SYBASE iAnywhere

QAnywhere

改訂 2007 年 3 月

版権と商標

目次

はじめに

このマニュアルの内容

このマニュアルでは QAnywhere について説明します。QAnywhere は、従来のデスクトップ・クライアントやラップトップ・クライアント用のメッセージング・プラットフォームであるほか、モバイル・クライアントや無線クライアント用のメッセージング・プラットフォームでもあります。

対象読者

このマニュアルは、モバイル・アプリケーションへのメッセージング機能の追加や、新しいモバイル・アプリケーション間メッセージング・ソリューションの構築を検討している、SQL Anywhere などのリレーショナル・データベースのユーザを対象としています。

SQL Anywhere のマニュアル

このマニュアルは、SQL Anywhere のマニュアル・セットの一部です。この項では、マニュアル・セットに含まれる各マニュアルと使用法について説明します。

SQL Anywhere のマニュアル

SQL Anywhere の完全なマニュアルは、各マニュアルをまとめたオンライン形式とマニュアル別の PDF ファイルで提供されます。いずれの形式のマニュアルも、同じ情報が含まれ、次のマニュアルから構成されます。

- **『SQL Anywhere 10 - 紹介』** このマニュアルでは、データの管理および交換機能を提供する包括的なパッケージである SQL Anywhere 10 について説明します。SQL Anywhere を使用すると、サーバ環境、デスクトップ環境、モバイル環境、リモート・オフィス環境に適したデータベース・ベースのアプリケーションを迅速に開発できるようになります。
- **『SQL Anywhere 10 - 変更点とアップグレード』** このマニュアルでは、SQL Anywhere 10 とそれ以前のバージョンに含まれる新機能について説明します。
- **『SQL Anywhere サーバ - データベース管理』** このマニュアルでは、SQL Anywhere データベースの実行、管理、設定について説明します。管理ユーティリティとオプションのほか、データベース接続、データベース・サーバ、データベース・ファイル、バックアップ・プロシージャ、セキュリティ、高可用性、Replication Server を使用したレプリケーションについて説明します。
- **『SQL Anywhere サーバ - SQL の使用法』** このマニュアルでは、データベースの設計と作成の方法、データのインポート・エクスポート・変更の方法、データの検索方法、ストアド・プロシージャとトリガの構築方法について説明します。
- **『SQL Anywhere サーバ - SQL リファレンス』** このマニュアルは、SQL Anywhere で使用する SQL 言語の完全なリファレンスです。また、SQL Anywhere のシステム・ビューとシステム・プロシージャについても説明しています。
- **『SQL Anywhere サーバ - プログラミング』** このマニュアルでは、C、C++、Java プログラミング言語、Visual Studio .NET を使用してデータベース・アプリケーションを構築、配備する方法について説明します。Visual Basic や PowerBuilder などのツールのユーザは、それらのツールのプログラミング・インタフェースを使用できます。
- **『SQL Anywhere 10 - エラー・メッセージ』** このマニュアルでは、SQL Anywhere エラー・メッセージの完全なリストを、その診断情報とともに説明します。
- **『Mobile Link - クイック・スタート』** このマニュアルでは、セッションベースのリレーショナル・データベース同期システムである Mobile Link について説明します。Mobile Link テクノロジは、双方向レプリケーションを可能にし、モバイル・コンピューティング環境に非常に適しています。
- **『Mobile Link - サーバ管理』** このマニュアルでは、Mobile Link アプリケーションを設定して管理する方法について説明します。

- **『Mobile Link - クライアント管理』** このマニュアルでは、Mobile Link クライアントを設定、構成、同期する方法について説明します。Mobile Link クライアントには、SQL Anywhere または Ultra Light のいずれかのデータベースを使用できます。
- **『Mobile Link - サーバ起動同期』** このマニュアルでは、Mobile Link のサーバによって開始される同期について説明します。サーバによって開始される同期とは、統合データベースから同期またはその他のリモート・アクションの開始を可能にする Mobile Link の機能です。
- **『QAnywhere』** このマニュアルでは QAnywhere について説明します。QAnywhere は、従来のデスクトップ・クライアントやラップトップ・クライアント用のメッセージング・プラットフォームであるほか、モバイル・クライアントや無線クライアント用のメッセージング・プラットフォームでもあります。
- **『SQL Remote』** このマニュアルでは、モバイル・コンピューティング用の SQL Remote データ・レプリケーション・システムについて説明します。このシステムによって、SQL Anywhere の統合データベースと複数の SQL Anywhere リモート・データベースの間で、電子メールやファイル転送などの間接的リンクを使用したデータ共有が可能になります。
- **『SQL Anywhere 10 - コンテキスト別ヘルプ』** このマニュアルには、[接続] ダイアログ、クエリ・エディタ、Mobile Link モニタ、SQL Anywhere コンソール・ユーティリティ、インデックス・コンサルタント、Interactive SQL のコンテキスト別のヘルプが収録されています。
- **『Ultra Light - データベース管理とリファレンス』** このマニュアルでは、小型デバイス用 Ultra Light データベース・システムの概要を説明します。
- **『Ultra Light - AppForge プログラミング』** このマニュアルでは、Ultra Light for AppForge について説明します。Ultra Light for AppForge を使用すると、Palm OS、Symbian OS、または Windows CE を搭載しているハンドヘルド、モバイル、または埋め込みデバイスに対してデータベース・アプリケーションを開発、配備できます。
- **『Ultra Light - .NET プログラミング』** このマニュアルでは、Ultra Light.NET について説明します。Ultra Light.NET を使用すると、PC、ハンドヘルド、モバイル、埋め込みデバイスのデータベース・アプリケーションを開発し、これらのデバイスに配備できます。
- **『Ultra Light - M-Business Anywhere プログラミング』** このマニュアルは、Ultra Light for M-Business Anywhere について説明します。Ultra Light for M-Business Anywhere を使用すると、Palm OS、Windows CE、または Windows XP を搭載しているハンドヘルド、モバイル、または埋め込みデバイスに対して Web ベースのデータベース・アプリケーションを開発、配備できます。
- **『Ultra Light - C/C++ プログラミング』** このマニュアルでは、Ultra Light C および Ultra Light C++ のプログラミング・インタフェースについて説明します。Ultra Light を使用すると、ハンドヘルド、モバイル、埋め込みデバイスに対してデータベース・アプリケーションを開発、配備できます。

マニュアルの形式

SQL Anywhere のマニュアルは、次の形式で提供されています。

- **オンライン・マニュアル** オンライン・マニュアルには、SQL Anywhere の完全なマニュアルがあり、SQL Anywhere ツールに関する印刷マニュアルとコンテキスト別のヘルプの両方が含

まれています。オンライン・マニュアルは、製品のメンテナンス・リリースごとに更新されます。これは、最新の情報を含む最も完全なマニュアルです。

Windows オペレーティング・システムでオンライン・マニュアルにアクセスするには、[スタート] − [プログラム] − [SQL Anywhere 10] − [オンライン・マニュアル] を選択します。オンライン・マニュアルをナビゲートするには、左ウィンドウ枠で HTML ヘルプの目次、索引、検索機能を使用し、右ウィンドウ枠でリンク情報とメニューを使用します。

UNIX オペレーティング・システムでオンライン・マニュアルにアクセスするには、SQL Anywhere のインストール・ディレクトリまたはインストール CD に保存されている HTML マニュアルを参照してください。

- **PDF ファイル** SQL Anywhere の完全なマニュアル・セットは、Adobe Reader で表示できる Adobe Portable Document Format (pdf) 形式のファイルとして提供されています。

  Windows では、PDF 形式のマニュアルはオンライン・マニュアルの各ページ上部にある PDF のリンクから、または Windows の [スタート] メニュー ([スタート] − [プログラム] − [SQL Anywhere 10] − [オンライン・マニュアル - PDF フォーマット]) からアクセスできます。

  UNIX では、PDF 形式のマニュアルはインストール CD にあります。

表記の規則

この項では、このマニュアルで使用されている書体およびグラフィック表現の規則について説明します。

SQL 構文の表記規則

SQL 構文の表記には、次の規則が適用されます。

- **キーワード**　SQL キーワードはすべて次の例に示す ALTER TABLE のように大文字で表記します。

  ALTER TABLE [*owner.*]*table-name*

- **プレースホルダ**　適切な識別子または式で置き換えられる項目は、次の例に示す *owner* や *table-name* のように表記します。

  ALTER TABLE [*owner.*]*table-name*

- **繰り返し項目**　繰り返し項目のリストは、次の例に示す *column-constraint* のように、リストの要素の後ろに省略記号 (ピリオド 3 つ …) を付けて表します。

  ADD *column-definition* [*column-constraint*, …]

  複数の要素を指定できます。複数の要素を指定する場合は、各要素間をカンマで区切る必要があります。

- **オプション部分**　文のオプション部分は角カッコで囲みます。

  RELEASE SAVEPOINT [*savepoint-name*]

  この例では、角カッコで囲まれた *savepoint-name* がオプション部分です。角カッコは入力しないでください。

- **オプション**　項目リストから 1 つだけ選択する場合や、何も選択しなくてもよい場合は、項目間を縦線で区切り、リスト全体を角カッコで囲みます。

  [**ASC** | **DESC**]

  この例では、ASC と DESC のどちらか 1 つを選択しても、選択しなくてもかまいません。角カッコは入力しないでください。

- **選択肢**　オプションの中の 1 つを必ず選択しなければならない場合は、選択肢を中カッコで囲み、縦棒で区切ります。

  [**QUOTES** { **ON** | **OFF** }]

  QUOTES オプションを使用する場合は、ON または OFF のどちらかを選択する必要があります。角カッコと中カッコは入力しないでください。

オペレーティング・システムの表記規則

- **Windows**　デスクトップおよびラップトップ・コンピュータ用の Microsoft Windows オペレーティング・システムのファミリのことです。Windows ファミリには Windows Vista や Windows XP も含まれます。
- **Windows CE**　Microsoft Windows CE モジュラ・オペレーティング・システムに基づいて構築されたプラットフォームです。Windows Mobile や Windows Embedded CE などのプラットフォームが含まれます。

  Windows Mobile は Windows CE 上に構築されています。これにより、Windows のユーザ・インタフェースや、Word や Excel といったアプリケーションの小規模バージョンなどの追加機能が実現されています。Windows Mobile は、モバイル・デバイスで最も広く使用されています。

  SQL Anywhere の制限事項や相違点は、基盤となっているオペレーティング・システム (Windows CE) に由来しており、使用しているプラットフォーム (Windows Mobile など) に依存していることはほとんどありません。
- **UNIX**　特に記述がないかぎり、UNIX は Linux プラットフォームと UNIX プラットフォームの両方のことです。

ファイルの命名規則

マニュアルでは、パス名やファイル名などのオペレーティング・システムに依存するタスクと機能を表すときは、通常 Windows の表記規則が使用されます。ほとんどの場合、他のオペレーティング・システムで使用される構文に簡単に変換できます。

- **ディレクトリ名とパス名**　マニュアルでは、ドライブを示すコロンや、ディレクトリの区切り文字として使用する円記号など、Windows の表記規則を使用して、ディレクトリ・パスのリストを示します。次に例を示します。

  ```
  MobiLink¥redirector
  ```

  UNIX、Linux、Mac OS X では、代わりにスラッシュを使用してください。次に例を示します。

  ```
  MobiLink/redirector
  ```

  SQL Anywhere がマルチプラットフォーム環境で使用されている場合、プラットフォーム間でのパス名の違いに注意する必要があります。
- **実行ファイル**　マニュアルでは、実行ファイルの名前は、Windows の表記規則が使用され、拡張子 *.exe* が付きます。UNIX、Linux、Mac OS X では、実行ファイルの名前には拡張子は付きません。NetWare では、実行ファイルの名前には、拡張子 *.nlm* が付きます。

  たとえば、Windows では、ネットワーク・データベース・サーバは *dbsrv10.exe* です。UNIX、Linux、Mac OS X では、*dbsrv10* になります。NetWare では、*dbsrv10.nlm* になります。
- **install-dir**　インストール・プロセスでは、SQL Anywhere をインストールするロケーションを選択できます。マニュアルでは、このロケーションは *install-dir* という表記で示されます。

インストールが完了すると、環境変数 SQLANY10 によって SQL Anywhere コンポーネントがあるインストール・ディレクトリのロケーション (*install-dir*) が指定されます。SQLANYSH10 は、SQL Anywhere が他の Sybase アプリケーションと共有しているコンポーネントがあるディレクトリのロケーションを指定します。

オペレーティング・システム別の *install-dir* のデフォルト・ロケーションの詳細については、「SQLANY10 環境変数」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **samples-dir** インストール・プロセスでは、SQL Anywhere に含まれるサンプルをインストールするロケーションを選択できます。マニュアルでは、このロケーションは *samples-dir* という表記で示されます。

  インストールが完了すると、環境変数 SQLANYSAMP10 によってサンプルがあるディレクトリのロケーション (*samples-dir*) が指定されます。Windows の [スタート] メニューから、[プログラム] – [SQL Anywhere 10] – [サンプル・アプリケーションおよびプロジェクト] を選択すると、このディレクトリで [Windows エクスプローラ] ウィンドウが表示されます。

  オペレーティング・システム別の *samples-dir* のデフォルト・ロケーションの詳細については、「サンプル・ディレクトリ」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **環境変数** マニュアルでは、環境変数設定が引用されます。Windows では、環境変数を参照するのに、構文 *%envvar%* が使用されます。UNIX、Linux、Mac OS X では、環境変数を参照するのに、構文 *$envvar* または *${envvar}* が使用されます。

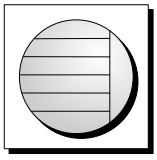
  UNIX、Linux、Mac OS X 環境変数は、*.cshrc* や *.tcshrc* などのシェルとログイン・スタートアップ・ファイルに格納されます。

グラフィック・アイコン

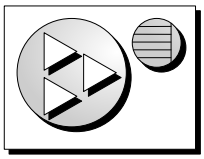
このマニュアルでは、次のアイコンを使用します。

- クライアント・アプリケーション

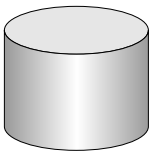
- SQL Anywhere などのデータベース・サーバ



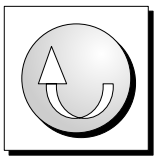
- Ultra Light アプリケーション



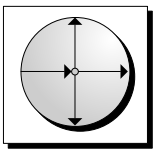
- データベース。高度な図では、データベースとデータベースを管理するデータ・サーバの両方をこのアイコンで表します。



- レプリケーションまたは同期のミドルウェア。ソフトウェアのこれらの部分は、データベース間のデータ共有を支援します。たとえば、Mobile Link サーバ、SQL Remote Message Agent などが挙げられます。



- Sybase Replication Server



- プログラミング・インタフェース

詳細情報の検索／フィードバックの提供

詳細情報の検索

詳しい情報やリソース (コード交換など) については、iAnywhere Developer Network (http://www.ianywhere.com/developer/) を参照してください。

ご質問がある場合や支援が必要な場合は、次に示す Sybase iAnywhere ニュースグループのいずれかにメッセージをお寄せください。

ニュースグループにメッセージをお送りいただく際には、ご使用の SQL Anywhere バージョンのビルド番号を明記し、現在発生している問題について詳しくお知らせくださいますようお願いいたします。バージョン情報は、コマンド・プロンプトで **dbeng10 -v** と入力して確認できます。

ニュースグループは、ニュース・サーバ *forums.sybase.com* にあります (ニュースグループにおけるサービスは英語でのみの提供となります)。以下のニュースグループがあります。

- sybase.public.sqlanywhere.general
- sybase.public.sqlanywhere.linux
- sybase.public.sqlanywhere.mobilink
- sybase.public.sqlanywhere.product_futures_discussion
- sybase.public.sqlanywhere.replication
- sybase.public.sqlanywhere.ultralite
- ianywhere.public.sqlanywhere.qanywhere

ニュースグループに関するお断り
iAnywhere Solutions は、ニュースグループ上に解決策、情報、または意見を提供する義務を負うものではありません。また、システム・オペレータ以外のスタッフにこのサービスを監視させて、操作状況や可用性を保証する義務もありません。
iAnywhere のテクニカル・アドバイザとその他のスタッフは、時間のある場合にかぎりニュースグループでの支援を行います。こうした支援は基本的にボランティアで行われるため、解決策や情報を定期的に提供できるとはかぎりません。支援できるかどうかは、スタッフの仕事量に左右されます。

フィードバック

このマニュアルに関するご意見、ご提案、フィードバックをお寄せください。

マニュアルに関するご意見、ご提案は、SQL Anywhere ドキュメンテーション・チームの iasdoc@ianywhere.com 宛てに電子メールでお寄せください。このアドレスに送信された電子メールに返信はいたしませんが、お寄せいただいたご意見、ご提案は必ず読ませていただきます。

マニュアルまたはソフトウェアについてのフィードバックは、上記のニュースグループを通してお寄せいただいてもかまいません。

パート I. QAnywhere アプリケーションの作成

パート I では、QAnywhere の設定方法とクライアント・アプリケーションの作成方法について説明します。

第 1 章

QAnywhere の概要

目次

QAnywhere アプリケーション間メッセージング

QAnywhere は、モバイル・ユーザ向けの総合的なアプリケーション間メッセージング・システムです。さまざまなデバイスの Windows または Windows CE オペレーティング・システムで動作しているリモート・アプリケーションとメッセージを交換するアプリケーションを作成するときに、インフラストラクチャとして使用できます。

アプリケーション間メッセージングを使用すると、モバイル・デバイスや無線デバイスで動作しているカスタム・プログラムと、集中管理されているサーバ・アプリケーションとの間で通信できます。QAnywhere メッセージングは、次のような場合に便利です。

- モバイル・デバイス間やモバイル・デバイスとエンタープライズ間でのメッセージングの実現
- 随時接続環境での通信の実現

  メッセージングは蓄積転送機能を備えています。つまり、送信先アプリケーションにネットワークを介して接続できない場合でも、メッセージを作成しておくと、後でネットワークが使用可能になったときにそのメッセージが配信されます。

  QAnywhere メッセージは中央のサーバを経由して交換されるので、メッセージが交換されるとき、送信者と受信者が同時にネットワークに接続している必要はありません。
- ネットワークに依存しない通信の実現

  QAnywhere メッセージは、TCP/IP、HTTP、HTTPS の各プロトコルを使用して転送できます。また、ActiveSync を使用して、Windows CE ハンドヘルド・デバイスから転送することもできます。QAnywhere メッセージ自体はネットワーク・プロトコルに依存しないので、受信側アプリケーションは、通信に使用しているネットワークが送信側と同じでなくても、QAnywhere メッセージを受信できます。

QAnywhere は、接続速度が遅い、場所によっては接続できないことがある、接続が切断されるなどの無線ネットワーク特有の問題に対応しています。また、パブリック・ネットワークを介して送信されるすべてのメッセージを暗号化して、機密情報を保護します。転送ルールを使用してメッセージ配信をカスタマイズすれば、ピーク時以外の時間にメッセージが配信されるようにすることなどもできます。

QAnywhere は、モバイル・アプリケーションとエンタープライズ・サーバの間で送信されるデータを圧縮し、暗号化します (暗号化はオプション)。さらに、蓄積転送メッセージング・パラダイムが実装されているので、メッセージの確実な配信を保証できます。

QAnywhere は、さまざまなハンドヘルド・デバイスを使用したメッセージング・ソリューションを実現できるように設計されています。プログラミング・インタフェースとして QAnywhere C++、Java、.NET、SQL API が用意されているので、開発者は自分のスキルに応じてソリューションを選択できます。

QAnywhere を使用すると、JMS インタフェースを備えた他のメッセージング・システムとシームレスに通信できます。これによって、J2EE アプリケーションとの統合が可能になります。

QAnywhere の機能

QAnywhere は、以下のアプリケーション間通信機能とコンポーネントで構成されています。

- **QAnywhere API**　オブジェクト指向の QAnywhere API は、Windows デスクトップ向けや Windows CE デバイス向けのメッセージング・アプリケーションを構築するためのインフラストラクチャとして使用できます。QAnywhere API は Java、C++、.NET、SQL で使用できます。
- **蓄積転送**　QAnywhere アプリケーションは、クライアントとサーバの間の接続が確立されてデータ転送が可能になるまで、ローカルにメッセージを格納しておきます。
- **データ同期の補完**　QAnywhere アプリケーションは、リレーショナル・データベースを一時的なメッセージ・ストアとして使用します。リレーショナル・データベースを使用することで、メッセージ・ストアのセキュリティ保護、トランザクション・ベースのコンピューティング、リレーショナル・データベースのその他の利点が得られます。

  SQL Anywhere リレーショナル・データベースをメッセージ・ストアとして使用すると、QAnywhere とデータ同期ソリューションを容易に組み合わせることができます。これは、どちらも Mobile Link 同期を使用して、クライアント／サーバ間情報交換の基本メカニズムを実現しているからです。
- **外部メッセージング・システムとの統合**　QAnywhere アプリケーション間でメッセージ交換を実現できるだけでなく、JMS インタフェースをサポートする外部メッセージング・システムに QAnywhere クライアントを統合することもできます。
- **暗号化**　メッセージは、トランスポート・レイヤ・セキュリティを使用して暗号化して送信できます。また、単純暗号化や FIPS 承認の AES アルゴリズムを使用してメッセージ・ストアを暗号化することもできます。
- **圧縮**　メッセージの内容は、一般的な ZLIB 圧縮ライブラリを使用して圧縮してから格納できます。
- **認証**　QAnywhere クライアントの認証は、組み込みの機能を使用したり、組織で使用している認証サーバなどのカスタム認証スクリプトを使用したりして実行できます。
- **複数ネットワーク対応**　QAnywhere は、TCP/IP または HTTP をサポートしている有線ネットワークまたは無線ネットワークのすべてで動作します。
- **フェールオーバ**　複数の Mobile Link サーバを実行し、いずれかのサーバで障害が発生したときに代替サーバに処理が引き継がれるようにすることができます。
- **管理**　QAnywhere アプリケーションは、クライアント側とサーバ側でメッセージの参照と操作を実行できます。
- **複数のキュー**　クライアント・デバイス上で任意の名前のキューを複数使用できるので、1 つのデバイスで複数のクライアント・アプリケーションを実行できます。アプリケーションは、任意の数のキューを使用してメッセージを送受信できます。メッセージは、同一デバイスで動作しているアプリケーションの間だけでなく、それぞれ異なるデバイスで動作しているアプリケーションの間でも交換できます。

