケンス番号に設定されます。
- JMSTimestamp - メッセージが送信された時刻に設定されます。
  - iPlanet Messaging Server および iBiff の場合、`timestamp` パラメータに相当します。
  - iPlanet Calendar Server では、未使用です。
- JMSType - 通知の種類。
  - iPlanet Messaging Server および iBiff では、`evtType` パラメータに相当します。

- iPlanet Calendar Server では、未使用です。

- 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ります。 すべてのプロパティ値は、String 型です。
- 未使用ヘッダーは、JMSExpiration、JMSpriority、JMSReplyTo です。

その他

- MessageSelectors は実装されていません。
- JMS は、永続的サブスクライバおよび非永続的サブスクライバという概念を使用しています。 永続的サブスクライバは、オフライン、または破壊的な問題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です。破壊的な問題とは、ENS サーバがパブリッシャから通知を受け取ったが、サブスクライバへの配信を行う前にダウンした、というような状況です。
  - 非永続的サブスクライバは実装されています。
  - また、永続的サブスクライバを使用することもできますが、永続的サブスクライバとしての完全な機能は実装されていません。
  - 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アクノリッジされるという点で、永続的サブスクライバは実装されています。
  - 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 という点で、永続的サブスクライバは実装されていません。 特に、JMSRedelivered は常に false に設定されています。

第 4 章

iPlanet 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 iPlanet Calendar Server に固有の事柄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 Calendar Server 通知
- iPlanet Calendar Server サンプルコード

iPlanet Calendar Server 通知

iPlanet 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がサポートされています。

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と、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。 次に、両方の種類のカレンダー通知について説明します。

- **アラーム通知**　これらの通知は、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。 iPlanet Messaging and Collaboration イベント通知サービスでは、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、アクノリッジされます (高信頼)。

  また、追加の通知を各リマインダにつき 1 つ生成するようサーバを構成して、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できます。

表 4-1 に、2 種類のアラームを使用可能にする方法、そのベースイベント URI、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 (68 ページの「カレンダー通知の形式」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	バイナリ
オプションのアラーム通知	ics.conf の caldb.serveralarms.contenttype に NULL 以外の値がある	ics.conf の、caldb.serveralarms.url の値	ics.conf の、caldb.serveralarms.content の値

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**　これらの通知は、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して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 パラメータ (すべてのパラメータのデフォルトは「yes」)	ベースイベント URL および ics.conf
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

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 パラメータ (すべてのパラメータのデフォルトは「yes」)	ベースイベント URL および ics.conf
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
イベント作成	caldb.berkeleydb.ensmsg.createevent	caldb.berkeleydb.ensmsg.createevent.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 種類のデータ形式がサポートされています。

iPlanet Calendar Server サンプルコード

iPlanet 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 iPlanet Calendar Server をカスタマイズすることができます。 次の 4 つのコードサンプル (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
```

パブリッシャとサブスクライバのサンプルコード

次のコーディング例のペアでは、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ドサンプル

```
/*

 * Copyright 2000 by Sun Microsystems, Inc.
 * All rights reserved
 *
 * apub : 簡単な対話形式の非同期パブリッシャ
 * 構文 :
 *   apub ホストポート
 */
```

```
#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("\n 使用法 :\napub ホストポート \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(" 状態 %d のため、パブリッシャの作成に失敗しました \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(" 状態 %d のため、公開に失敗しました \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; /* \n を取り除く */
                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("パブリッシャを作成できません");

    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ドサンプル

```
/*
 * Copyright 1997 by Sun Microsystems, Inc.
 * All rights reserved
 *
 * asub : 非同期サブスクライバの例
 *
 * 構文：
 *   asub ホストポート
 */

#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("\n 使用法 :\nasub ホストポート \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(" 購読に成功しました \n");
    } else {
        printf(" 購読に失敗しました - 状態 %d\n", rc);
        pas_shutdown(disp);
    }
}

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

    (void *)ignored;
    (void *)arg;

    if (rc != 0)
    {
        printf(" 購読の解除に失敗しました - 状態 %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(" 状態 %d のため、サブスクライバの作成に失敗しました \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(" パブリッシャを作成できません ");

    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

次のコーディング例のペアでは、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ドサンプル

```
/*
 * Copyright 2000 by Sun Microsystems, Inc.
 * All rights reserved
 *
 * rpub : 簡単で * 信頼性の高い * 対話形式の非同期パブリッシャ
 * rsub （信頼性の高いサブスクライバ） と組み合わせて
 * 使用するように設計されています。
 *
 * 構文 :
 *   rpub ホストポート
 */

#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("\n 使用法 :\nrpub ホストポート \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("RENL の作成に失敗しました - 状態 %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("パブリッシャの作成に失敗しました - 状態 %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("アクノリッジメントのタイムアウト\n");
    } else if ( rc == 0) {
       printf("アクノリッジメントを受信しました\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; /* \n を取り除く */

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

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

          /* タイムアウト 5 秒 */
          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("パブリッシャを作成できません");