- **サーバによって開始されるメッセージの送受信**　QAnywhere を使用すると、クライアント・デバイスにメッセージをプッシュできるので、クライアント・アプリケーションにメッセージ駆動型のロジックを実装できます。
- **転送ルール**　転送ルールを作成すると、メッセージの転送を実行するタイミングを指定できます。
- **再開可能なダウンロード**　サイズの大きなメッセージやまとまった量のメッセージがある場合、それらのメッセージは少しずつ QAnywhere クライアントに送信されます。これによって、ネットワーク障害が発生した場合のデータの再転送が最小限に抑えられます。
- **配信の保証**　QAnywhere を使用すると、メッセージが確実に配信されます。
- **モバイル Web サービス**　モバイル Web サービスは、QAnywhere 上の Web サービス要求と応答の転送を容易にします。

QAnywhere アーキテクチャ

この項では、QAnywhere メッセージング・アプリケーションのアーキテクチャについて説明します。まず、簡単なメッセージング・シナリオを取り上げ、次に、より高度なシナリオについて解説します。

クライアント・アプリケーションは、QAnywhere API を使用してメッセージの送受信を行います。メッセージは、クライアント・メッセージ・ストア内にキューイングされます。メッセージ転送とは、クライアント・メッセージ・ストアと、中央の QAnywhere サーバ・メッセージ・ストアとの間で、メッセージをやり取りすることを指します。

QAnywhere でサポートされている典型的なメッセージング・シナリオを以下に示します。

- **簡単なメッセージング**　QAnywhere クライアント間でメッセージを交換する場合です。クライアント・アプリケーションが、クライアント・メッセージ・ストアとサーバ・メッセージ・ストアの間でメッセージを転送するタイミングを制御します。

  「簡単なメッセージング・シナリオ」 7 ページを参照してください。

- **Push 通知によるメッセージング**　QAnywhere クライアント間でメッセージを交換する場合です。このシナリオでは、Mobile Link サーバがクライアント間のメッセージ転送を開始できます。これは、クライアントとサーバのメッセージ・ストア間でメッセージを交換することによって行われます。

  「Push 通知によるメッセージングのシナリオ」 9 ページを参照してください。

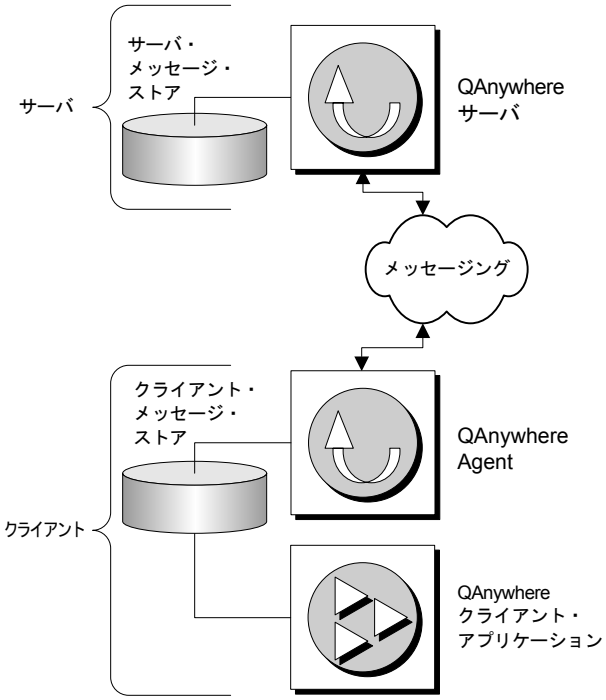
- **外部メッセージング・システムとのメッセージング**　JMS プロバイダを提供する外部システム (BEA WebLogic や Sybase EAServer など) を介して QAnywhere クライアント間でメッセージ交換を行う場合です。

  「外部メッセージング・システムとのメッセージングのシナリオ」 10 ページを参照してください。

Push 通知と外部メッセージング・システムを組み合わせて使用すると、最も汎用的なソリューションを実現できます。

簡単なメッセージング・シナリオ

次の図は、簡単な QAnywhere メッセージングの設定を表したものです。わかりやすくするため、クライアントを 1 つだけにしています。ただし、典型的なシナリオでは複数のクライアントが存在し、サーバ・メッセージ・ストアによってそれらの間でメッセージが転送されます。



このアーキテクチャは、以下のコンポーネントで構成されています。

- **サーバ・メッセージ・ストア**　サーバ側ではメッセージがリレーショナル・データベースに格納されます。このデータベースは、Mobile Link 統合データベースとして設定する必要があります。サポートされている統合データベースのいずれか (SQL Anywhere、Adaptive Server Enterprise、Microsoft SQL Server、DB2、または Oracle) を使用できます。
- **クライアント・メッセージ・ストア**　各クライアントのメッセージもリレーショナル・データベースに格納されます。この SQL Anywhere データベースは QAnywhere メッセージ専用です。
- **QAnywhere サーバ**　QAnywhere サーバは、メッセージングが有効になっている Mobile Link サーバです。Mobile Link 同期は、QAnywhere クライアントと QAnywhere サーバの間で、メッセージの転送や追跡のための手段として使用されます。Mobile Link は、セキュリティ、認証、暗号化、柔軟性を実現します。また、Mobile Link が使用されている場合、メッセージングとデータ同期を組み合わせることができます。

  QAnywhere サーバを起動するには、-m オプションを指定して Mobile Link サーバを起動します。「QAnywhere サーバの起動」 33 ページを参照してください。
- **QAnywhere Agent**　QAnywhere Agent は、クライアント側のメッセージ送信を管理します。この処理は QAnywhere クライアント・アプリケーションから独立しています。

  「QAnywhere Agent の実行」 38 ページを参照してください。
- **QAnywhere クライアント・アプリケーション**　QAnywhere C++、Java、または .NET API を使用して作成されたアプリケーションから、メッセージを送受信するためのメソッドが呼び出

されます。クライアント・アプリケーションで使用される基本オブジェクトは QAManager です。

「QAnywhere クライアント・アプリケーションの作成」 51 ページを参照してください。

メッセージの送信と受信は、QAnywhere クライアントによって開始されます。サーバ側のメッセージは、クライアントがメッセージ転送を開始するまで取り出されません。QAnywhere は、**ポリシー**に基づいて、メッセージ転送を実行するタイミングを決定します。ポリシーには、オンデマンド、自動、スケジュール済み、カスタムがあります。オンデマンド・ポリシーでは、ユーザがメッセージ転送を制御できます。自動ポリシーでは、クライアントとの間でメッセージの配信準備が行われるたびにメッセージの転送が開始されます。

「クライアントにメッセージを転送するタイミングの決定」 40 ページを参照してください。

Push 通知によるメッセージングのシナリオ

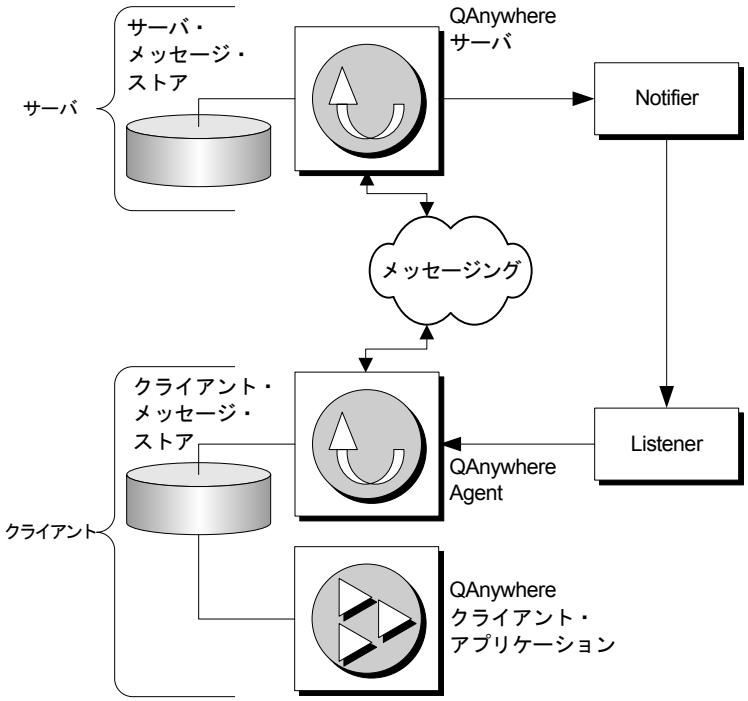
Push 通知は、サーバから QAnywhere クライアントに配信される特殊なメッセージです。Push 通知は、メッセージがサーバ・メッセージ・ストアに着信したときに送信されます。メッセージング・サーバは、受信側クライアントの Listener に Push 要求を自動的に通知します。クライアントは通知を受信するとメッセージ転送を開始し、サーバで待機しているメッセージを受信したり、カスタム・アクションを実行したりします。

Push 通知に対するクライアントの応答の詳細については、「クライアントにメッセージを転送するタイミングの決定」 40 ページを参照してください。

Push 通知を使用できるようにするため、QAnywhere アーキテクチャには 2 つのコンポーネントが追加されています。サーバ側では、QAnywhere Notifier が Push 通知を送信します。クライアント側では、QAnywhere Listener が Push 通知を受け取り、QAnywhere Agent に渡します。

Push 通知を使用しなくてもメッセージはサーバ・メッセージ・ストアからクライアント・メッセージ・ストアに送信されます。ただし、その場合は、スケジュール転送ポリシーなどを使用してクライアント側からメッセージ転送を開始する必要があります。

Push 通知によるメッセージングのアーキテクチャは、「簡単なメッセージング・シナリオ」 7 ページで説明したアーキテクチャを拡張したものです。次の図は、このアーキテクチャを示しています。



Push 通知を使用できるようにするために、次のコンポーネントが「簡単なメッセージング・シナリオ」 7 ページに追加されています。

- **QAnywhere Notifier** QAnywhere Notifier は Mobile Link サーバのコンポーネントで、Push 通知を配信するために使用されています。

  QAnywhere Notifier は、メッセージの配信準備が完了したら Push 通知を送信するように特別に構成された Notifier インスタンスです。

- **Listener** Listener は、クライアント側で動作する独立のプロセスです。Push 通知を受信して、QAnywhere Agent に渡す役割を果たします。QAnywhere Agent ポリシーは、Push 通知の受信によりメッセージ転送が自動的に行われるかどうかを決定します。

  「クライアントにメッセージを転送するタイミングの決定」 40 ページを参照してください。

参照

- 「Push 通知の使用方法」 45 ページ
- 「非同期的なメッセージ受信」 83 ページ
- 「サーバ起動同期の概要」 『Mobile Link - サーバ起動同期』

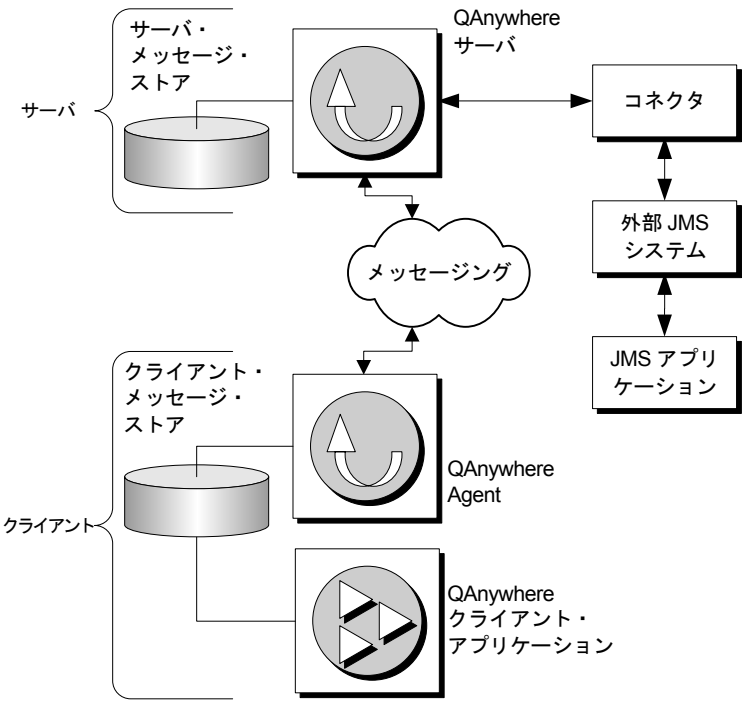
外部メッセージング・システムとのメッセージングのシナリオ

コネクタと呼ばれる特別に構成されたクライアントを使用すると、QAnywhere アプリケーション間でのメッセージ交換だけでなく、JMS インタフェースを備えたシステムとのメッセージ交換

も可能になります。JMS とは、Java アプリケーションにメッセージング機能を追加する役割を果たす、Java Message Service API のことです。

外部メッセージング・システムは、特別なクライアントとして動作するように設定されます。これには、固有のアドレスと構成を使用します。

外部メッセージング・システムを含むメッセージング・アーキテクチャは、「簡単なメッセージング・シナリオ」 7 ページで説明したアーキテクチャを拡張したものです。次の図は、このアーキテクチャを示しています。



外部メッセージング・システムとのメッセージングを可能にするために「簡単なメッセージング・シナリオ」 7 ページに追加されたコンポーネントは、以下のとおりです。

- **QAnywhere JMS コネクタ**　JMS コネクタは、QAnywhere と外部メッセージング・システムの間のインタフェースを提供します。

  JMS コネクタは、QAnywhere と外部 JMS システムの間でメッセージ転送を行うための特殊な QAnywhere クライアントです。

参照

- 「JMS コネクタ」 135 ページ
- 「チュートリアル : JMS コネクタの使用」 152 ページ

QAnywhere メッセージ配信

メッセージはクライアント・メッセージ・ストアからサーバ・メッセージ・ストアに送信されてから、別のクライアント・メッセージ・ストアに送信されます。QAnywhere ではキューを使用してこれを行います。メッセージはまず、クライアント・メッセージ・ストアのキューに登録されます。次にサーバ・メッセージ・ストアで受信されると、1 つまたは複数のクライアント・メッセージ・ストアに配信するためのキューに登録されます。最後にクライアント・メッセージ・ストアで受信されると、取り出し用のキューに登録されます。

一度送信されたメッセージは、次のいずれかの場合を除き、確実に配信されます。

- メッセージの有効期限が切れた場合 (有効期限が指定されている場合のみ)
- Sybase Center からメッセージがキャンセルされた場合
- メッセージを送信したデバイスが、サーバ・メッセージ・ストアと同期する前に失われてリカバリ不能になった場合、または何らかの理由により同期できない場合

メッセージが配信されるのは 1 度だけです。アプリケーションによって受信が正常に確認またはコミットされたメッセージは、もう一度配送することはできません。ただし、JMS サーバの場合は例外があり、Mobile Link サーバまたは JMS サーバがクラッシュした場合は、メッセージが 2 度配信される可能性があります。

QAnywhere プラグイン

Sybase Central QAnywhere プラグインを使用して、QAnywhere アプリケーションの作成や管理が行えます。このプラグインでは、次の作業を実行できます。

- クライアント・メッセージ・ストアとサーバ・メッセージ・ストアの作成
- QAnywhere Agent の設定ファイルの作成と管理
- QAnywhere Agent のログ・ファイルのブラウズ
- 送信先エイリアスの作成または変更
- JMS コネクタと Web サービス・コネクタの作成
- 転送ルール・ファイルの作成と管理
- メッセージ・ストアのリモートでのブラウズ
- メッセージの追跡

- **QAnywhere プラグインを起動するには、次の手順に従います。**

1. Sybase Central を起動します。

   [スタート] – [プログラム] – [SQL Anywhere 10] – [Sybase Central] を選択します。
2. [接続] メニューから、[QAnywhere 10 に接続] を選択します。
3. ODBC データ・ソース名または ODBC データ・ソース・ファイルを指定し、必要に応じてユーザ ID とパスワードを指定します。
4. [OK] をクリックします。

QAnywhere のクイック・スタート

次の手順には、QAnywhere メッセージングの設定と実行に必要なタスクの概要が記載されています。

♦ **QAnywhere メッセージングを設定して起動するには、次の手順に従います。**

1. サーバ・メッセージ・ストアを設定します。新しく設定しない場合は、既存の Mobile Link 統合データベースを使用する必要があります。

   「サーバ・メッセージ・ストアの設定」 32 ページを参照してください。

2. -m オプションを指定して Mobile Link サーバを起動し、サーバ・メッセージ・ストアへの接続を開始します。

   「QAnywhere サーバの起動」 33 ページを参照してください。

3. クライアント・メッセージ・ストアを設定します。このメッセージ・ストアは、メッセージを一時的に格納するための SQL Anywhere データベースです。

   「クライアント・メッセージ・ストアの設定」 36 ページを参照してください。

4. クライアントごとに、メッセージング・アプリケーションを作成します。

   「QAnywhere クライアント・アプリケーションの作成」 51 ページを参照してください。

5. 外部 JMS メッセージング・システムを統合する場合は、QAnywhere 用に JMS メッセージングを設定します。

   「JMS コネクタ」 135 ページを参照してください。

6. クライアントごとに、ローカルのクライアント・メッセージ・ストアに接続する QAnywhere Agent を起動します。

   「QAnywhere Agent の実行」 38 ページを参照してください。

モバイル Web サービスの設定の詳細については、「モバイル Web サービス」 195 ページを参照してください。

クイック・スタートのためのその他の資料

- 「チュートリアル：TestMessage の解説」 15 ページ
- 「チュートリアル：JMS コネクタの使用」 152 ページ
- *samples-dir¥QAnywhere* にサンプル・アプリケーションがあります。*samples-dir* の詳細については、「サンプル・ディレクトリ」 『SQL Anywhere サーバ - データベース管理』を参照してください。

第 2 章

チュートリアル：TestMessage の解説

目次

チュートリアルの概要

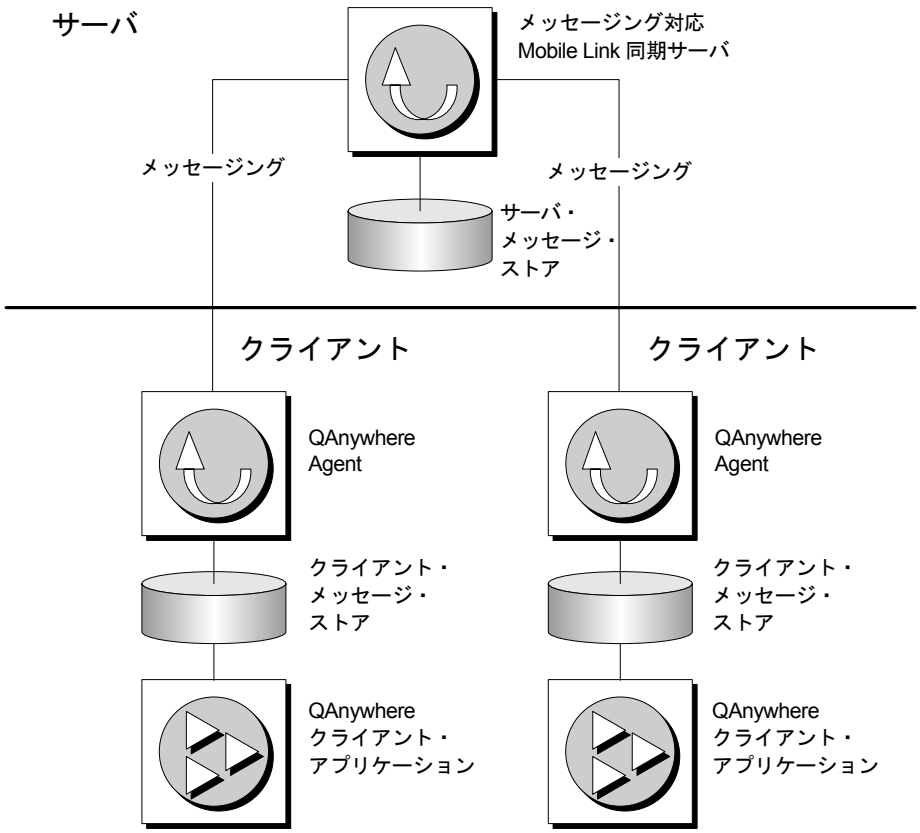
TestMessage は QAnywhere クライアント・アプリケーションのサンプルです。このアプリケーションは、QAnywhere を使用してユーザ独自のメッセージ・クライアント・アプリケーションを作成する方法を示しています。TestMessage では、単一のクライアント間インタフェースを使用して、メッセージを送信、受信、表示します。このサンプルでは、QAnywhere メッセージングの機能をわかりやすく示すため、人間が読めるテキスト・メッセージを使用しています。ただし、QAnywhere はテキスト以外のメッセージも扱うことができます。QAnywhere は、多数のクライアントの間でメッセージ・ベースの通信を実現できる、汎用のアプリケーション間メッセージング・システムです。

このチュートリアルは、Windows デスクトップ・システムを対象としています。これらのプラットフォームはデモ用に便利です。QAnywhere API は、Windows CE デバイスで動作するアプリケーションの作成にも使用できます。Windows CE で C++、Visual Basic .NET、C#、Java に使用できるソース・コードが提供されます。.NET Compact Framework を使用して作成された C# バージョンもあります。

レッスン 1：メッセージング対応 Mobile Link の起動

背景

QAnywhere は、Mobile Link 同期を使用してメッセージを送受信します。クライアント間でやり取りされるメッセージは、いずれも中央の Mobile Link サーバを介して配信されます。典型的なシステムのアーキテクチャを次の図に示します。このアーキテクチャでは、QAnywhere クライアントが 2 つだけ使用されています。



サーバ・メッセージ・ストアは、Mobile Link 統合データベースとして使用できるように構成されているデータベースです。TestMessage では、サーバ・メッセージ・ストアとして SQL Anywhere 統合データベースを使用します。

サーバ・メッセージ・ストアに必要なテーブルは Mobile Link システム・テーブルだけです。Mobile Link システム・テーブルは、Mobile Link 統合データベースとして設定された、サポートされているデータベースすべてに含まれています。

これらのシステム・テーブルは、Mobile Link によって管理されます。リレーショナル・データベースを使用すると、安全で高性能なメッセージ・ストアを実現できます。また、メッセージング機能を既存のデータ管理／同期システムに容易に統合できます。

通常、QAnywhere メッセージングは、複数の異なるコンピュータにまたがって実行されます。ただし、このチュートリアルでは、すべてのコンポーネントが 1 つのコンピュータで実行されます。サーバ上のアクティビティとクライアント上のアクティビティを混同しないように注意してください。

このレッスンでは、サーバ側でアクションを実行します。

アクティビティ

Mobile Link サーバは、-m オプションと、サーバ・メッセージ・ストアに接続するための接続文字列を指定して起動すると、メッセージングに対応した状態で起動できます。TestMessage では、サーバ・メッセージ・ストアとして QAnywhere サンプル・データベースを使用します。TestMessage サンプル・アプリケーションを使用するときには、コマンド・ライン・オプションを指定するか、SQL Anywhere インストール環境にあるサンプル・ショートカットをクリックするか、Sybase Central 用の QAnywhere プラグインを使用すると、Mobile Link サーバをメッセージングに対応した状態で起動できます。

♦ メッセージング・サーバを起動する

1. Windows で、[スタート] – [プログラム] – [SQL Anywhere 10] – [Mobile Link] – [メッセージ・サンプル付き Mobile Link] を選択します。

   また、コマンド・プロンプトで *samples-dir¥QAnywhere¥server* に移動し、次のコマンドを入力することもできます。

   ```
   mlsrv10 -m -c "dsn=QAnywhere 10 Demo" -vcrs -zu+
   ```

   このコマンド・ラインでは、以下に示す mlsrv10 のオプションを使用します。

オプション	説明
-m	メッセージングを有効にします。「-m オプション」 『Mobile Link - サーバ管理』を参照してください。
-c	サーバ・メッセージ・ストアに接続するための接続文字列を指定します。この例では、ODBC データ・ソース QAnywhere 10.0 Demo が接続文字列に含まれています。「-c オプション」 『Mobile Link - サーバ管理』を参照してください。
-vcrs	サーバのアクティビティに関する冗長ロギングを有効にします。開発時に指定すると便利です。「-v オプション」 『Mobile Link - サーバ管理』を参照してください。
-zu+	ユーザ名を自動的にシステムに追加します。このオプションは、チュートリアルや開発用に使用すると便利ですが、運用環境では通常使用しません。「-zu オプション」 『Mobile Link - サーバ管理』を参照してください。

2. Mobile Link サーバ・ウィンドウを画面中央に移動します。これは、このチュートリアルのサーバになります。

Mobile Link サーバが起動したら、次のレッスンに進んでください。

詳細情報

- 「QAnywhere サーバの起動」 33 ページ
- 「-m オプション」 『Mobile Link - サーバ管理』
- 「QAnywhere のクイック・スタート」 14 ページ
- 「簡単なメッセージング・シナリオ」 7 ページ

レッスン 2：TestMessage アプリケーションの実行

背景

TestMessage は、QAnywhere を使用してテキスト・メッセージを送受信する簡単なアプリケーションです。このチュートリアルでテキスト・メッセージを使用するのは、メッセージングのデモ用として簡単で理解しやすいからです。実際には、QAnywhere は、単なるテキスト・メッセージング・システムではなく、汎用のアプリケーション間メッセージングを実現できるシステムです。

このレッスンでは、クライアント側でアクティビティを行います。通常、クライアントはサーバとは別のコンピュータ上で動作します。

このレッスンでは、TestMessage サンプルに含まれているクライアント・メッセージ・ストアを開始します。レッスン 3 では、このメッセージ・ストアを使用して、メッセージを別のクライアント・メッセージ・ストアに送信します。

アクティビティ

♦ QAnywhere Agent を起動し TestMessage のクライアント・メッセージ・ストアを開始する

1. Windows で、[スタート] – [プログラム] – [SQL Anywhere 10] – [QAnywhere] – [Client1 用エージェント・サンプル] を選択します。

   QAnywhere Agent のインスタンスが起動されます。この Agent は TestMessage の最初のサンプル・クライアント・メッセージ・ストアに接続し、そのメッセージ・ストア間とのメッセージ転送を管理します。

2. 最初の QAnywhere Agent ウィンドウを画面右側に移動します。これは、このチュートリアルの最初のクライアントになります。

3. Windows で、[スタート] – [プログラム] – [SQL Anywhere 10] – [QAnywhere] – [Client2 用エージェント・サンプル] を選択します。

   QAnywhere Agent の別のインスタンスが起動されます。この Agent は TestMessage の 2 番目のサンプル・クライアント・メッセージ・ストアに接続し、そのメッセージ・ストア間とのメッセージ転送を管理します。

4. 2 番目の QAnywhere Agent ウィンドウを画面左側に移動します。これは、このチュートリアルの 2 番目のクライアントになります。

5. 各 QAnywhere Agent ウィンドウに、client1 と client2 というクライアント・メッセージ・ストア ID が表示されます。

♦ TestMessage を開始する

1. Windows で、[スタート] – [プログラム] – [SQL Anywhere 10] – [QAnywhere] – [Client1 用 TestMessage サンプル] を選択します。

   TestMessage のウィンドウが表示されます。このアプリケーションは、前の手順で開始した TestMessage の最初のクライアント・メッセージ・ストアに接続されます。

2. TestMessage のウィンドウを、最初の QAnywhere Agent と同じように、画面右側に移動します。これらは、両方とも最初のクライアント上のコンポーネントです。
3. メッセージ・キューを確認します。

   TestMessage サンプル・アプリケーションで、クライアント 1 の [Tools] − [Options] を選択します。キュー名として **testmessage** と指定されていることが確認できます。このキューを使用して、TestMessage アプリケーションが着信メッセージを受信します。この名前は変更しないでください。
4. Windows で、[スタート] − [プログラム] − [SQL Anywhere 10] − [QAnywhere] − [Client2 用 TestMessage サンプル] を選択します。

   TestMessage のウィンドウが表示されます。このアプリケーションは、前の手順で開始した TestMessage の 2 番目のクライアント・メッセージ・ストアに接続されます。
5. TestMessage のウィンドウを、2 番目の QAnywhere Agent と同じように、画面左側に移動します。これらは、両方とも 2 番目のクライアント上のコンポーネントです。
6. メッセージ・キューを確認します。

   TestMessage サンプル・アプリケーションで、クライアント 2 の [Tools] − [Options] を選択します。キュー名として **testmessage** と指定されていることが確認できます。このキューを使用して、TestMessage アプリケーションが着信メッセージを受信します。この名前は変更しないでください。

説明

QAnywhere Agent のメッセージ・モニタリングの方法を設定するには、メッセージ転送ポリシーを設定します。このサンプルでは、自動ポリシーまたはスケジュール済みポリシーだけを使用できます。QAnywhere Agent は自動ポリシーを使用して起動します。QAnywhere のポリシーは次のとおりです。

- **scheduled**　このポリシー設定は、メッセージを定期的に転送するように QAnywhere Agent に指示します。転送間隔が指定されなかった場合、デフォルトの 15 秒間隔になります。
- **automatic**　これはデフォルトのポリシー設定です。QAnywhere Agent は、メッセージがクライアント・メッセージ・ストアとの間で配信できる準備ができた時点で転送を行います。
- **ondemand**　このポリシーを設定すると、QAnywhere Agent はメッセージを転送するようにアプリケーションから指示が出された場合にだけ、転送を行います。
- **custom**　このモードでは、ルール・セットを設定して、より複雑な転送動作を指定できます。

QAnywhere メッセージは QAnywhere アドレス宛に配信されます。QAnywhere アドレスは、クライアント・メッセージ・ストア ID とキュー名から成ります。デフォルトの ID は、QAnywhere Agent が動作しているコンピュータの名前です。メッセージ・ストアごとに、独自の QAnywhere Agent が必要です。各アプリケーションは、複数のキューで受信することができます。ただし、各キューは 1 つのアプリケーションに割り当てられている必要があります。

詳細情報

- 「QAnywhere Agent の実行」 38 ページ
- 「クライアントにメッセージを転送するタイミングの決定」 40 ページ
- 「qaagent 構文」 156 ページ
- 「QAnywhere 転送ルールと削除ルール」 241 ページ
- 「QAnywhere クライアント・アプリケーションの作成」 51 ページ
- *samples-dir¥QAnywhere* に QAnywhere のサンプルがあります。*samples-dir* の詳細については、「サンプル・ディレクトリ」『SQL Anywhere サーバ - データベース管理』を参照してください。

レッスン 3：メッセージの送信

背景

TestMessage サンプル・アプリケーションには、レッスン 1 で開始した 2 つのクライアント・メッセージ・ストアが含まれています。このレッスンでは、TestMessage の client1 アプリケーションから client2 アプリケーションにメッセージを送信します。

アクティビティ

- **TestMessage からメッセージを送信する**

1. TestMessage サンプル・アプリケーションで、client 1 の [Message] – [New] を選択します。[New Message] ウィンドウが表示されます。
2. [Destination ID] フィールドに「client2」と入力します ([Destination Queue] フィールドには testmessage を残します)。
3. [Subject] フィールドと [Massage] フィールドにサンプル・テキストを入力して、[Send] をクリックします。

   メッセージングをテストするときは、現在時刻をメッセージの件名として使用すると、個々のメッセージを追跡しやすくなるので便利です。

   警告が表示されます。
4. メッセージを読みます。

   TestMessage サンプル・アプリケーションの client2 のウィンドウに切り替えます。メッセージを選択すると、その内容が下部にあるウィンドウ枠に表示されます。

説明

他の QAnywhere アプリケーション同様、TestMessage も QAnywhere API を使用してメッセージを管理しています。QAnywhere API は、C++ API、Java API、Microsoft .NET API、SQL API として提供されています。

詳細情報

- 「QAnywhere メッセージ・アドレス」 56 ページ
- 「QAnywhere メッセージの送信」 73 ページ
- 「メッセージの削除ルール」 256 ページ

レッスン 4：TestMessage クライアントのソース・コードの概要

背景

この項では、TestMessage クライアント・アプリケーションのソース・コードの内容について簡単に説明します。

TestMessage クライアント・アプリケーションの大部分は、メッセージを送信、受信、表示するための Windows インタフェースの実装です。しかし、ここでは、QAnywhere の機能を実装しているコードを中心に解説していきます。

TestMessage のソース・コードは、*samples-dir¥QAnywhere* にあります。

TestMessage のソース・コードにはいくつかのバージョンがあります。Windows 2000 と Windows XP 用に次のバージョンが用意されています。

- Microsoft Foundation Classes を使用して作成された C++ バージョンが、*Samples¥QAnywhere ¥Desktop¥MFC¥TestMessage¥TestMessage.sln* として提供されています。
- .NET Framework を使用して作成された Visual Basic .NET バージョンが、*Samples¥QAnywhere ¥Desktop¥.NET¥VB¥TestMessage¥TestMessage.sln* として提供されています。
- .NET Framework を使用して作成された C# バージョンが、*Samples¥QAnywhere¥Desktop¥.NET ¥CS¥TestMessage¥TestMessage.sln* として提供されています。
- Java バージョンが、*Samples¥QAnywhere¥Java¥TestMessage¥TestMessage.java* として提供されています。

.NET Compact Framework については、次のバージョンがあります。

- .NET Compact Framework を使用して作成された C# バージョンが、*Samples¥QAnywhere ¥PocketPC¥.NET¥CS¥TestMessage¥TestMessage.sln* として提供されています。

必要なソフトウェア

ソリューション・ファイルを開いたり、.NET Framework プロジェクトや .NET Compact Framework プロジェクトをビルドしたりする場合は、Visual Studio .NET 2003 以降が必要です。

C# ソースの確認

この項では、C# バージョンのソース・コードについて解説します。C# バージョンと Visual Basic .NET バージョンは、構成が非常によく似ています。

このレッスンでは、アプリケーションのすべての行を順に確認することはしません。QAnywhere アプリケーションの理解に特に役立つ行だけを確認します。解説には C# バージョンのコードを使用します。

1. いずれかのバージョンの TestMessage プロジェクトを開きます。

ソリューション・ファイルをダブルクリックして、Visual Studio .NET 内で目的のプロジェクトを開きます。たとえば、*Samples¥QAnywhere¥Desktop¥.NET¥CS¥TestMessage ¥TestMessage.sln* はソリューション・ファイルです。さまざまな環境向けに複数のソリューション・ファイルが用意されています。

2. ソリューション・エクスプローラが開いていることを確認します。

   ソリューション・エクスプローラは、[表示] メニューから開くことができます。

3. [ソース ファイル] フォルダの内容を確認します。

   特に重要なファイルが 2 つあります。*MessageList* ファイル (*MessageList.cs*) は、メッセージを受信して、表示する機能を実現します。*NewMessage* ファイル (*NewMessage.cs*) は、メッセージの作成と送信を可能にします。

4. ソリューション・エクスプローラで、*MessageList* ファイルを開きます。

5. インクルードされているネームスペースを調べます。

   QAnywhere アプリケーションは、いずれも iAnywhere.QAnywhere.Client ネームスペースをインクルードしている必要があります。このネームスペースを定義しているアセンブリは、DLL ファイル *iAnywhere.QAnywhere.Client.dll* として提供されています。このファイルは、SQL Anywhere のインストール・ディレクトリを基準とした相対ディレクトリに格納されています。

   - .NET Framework 1.1：*¥Assembly¥v1*
   - .NET Framework 2.0：*¥Assembly¥v2*
   - .NET Compact Framework 1.0：*ce¥Assembly¥v1*
   - .NET Compact Framework 2.0：*ce¥Assembly¥v2*

   ユーザ独自のプロジェクトでは、コンパイル時にこの DLL への参照をインクルードする必要があります。このネームスペースは、各ファイルの先頭行で、次のようにインクルードされています。

   ```
   using iAnywhere.QAnywhere.Client;
   ```

6. startReceiver メソッドを調べます。

   MessageList_Load メソッドは、以下に示す、各 QAnywhere アプリケーションに共通する初期化処理を実行します。

   - QAManager オブジェクトを作成する。

     ```
     _qaManager =
     QAManagerFactory.Instance.CreateQAManager( null );
     ```

     QAnywhere には、QAManager オブジェクトを作成するための QAManagerFactory オブジェクトが用意されています。QAManager オブジェクトは、QAnywhere のメッセージング操作を処理します。具体的には、メッセージの受信 (キューからのメッセージの取り出し) とメッセージの送信 (キューへのメッセージの登録) を処理します。

     QAnywhere には、QAManager と QATransactionalManager の 2 種類の Manager が用意されています。QATransactionalManager を使用すると、送信と受信はすべてトランザクショ

ン内で処理されます。したがって、すべてのメッセージが送信 (受信) されるか、何も送信 (受信) されないかのどちらかになります。

- メッセージを処理するメソッドを記述する。

  システム関連メッセージではない通常のメッセージを処理するための onMessage() メソッドが、QAnywhere から呼び出されます。このメソッドが受信するメッセージは、QAMessage オブジェクトとしてエンコードされています。この QAMessage クラスとその子クラス (QATextMessage と QABinaryMessage) のプロパティとメソッドに、QAnywhere アプリケーションが必要とするメッセージ関連情報が格納されています。

```
private void onMessage( QAMessage msg ) {
        Invoke( new onMessageDelegate( onMessageReceived ),
    new Object [] { msg } );
}
```

  このコードは Form の Invoke メソッドを使用して、基本のウィンドウを実行するスレッドでイベントが処理されるようにします。これにより、ユーザ・インタフェースが更新されてメッセージが表示できるようになります。これは、QAManger を作成したスレッドでもあります。一部の例外を除き、QAManager にアクセスできるのは、その QAManager を作成したスレッドだけです。

- MessageListener を宣言する。

```
_receiveListener = new
  QAManager.MessageListener( onMessage );
```

  メッセージが QAnywhere Agent によって受信され、アプリケーションが待機しているキューに登録されると、OnMessage() メソッドが呼び出されます。

> **メッセージ・リスナと通知リスナ**
> メッセージ・リスナは、「Push 通知によるメッセージングのシナリオ」 9 ページで説明した Listener コンポーネントとは異なります。この Listener コンポーネントが通知を受信するのに対し、メッセージ・リスナ・オブジェクトはキューからメッセージを取り出します。

メッセージ・リスナがキューに割り当てられると、QAnywhere Manager は、キューに着信したメッセージを、そのキューに割り当てられているリスナに渡すようになります。1 つのキューに複数のリスナを割り当てることはできません。キューに NULL リスナを割り当てると、そのキューへのリスナの割り当てが解除されます。

MessageListener 実装では、メッセージは非同期に受信されます。メッセージを同期的に受信することもできます。同期的なメッセージ受信では、アプリケーションは、キューにメッセージが着信した時点で通知を受けるのではなく、キューに登録されたメッセージを明示的に (たいていの場合は [再表示] ボタンのクリックなどのユーザ・アクションに応答して) 検索します。

その他の初期化タスクには、次のものがあります。

- QAManager オブジェクトをオープンして開始する。

```
_qaManager.Open(
      AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT );
_qaManager.Start();
```

AcknowledgementMode 列挙定数は、メッセージの受信確認応答を送信元に返す方法を決定します。EXPLICIT_ACKNOWLEDGEMENT 定数は、QAManager のいずれかの受信確認メソッドが呼び出されるまでメッセージが受信確認されないことを意味します。

- キュー内で待ち状態にあるメッセージをすべてロードする。

  ```
  loadMessages();
  ```

- 以降のメッセージが着信するキューにリスナを割り当てる。

  リスナは、MessageList_Load() メソッド内で宣言されています。

  ```
  _qaManager.SetMessageListener(
    _options.ReceiveQueueName,
    _receiveListener );
  ```

  Options の ReceiveQueueName プロパティには、文字列 **testmessage** が含まれています。これは、TestMessage アプリケーションの [Options] ダイアログで設定された TextMessage アプリケーション用のキューです。

7. 同じファイルにある addMessage() メソッドを調べます。

   このメソッドは、アプリケーションがメッセージを受信するたびに呼び出されます。メッセージのプロパティ (返信アドレスや適切な名前など) と送信時刻 (タイムスタンプ) を取得して、TestMessage のユーザ・インタフェースにこれらの情報を表示します。次のコードは、受信メッセージを QATextMessage オブジェクトにキャストして、メッセージの返信アドレスを取得します。

   ```
   text_msg = ( QATextMessage )msg;
   from = text_msg.ReplyToAddress;
   ```

   MessageList ファイルに基づいて実行される主要なタスクの紹介は、これで終わりです。

8. ソリューション・エクスプローラで、*NewMessage* ファイルを開きます。

9. sendMessage() メソッドを調べます。

   このメソッドは、[New Message] ダイアログに入力された情報を引数として、QATextMessage オブジェクトを生成します。QAManager オブジェクトは、送信メッセージをキューに登録します。

   次のコードでは、QATextMessage オブジェクトを生成し、その ReplyToAddress プロパティを設定しています。

   ```
   qa_manager = MessageList.GetQAManager();
   msg = qa_manager.CreateTextMessage();
   msg.ReplyToAddress = MessageList.getOptions().ReceiveQueueName;
   ```

   次のコード行では、送信メッセージをキューに登録しています。変数 dest は宛先アドレスです。宛先アドレスは、この関数の引数として渡されます。

   ```
   qa_manager.PutMessage( dest, msg );
   ```


詳細情報

- 「QAnywhere C++ API リファレンス」 421 ページ
- 「iAnywhere.QAnywhere.Client ネームスペース (.NET 1.0)」 262 ページ
- 「QAnywhere クライアント・アプリケーションの作成」 51 ページ
- TestMessage のサンプルは、*samples-dir¥QAnywhere* にあります。*samples-dir* の詳細については、「サンプル・ディレクトリ」『SQL Anywhere サーバ - データベース管理』を参照してください。

チュートリアルのクリーンアップ

TestMessage、QAnywhere Agent、Mobile Link サーバのすべてのインスタンスを停止します。

第 3 章

QAnywhere メッセージングの設定

目次

サーバ・サイド・コンポーネントの設定

♦ **QAnywhere サーバ・サイド・コンポーネントの設定の概要**

1. サーバ・メッセージ・ストアを設定し、開始します。いずれかの Mobile Link 統合データベースを使用できます。

   「サーバ・メッセージ・ストアの設定」 32 ページを参照してください。

2. -m オプションを指定して mlsrv10 を起動し、サーバ・メッセージ・ストアへの接続を開始します。

   「QAnywhere サーバの起動」 33 ページを参照してください。

3. クライアント・ユーザ名をサーバ・メッセージ・ストアに追加します。

   「QAnywhere クライアント・ユーザ名の登録」 34 ページを参照してください。

注意
サーバ・メッセージ・ストアを作成および管理するには、Sybase Central を使用する方法が最も簡単です。QAnywhere プラグインのタスクのウィンドウ枠から、[サーバ・メッセージ・ストアを使用] を選択します。