    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ドサンプル

```
/*
 * Copyright 1997 by Sun Microsystems, Inc.
 * All rights reserved
 *
 * asub : 非同期サブスクライバの例
 *
 * 構文:
 *   asub ホストポート
 */

#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("\n 使用法 :\nasub ホストポート \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(" 購読に成功しました \n");
        _renl = renl_create_subscriber(_subscription, "renl_id",
NULL);
    } else {
        printf(" 購読に失敗しました - 状態 %d\n", rc)
        pas_shutdown(disp);
    }
}

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

    if (rc != 0)
    {
        printf(" 購読の解除に失敗しました - 状態 %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(" 状態 %d のため、サブスクライバの作成に失敗しました \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(" パブリッシャを作成できません ");
```

```
    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しました。
- DeleteMsg - ユーザがメールボックスからメッセージを削除しました (IMAP プロトコルでは、「expunge (抹消)」が実行されました)。
- 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されました。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しました。
- ReadMsg - メールボックス内のメッセージが読まれました (IMAP プロトコルでは、メッセージに「Seen (既読)」のマークが付けられました)。
- PurgeMsg - メッセージは、メールボックスからパージされました (IMAP プロトコルでは、「expunge (抹消)」が実行されました)。

上記のサポートされるイベントに対して、以下が適用されます。

- すべてのイベントは、INBOX にのみ関係します。
- 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ます (「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) に対応しています。 IMAP プロトコルで削除とマークが付けられたときではありません。 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、という点です。 `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します (たとえば、メッセージの期限が切れた場合)。
- 通知には、イベントの種類に応じて、さまざまな情報を運びます。たとえば、`NewMsg` は、新しいメッセージの IMAP uid を示します。
- POP3 クライアントのアクセスに対するイベントは、生成されません。

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 (?) を使っ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。

表 5-1 で、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	存在する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
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 - 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 - 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許すべきです。 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) サイズに制限されます。次の各節に示す構成パラメータを参照してください。
- 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です。次の各節に示す構成パラメータを参照してください。

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 各イベントの種類で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
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。本体の部分は縮められています。

```
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)
ontent-type: text/plain; charset=us-ascii
Content-langua
```

次に別の例として、DeleteMsg イベントの例を示します (ここでも 1 行を読みやすくするために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 1 行を読みやすくするために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。 iPlanet Messaging Server で ENS 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ドサンプル

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与えます。

```
/*
 * Copyright 2000 by Sun Microsystems, Inc.
 * All rights reserved
 */
```

```
/*
 *
 *                                apub
 *                                 --
 *          簡単な対話形式の非同期パブリッシャ
 *                                 --
 *
 * この単純化されたプログラムは、
 * ハードコード化された以下のイベント参照を使用してイベントを公開します。
 *      enp://127.0.0.1/store
 * また、通知ペイロードとしてプロンプトに入力されたデータを使用します。
 * プログラムを終了するには、"." を入力します。
 *
 * 対応するサブスクライバの asub を同じ通知サーバ上で実行すると、
 * asub ウィンドウに送信データが
 * 出力されます。
 *
 * 構文：
 *      $ apub <host> <port>
 * 説明
 *      <host> は、通知サーバのホスト名
 *      <port> は、通知サーバの IP ポート番号
 */

#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("\n 使用法 :\napub ホストポート \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("状態 %d のため、パブリッシャの作成に失敗しました \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("状態 %d のため、公開に失敗しました \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; /* \n を取り除く */

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("パブリッシャを作成できません");

    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ドサンプル

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
 *                                 --
 *                          簡単なサブスクライバ
 *                                 --
 *
 * この単純化されたプログラムは、ハードコード化された次のイベント参照
 * に一致するイベントを購読します。
 *      enp://127.0.0.1/store
 * 続いて、apub プロセスのいずれかが使用されて、apub プロセスが放出した
 * メッセージを受信したら、各々の受信した通知のペイロードを
 * 標準出力へ表示します。
 *
 * 構文：
 *      $ asub <host> <port>
 * 説明
 *      <host> は、通知サーバのホスト名
 *      <port> は、通知サーバの IP ポート番号
 */

#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("\n 使用法 :\nasub ホストポート \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(" 購読に成功しました \n");
         subscriber_keepalive(_subscriber, 30000);
     } else {
         printf(" 購読に失敗しました - 状態 %d\n", rc);
```

```
        pas_shutdown(disp);
    }
}

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

    if (rc != 0) {
        printf("購読の解除に失敗しました - 状態 %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("状態 %d のため、サブスクライバの作成に失敗しました \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("パブリッシャを作成できません");

    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と他のすべてのユーザのメールの一部分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、少なくともファイアウォールの「安全」な側に配置することをお勧めします。

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みます。

iPlanet 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。

RENL (Reliable Event Notification Link)
RENL は、パブリッシャ、サブスクライバ、およびアクノリッジメントの対象となる通知を識別する一意の ID で構成されます。

todo
サーバ側の iPlanet 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 は仕事とも呼ばれます。

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

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します。RFC 2396 で定義されている URI 構文に準拠しています。

イベントパブリッシャ **(event publisher)**
他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、通知サービスにイベントを通知します。

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みます。

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を望むことを通知サービスに知らせます。

購読の解除 **(unsubscription)**
このメッセージは、存在する購読をキャンセル (購読を解除) します。イベントサブスクライバは、特定のイベントについて通知の中継を停止することを通知サービスに知らせます。

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、特定のイベントについて通知の中継を停止することを通知サービスに知らせます。

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います。

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、タスクは todo とも呼ばれます。

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、非透過であるオプションデータを含みます。

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ます。

通知サービス **(notification service)**
他のサーバから購読と通知を受け取ります。通知をサブスクライバに中継します。

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ます。

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、カレンダーはリソースです。

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

索引

A

E

か

き

こ

し

つ