サーバ・メッセージ・ストアの設定

サーバ・メッセージ・ストアはサーバ上のリレーショナル・データベースです。このデータベースは、メッセージを、クライアント・メッセージ・ストア、Web サービス、または JMS システムに転送されるまで一時的に格納します。メッセージは、サーバ・メッセージ・ストアを介して、クライアント間で交換されます。

サーバ・メッセージ・ストアは Mobile Link 統合データベースです。したがって、サーバ・メッセージ・ストアとして使用できるのは、Mobile Link がサポートしているいずれかの RDBMS (SQL Anywhere、Adaptive Server Enterprise、Microsoft SQL Server、Oracle、または DB2) です。新しくデータベースを作成して使用することも、既存のデータベースを使用することもできます。

Mobile Link 統合データベース (つまりサーバ・メッセージ・ストア) として使用できるようにデータベースを設定するには、設定スクリプトを実行します。[同期モデル作成] ウィザードを使用して統合データベースを作成した場合は、設定は自動的に行われます。

「統合データベースの設定」 『Mobile Link - サーバ管理』を参照してください。

SQL Anywhere データベースの作成の詳細については、「初期化ユーティリティ (dbinit)」 『SQL Anywhere サーバ - データベース管理』を参照してください。

10.0.0 よりも前のバージョンで作成された SQL Anywhere データベースを使用している場合は、データベースをアップグレードする必要があります。

データベースのアップグレードの詳細については、「SQL Anywhere 10 へのアップグレード」 『SQL Anywhere 10 - 変更点とアップグレード』を参照してください。

注意
サーバ・メッセージ・ストアを作成および管理するには、Sybase Central を使用する方法が最も簡単です。QAnywhere プラグインのタスクのウィンドウ枠から、[サーバ・メッセージ・ストアを使用] を選択します。

例

qanytest.db という名前の SQL Anywhere データベースを作成するには、コマンド・プロンプトで次のように入力します。

```
dbinit -s qanytest.db
```

次のように入力して、データベース上で Mobile Link 設定スクリプトを実行します。

```
%SQLANY10%¥MobiLink¥setup¥syncsa.sql
```

このデータベースは、サーバ・メッセージ・ストアとして使用できます。

QAnywhere サーバの起動

QAnywhere は、Mobile Link 同期を使用してメッセージを転送します。QAnywhere サーバは、メッセージングが有効になっている Mobile Link サーバです。

QAnywhere サーバを実行するには、次のオプションを指定して Mobile Link サーバ (mlsrv10) を起動します。

- **-c *connection-string*** サーバ・メッセージ・ストアに接続するための接続文字列を指定します。これは mlsrv10 に必須のオプションです。

  「-c オプション」 『Mobile Link - サーバ管理』を参照してください。

- **-m** QAnywhere のメッセージング機能を有効にします。

  「-m オプション」 『Mobile Link - サーバ管理』を参照してください。

上記以外にも、Mobile Link サーバの動作をカスタマイズするためのオプションがいくつかあります。詳細については、「mlsrv10 の構文」 『Mobile Link - サーバ管理』を参照してください。

注意

- JMS メッセージング・システムと統合する場合は、Mobile Link サーバの起動時に上記以外のオプションを指定する必要があります。

  「JMS 統合用 Mobile Link サーバの起動」 140 ページを参照してください。

例

サンプルのサーバ・メッセージ・ストア *qanyserv.db* を使用して QAnywhere のメッセージングを開始するには、*samples-dir¥QAnywhere¥server* に移動し、コマンド・プロンプトで次のように入力します。

```
mlsrv10 -m -c "dsn=QAnywhere 10 Demo"
```


samples-dir の詳細については、「サンプル・ディレクトリ」 『SQL Anywhere サーバ - データベース管理』を参照してください。

QAnywhere クライアント・ユーザ名の登録

QAnywhere の各クライアント・メッセージ・ストアには、識別のためのユニークな ID が付けられます。また、クライアント・メッセージ・ストアには、クライアント・メッセージ・ストアを Mobile Link サーバで認証するためにオプションで使用できる Mobile Link ユーザ名があります。Mobile Link ユーザ名は、qaagent の -mu オプションを使用して指定できます。このオプションを指定しないと、クライアント・メッセージ・ストア ID と同じ名前のユーザ名が作成されます。

Mobile Link ユーザ名は、サーバ・メッセージ・ストアに登録する必要があります。これを実行するには、いくつかの方法があります。

- mluser ユーティリティを使用する。

  「Mobile Link ユーザ認証ユーティリティ [mluser]」 『Mobile Link - サーバ管理』を参照してください。

- Sybase Central で Mobile Link 管理モードを使用する。
- mlsrv10 で -zu+ オプションを指定する。この場合、最初に同期するときに、統合データベースに追加されていない既存の Mobile Link ユーザが追加されます。このオプションは、開発時には便利ですが、運用環境ではおすすめできません。

  「-zu オプション」 『Mobile Link - サーバ管理』を参照してください。

Mobile Link ユーザ名の詳細については、「Mobile Link ユーザの概要」 『Mobile Link - クライアント管理』を参照してください。

クライアント・メッセージ・ストア ID の詳細については、「-id オプション」 163 ページを参照してください。

QAnywhere サーバ上でのクライアントのプロパティの設定

QAnywhere プラグインを使用して、QAnywhere サーバ上で QAnywhere クライアントのプロパティを設定できます。この操作を行う場合、サーバにクライアントを追加する必要があります。クライアントに対して最初の同期を実行すると、プロパティがダウンロードされます。

- **Sybase Central でクライアント・ユーザ名を追加するには、次の手順に従います。**

1. Sybase Central を起動します。
   - [スタート] – [プログラム] – [SQL Anywhere 10] – [Sybase Central] を選択します。
   - [接続] メニューから、[QAnywhere 10 に接続] を選択します。

- ODBC データ・ソース名または ODBC データ・ソース・ファイルを指定し、必要に応じてユーザ ID とパスワードを指定します。[OK] をクリックします。

2. [ファイル] – [新規] – [クライアント] を選択します。
3. クライアント名を入力します。
4. [OK] をクリックします。

参照

- 「QAnywhere クライアント・ユーザ名の登録」 34 ページ

QAnywhere サーバのログ

QAnywhere サーバは、メッセージングが有効になっている Mobile Link サーバです。QAnywhere サーバのログ・ファイルは、Mobile Link のログ・ファイルです。

Mobile Link のログ・ファイルの詳細については、「Mobile Link サーバの動作のロギング」『Mobile Link - サーバ管理』を参照してください。

Mobile Link サーバ・ログ・ファイル・ビューワ

QAnywhere サーバのログ・ファイルを表示するには、Sybase Central を開き、[ツール] – [QAnywhere 10] – [Mobile Link サーバ・ログ・ファイル・ビューワ] を選択します。表示するログ・ファイルの選択が要求されます。

ログ・ビューワでは、Mobile Link のログ・ファイルに格納されている情報が読み込まれます。Mobile Link サーバに接続したり、ログ・ファイルの構成が変更されたりはしません。

ログ・ビューワでは、表示する情報をフィルタできます。また、ログ内の情報に基づく統計値も表示できます。

クライアント・サイド・コンポーネントの設定

♦ **クライアント・サイド・コンポーネントの設定の概要**

1. SQL Anywhere データベースを作成し、クライアント・メッセージ・ストアとして初期化します。

   「クライアント・メッセージ・ストアの設定」 36 ページを参照してください。

2. クライアント・アプリケーションを作成します。

   「QAnywhere クライアント・アプリケーションの作成」 51 ページを参照してください。

3. QAnywhere Agent を起動します。

   「QAnywhere Agent の実行」 38 ページを参照してください。

注意
クライアント・メッセージ・ストアを作成および管理するには、Sybase Central を使用する方法が最も簡単です。QAnywhere プラグインのタスクのウィンドウ枠から、[クライアント・メッセージ・ストアを使用] を選択します。

クライアント・メッセージ・ストアの設定

クライアント・メッセージ・ストアは、リモート・デバイス上の SQL Anywhere データベースです。アプリケーションは、QAnywhere API を使用してこのメッセージ・ストアに接続します。

クライアントのメッセージ・ストアは、QAnywhere アプリケーション専用にする必要があります。ただし、データベース・サーバ内で別のデータベースを実行することはできます。これは、QAnywhere のクライアント・メッセージ・ストアと Mobile Link 同期クライアントを同じデバイスで実行する場合に便利です。

リレーショナル・データベースをメッセージ・ストアとして使用することで、安全で高性能なストアを実現できます。

「セキュリティ保護されたクライアント・メッセージ・ストアの作成」 190 ページを参照してください。

♦ **クライアント・メッセージ・ストアを作成するには、次の手順に従います。**

1. 新しい SQL Anywhere データベースを作成します。

   「データベースの作成」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

2. 各クライアント・メッセージ・ストアを初期化します。この初期化を行うには、次のオプションを指定して QAnywhere Agent (qaagent) を実行します。

   ♦ **-c オプション**　作成したデータベースに接続するための接続文字列を指定します。

「-c オプション」 159 ページを参照してください。

- **-si オプション**　データベースを初期化します。-si オプションを指定すると、デフォルトのデータベース・ユーザとパスワードが作成されます。QAnywhere Agent は、データベースが初期化された後で停止します。

  「-si オプション」 181 ページを参照してください。

- **-id オプション**　クライアント・メッセージ・ストア ID を事前に割り当てます (オプション)。

  「クライアント・メッセージ・ストア ID の作成」 37 ページと「-id オプション」 163 ページを参照してください。

- **-mu オプション**　Mobile Link サーバでの認証で使用するユーザ名を作成します (オプション)。ここで -mu オプションを使用しなくても、QAnywhere Agent の起動時にいつでも指定できます。ユーザ名が既存していない場合は、自動的に作成されます。

3. -mu オプションを指定してユーザ名を作成した場合は、その名前をサーバ・メッセージ・ストアに追加する必要があります。-zu+ オプションを指定して mlsrv10 を実行すると、この処理が自動的に行われるようになります。この処理は、それ以外の方法でも実行できます。

   「QAnywhere クライアント・ユーザ名の登録」 34 ページを参照してください。

4. デフォルトのパスワードを変更します。また、クライアント・メッセージ・ストアのセキュリティを保護するための手順も実行します。

   「セキュリティ保護されたクライアント・メッセージ・ストアの作成」 190 ページを参照してください。

旧バージョンの QAnywhere で作成したクライアント・メッセージ・ストアをアップグレードすることもできます。

「-su オプション」 183 ページと「-sur オプション」 184 ページを参照してください。

注意
クライアント・メッセージ・ストアを作成および管理するには、Sybase Central を使用する方法が最も簡単です。QAnywhere プラグインのタスクのウィンドウ枠から、[クライアント・メッセージ・ストアを使用] を選択します。

クライアント・メッセージ・ストア ID の作成

クライアント・メッセージ・ストア ID を指定しなかった場合には、-si オプションを指定して qaagent を実行した後、最初に qaagent を実行したときに、デバイス名がクライアント・メッセージ・ストア ID として割り当てられます。ID は QAnywhere Agent ウィンドウに表示されます。

ID は手動で指定するほうが便利です。ID は次の方法で指定できます。

- qaagent -si オプションを使用してクライアント・メッセージ・ストアを初期化するときに、qaagent -id オプションを使用して ID を指定する。

- クライアント・メッセージ・ストアの初期化の後、最初に qaagent を実行するときに、-id オプションを使用して ID を指定する。

「QAnywhere Agent」 155 ページを参照してください。

クライアント・メッセージ・ストア ID は、大文字と小文字以外の違いがある必要があります。たとえば、2 つのメッセージ・ストア ID を AAA と aaa にすることはできません。

クライアント・メッセージ・ストア ID は 128 文字以内である必要があります。

トランザクション・ログ

トランザクション・ログは使用することをおすすめします。SQL Anywhere データベースはトランザクション・ログを使用したときに実行効率が最もよく、またトランザクション・ログはデータベース障害が発生したときにデータベースを保護する手段になるからです。しかし、トランザクション・ログは非常に大きくなる可能性があります。このため、QAnywhere Agent では、トランザクション・ログの内容がチェックポイントで削除されるように、デフォルトで dbsrv10 -m オプションが設定されます。この設定をおすすめします。qaagent -c オプションで StartLine パラメータを指定した場合は、-m を指定してください。

クライアント・メッセージ・ストアの保護

バックアップとリカバリの詳細については、「バックアップとリカバリのプランの設計」 『SQL Anywhere サーバ - データベース管理』を参照してください。

クライアント・メッセージ・ストアの作成例

次のコマンドを実行すると、SQL Anywhere データベース qanyclient.db が作成されます (dbinit -i と -s オプションは必須ではありませんが、小型デバイスでは指定することをおすすめします。

```
dbinit -i -s qanyclient.db
```

次のコマンドを実行すると、*qanyclient.db* への接続が実行され、このデータベースが QAnywhere クライアント・データベースとして初期化されます。

```
qaagent -si -c "DBF=qanyclient.db"
```

「初期化ユーティリティ (dbinit)」 『SQL Anywhere サーバ - データベース管理』と「QAnywhere Agent」 155 ページを参照してください。

QAnywhere Agent の実行

QAnywhere Agent (qaagent) は、クライアント・デバイス上で実行される独立したプロセスです。クライアント・メッセージ・ストアをモニタし、メッセージ転送を行うタイミングを決定します。

QAnywhere Agent は、サーバ・メッセージ・ストアとクライアント・メッセージ・ストアの間でメッセージを転送します。QAnywhere Agent の複数のインスタンスを 1 台のデバイス上で実行できます。ただし、各インスタンスが独自のメッセージ・ストアに接続されている必要があります。メッセージ・ストアごとにユニークなメッセージ・ストア ID が必要です。

コマンド・ライン・オプションを使用して、コマンド・ラインから Agent を実行できます。QAnywhere Agent を起動するときには、少なくとも次のオプションを指定する必要があります。

- **接続パラメータ**　クライアント・メッセージ・ストアに接続します。

  これは、エージェント設定ファイルの [プロパティ] ダイアログでは、[メッセージ・ストア] タブに表示される情報です。

  qaagent コマンド・ラインでは、-c オプションとともに指定されます。

  「-c オプション」 159 ページを参照してください。

- **クライアント・メッセージ・ストア ID**　クライアント・メッセージ・ストアを識別します。クライアント・メッセージ・ストアの初期化の後、最初に qaagent を実行するときにこのオプションを指定すると、クライアント・メッセージ・ストアに名前を付けることができます。このオプションを指定しないと、デバイス名がデフォルトのストア名として使用されます。以後、qaagent を起動するときには、その都度 -id オプションを使用してユニークなクライアント・メッセージ・ストア ID を指定する必要があります。

  これは、エージェント設定ファイルの [プロパティ] ダイアログの [一般] タブで指定されます。

  qaagent コマンド・ラインでは、-id オプションとともに指定されます。

  「-id オプション」 163 ページを参照してください。

- **ネットワーク・プロトコル・オプションとプロトコル・オプション**　Mobile Link サーバに接続します。デフォルトの通信パラメータを使用して Mobile Link サーバが QAnywhere Agent と同じデバイス上で動作している場合を除いて、このオプションは必須です。

  これは、エージェント設定ファイルの [プロパティ] ダイアログの [サーバ] タブに表示されるサーバ情報です。

  qaagent コマンド・ラインでは、-x オプションです。

  「-x オプション」 186 ページを参照してください。

QAnywhere Agent のオプションの完全なリストについては、「qaagent 構文」 156 ページを参照してください。

Windows CE 上での qaagent の起動

Windows CE 環境では、-qi オプションを指定して、QAnywhere Agent をクワイエット・モードで起動できます。

「-qi オプション」 180 ページを参照してください。

複数の QAnywhere Agent インスタンスの実行

qaagent の複数のインスタンスを 1 台のデバイス上で実行できます。ただし、2 番目のインスタンスを起動するときは、次の点に注意してください。

- 2 番目のインスタンスは別のデータベース・ファイルを指定して起動する必要がある。
- -id オプションを使用して、ユニークなメッセージ・ストア ID を指定する必要がある。

  「-id オプション」 163 ページを参照してください。

QAnywhere Agent の停止

QAnywhere Agent を停止するには、コンソールで [シャットダウン] をクリックします。

QAnywhere Agent をクワイエット・モードで起動した場合は、**qastop** を実行して停止する必要があります。

「-qi オプション」 180 ページを参照してください。

QAnywhere Agent によって開始されるプロセス

QAnywhere Agent は、さまざまなメッセージング・タスクを処理するための各種プロセスを開始します。これらの各プロセスは、QAnywhere Agent によってまとめて管理されるので、個別に管理する必要はありません。QAnywhere Agent を起動すると、次のプロセスが生成されます。

- **dbmlsync** dbmlsync 実行プログラムは Mobile Link 同期クライアントです。dbmlsync 実行プログラムを使用して、メッセージが送受信されます。

  警告
  dbmlsync を qaagent から独立して QAnywhere のメッセージ・ストアに対して実行しないでください。

- **dblsn** dblsn 実行プログラムは Listener ユーティリティです。Listener ユーティリティは、Push 通知を受信します。Push 通知を使用しない場合、アプリケーションを配備するときに dblsn 実行プログラムを加える必要はありません。この場合、qaagent は、-push **none** オプションを指定して実行する必要があります。

  「-push オプション」 177 ページを参照してください。

- **データベース・サーバ** クライアント・メッセージ・ストアは SQL Anywhere データベースです。QAnywhere Agent がデータベースを実行するためには、SQL Anywhere データベース・サーバが必要です。Windows CE では、データベース・サーバは *dbsrv10.exe* です。Windows では、パーソナル・データベース・サーバ *dbeng10.exe* がデータベース・サーバです。

  QAnywhere Agent は、データベース・サーバを生成するか、動作中のサーバに接続します。どちらになるかは、qaagent -c オプションに指定された通信パラメータによって決まります。

  「-c オプション」 159 ページを参照してください。

QAnywhere Agent の配備

詳細については、「QAnywhere アプリケーションの配備」 97 ページを参照してください。

クライアントにメッセージを転送するタイミングの決定

クライアント側で「**ポリシー**」を指定することによって、メッセージを転送するタイミングを決定できます。ポリシーは、クライアント・メッセージ・ストアからサーバ・メッセージ・ストアにメッセージを転送するタイミングを QAnywhere Agent に指示します。ポリシーを指定しない場合は、サーバへの配信対象メッセージがキューに登録されると、自動的に転送が行われます。こ

れがデフォルトの動作です。ポリシーには、スケジュール済み、自動、オンデマンドの 3 つの事前定義済みポリシーと、カスタム・ポリシーがあります。

ポリシーは、次の 2 つの方法で指定できます。

- Sybase Central で QAnywhere プラグインを使用し、[エージェント設定ファイルの新規作成] タスクを選択する。ポリシーは、コマンド・ファイルの [プロパティ] ダイアログの [一般] タブで指定します。

  カスタム・プロパティを指定するには、[エージェント・ルール・ファイルの新規作成] タスクも選択する必要があります。このタスクにより、拡張子 *.qar* の付いたファイルが作成されます。この拡張子は、Sybase Central の規則に従っています。

- -policy オプションを使用して、qaagent をコマンド・ラインから実行する。カスタム・ポリシーの場合は、ルール・ファイルを作成してから、それを指定します。

スケジュール済みポリシー

スケジュール済みポリシーは、指定された時間間隔で転送を実行するように QAnywhere Agent に指示します。

スケジュール済みポリシーを呼び出すには、コマンド・ファイルの [プロパティ] ダイアログで **scheduled** を選択するか、QAnywhere Agent を起動するときに、次のように scheduled キーワードを指定します。

qaagent -policy scheduled [*interval*] ...

interval に間隔を秒単位で指定します。

デフォルト値は 900 秒 (15 分) です。

スケジュール済みポリシーを指定すると、次のいずれかの条件が満たされたときに、*n* 秒間隔でメッセージが転送されます。

- 前回の時間間隔が経過した後、クライアント・メッセージ・ストアに新しいメッセージが着信した。
- 前回の時間間隔が経過した後、メッセージ・ステータスが変化した。この現象は、通常、アプリケーションがメッセージの受信を確認したときに起こります。

  受信確認の詳細については、次の項を参照してください。

  - .NET：「AcknowledgementMode 列挙」 262 ページ
  - C++：「AcknowledgementMode クラス」 422 ページ
  - Java：「AcknowledgementMode インタフェース」 532 ページ

- 前回の時間間隔が経過した後、Push 通知を受信した。
- 前回の時間間隔が経過した後、ネットワーク・ステータス変化通知を受信した。
- Push 通知が無効にされた。

時間間隔を無視するには、トリガの送受信メソッドを呼び出します。このメソッドを使用すると、時間間隔が経過する前にメッセージを転送できます。次の項を参照してください。

- .NET：「TriggerSendReceive メソッド」 338 ページ
- C++：「triggerSendReceive 関数」 488 ページ
- Java：「triggerSendReceive メソッド」 601 ページ
- SQL：「ml_qa_triggersendreceive」 714 ページ

自動ポリシー

自動ポリシーは、クライアント・メッセージ・ストアとサーバ・メッセージ・ストアをできるかぎり最新の状態に維持します。

自動ポリシーを使用すると、次のいずれかのイベントが発生したときにメッセージが転送されます。

- PutMessage() が呼び出された。次の項を参照してください。
  - .NET：「PutMessage メソッド」 322 ページ
  - C++：「putMessage 関数」 480 ページ
  - Java：「putMessage メソッド」 591 ページ
  - SQL：「ml_qa_putmessage」 713 ページ
- メッセージ・ステータスが変化した。この現象は、通常、アプリケーションがメッセージの受信を確認したときに起こります。次の項を参照してください。
  - .NET：「AcknowledgementMode 列挙」 262 ページ
  - C++：「AcknowledgementMode クラス」 422 ページ
  - Java：「AcknowledgementMode インタフェース」 532 ページ
  - SQL：SQL API を使用するメッセージングはすべてトランザクション志向です。
- Push 通知を受信した。

  「Push 通知の使用方法」 45 ページを参照してください。
- ネットワーク・ステータス変更通知を受信した。

  「Push 通知の通知」 59 ページを参照してください。
- TriggerSendReceive() が呼び出された。次の項を参照してください。
  - .NET：「TriggerSendReceive メソッド」 338 ページ
  - C++：「triggerSendReceive 関数」 488 ページ
  - Java：「triggerSendReceive メソッド」 601 ページ
  - SQL：「ml_qa_triggersendreceive」 714 ページ

オンデマンド・ポリシー

オンデマンド・ポリシーを使用すると、アプリケーションによって指示されたときにだけメッセージが転送されます。

アプリケーションで TriggerSendReceive() が呼び出されると、メッセージが強制的に転送されます。

Agent が Push 通知またはネットワーク・ステータス変更通知を受信すると、対応するメッセージがシステム・キューに送信されます。アプリケーションでは、このイベントを検出し、TriggerSendReceive() を呼び出してメッセージを強制的に転送することができます。次の項を参照してください。

- .NET：「TriggerSendReceive メソッド」 338 ページ
- C++：「triggerSendReceive 関数」 488 ページ
- Java：「triggerSendReceive メソッド」 601 ページ
- SQL：「ml_qa_triggersendreceive」 714 ページ

Push 通知とネットワーク・ステータス変更通知の処理の詳細については、「システム・キュー」 58 ページを参照してください。

カスタム・ポリシー

カスタム・ポリシーを使用すると、メッセージ転送のタイミングと、そのメッセージ転送で送信するメッセージを定義できます。カスタム・ポリシーは、転送ルール・セットを使って定義します。

各ルールは次の形式で定義します。

schedule = *condition*

condition には条件を指定し、*schedule* にはその条件を評価するタイミングを指定します。詳細については、「ルールの構文」 242 ページを参照してください。

condition の条件を満たしているメッセージがすべて転送されます。*schedule* に automatic と指定すると、次のいずれかのイベントが発生したときに条件が評価されます。

- PutMessage() が呼び出された。次の項を参照してください。
  - .NET：「PutMessage メソッド」 322 ページ
  - C++：「putMessage 関数」 480 ページ
  - Java：「putMessage メソッド」 591 ページ
  - SQL：「ml_qa_putmessage」 713 ページ
- メッセージ・ステータスが変化した。この現象は、通常、アプリケーションがメッセージの受信を確認したときに起こります。次の項を参照してください。
  - .NET：「AcknowledgementMode 列挙」 262 ページ
  - C++：「AcknowledgementMode クラス」 422 ページ
  - Java：「AcknowledgementMode インタフェース」 532 ページ
  - SQL：SQL API を使用するメッセージングはすべてトランザクション志向です。
- Push 通知を受信した。

  「Push 通知の使用方法」 45 ページを参照してください。
- ネットワーク・ステータス変更通知を受信した。

- TriggerSendReceive() が呼び出された。次の項を参照してください。
  - .NET：「TriggerSendReceive メソッド」 338 ページ
  - C++：「triggerSendReceive 関数」 488 ページ
  - Java：「triggerSendReceive メソッド」 601 ページ
  - SQL：「ml_qa_triggersendreceive」 714 ページ

参照

- 「メッセージ転送ルール」 251 ページ
- 「-policy オプション」 175 ページ

Push 通知の使用方法

Push 通知は、メッセージ転送を開始するよう QAnywhere クライアントに対して指示するために、サーバ・メッセージ・ストアから QAnywhere クライアントに配信される特殊なメッセージです。Push 通知はデフォルトでオンになっていますが、必須ではありません。Push 通知が行われるとき、QAnywhere アーキテクチャでは次の追加コンポーネントが動作します。

- サーバ側では、QAnywhere Notifier が Push 通知を送信します。
- クライアント側では、QAnywhere Listener が Push 通知を受け取り、QAnywhere Agent に渡します。
- クライアント側では、Push 通知ごとに通知がシステム・キューに送信されます。

QAnywhere Agent のスケジュール済みまたは自動のポリシーを使用する場合は、Push 通知によってクライアントが自動的にメッセージ転送を開始します。オンデマンド・ポリシーを使用する場合は、イベント・ハンドラを使用して Push 要求を手動で処理する必要があります。

Push 通知の手動処理の詳細については、「Push 通知の通知」 59 ページを参照してください。

QAnywhere Agent ポリシーの詳細については、「クライアントにメッセージを転送するタイミングの決定」 40 ページを参照してください。

Push 通知はデフォルトで有効になっています。つまり、qaagent -push オプションは、デフォルトで connected モードに設定されます。connected モードでは、Push 通信は永続的な TCP/IP 接続で送信されます。

UDP を使用して Push 通知を配信する場合、ほとんどは設定を行わなくても機能します。しかし、ActiveSync の UDP の実装には制限があるため、ActiveSync 上では機能しません。

参照

- 「Push 通知によるメッセージングのシナリオ」 9 ページ
- 「Push 通知の通知」 59 ページ
- 「-push オプション」 177 ページ

Push 通知の設定

Push 通知は、QAnywhere サーバのメッセージ・ストアに、QAnywhere クライアント宛てのメッセージが届いたときに、サーバからそのクライアントに送信される特殊なメッセージです。Push 通知は、サーバで実行される **Notifier** というプログラムから送信され、クライアントで実行される **Listener** というプログラムで受信されます。Push 通知はゲートウェイを経由して送信されます。クライアントで Push 通知が受信されると、サーバで待機しているメッセージを受信するためのメッセージ転送が開始されるか、カスタムの処理が行われます。

Notifier、Listener、ゲートウェイは、変更しなくても QAnywhere で正常に動作するように事前に設定されています。ただし、まれに設定が必要な場合があります。また、必要に応じて Notifier の一部の設定を変更することもできます。次の項を参照してください。

- 「QAnywhere Notifier の設定」 46 ページ
- 「Listener の設定」 48 ページ
- 「QAnywhere ゲートウェイの設定」 48 ページ

Push 通知を無効にし、Notifier または Listener が使用されないようにすることもできます。「-push オプション」 177 ページを参照してください。

Push 通知に対するクライアントの応答については、「クライアントにメッセージを転送するタイミングの決定」 40 ページを参照してください。

QAnywhere Notifier の設定

QAnywhere Notifier は Mobile Link の設定スクリプトによって作成され、-m オプションを指定して Mobile Link サーバを実行したときに起動します。QAnywhere Notifier は QAnyNotifier_client といいます。

QAnyNotifier_client では、「Mobile Link 通知プロパティ」 『Mobile Link - サーバ起動同期』に示すデフォルト値が使用されます。ただし、次の例外があります。

- gui プロパティは off に設定されるので、Notifier が実行されるコンピュータで Notifier のダイアログが表示されない。
- enable プロパティは no に設定されるので、Notifier を起動するには -m オプションを指定して mlsrv10 を実行する必要がある。
- poll_every プロパティは 5 に設定されるので、Notifier では 5 秒ごとにポーリングが実行され、Push 通知を送信する必要があるかどうかが確認される。

Notifier の次のプロパティを変更できます。

- poll_every プロパティ
- request_cursor プロパティの再送間隔
- request_cursor プロパティの存続期間

> **注意**
> 上記以外の Notifier のプロパティは変更しないでください。request_cursor の他のカラムは変更しないでください。

poll_every プロパティ

次のコード内で値 5 を変更し、統合データベースに対して実行することで、QAnyNotifier_client のデフォルトのポーリング間隔を変更できます。

```
CALL ml_add_property( 'SIS', 'Notifier(QAnyNotifier_client)', 'poll_every', '5' )
```

「poll_every プロパティ」 『Mobile Link - サーバ起動同期』を参照してください。

再送間隔と存続期間

QAnywhere Notifier には、デフォルトの request_cursor が含まれます。request_cursor は、どの情報を Push 要求で送信するか、誰が、いつ、どこで情報を受信するかを決定します。再送間隔と存続期間以外のデフォルト値は変更しないでください。再送間隔は、未受信の Push 通知を再送する間隔を指定します。この間隔はデフォルトでは 5 分です。存続期間は、未受信の Push 通知の再送を続ける時間を指定します。この時間はデフォルトでは 3 時間です。ほとんどの場合はこれらのデフォルト値が最適です。QAnyNotifier_client のデフォルトの request_cursor は次のとおりです。

```
SELECT
  u.user_id,
  "Default-DeviceTracker",
  "qa",
  u.name,
  u.name,
  "5M",
  "3H"
    FROM ml_qa_notifications u
    WHERE EXISTS( SELECT *
                  FROM ml_listening l
                  WHERE l.name = u.name AND l.listening = "y")
```

request_cursor のカラムの詳細については、「Push 要求テーブルの作成」 『Mobile Link - サーバ起動同期』を参照してください。

再送間隔をデフォルト値の 5 分から別の値に変更するには、次のコード内で値 5M を変更します。存続期間をデフォルト値の 3 時間から別の値に変更するには、値 3H を変更します。

```
CALL ml_add_property(
  'SIS',
  'Notifier(QAnyNotifier_client)',
  'request_cursor',
  'select u.user_id,
  "Default-DeviceTracker",
  "qa",
  u.name,
  u.name,
  "5M",
  "3H"
    FROM ml_qa_notifications u
    WHERE EXISTS(
      SELECT *
      FROM ml_listening l WHERE l.name = u.name AND l.listening = "y")' )
```

詳細については、「request_cursor プロパティ」 『Mobile Link - サーバ起動同期』を参照してください。

参照

- 「Notifier の設定」 『Mobile Link - サーバ起動同期』
- 「Mobile Link 通知プロパティ」 『Mobile Link - サーバ起動同期』
- 「Notifier」 『Mobile Link - サーバ起動同期』
- 「ml_add_property」 『Mobile Link - サーバ管理』
- 「Push 要求」 『Mobile Link - サーバ起動同期』

Listener の設定

Listener は、クライアント・メッセージ・ストアと同じデバイスで実行されます。Listener は、サーバから Push 通知を受信して、QAnywhere Agent に渡す役割を果たします。

Listener は QAnywhere で正常に動作するように事前に設定されています。ただし、まれにデフォルトの動作を変更する必要がある場合があります。

たとえば、QAnywhere で使用するゲートウェイを SMS ゲートウェイに変更する場合は、異なるオプションを指定して Listener を手動で起動する必要があります。QAnywhere のメッセージ・ストア ID が mystore で、Mobile Link ホストが *acme.com* であり、SMS ライブラリ *maac555.dll* を使用して AirCard 555 上で SMS メッセージを受信するように Listener を起動するとします。この場合、次のコマンドを使用して Listener を起動する必要があります。

```
dblsn.exe -u ias_mystore_lsn -e mystore -t+ mystore
  -x "tcpip(host=acme.com)" -pc- -d lsn_udp.dll -a "port=5001" -d maac555.dll
  -i 60
```

起動した Listener が QAnywhere Agent で認識されるためには、次のように入力して QAnywhere Agent も再起動する必要があります。

```
qaagent -c "dbf=mystore.db;eng=mystore;dbn=mystore" -id mystore
  -lp 5001-x tcpip(host=acme.com)
```

参照

- 「Listener」 『Mobile Link - サーバ起動同期』
- 「Listener 構文」 『Mobile Link - サーバ起動同期』
- 「QAnywhere ゲートウェイの設定」 48 ページ

QAnywhere ゲートウェイの設定

ゲートウェイは、Push 通知が送信される方法です。QAnywhere では、デフォルトで Device Tracker ゲートウェイが使用されます。Device Tracker ゲートウェイでは、まず SYNC ゲートウェイを使用しようとします。SYNC ゲートウェイは、Mobile Link 同期と同じプロトコルを使用し、永続的です。ほとんどの場合、デフォルトの Device Tracker ゲートウェイが、Push 通知を送信する最適な方法です。ただし、SMS ゲートウェイまたは UDP ゲートウェイを使用することもできます。

ゲートウェイを設定する方法については、「ゲートウェイ・プロパティ」 『Mobile Link - サーバ起動同期』と「プロパティの設定」 『Mobile Link - サーバ起動同期』を参照してください。

SMS ゲートウェイを使用するには、新しいオプションを指定して Listener を起動する必要があります。「Listener の設定」 48 ページを参照してください。

UDP ゲートウェイを使用するには、qaagent の -push disconnected オプションを設定する必要があります。「-push オプション」 177 ページを参照してください。

参照

- 「ゲートウェイと Carrier」 『Mobile Link - サーバ起動同期』

フェールオーバ・メカニズムの設定

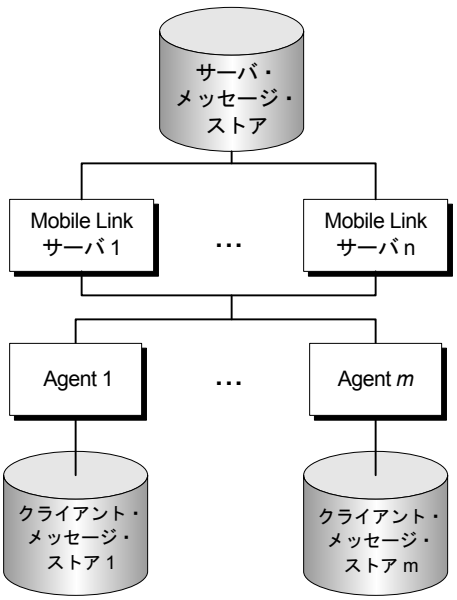
QAnywhere アプリケーションは、フェールオーバ・メカニズムを使用するように設定できます。フェールオーバ・メカニズムを使用すると、Mobile Link サーバで障害が発生しても代替サーバを使用できます。フェールオーバをサポートするには、各 QAnywhere Agent を起動するときに、Mobile Link サーバのリストを指定する必要があります。リストの先頭に指定された Mobile Link サーバがプライマリ・サーバになります。残りのサーバは代替サーバになります。

たとえば、リモート・デバイス上で次のコマンドを実行すると、1 台のプライマリ・サーバと 1 台の代替サーバが存在する QAnywhere Agent が起動されます。

```
qaagent -x tcpip(host=ml1.ianywhere.com)
        -x tcpip(host=ml2.ianywhere.com)
```

QAnywhere Agent ごとに異なるプライマリ・サーバを指定できます。

次の図に、複数の Mobile Link サーバと複数の QAnywhere Agent を使用したフェールオーバ構成を示します。複数のクライアント・メッセージ・ストアが存在する一方で、Mobile Link サーバはいずれも同一のサーバ側メッセージ・ストアに接続されています。



この構成には次のような特徴があります。

- メッセージ転送が発生すると、QAnywhere Agent が接続されているサーバに関係なく、サーバ・メッセージ・ストア内のすべてのメッセージがクライアント・メッセージ・ストアに配信される。
- Push 通知は、QAnywhere Agent がプライマリ・サーバに接続されているときだけ、その QAnywhere Agent に送信される。

- 単一障害点 (Single-Point-of-Failure) が存在する。サーバ・メッセージ・ストアが存在するマシンが使用できなくなると、メッセージを転送できない。

フェールオーバ Mobile Link サーバが設定されている場合、デフォルトでは、プライマリ・サーバへの接続で障害が発生するとすぐに QAnywhere Agent は代替サーバへの接続を試行します。このデフォルト動作を変更するには、QAnywhere Agent の -fr オプションを使用します。-fr オプションを指定すると、QAnywhere Agent は代替サーバに接続する前に、プライマリ・サーバへの接続を再試行します。再試行の回数も指定できます。-fd オプションを併せて使用することで、プライマリ・サーバへの接続試行の間隔も指定できます。

-fr オプションと -fd オプションが適用されるのは、プライマリ・サーバだけです。指定された回数だけ再試行してもプライマリ・サーバに接続できない場合、QAnywhere Agent は代替サーバへの接続を試行します。各代替サーバへの接続は 1 回のみ試行されます。代替サーバへの接続が確立できない場合は、エラーが発行されます。

参照

- 「-x オプション」 186 ページ
- 「-fd オプション」 161 ページ
- 「-fr オプション」 162 ページ
- 「QAnywhere Agent の実行」 38 ページ

第 4 章

QAnywhere クライアント・アプリケーションの作成

目次

QAnywhere インタフェースの概要

QAnywhere クライアント・アプリケーションは、QAnywhere メッセージの送信と受信を管理します。このアプリケーションは、次のいずれかの QAnywhere API を使用して作成できます。

- QAnywhere .NET API
- QAnywhere C++ API
- QAnywhere Java API
- QAnywhere SQL API

各種のクライアントを QAnywhere システム内で組み合わせて使用できます。たとえば、QAnywhere SQL で生成されたメッセージを、.NET API、C++ API、Jana API を使用して作成されたクライアントで受信することもできます。JMS コネクタがサーバ上で設定されている場合は、JMS クライアントがこのメッセージを受信することもできます。同様に、QAnywhere .NET、C++、Java、または JMS のクライアントで生成されたメッセージを、QAnywhere SQL で受信することもできます。

QAnywhere .NET API

QAnywhere .NET API は、Microsoft .NET Framework を使用している Windows コンピュータや、Microsoft .NET Compact Framework が動作しているハンドヘルド・デバイスに、QAnywhere クライアント・アプリケーションを配備するためのプログラミング・インタフェースです。QAnywhere .NET API は、iAnywhere.QAnywhere.Client ネームスペースとして提供されています。

QAnywhere は Microsoft Visual Studio .NET 2003 と 2005 をサポートしています。

注意
この章で示す QAnywhere .NET API のサンプル・コードで使用するプログラミング言語は C# だけですが、QAnywhere .NET API には、Microsoft .NET でサポートされているすべてのプログラミング言語を使用してアクセスすることができます。

TestMessage サンプル・アプリケーションには、Java、C#、Visual Basic.NET で作成されたバージョンがあります。.NET Compact Framework で作成されたサンプルもあります。

.NET バージョンの TestMessage サンプル・アプリケーションの詳細については、「レッスン 4 : TestMessage クライアントのソース・コードの概要」 24 ページを参照してください。

「iAnywhere.QAnywhere.Client ネームスペース (.NET 1.0)」 262 ページを参照してください。

QAnywhere C++ API

QAnywhere C++ API は、Microsoft Visual C++ 6.0、Microsoft Visual Studio .NET 2003、Microsoft eMbedded Visual C++ 3.0、Microsoft eMbedded Visual C++ 4.0、Visual Studio 2005 をサポートしています。

QAnywhere C++ API の構成ファイルは次のとおりです。

- SQL Anywhere インストール環境のサブディレクトリ *QAnywhere¥h* にある一連のヘッダ・ファイル (メインのヘッダ・ファイルは *qa.hpp*)
- SQL Anywhere インストール環境のサブディレクトリ *ce¥arm.30¥lib*、*ce¥arm.50¥lib*、*QAnywhere¥lib*、*ce¥x86.30¥lib* にあるインポート・ライブラリ (*qany10.lib*)
- SQL Anywhere インストール環境のサブディレクトリ *win32*、*ce¥arm.30*、*ce¥arm.50¥lib*、*ce¥x86.30* にあるランタイム DLL (*qany10.dll*)

この API を使用するには、ソース・コード・ファイルに上記のヘッダ・ファイルをインクルードする必要があります。インポート・ライブラリは、アプリケーションをランタイム DLL にリンクするために使用します。ランタイム DLL は、アプリケーションとともに配備する必要があります。

C++ バージョンの TestMessage サンプル・アプリケーションは、*samples-dir¥QAnywhere¥Desktop¥MFC* にあります。*samples-dir* の詳細については、「サンプル・ディレクトリ」『SQL Anywhere サーバ - データベース管理』を参照してください。

「QAnywhere C++ API リファレンス」 421 ページを参照してください。

QAnywhere Java API

QAnywhere Java API は JRE 1.4.2 と 1.5.0 をサポートしています。

QAnywhere Java API の構成ファイルは次のとおりです。

- API リファレンス。このマニュアルから情報を入手することも、SQL Anywhere 10 インストール環境のサブディレクトリ *docs¥en¥javadocs¥QAnywhere* から Javadoc フォーマットで入手することもできます。
- SQL Anywhere インストール環境のサブディレクトリ *win32* にあるランタイム DLL (*qany10jni.dll and qany10.dll*)
- SQL Anywhere 10 インストール環境のサブディレクトリ *java* にあるクラス・ファイルのアーカイブ (*qaclient.jar*)

クラス・ファイルのアーカイブは、アプリケーションのコンパイル時にパスに含まれている必要があります。ランタイム DLL は、アプリケーションとともに配備する必要があります。

Java バージョンの TestMessage サンプル・アプリケーションは、*samples-dir¥QAnywhere¥Java¥* にあります。*samples-dir* の詳細については、「サンプル・ディレクトリ」『SQL Anywhere サーバ - データベース管理』を参照してください。

「QAnywhere Java API リファレンス」 531 ページを参照してください。

QAnywhere SQL API

QAnywhere SQL API は、SQL を使用してメッセージング API を実装するストアド・プロシージャの集まりです。QAnywhere SQL API を使用して、メッセージの作成、メッセージのプロパティと内容の取得や設定、メッセージの送受信、メッセージ同期のトリガ、メッセージ・ストア・プロパティの取得と設定を実行できます。

「QAnywhere SQL API リファレンス」 675 ページを参照してください。

JMS コネクタ

QAnywhere には、QAnywhere と JMS アプリケーション間を接続する JMS コネクタが含まれています。次の項を参照してください。

- 「外部メッセージング・システムとのメッセージングのシナリオ」 10 ページ
- 「JMS コネクタの概要」 136 ページ
- 「チュートリアル：JMS コネクタの使用」 152 ページ

モバイル Web サービス・コネクタ

QAnywhere には、QAnywhere と Web サービス間でメッセージングを行うためのモバイル Web サービス・コネクタが含まれています。

「モバイル Web サービス」 195 ページを参照してください。

クライアント・アプリケーション作成のクイック・スタート

♦ **クライアント・アプリケーション設定の概要**

1. 適切な QAnywhere API を初期化します。次の項を参照してください。

   - 「.NET アプリケーションの設定」 61 ページ
   - 「C++ アプリケーションの設定」 63 ページ
   - 「Java アプリケーションの設定」 65 ページ
   - 「SQL アプリケーションの設定」 66 ページ

2. QAnywhere Manager の設定プロパティを設定します。「QAnywhere Manager の設定プロパティ」 69 ページを参照してください。

3. アプリケーション・コードを作成し、コンパイルします。次の項を参照してください。

   - 「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページ
   - 「クライアント・メッセージ・ストア・プロパティ」 230 ページ
   - 「QAnywhere メッセージの送信」 73 ページ
   - 「QAnywhere メッセージの受信」 82 ページ
   - 「サイズの大きなメッセージの読み込み」 87 ページ
   - 「トランザクション志向メッセージングの実装」 75 ページ
   - 「QAnywhere の停止」 96 ページ

4. アプリケーションをターゲット・デバイスに配備します。

   「QAnywhere アプリケーションの配備」 97 ページを参照してください。

クイック・スタートのためのその他の資料

- 「チュートリアル：TestMessage の解説」 15 ページ
- *samples-dir¥QAnywhere* にサンプル・アプリケーションがあります。*samples-dir* の詳細については、「サンプル・ディレクトリ」『SQL Anywhere サーバ - データベース管理』を参照してください。

QAnywhere メッセージ・アドレス

QAnywhere メッセージのアドレスは、クライアント・メッセージ・ストア ID とアプリケーション・キュー名の 2 つの部分で構成されています。

```
id¥queue-name
```

キュー名は、アプリケーション内で指定します。他のデバイスの送信アプリケーションのインスタンスも、このキュー名を認識していなければなりません。クライアント・メッセージ・ストア ID の詳細については、「クライアント・メッセージ・ストアの設定」 36 ページを参照してください。

アプリケーション内でアドレスを文字列として指定する場合は、必要に応じて円記号をエスケープする必要があります。使用しているプログラミング言語の文字列エスケープ規則に従ってください。JMS の送信先に円記号が含まれている場合は、円記号をもう 1 つ追加して、この円記号をエスケープする必要があります。

アドレスの長さは 255 文字以内です。

システム・キュー

通知とネットワーク・ステータスの変化は、どちらも**システム・メッセージ**として QAnywhere アプリケーションに送信されます。システム・メッセージは、**system** という名前の専用のキューで受信される点を除けば、他のメッセージと変わりません。

「システム・キュー」 58 ページを参照してください。

JMS コネクタへのメッセージの送信

QAnywhere から JMS に送信されるメッセージの送信先アドレスは、次の 2 つの部分で構成されています。

- コネクタ・アドレス。これは、ianywhere.connector.address プロパティの値です。

  「JMS コネクタ・プロパティ」 141 ページを参照してください。

- JMS キュー名。JMS 管理ツールを使用して作成したキューの名前です。

送信先アドレスの形式は次のとおりです。

```
connector-address¥JMS-queue-name
```

JMS アプリケーション内でメッセージ・アドレスを指定する方法の詳細については、次の項を参照してください。

- 「QAnywhere から JMS に送信されるメッセージのアドレス指定」 146 ページ
- 「JMS から QAnywhere に送信されるメッセージのアドレス指定」 149 ページ
- 「JMS コネクタ」 135 ページ

送信先エイリアス

「**送信先エイリアス**」は、メッセージ・アドレスとその他の送信先エイリアスのリストです。送信先エイリアスに送信されるメッセージは、このリストのすべてのメンバに送信されます。

送信先エイリアスのメンバに対して、配信条件を関連付けることができます。この場合、条件に一致するメッセージだけが該当するメンバに転送されます。

例

client1 と client2 というメンバを含む送信先エイリアス all_clients を定義します。

client1 に次の配信条件を定義します。

```
ias_Priority=1
```

client2 に次の配信条件を定義します。

```
ias_Priority=9
```

優先度が 1 のメッセージだけが client1 に送信され、優先度が 9 のメッセージだけが client2 に送信されます。

送信先エイリアスの作成

送信先エイリアスは、次のいずれかの方法で作成および管理できます。

- サーバ管理要求

  「サーバ管理要求を使用した送信先エイリアスの作成」 125 ページを参照してください。

- Sybase Central

  「Sybase Central を使用した送信先エイリアスの作成」 57 ページを参照してください。

Sybase Central を使用した送信先エイリアスの作成

Sybase Central を使用して、送信先アドレスを作成したり変更したりできます。

- **Sybase Central を使用して送信先エイリアスを作成するには、次の手順に従います。**

1. Sybase Central を起動します。
   - [スタート] – [プログラム] – [SQL Anywhere 10] – [Sybase Central] を選択します。
   - [接続] – [QAnywhere 10 に接続] を選択します。
   - ODBC データ・ソース名または ODBC データ・ソース・ファイルを指定し、必要に応じてユーザ ID とパスワードを指定します。
   - [OK] をクリックします。
2. [ファイル] – [新規] – [送信先エイリアス] を選択します。

3. [エイリアス] フィールドにエイリアス名を入力します。
4. [送信先] フィールドで、各送信先の行に送信先名を入力します。
5. [OK] をクリックします。

システム・キュー

システム・キューは、QAnywhere システム・メッセージを受信するための特別なキューです。システム・キューに送信されるメッセージには、次の 2 種類があります。

- 「ネットワーク・ステータス通知」 59 ページ
- 「Push 通知の通知」 59 ページ

例

次の C# コードは、システム・メッセージと通常のメッセージを処理します。これは、オンデマンド・ポリシーを使用している場合に便利です。この例では、メッセージ処理のためのアプリケーション論理を実装したメッセージ処理メソッド onMessage() と onSystemMessage() が、すでに定義されているとものとします。

```
// Declare the message listener and system listener.
private QAManager.MessageListener _receiveListener;
private QAManager.MessageListener _systemListener;
...

// Create a MessageListener that uses the appropriate message handlers.
_receiveListener = new QAManager.MessageListener( onMessage );
_systemListener = new QAManager.MessageListener( onSystemMessage );
...

// Register the message handler.
mgr.SetMessageListener( queue-name, _receiveListener );
mgr.SetMessageListener( "system", _systemListener );
```

システム・メッセージ・ハンドラは、メッセージ・プロパティを参照して、プロパティに設定されている情報を確認します。メッセージ・タイプ・プロパティの値によって、メッセージがネットワーク・ステータス通知を保持しているかどうかを判定できます。たとえば、メッセージ msg に対して次の処理を実行できます。

```
msg_type = (MessageType)msg.GetIntProperty( MessageProperties.MSG_TYPE );
if( msg_type == MessageType.NETWORK_STATUS_NOTIFICATION ) {
  // Process a network status change.
  mgr.TriggerSendReceive( );
} else if ( msg_type == MessageType.PUSH_NOTIFICATION ) {
  // Process a push notification.
  mgr.TriggerSendReceive( );
} else if ( msg_type == MessageType.REGULAR ) {
  // This message type should not be received on the
  // system queue. Take appropriate action here.
}
```


ネットワーク・ステータス通知

ネットワーク・ステータスが変化すると、メッセージ・タイプ NETWORK_STATUS_NOTIFICATION がシステム・キューに送信されます。このメッセージ・タイプの有効期限は 1 分です。この有効期限は変更できません。

デバイスがネットワーク・カバレッジの範囲内に入ったり、範囲外になると、次の情報を含むメッセージがシステム・キューに送信されます。

- **ias_Adapters** 文字列。Mobile Link サーバへの接続で使用できるネットワーク・アダプタのリスト。各文字列は縦棒で区切られます。このプロパティは読み込み可能ですが、設定はできません。次の項を参照してください。
  - .NET :「ADAPTER フィールド」 266 ページ
  - C++ :「ADAPTER 変数」 425 ページ
  - Java :「ADAPTERS 変数」 535 ページ
- **ias_RASNames** 文字列。Mobile Link サーバへの接続で使用できるネットワーク名のリスト。各文字列は縦棒で区切られます。次の項を参照してください。
  - .NET :「RASNAMES フィールド」 271 ページ
  - C++ :「RASNAMES 変数」 429 ページ
  - Java :「RASNAMES 変数」 538 ページ
- **ias_NetworkStatus** 整数。ネットワーク接続のステータス。値 1 は 接続済み、0 はそれ以外を意味します。次の項を参照してください。
  - .NET :「NETWORK_STATUS フィールド」 269 ページ
  - C++ :「NETWORK_STATUS 変数」 428 ページ
  - Java :「NETWORK_STATUS 変数」 537 ページ

ネットワークの可用性のモニタリング

ネットワーク・ステータス通知を使用すると、ネットワークの可用性をモニタリングして、デバイスがネットワーク・カバレッジの範囲内に入ったときに何らかのアクションを実行できます。たとえば、タイプ NETWORK_STATUS_NOTIFICATION のシステム・キュー・メッセージを ias_NetworkStatus=1 のステータスで受信した場合、オンデマンド・ポリシーを使用して QAManagerBase triggerSendReceive を呼び出します。

参照

- 「事前に定義されたメッセージ・プロパティ」 224 ページの ias_MessageType
- 「システム・キュー」 56 ページ

Push 通知の通知

Push 通知がサーバから受信されると、PUSH_NOTIFICATION メッセージ・タイプがシステム・キューに送信されます。これは、メッセージがサーバのキューに登録されたことを通知するメッセージです。このメッセージ・タイプの有効期限は 1 分です。この有効期限は変更できません。

オンデマンド・ポリシーを使用している場合は、このタイプのシステム・メッセージが便利です。たとえば、タイプ PUSH_NOTIFICATION のシステム・キュー・メッセージを受信した場合に、QAManagerBase triggerSendReceive を呼び出すことができます。

参照

- 「Push 通知によるメッセージングのシナリオ」 9 ページ
- 「Push 通知の使用方法」 45 ページ
- 「システム・キュー」 56 ページ
- 「非同期的なメッセージ受信」 83 ページ
- 「事前に定義されたメッセージ・プロパティ」 224 ページの ias_MessageType
- .NET：「MessageProperties クラス」 264 ページ
- C++：「MessageProperties クラス」 424 ページ
- Java：「MessageProperties インタフェース」 533 ページ

QAnywhere API の初期化

QAnywhere を使用してメッセージを送信または受信する前に、次の初期化処理を実行する必要があります。

.NET アプリケーションの設定

QAnywhere .NET クライアントを使用してメッセージを送信または受信する前に、次の初期化処理を実行する必要があります。

Visual Studio .NET プロジェクトを使用するには、次の 2 つの変更を行う必要があります。

- QAnywhere .NET DLL への参照を追加する。参照を追加すると、QAnywhere .NET API のコードを検索するためにインクルードする必要がある DLL を Visual Studio.NET に認識させることができます。
- QAnywhere .NET API のクラスを参照する行をソース・コードに追加する。QAnywhere .NET API を使用するには、データ・プロバイダを参照する行もソース・コードに追加する必要があります。C# では、Visual Basic .NET 用とは異なる行を追加します。

これらの作業を行った上で、QAnywhere .NET API を初期化してください。

♦ Visual Studio .NET のプロジェクトに QAnywhere .NET API への参照を追加するには、次の手順に従います。

1. Visual Studio .NET を起動し、プロジェクトを開きます。
2. [ソリューション エクスプローラ] ウィンドウで、[参照設定] フォルダを右クリックし、ポップアップ・メニューから [参照の追加] を選択します。

   [参照の追加] ダイアログが表示されます。
3. [.NET] タブで [参照] をクリックして、iAnywhere.QAnywhere.Client.dll を見つけます。デフォルト・ロケーションは、SQL Anywhere のインストール・ディレクトリを基準とした相対ディレクトリにあります。

   - .NET Framework 1.1：*Assembly¥v1*
   - .NET Framework 2.0：*Assembly¥v2*
   - .NET Compact Framework 1.0：*ce¥Assembly¥v1*
   - .NET Compact Framework 2.0：*ce¥Assembly¥v2*

   DLL を選択して [開く] をクリックします。
4. DLL がプロジェクトに追加されているかどうかを確認できます。[参照の追加] ダイアログを開き、[.NET] タブをクリックします。[選択されたコンポーネント] リストに iAnywhere.QAnywhere.Client.dll が表示されます。[OK] をクリックしてダイアログを閉じます。

ソース・コードのデータ・プロバイダ・クラスを参照する

♦ **コード内で QAnywhere .NET API のクラスを参照するには、次の手順に従います。**

1. Visual Studio .NET を起動し、プロジェクトを開きます。
2. C# を使用している場合は、ファイルの先頭にある using ディレクティブのリストに次の行を追加します。

   ```
   using iAnywhere.QAnywhere.Client;
   ```

3. Visual Basic .NET を使用している場合は、ファイルの先頭にある imports リストに次の行を追加します。

   ```
   Imports iAnywhere.QAnywhere.Client
   ```

   この行は必須ではありません。しかし、この行を追加すると、QAnywhere クラスの省略形を使用できます。この行を追加しなくても、完全に修飾されたクラス名をコードで使用できます。例：

   ```
   iAnywhere.QAnywhere.Client.QAManager
   mgr =
    new iAnywhere.QAnywhere.Client.QAManagerFactory.Instance.CreateQAManager(
   "qa_manager.props" );
   ```

   一方、追加した場合には、次のようにクラス名を指定できます。

   ```
   QAManager mgr = QAManagerFactory.Instance.CreateQAManager(
    "qa_manager.props" );
   ```

♦ **QAnywhere .NET API を初期化するには、次の手順に従います。**

1. 前の手順の説明に従って、iAnywhere.QAnywhere.Client ネームスペースをインクルードします。

   ```
   using iAnywhere.QAnywhere.Client;
   ```

2. QAManager オブジェクトを作成します。

   たとえば、デフォルトの QAManager オブジェクトを作成するには、パラメータに null を指定して、CreateQAManager を呼び出します。

   ```
   QAManager mgr;
   mgr = QAManagerFactory.Instance.CreateQAManager( null );
   ```

   ヒント
   同時実行性の利点を最大限に生かすために、マルチスレッド・アプリケーションでは、スレッドごとに QAManager を作成する必要があります。「マルチスレッド QAManager」 68 ページを参照してください。

   QAManagerFactory の詳細については、「QAManagerFactory クラス」 339 ページを参照してください。

プロパティ・ファイルを使用して、カスタマイズされた QAManager オブジェクトを作成することもできます。次のように、CreateQAManager メソッドの引数としてプロパティ・ファイルを指定します。

```
mgr = QAManagerFactory.Instance.CreateQAManager(
  "qa_mgr.props" );
```

qa_mgr.props は、リモート・デバイス上に存在するプロパティ・ファイルの名前です。

3. QAManager オブジェクトを初期化します。次に例を示します。

```
mgr.Open(
  AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
```

open メソッドの引数には、メッセージの受信確認方法を示す受信確認モードを指定します。これは、IMPLICIT_ACKNOWLEDGEMENT または EXPLICIT_ACKNOWLEDGEMENT のどちらかにする必要があります。前者は暗黙的な受信確認モードを意味します。このモードの場合、メッセージはクライアントで受信されるとすぐに受信確認されます。後者は明示的な受信確認モードを意味します。このモードの場合、QAManager の Acknowledge メソッドを呼び出してメッセージを受信確認する必要があります。

受信確認モードの詳細については、「AcknowledgementMode 列挙」 262 ページを参照してください。

これで、メッセージを送信する準備ができました。

QAManager を作成する代わりに、QATransactionalManager を作成できます。「トランザクション志向メッセージングの実装 (.NET クライアントの場合)」 75 ページを参照してください。

参照

- 「iAnywhere.QAnywhere.Client ネームスペース (.NET 1.0)」 262 ページ

C++ アプリケーションの設定

QAnywhere C++ クライアントを使用してメッセージを送信または受信する前に、次の初期化処理を実行する必要があります。

♦ **QAnywhere C++ API を初期化するには、次の手順に従います。**

1. QAnywhere ヘッダ・ファイルをインクルードします。

```
#include <qa.hpp>
```

qa.hpp には、QAnywhere のさまざまなクラスが定義されています。

2. QAnywhere を初期化します。

そのためには、QAManager オブジェクトを作成するためのファクトリを初期化します。


```
QAManagerFactory * factory;

factory = QAnywhereFactory_init();
if( factory == NULL ) {
    // Fatal error.
}
```

QAManagerFactory の詳細については、「QAManagerFactory クラス」 490 ページを参照してください。

3. QAManager インスタンスを作成します。

   デフォルトの QAManager オブジェクトを作成するには、次のようにします。

```
QAManager *  mgr;

// Create a manager
mgr = factory->createQAManager( NULL );
if( mgr == NULL ) {
  // fatal error
}
```

「QAManager クラス」 455 ページを参照してください。

> **ヒント**
> 同時実行性の利点を最大限に生かすために、マルチスレッド・アプリケーションでは、スレッドごとに QAManager を作成する必要があります。「マルチスレッド QAManager」 68 ページを参照してください。

QAManager オブジェクトは、プログラムで、またはプロパティ・ファイルを使用してカスタマイズできます。

- QAManager をプログラムでカスタマイズする場合は、setProperty() を使用します。

  「プログラムによる QAnywhere Manager の設定プロパティの設定」 71 ページを参照してください。

- プロパティ・ファイルを使用する場合は、createQAManager() でプロパティ・ファイルを指定します。

```
mgr = factory->createQAManager( "qa_mgr.props" );
```

  qa_mgr.props は、リモート・デバイス上のプロパティ・ファイルの名前です。

  「QAnywhere Manager の設定プロパティをファイルに設定する」 70 ページを参照してください。

4. QAManager オブジェクトを初期化します。

```
qa_bool  rc;
rc=mgr->open(
    AcknowledgementMode::IMPLICIT_ACKNOWLEDGEMENT );
```

open メソッドの引数には、メッセージの受信確認方法を示す受信確認モードを指定します。これは、**IMPLICIT_ACKNOWLEDGEMENT** または **EXPLICIT_ACKNOWLEDGEMENT** のどちらかにする必要があります。前者は暗黙的な受信確認モードを意味します。このモードの場合、メッセージはクライアントで受信されるとすぐに受信確認されます。後者は明示的な受信確認モードを意味します。このモードの場合、QAManager のいずれかの acknowledgement メソッドを呼び出してメッセージを受信確認する必要があります。

受信確認モードの詳細については、「AcknowledgementMode クラス」 422 ページを参照してください。

QAManager を作成する代わりに、QATransactionalManager を作成できます。「トランザクション志向メッセージングの実装 (C++ クライアントの場合)」 77 ページを参照してください。

これで、メッセージを送信する準備ができました。

参照

- 「QAnywhere C++ API リファレンス」 421 ページ

Java アプリケーションの設定

QAnywhere Java クライアントを使用してメッセージを送信または受信する前に、次の初期化処理を実行する必要があります。

♦ QAnywhere Java API を初期化するには、次の手順に従います。

1. *qaclient.jar* のロケーションをクラスパスに追加します。このファイルは、デフォルトで SQL Anywhere インストール環境の *java* サブディレクトリにあります。
2. ianywhere.qanywhere.client パッケージをインポートします。

   ```
   import ianywhere.qanywhere.client.*;
   ```

3. QAManager オブジェクトを作成します。

   ```
   QAManager mgr;
   mgr = QAManagerFactory.getInstance().createQAManager(null);
   ```

   createQAManager メソッドにプロパティ・ファイルを指定して、QAManager オブジェクトをカスタマイズすることもできます。

   ```
   mgr = QAManagerFactory.getInstance().createQAManager("qa_mgr.props.");
   ```

ヒント
同時実行性の利点を最大限に生かすために、マルチスレッド・アプリケーションでは、スレッドごとに QAManager を作成する必要があります。「マルチスレッド QAManager」 68 ページを参照してください。

4. QAManager オブジェクトを初期化します。

```
mgr.open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
```

open メソッドの引数には、メッセージの受信確認方法を示す受信確認モードを指定します。これは、IMPLICIT_ACKNOWLEDGEMENT または EXPLICIT_ACKNOWLEDGEMENT のどちらかにする必要があります。前者は暗黙的な受信確認モードを意味します。このモードの場合、メッセージはクライアントで受信されるとすぐに受信確認されます。後者は明示的な受信確認モードを意味します。このモードの場合、QAManager のいずれかの acknowledgement メソッドを呼び出してメッセージを受信確認する必要があります。

受信確認モードの詳細については、「AcknowledgementMode インタフェース」 532 ページを参照してください。

> QAManager を作成する代わりに、QATransactionalManager を作成できます。「トランザクション志向メッセージングの実装 (Java クライアントの場合)」 78 ページを参照してください。

これで、メッセージを送信する準備ができました。

参照

- 「QAnywhere Java API リファレンス」 531 ページ

SQL アプリケーションの設定

QAnywhere SQL を使用すると、QAnywhere .NET、C++、Java API のほとんどのメッセージング機能を SQL で実行できます。たとえば、メッセージの作成、メッセージ・プロパティと内容の設定や取得、メッセージの送受信、メッセージ同期のトリガ、メッセージ・ストア・プロパティの設定と取得などの機能を実行できます。

QAnywhere SQL で生成されたメッセージを、プログラミング API で作成されたクライアントで受信することもできます。JMS コネクタがサーバ上で設定されている場合は、JMS クライアントがこのメッセージを受信することもできます。同様に、QAnywhere .NET、C++、Java API、または JMS のクライアントで生成されたメッセージを、QAnywhere SQL で受信することもできます。

QAnywhere SQL メッセージングは、ユーザ・トランザクションと共存できます。つまり、トランザクションをコミットすると、その接続でのすべての QAnywhere 操作がコミットされます。

「QAnywhere クライアント・アプリケーションの作成」 51 ページを参照してください。

パーミッション

QAnywhere ストアド・プロシージャを実行するパーミッションを自動的に与えられるのは、DBA 権限を持つユーザだけです。ユーザにパーミッションを付与するには、DBA 権限を持つユーザが ml_qa_grant_messaging_permissions プロシージャを呼び出す必要があります。

「ml_qa_grant_messaging_permissions」 711 ページを参照してください。

受信確認モード

QAnywhere SQL API では、IMPLICIT_ACKNOWLEDGEMENT と EXPLICIT_ACKNOWLEDGEMENT のどちらのモードもサポートしません。SQL API を介するメッセージは、すべてトランザクション志向です。

例

次の例では、在庫テーブルのトリガを作成します。このトリガは、ある品目の在庫が特定のしきい値を下回ると、メッセージを送信します。メッセージは、トリガを呼び出したトランザクションがコミットされた後で送信されます。トランザクションがロールバックされた場合、メッセージは送信されません。

```
CREATE TRIGGER inventory_trigger AFTER UPDATE ON inventory
REFERENCING old AS oldinv new AS newinv
FOR EACH ROW
begin
  DECLARE msgid VARCHAR(128);
  IF oldinv.quantity > newinv.quantity AND newinv.quantity < 10 THEN
    -- Create the message
    SET msgid = ml_qa_createmessage();
    -- Set the message content
    CALL ml_qa_settextcontent( msgid,
      'Inventory of item ' || newinv.itemname
        || ' has fallen to only ' || newinv.quantity );
    -- Make the message high priority
    CALL ml_qa_setpriority( msgid, 9 );
    -- Set a message subject
    CALL ml_qa_setstringproperty( msgid,
      'tm_Subject', 'Inventory low!' );
    -- Send the message to the inventoryManager queue
    CALL ml_qa_putmessage( msgid,
      'inventoryManager' );
  end if;
end
```

参照

- 「QAnywhere SQL API リファレンス」 675 ページ

マルチスレッド QAManager

QAManager へのアクセスは直列化されます。1 つの QAManager へのアクセスにマルチスレッドを使用すると、あるスレッドが QAManager に対してメソッド呼び出しを実行している間、他のスレッドはブロックします。同時実行性を最大化するためには、スレッドごとに異なる QAManager を使用します。QAManager のインスタンスに同時にアクセスできるスレッドは、1 つだけです。その他のスレッドは、最初のスレッドによって呼び出された QAManager メソッドが戻るまで、ブロックします。

QAnywhere Manager の設定プロパティ

QAnywhere Manager の設定プロパティは、次のいずれかの方法で設定できます。

- QAnywhere Manager の設定プロパティが定義されているプロパティ・テキスト・ファイルを作成し、このプロパティ・ファイルを 1 つの QAnywhere Manager インスタンスで使用する。

  「QAnywhere Manager の設定プロパティをファイルに設定する」 70 ページを参照してください。

- プログラムで QAnywhere Manager の設定プロパティを設定する。

  「プログラムによる QAnywhere Manager の設定プロパティの設定」 71 ページを参照してください。

次に、QAnywhere Manager の設定プロパティを示します。

- **COMPRESSION_LEVEL=n**　圧縮レベルを設定します。

  n は圧縮率です。これは 0 ー 9 の整数で指定し、0 は圧縮なし、9 は最大の圧縮率を表します。

- **CONNECT_PARAMS=connect-string**　QAnywhere Manager がメッセージ・ストア・データベースに接続する場合に使用する接続文字列を指定します。接続オプションは *keyword=value* の形式で指定します。複数のオプションを指定する場合は、各オプションをセミコロンで区切ります。

  デフォルトは "eng=qanywhere;uid=ml_qa_user;pwd=qanywhere" です。

  オプションのリストについては、「接続パラメータ」 『SQL Anywhere サーバ - データベース管理』を参照してください。

  データベース・ユーザとパスワードの管理の詳細については、「セキュリティ保護されたメッセージング・アプリケーションの作成」 189 ページを参照してください。

- **LOG_FILE=filename**　ログ・メッセージを書き込むファイルの名前を指定します。このオプションを指定すると、暗黙的にロギングが有効になります。

- **MAX_IN_MEMORY_MESSAGE_SIZE=n**　メッセージの読み込み時、バッファに割り当てられるメッセージの最大値を *n* バイトにします。サイズが *n* バイトを超えるメッセージは、ストリーム操作を使用して読み込む必要があります。デフォルト値は、Windows では 1MB、Windows CE では 64KB です。

QAnywhere Manager の設定プロパティをファイルに設定する

> **注意**
> QAnywhere Manager の設定ファイルは Sybase Central で作成したり、開いたりできます。QAnywhere プラグインのタスクのウィンドウ枠で、[エージェント設定ファイルの新規作成] を選択します。ファイル名とロケーションを選択したら、設定ファイルの [プロパティ] ダイアログが開き、このダイアログでプロパティを設定できます。

QAnywhere Manager のプロパティ・ファイルに記述された情報は、1 つの QAManager インスタンスに固有のものです。

プロパティ・ファイルを使用する場合は、配備したアプリケーションごとに、リモート・デバイス上でプロパティ・ファイルの設定とインストールを行う必要があります。

プロパティ・ファイル名の指定の詳細については、次の項を参照してください。

- .NET API：「CreateQAManager メソッド」 341 ページ
- C++ API：「createQAManager 関数」 490 ページ
- Java API：「createQAManager メソッド」 602 ページ
- SQL API：QAnywhere SQL API を使用して、プロパティをファイルに設定することはできません。「プログラムによる QAnywhere Manager の設定プロパティの設定」 71 ページを参照してください。

使用するプロパティ・ファイルがクライアントの実行プログラムと同じディレクトリに存在しない場合は、絶対パスで指定する必要があります。すべてのプロパティでデフォルトの設定を使用する場合は、ファイル名の代わりに NULL を指定します。

プロパティ・ファイル内で値を設定することによって、自動メッセージ圧縮やロギングなどの QAnywhere の一部の機能を有効または無効にすることができます。

QAnywhere Manager の設定プロパティ・ファイルのエントリは、*name*=*value* の形式を取ります。プロパティ名のリストについては、「QAnywhere Manager の設定プロパティ」 69 ページを参照してください。*value* にスペースが含まれる場合は、スペースを二重引用符で囲みます。コメント行の先頭には **#** を付けます。次に例を示します。

```
# contents of QAnywhere manager configuration properties file
LOG_FILE=.¥sender.ini.txt
# A comment
CONNECT_PARAMS=eng=qanywhere;uid=ml_qa_user;pwd=qanywhere
MAX_IN_MEMORY_MESSAGE_SIZE=2048
COMPRESSION_LEVEL=0
```

設定ファイルの参照

次のような内容の QAnywhere Manager の設定プロパティ・ファイル *mymanager.props* があるとします。

```
COMPRESSION_LEVEL=9
CONNECT_PARMS=DBF=mystore.db
```

QAManager を生成するときに、このファイルの名前を引数に指定します。

次は、C# のコード例です。

```
QAManager     mgr;
mgr = QAManagerFactory.Instance.CreateQAManager( “mymanager.props” );
mgr.Open( AcknowledgeMode.EXPLICIT_ACKNOWLEDGEMENT );
```

.NET API については、「QAManager インタフェース」 292 ページと「QAManagerFactory クラス」 339 ページを参照してください。

次は、C++ のコード例です。

```
QAManagerFactory *  qa_factory;
QAManager *    mgr;
qa_factory = QAnywhereFactory_init();
mgr = qa_factory->createQAManager( “mymanager.props” );
mgr->open( AcknowledgementMode::EXPLICIT_ACKNOWLEDGEMENT );
```

C++ API については、「QAManager クラス」 455 ページと「QAManagerFactory クラス」 490 ページを参照してください。

次は、Java のコード例です。

```
QAManager mgr;
mgr = QAManagerFactory.getInstance().createQAManager("mymanager.props");
mgr.open( AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT );
```

Java API については、「QAManagerFactory クラス」 602 ページと「QAManager インタフェース」 566 ページを参照してください。

プログラムによる QAnywhere Manager の設定プロパティの設定

QAnywhere API では QAManagerBase set プロパティ・メソッドを使用して、プロパティをプログラムで設定できます。QAnywhere Manager の設定プロパティをプログラムで設定する場合は、QAManager インスタンスの open メソッドを呼び出す前に設定する必要があります。

QAManager のプロパティの詳細については、「QAnywhere Manager の設定プロパティ」 69 ページを参照してください。

例

次の C# のコードでは、各プロパティをプログラムで直接設定しています。QAManager を作成した後すぐにプロパティの設定を行っています。

```
QAManager     mgr;
mgr = QAManagerFactory.Instance.CreateQAManager( null );
mgr.SetProperty( “COMPRESSION_LEVEL”, “9” );
mgr.SetProperty( “CONNECT_PARAMS”, “DBF=mystore.db” );
mgr.Open( AcknowledgeMode.EXPLICIT_ACKNOWLEDGEMENT );
```

.NET API については、「QAManager インタフェース」 292 ページと「QAManagerFactory クラス」 339 ページを参照してください。

次の C++ のコードでは、各プロパティをプログラムで直接設定しています。QAManager を作成した後すぐにプロパティの設定を行っています。

```
QAManagerFactory *  qa_factory;
QAManager *    mgr;
```



```
qa_factory = QAnywhereFactory_init();
mgr = qa_factory->createQAManager( NULL );
mgr->setProperty( “COMPRESSION_LEVEL”, “9” );
mgr->setProperty( “CONNECT_PARAMS”, “DBF=mystore.db” );
mgr->open( AcknowledgementMode::EXPLICIT_ACKNOWLEDGEMENT );
```

C++ API については、「QAManager クラス」 455 ページと「QAManagerFactory クラス」 490 ページを参照してください。

次の Java のコードでは、各プロパティをプログラムで直接設定しています。QAManager を作成した後すぐにプロパティの設定を行っています。

```
QAManager mgr;
mgr = QAManagerFactory.getInstance().createQAManager(null);
mgr.setProperty("COMPRESSION_LEVEL", 9);
mgr.setStringProperty("CONNECT_PARMS", "DBF=mystore.db");
mgr.open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
```

Java API については、「QAManagerFactory クラス」 602 ページと「QAManager インタフェース」 566 ページを参照してください。

QAnywhere メッセージの送信

以下の手順では、QAnywhere アプリケーションからメッセージを送信する方法を説明します。これらの手順では、QAManager オブジェクトがすでに作成され、オープンされているものとします。

アプリケーションからメッセージを送信しても、そのメッセージがデバイスから配信されるという保証はありません。アプリケーションは、メッセージ配信のためにキューにメッセージを登録するだけです。QAnywhere Agent がそのメッセージを Mobile Link サーバに送信し、その後 Mobile Link サーバが実際に送信先にメッセージを配信します。

メッセージ転送が行われるタイミングの詳細については、「クライアントにメッセージを転送するタイミングの決定」 40 ページを参照してください。

♦ **メッセージを送信するには、次の手順に従います (.NET の場合)。**

1. 新しいメッセージを作成します。

   CreateTextMessage() を使用してテキスト・メッセージを作成することも、CreateBinaryMessage() を使用してバイナリ・メッセージを作成することもできます。

   ```
   QATextMessage    msg;
   msg = mgr.CreateTextMessage();
   ```

2. メッセージのプロパティを設定します。

   QATextMessage クラスまたは QABinaryMessage クラスのメソッドを使用してプロパティを設定します。

   「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

3. メッセージをキューに登録します。これで送信準備が完了します。

   ```
   mgr.PutMessage( "store-id¥¥queue-name", msg );
   ```

   store-id と *queue-name* を連結した文字列が送信先アドレスになります。

   「PutMessage メソッド」 322 ページと「クライアントにメッセージを転送するタイミングの決定」 40 ページを参照してください。

♦ **メッセージを送信するには、次の手順に従います (C++ の場合)。**

1. 新しいメッセージを作成します。

   createTextMessage() を使用してテキスト・メッセージを作成することも、createBinaryMessage() を使用してバイナリ・メッセージを作成することもできます。

   ```
   QATextMessage *   msg;
   msg = mgr->createTextMessage();
   ```

2. メッセージのプロパティを設定します。

   QATextMessage クラスまたは QABinaryMessage クラスのメソッドを使用して、メッセージのプロパティを設定します。

「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

3. メッセージをキューに登録します。これで送信準備が完了します。

```
if( msg != NULL ) {
  if( !mgr->putMessage( "store-id¥¥queue-name", msg ) ) {
    // Display error using mgr->getLastErrorMsg().
    }
    mgr->deleteMessage( msg );
}
```

store-id と *queue-name* を連結した文字列が送信先アドレスになります。

「putMessage 関数」 480 ページと「クライアントにメッセージを転送するタイミングの決定」 40 ページを参照してください。

♦ **メッセージを送信するには、次の手順に従います (Java の場合)。**

1. 新しいメッセージを作成します。

   QAManagerBase.createTextMessage() を使用してテキスト・メッセージを作成することも、QAManagerBase.createBinaryMessage() を使用してバイナリ・メッセージを作成することもできます。

```
QATextMessage msg;
msg = mgr.createTextMessage();
```

2. メッセージのプロパティを設定します。

   QATextMessage メソッドまたは QABinaryMessage メソッドを使用して、メッセージのプロパティを設定します。

「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

3. メッセージをキューに登録します。

```
mgr.putMessage("store-id¥¥queue-name", msg);
```

「putMessage メソッド」 591 ページと「クライアントにメッセージを転送するタイミングの決定」 40 ページを参照してください。

♦ **メッセージを送信するには、次の手順に従います (SQL の場合)。**

1. メッセージ ID を格納する変数を宣言します。

```
begin
   declare @msgid varchar(128);
```

2. 新しいメッセージを作成します。

```
   set @msgid = ml_qa_createmessage();
```

3. メッセージのプロパティを設定します。

   詳細については、「メッセージ・プロパティ」 686 ページを参照してください。

4. メッセージをキューに登録します。

```
    call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
    commit;
end
```

「ml_qa_putmessage」 713 ページと「クライアントにメッセージを転送するタイミングの決定」 40 ページを参照してください。

トランザクション志向メッセージングの実装

トランザクション志向メッセージングを使用すると、メッセージがグループにまとめられ、グループ内のすべてのメッセージが配信されるか、すべてのメッセージが配信されないかのどちらかになることが保証されます。このような処理は、一般に**トランザクション**と呼ばれています。

トランザクション志向メッセージングを実装するには、QATransactionalManager と呼ばれる特殊な QAManagerBase オブジェクトを作成します。

詳細については、次の項を参照してください。

- .NET クライアント：「QATransactionalManager インタフェース」 367 ページ
- C++ クライアント：「QATransactionalManager クラス」 521 ページ
- Java クライアント：「QATransactionalManager インタフェース」 632 ページ
- SQL クライアント：SQL クライアントでは、すべてのメッセージングがトランザクション志向であるため、トランザクション志向マネージャは不要です。

トランザクション志向メッセージングの実装 (.NET クライアントの場合)

- **トランザクション志向マネージャを作成するには、次の手順に従います。**

1. QAnywhere を初期化します。

   この手順は、非トランザクション志向メッセージングの場合と同じです。

   ```
   using iAnywhere.QAnywhere.Client;
   ```

2. QATransactionalManager オブジェクトを作成します。

   たとえば、デフォルトの QATransactionalManager オブジェクトを作成するには、パラメータに null を指定して、CreateQATransactionalManager を呼び出します。

   ```
   QAManager mgr;
   mgr =
     QAManagerFactory.Instance.CreateQATransactionalManager(
     null );
   ```

   「QAManagerFactory クラス」 339 ページを参照してください。

   プロパティ・ファイルを使用して、カスタマイズされた QATransactionalManager オブジェクトを作成することもできます。次のように、CreateQATransactionalManager メソッドの引数としてプロパティ・ファイルを指定します。


```
mgr =
  QAManagerFactory.Instance.CreateQATransactionalManager(
  "qa_mgr.props" );
```

qa_mgr.props は、リモート・デバイス上に存在するプロパティ・ファイルの名前です。

3. QAManager オブジェクトを初期化します。

```
mgr.Open();
```

これで、メッセージを送信する準備ができました。次に、1 つのトランザクション内で 2 つのメッセージを送信する手順を示します。

♦ **複数のメッセージを 1 つのトランザクションで送信するには、次の手順に従います。**

1. メッセージ・オブジェクトを初期化します。

```
QATextMessage msg_1;
QATextMessage msg_2;
```

2. メッセージを送信します。

次のコードでは、1 つのトランザクション内で 2 つのメッセージを送信しています。

```
msg_1 = mgr.CreateTextMessage();
if( msg_1 != null ) {
  msg_2 = mgr.CreateTextMessage();
  if( msg_2 != null ) {
    if( !mgr.PutMessage( "jms_1¥¥queue_name", msg_1 ) ) {
      // Display message using mgr.GetLastErrorMsg().
    } else {
      if( !mgr.PutMessage( "jms_1¥¥queue_name", msg_2 ) ) {
        // Display message using mgr.GetLastErrorMsg().
      } else {
        mgr.Commit();
      }
    }
  }
}
```

Commit メソッドは、現在のトランザクションをコミットして、新しいトランザクションを開始します。このメソッドは、PutMessage() メソッドと GetMessage() メソッドのすべての呼び出しをコミットします。

注意
最初のトランザクションは、open メソッドの呼び出しで開始されます。

参照

♦ 「QATransactionalManager インタフェース」 367 ページ

トランザクション志向メッセージングの実装 (C++ クライアントの場合)

♦ **トランザクション志向マネージャを作成するには、次の手順に従います。**

1. QAnywhere を初期化します。

   この手順は、非トランザクション志向メッセージングの場合と同じです。

   ```
   #include <qa.hpp>
   QAManagerFactory *   factory;

   factory = QAnywhereFactory_init();
   if( factory == NULL ) {
      // Fatal error.
   }
   ```

2. トランザクション志向マネージャを作成します。

   ```
   QATransactionalManager *  mgr;
   mgr = factory->createQATransactionalManager( NULL );
   if( mgr == NULL ) {
      // Fatal error.
   }
   ```

   非トランザクション志向マネージャの場合と同様に、プロパティ・ファイルを指定することで QAnywhere の動作をカスタマイズできます。この例では、プロパティ・ファイルは使用していません。

3. マネージャを初期化します。

   ```
   if( !mgr->open() ) {
      // Display message using mgr->getLastErrorMsg().
   }
   ```

これで、メッセージを送信する準備ができました。次に、1 つのトランザクション内で 2 つのメッセージを送信する手順を示します。

♦ **複数のメッセージを 1 つのトランザクションで送信するには、次の手順に従います。**

1. メッセージ・オブジェクトを初期化します。

   ```
   QATextMessage *  msg_1;
   QATextMessage *  msg_2;
   ```

2. メッセージを送信します。

   次のコードでは、1 つのトランザクション内で 2 つのメッセージを送信しています。

   ```
   msg_1 = mgr->createTextMessage();
   if( msg_1 != NULL ) {
     msg_2 = mgr->createTextMessage();
     if( msg_2 != NULL ) {
       if( !mgr->putMessage( "jms_1¥¥queue_name", msg_1 ) ) {
         // Display message using mgr->getLastErrorMsg().
       } else {
         if( !mgr->putMessage( "jms_1¥¥queue_name", msg_2 ) ) {
           // Display message using mgr->getLastErrorMsg().
         } else {
           mgr->commit();
   ```

```
        }
      }
      mgr->deleteMessage( msg_2 );
    }
    mgr->deleteMessage( msg_1 );
  }
```

commit メソッドは、現在のトランザクションをコミットして、新しいトランザクションを開始します。このメソッドは、putMessage() メソッドと getMessage() メソッドのすべての呼び出しをコミットします。

> **注意**
> 最初のトランザクションは、open メソッドの呼び出しで開始されます。

参照

- C++ :「QATransactionalManager クラス」 521 ページ
- Java :「QATransactionalManager インタフェース」 632 ページ

トランザクション志向メッセージングの実装 (Java クライアントの場合)

♦ **トランザクション志向マネージャを作成するには、次の手順に従います。**

1. QAnywhere を初期化します。

   この手順は、非トランザクション志向メッセージングの場合と同じです。

   ```
   import ianywhere.qanywhere.client;
   QAManagerFactory factory = new QAManagerFactory();
   ```

   「QAManagerFactory クラス」 339 ページを参照してください。

2. QATransactionalManager オブジェクトを作成します。

   たとえば、デフォルトの QATransactionalManager オブジェクトを作成するには、パラメータに null を指定して、createQATransactionalManager を呼び出します。

   ```
   QAManager mgr;
   mgr = factory.createQATransactionalManager( null );
   ```

   プロパティ・ファイルを使用して、カスタマイズされた QATransactionalManager オブジェクトを作成することもできます。次のように、createQATransactionalManager メソッドの引数としてプロパティ・ファイルを指定します。

   ```
   mgr = factory.createQATransactionalManager( "qa_mgr.props" );
   ```

   qa_mgr.props は、リモート・デバイス上に存在するプロパティ・ファイルの名前です。

3. QAManager オブジェクトを初期化します。

   ```
   mgr.open();
   ```

これで、メッセージを送信する準備ができました。次に、1 つのトランザクション内で 2 つのメッセージを送信する手順を示します。

♦ **複数のメッセージを 1 つのトランザクションで送信するには、次の手順に従います。**

1. メッセージ・オブジェクトを初期化します。

```
QATextMessage msg_1;
QATextMessage msg_2;
```

2. メッセージを送信します。

   次のコードでは、1 つのトランザクション内で 2 つのメッセージを送信しています。

```
msg_1 = mgr.createTextMessage();
if( msg_1 != null ) {
  msg_2 = mgr.createTextMessage();
  if( msg_2 != null ) {
    if( !mgr.putMessage( "jms_1¥¥queue_name", msg_1 ) ) {
      // Display message using mgr.getLastErrorMsg().
    } else {
      if( !mgr.putMessage( "jms_1¥¥queue_name", msg_2 ) ) {
        // Display message using mgr.getLastErrorMsg().
      } else {
        mgr.commit();
      }
    }
  }
}
```

   commit メソッドは、現在のトランザクションをコミットして、新しいトランザクションを開始します。このメソッドは、putMessage() メソッドと getMessage() メソッドのすべての呼び出しをコミットします。

> **注意**
> 最初のトランザクションは、open メソッドの呼び出しで開始されます。

QAnywhere メッセージのキャンセル

QAnywhere メッセージをキャンセルすると、メッセージは転送される前にキャンセル済みの状態になります。QAnywhere Agent のデフォルトの削除ルールにより、キャンセル済みメッセージは最終的にメッセージ・ストアから削除されます。メッセージがすでに最終ステータスになっている場合や、中央のメッセージング・サーバに転送済みである場合は、QAnywhere メッセージをキャンセルできません。

次の手順では、QAnywhere メッセージをキャンセルする方法について説明します。

注意
QAnywhere SQL API を使用してメッセージをキャンセルすることはできません。

♦ **メッセージをキャンセルするには、次の手順に従います (.NET の場合)。**

1. キャンセルするメッセージの ID を取得します。

```
// msg is a QAMessage instance that has not been
// transmitted.
string msgID = msg.getMessageID();
```

2. キャンセルするメッセージの ID を指定して、CancelMessage を呼び出します。

```
mgr.CancelMessage(msgID);
```

「CancelMessage メソッド」 306 ページを参照してください。

♦ **メッセージをキャンセルするには、次の手順に従います (C++ の場合)。**

1. キャンセルするメッセージの ID を取得します。

```
// msg is a QAMessage instance that has not been
// transmitted.
qa_string msgID = msg->getMessageID();
```

2. キャンセルするメッセージの ID を指定して、cancelMessage を呼び出します。

```
bool result = mgr->cancelMessage(msgID);
```

「cancelMessage 関数」 466 ページを参照してください。

♦ **メッセージをキャンセルするには、次の手順に従います (Java の場合)。**

1. キャンセルするメッセージの ID を取得します。

```
// msg is a QAMessage instance that has not been
// transmitted.
String msgID = msg.getMessageID();
```

2. キャンセルするメッセージの ID を指定して、cancelMessage を呼び出します。

```
boolean result = mgr.cancelMessage(msgID);
```

「cancelMessage メソッド」 575 ページを参照してください。

QAnywhere メッセージの受信

次のトピックでは、QAnywhere メッセージの受信方法について説明します。

同期的なメッセージ受信

メッセージを同期的に受信するには、アプリケーションから明示的にキューをポーリングしてメッセージの有無を確認します。キューのポーリングは、定期的に、またはユーザが特定のアクション ([再表示] ボタンのクリックなど) を開始したときに実行できます。

♦ **同期的にメッセージを受信するには、次の手順に従います (.NET の場合)。**

1. 着信メッセージを格納するメッセージ・オブジェクトを宣言します。

```
QAMessage msg;
QATextMessage text_msg;
```

2. メッセージ・キューをポーリングして、メッセージを収集します。

```
if(mgr.start())  {
  for(;;) {
    msg = mgr.GetMessageNoWait("queue-name");
    if( msg == null ) {
      break;
    }
    addMessage( msg );
  }
  mgr.stop();
}
```

「GetMessageNoWait メソッド」 316 ページを参照してください。

♦ **同期的にメッセージを受信するには、次の手順に従います (C++ の場合)。**

1. 着信メッセージを格納するメッセージ・オブジェクトを宣言します。

```
QAMessage * msg;
QATextMessage * text_msg;
```

2. メッセージ・キューをポーリングして、メッセージを収集します。

```
if( mgr->start() ) {
  for( ;; ) {
    msg = mgr->getMessageNoWait( "queue-name" );
    if( msg == NULL ) {
      break;
    }
    addMessage(msg);
  }
  mgr->stop();
}
```

「getMessageNoWait 関数」 476 ページを参照してください。

♦ **同期的にメッセージを受信するには、次の手順に従います (Java の場合)。**

1. 着信メッセージを格納するメッセージ・オブジェクトを宣言します。

```
QAMessage msg;
QATextMessage text_message;
```

2. メッセージ・キューをポーリングして、メッセージを収集します。

```
if(mgr.start()) {
  for ( ;; ) {
    msg = mgr.getMessageNoWait("queue-name");
    if ( msg == null ) {
      break;
    }
    addMessage(msg);
  }
  mgr.stop();
}
```

「getMessageNoWait メソッド」 585 ページを参照してください。

♦ **同期的にメッセージを受信するには、次の手順に従います (SQL の場合)。**

1. メッセージ ID を格納するオブジェクトを宣言します。

```
begin
  declare @msgid varchar(128);
```

2. メッセージ・キューをポーリングして、メッセージを収集します。

```
  loop
    set @msgid = ml_qa_getmessagenowait( 'myaddress' );
    if @msgid is null then leave end if;
    message 'a message with content ' || ml_qa_gettextcontent( @msgid ) || ' has been received';
  end loop;
  commit;
end
```

次の項を参照してください。

- 「ml_qa_getmessagenowait」 709 ページ
- 「ml_qa_getmessagetimeout」 710 ページ
- 「ml_qa_getmessage」 707 ページ

非同期的なメッセージ受信

.NET、C++、Java の API を使用して、非同期的にメッセージを受信するには、メッセージ・リスナ関数を作成して登録します。メッセージがキューに登録されると、このメッセージ・リスナ関数が QAnywhere によって呼び出されます。メッセージ・リスナは、着信メッセージを引数とします。メッセージ・リスナ内で実行する処理は、アプリケーションによって異なります。たとえば、TestMessage サンプル・アプリケーションのメッセージ・リスナは、TestMessage のメイン・ウィンドウのメッセージ・リストに着信メッセージを追加します。

.NET、C++、Java での開発のヒント

受信メッセージの処理過程、メッセージの受信確認中に発生する可能性のあるアプリケーション・エラーを防ぐために、QAManager を EXPLICIT_ACKNOWLEDGEMENT モードで使用することをおすすめします。

QAManager が EXPLICIT_ACKNOWLEDGEMENT モードでオープンされている場合、メッセージは正常に処理された後でのみ onMessage メソッドで受信確認されます。この方法では、メッセージの処理中にエラーが発生した場合は受信確認が行われないため、メッセージを再受信できます。

QAManager が IMPLICIT_ACKNOWLEDGEMENT モードでオープンされている場合、onMessage に渡されたメッセージは、onMessage がメッセージを戻すときに暗黙的に受信確認されます。このため、メッセージの処理中にユーザ・アプリケーションでエラーが発生した場合でも受信確認が行われるため、メッセージは再受信できません。

♦ **非同期的にメッセージを受信するには、次の手順に従います (.NET の場合)。**

1. メッセージ・ハンドラ・メソッドを実装します。

```
private void onMessage(QAMessage msg) {
   // Process message.
}
```

2. メッセージ・ハンドラを登録します。

   メッセージ・ハンドラを登録するには、メッセージ・ハンドラを引数として指定して QAManager.MessageListener オブジェクトを作成します。次に、QAManager.SetMessageListener 関数を使用して、MessageListner を特定のキューに登録します。次の例では、*queue-name* は QAManager オブジェクトが受信するキューの名前を示す文字列です。

```
MessageListener listener;
listener = new MessageListener( onMessage );
mgr.SetMessageListener( "queue-name", listener );
```

「MessageListener 委任」 264 ページと「SetMessageListener メソッド」 329 ページを参照してください。

♦ **非同期的にメッセージを受信するには、次の手順に従います (C++ の場合)。**

1. QAMessageListener インタフェースを実装するクラスを作成します。

```
class MyClass: public QAMessageListener {
  public:
  void onMessage( QAMessage * Msg);
};
```

「QAMessageListener クラス」 516 ページを参照してください。

2. onMessage メソッドを実装します。

   QAMessageListener インタフェースには onMessage という名前のメソッドがあります。キューにメッセージが着信するたびに、QAnywhere ライブラリはこのメソッドを呼び出します。そのとき、着信メッセージが唯一の引数として渡されます。

```
void MyClass::onMessage(QAMessage * msg) {
    // Process message.
}
```

3. メッセージ・リスナを登録します。

```
my_listener = new MyClass();
mgr->setMessageListener( "queue-name", my_listener );
```

「setMessageListener 関数」 484 ページを参照してください。

♦ **非同期的にメッセージを受信するには、次の手順に従います (Java の場合)。**

1. メッセージ・ハンドラ・メソッドと例外ハンドラ・メソッドを実装します。

```
class MyClass implements QAMessageListener {
  public void onMessage(QAMessage message) {
    // Process the message.
  }
  public void onException(
    QAException exception, QAMessage message) {
    // Handle the exception.
  }
}
```

2. メッセージ・ハンドラを登録します。

```
MyClass listener = new MyClass();
mgr.setMessageListener("queue-name", listener);
```

「QAMessageListener インタフェース」 624 ページと「setMessageListener メソッド」 596 ページを参照してください。

♦ **非同期的にメッセージを受信するには、次の手順に従います (SQL の場合)。**

- **ml_qa_listener_*queue*** という名前でストアド・プロシージャを作成します。*queue* はメッセージ・キューの名前です。

  このプロシージャは、指定されたキューにメッセージが登録されるたびに呼び出されます。

  「ml_qa_listener_queue」 712 ページを参照してください。

セレクタを使用したメッセージの受信

「**メッセージ・セレクタ**」を使用すると、受信するメッセージを選択できます。メッセージ・セレクタは SQL に似た式で、受信操作でメッセージのサブセットを選択するための条件を指定します。

メッセージ・セレクタの構文とセマンティックは、転送ルールの条件部分とまったく同じです。

「条件構文」 244 ページを参照してください。

例

次の C# の例では、intprop というメッセージ・プロパティに値 1 が設定されている receiveQueue から、次のメッセージを取得します。


```
msg = receiver.GetMessageBySelectorNoWait(
  receiveQueue, "intprop=1" );
```

次の C++ の例では、intprop というメッセージ・プロパティに値 1 が設定されている receiveQueue から、次のメッセージを取得します。

```
msg = receiver->getMessageBySelectorNoWait(
  receiveQueue, "intprop=1" );
```

次の Java の例では、intprop というメッセージ・プロパティに値 1 が設定されている receiveQueue から、次のメッセージを取得します。

```
msg = receiver.getMessageBySelectorNoWait(
  receiveQueue, "intprop=1");
```

参照

- .NET：「GetMessageBySelector メソッド」 313 ページと「GetMessageBySelectorNoWait メソッド」 314 ページ
- C++：「getMessageBySelector 関数」 474 ページと「getMessageBySelectorNoWait 関数」 475 ページ
- Java：「getMessageBySelector メソッド」 582 ページと「getMessageBySelectorNoWait メソッド」 582 ページ
- SQL：SQL API では、セレクタを使用したメッセージ受信はサポートしていません。

サイズの大きなメッセージの読み込み

メッセージのサイズが大き過ぎて、QAManager のプロパティ MAX_IN_MEMORY_MESSAGE_SIZE に設定されている制限値、またはデフォルトの制限値 (Windows の場合は 1MB、Windows CE の場合は 64K) を超える場合があります。この場合、メモリ内のすべての内容をメッセージ・オブジェクトに格納することはできません。したがって、readInt() や readString() など、メモリにロードされているメッセージのすべての内容を必要とするメソッドを使用できません。しかし、大きなメッセージを分割してメッセージ・ストアから直接読み込むことはできます。それには、QATextMessage.readText() メソッドまたは QABinaryMessage.readBinary() メソッドを使用してループ処理します。

詳細については、次の項を参照してください。

- .NET：「ReadBinary メソッド」 278 ページと「ReadText メソッド」 366 ページ
- C++：「readBinary 関数」 437 ページと「readText 関数」 519 ページ
- Java：「readBinary メソッド」 547 ページと「readText メソッド」 630 ページ
- SQL：SQL API では、サイズの大きなメッセージを受信できません。

この方法でメッセージを読み込む場合、IMPLICIT_ACKNOWLEDGEMENT を指定してオープンした QAManager は使用できません。EXPLICIT_ACKNOWLEDGEMENT を指定してオープンした QAManager を使用する必要があります。また、メッセージの受信確認は、readText() または readBinary() の呼び出しがすべて完了してから行います。

「受信確認モード」 67 ページを参照してください。

QAnywhere メッセージの参照

入力キューと出力キューに登録されているメッセージを参照できます。メッセージを参照しても、メッセージ・ステータスは変わりません。

メッセージ・ステータスの詳細については、「事前に定義されたメッセージ・プロパティ」 224 ページの ias_Status を参照してください。

次のトピックでは、QAnywhere メッセージの参照方法について説明します。

すべてのメッセージの参照

すべてのキューに登録されているメッセージを参照するには、適切な browseMessages() メソッドを呼び出します。

次の .NET の例では、QAManager.BrowseMessages() メソッドを使用して、すべてのキューのメッセージを参照します。

```
QAMessage msg;
IEnumerator msgs = mgr.BrowseMessages();
while( msgs.MoveNext() ) {
    msg = (QAMessage)msgs.Current;
    // Process message.
}
```

次の C++ の例では、QAManager browseMessages 関数を使用して、すべてのキューのメッセージを参照します。

```
QAMessage *msg;
qa_browse_handle bh = mgr->browseMessages();
for (;;) {
  msg = mgr->browseNextMessage( bh );
  if( msg == qa_null ) {
    break;
  }
  // Process message.
  mgr->browseClose( bh );
}
```

次の Java の例では、QAManager.browseMessages メソッドを使用して、すべてのキューのメッセージを参照します。

```
QAMessage msg;
java.util.Enumeration msgs = mgr.browseMessages();
while( msgs.hasMoreElements() ) {
    msg = (QAMessage)msgs.nextElement();
  // Process message.
}
```

参照

- .NET：「BrowseMessages メソッド」 302 ページ
- C++：「browseMessages 関数」 462 ページ
- Java：「browseMessages メソッド」 572 ページ
- SQL：SQL API では、メッセージを参照できません。

特定のキューに登録されているメッセージの参照

特定のキューに登録されているメッセージを参照するには、目的のキューの名前を指定して、適切な browseMessagesByQueue() メソッドを呼び出します。

次の .NET の例では、QAManager.BrowseMessagesByQueue メソッドを使用して、特定のキューのメッセージを参照します。

```
QAMessage msg;
IEnumerator msgs = mgr.BrowseMessagesByQueue( "q1" );
while( msgs.MoveNext() ) {
  msg = (QAMessage)msgs.Current;
  // Process message.
}
```

次の C++ の例では、QAManager browseMessagesByQueue 関数を使用して、特定のキューのメッセージを参照します。

```
QAMessage *msg;
qa_browse_handle bh = mgr->browseMessagesByQueue( _T("q1") );
for (;;) {
  msg = mgr->browseNextMessage( bh );
  if( msg == qa_null ) {
    break;
  }
  // Process message.
}
mgr->browseClose( bh );
```

次の Java の例では、QAManager.browseMessagesByQueue メソッドを使用して、特定のキューのメッセージを参照します。

```
QAMessage msg;
java.util.Enumeration msgs = mgr.browseMessagesByQueue( "q1" );
while( msgs.hasMoreElements() ) {
  msg = (QAMessage)msgs.nextElement();
  // Process message.
}
```

参照

- .NET：「BrowseMessagesByQueue メソッド」 304 ページ
- C++：「browseMessagesByQueue 関数」 464 ページ
- Java：「browseMessagesByQueue メソッド」 574 ページ
- SQL：SQL API では、メッセージを参照できません。

ID の指定によるメッセージの参照

特定のメッセージを参照するには、そのメッセージ ID を指定して、browseMessagesbyID() メソッドを呼び出します。

次の .NET の例では、QAManager.BrowseMessageByID メソッドを使用して、特定のメッセージを参照します。

```
QAMessage msg;
IEnumerator msgs = mgr.BrowseMessagesByID( "ID:123" );
if( msgs.MoveNext() ) {
```



```
  msg = (QAMessage)msgs.Current;
  // Process message.
}
```

次の C++ の例では、QAManager browseMessageByID 関数を使用して、特定のメッセージを参照します。

```
QAMessage *msg;
qa_browse_handle bh = mgr->browseMessagesByID( _T( "ID:123" ) );
msg = mgr->browseNextMessage( bh );
if( msg != qa_null ) {
  // Process message.
}
mgr->browseClose( bh );
```

次の Java の例では、QAManager.browseMessageByID メソッドを使用して、特定のメッセージを参照します。

```
QAMessage msg;
java.util.Enumeration msgs = mgr.browseMessagesByID( "ID:123" );
if( msgs.hasMoreElements() ) {
  msg = (QAMessage)msgs.nextElement();
  // Process message.
}
```

参照

- .NET：「BrowseMessagesByID メソッド」 304 ページ
- C++：「browseMessagesByID 関数」 463 ページ
- Java：「browseMessagesByID メソッド」 573 ページ
- SQL：SQL API では、メッセージを参照できません。

セレクタを使用したメッセージの参照

「**メッセージ・セレクタ**」を使用すると、参照するメッセージを選択できます。メッセージ・セレクタは SQL に似た式で、参照操作でメッセージのサブセットを選択するための条件を指定します。

メッセージ・セレクタの構文とセマンティックは、転送ルールの条件部分とまったく同じです。

「条件構文」 244 ページを参照してください。

次の .NET の例では、intprop というプロパティに値 1 が設定されているメッセージ・ストア内のすべてのメッセージを参照します。

```
QAMessage msg;
IEnumerator msgs = mgr.BrowseMessagesBySelector( "intprop = 1" );
while( msgs.MoveNext() ) {
  msg = (QAMessage)msgs.Current;
  // Process message.
}
```

次の C++ の例では、intprop というプロパティに値 1 が設定されているメッセージ・ストア内のすべてのメッセージを参照します。

```
QAMessage *msg;
qa_browse_handle bh = mgr->browseMessagesBySelector( _T("intprop = 1") );
```



```
for (;;) {
  msg = mgr->browseNextMessage( bh );
  if( msg == qa_null ) {
    break;
  }
  // Process message.
}
mgr->browseClose( bh );
```

次の Java の例では、intprop というプロパティに値 1 が設定されているメッセージ・ストア内のすべてのメッセージを参照します。

```
QAMessage msg;
java.util.Enumeration msgs = mgr.browseMessagesBySelector( "intprop = 1" );
while( msgs.hasMoreElements() ) {
  msg = (QAMessage)msgs.nextElement();
  // Process message.
}
```

参照

- .NET：「BrowseMessagesBySelector メソッド」 305 ページ
- C++：「browseMessagesBySelector 関数」 464 ページ
- Java：「browseMessagesBySelector メソッド」 574 ページ
- SQL：SQL API では、メッセージを参照できません。

QAnywhere 例外の処理

QAnywhere C++、Java、.NET API には、例外処理のための特別なオブジェクトとプロパティがあります。

.NET 例外

QAException クラスは、QAnywhere クライアント・アプリケーションの例外をカプセル化します。QAnywhere 例外をキャッチした後で、QAException ErrorCode プロパティと Message プロパティを使用して、エラー・コードとエラー・メッセージを特定できます。

メッセージ・リスナ委任で QAException がスローされた場合で、例外がメッセージ・リスナでキャッチされないときは、その内容が QAManager ログ・ファイルに記録されることに注意してください。キャッチされなかった QAException はログに記録されるだけなので、すべての例外がメッセージ・リスナ委任内または例外リスナ委任によって処理されるようにすることをおすすめします。こうすることで、例外は適切に処理されます。

メッセージ・リスナ委任と例外リスナ委任の詳細については、次の項を参照してください。

- 「MessageListener 委任」 264 ページ
- 「MessageListener2 委任」 264 ページ
- 「ExceptionListener 委任」 263 ページ
- 「ExceptionListener2 委任」 263 ページ

ログ・ファイルの詳細については、「QAnywhere Manager の設定プロパティ」 69 ページを参照してください。

QAException がスローされると、現在のトランザクションはロール・バックされます。QATransactionalManager を使用するメッセージ・リスナでこれが発生すると、QAException がスローされたときに処理されていたメッセージはキューに戻されます。これにより、メッセージが再度受信されることになります。無限ループを回避するため、メッセージ・ストア・プロパティ ias_MaxDeliveryAttempts を使用できます。

プロパティ ias_MaxDeliveryAttempts が、mgr.SetIntStoreProperty("ias_MaxDeliveryAttempts", 5) のように QAnywhere アプリケーションによって正の整数 *n* に設定されると、QAnywhere クライアントは、メッセージのステータスを受信不可にする前に、受信確認されていないメッセージを最大 *n* 回まで受信するように試みます。プロパティ ias_MaxDeliveryAttempts が設定されていないか、負の値に設定されると、QAnywhere クライアントは、制限なく、メッセージの受信を試みます。

詳細については、次の項を参照してください。

- 「QAException クラス」 290 ページ
- 「ErrorCode プロパティ」 292 ページ
- 「事前に定義されたクライアント・メッセージ・ストア・プロパティ」 230 ページ

C++ 例外

C++ の場合は、QAError クラスが QAnywhere クライアント・アプリケーションの例外をカプセル化します。QAManagerBase::getLastError() メソッドまたは QAManagerFactory::getLastError() メ

ソッドを使用すると、最後に実行されたメソッドに関連するエラー・コードを特定できます。また、対応する getLastErrorMessage() メソッドを使用すると、エラー・テキストを取得できます。

エラー・コードのリストと詳細については、「QAError クラス」 448 ページを参照してください。

getLastError と getLastErrorMessage の詳細については、次の項を参照してください。

- QAManagerBase：「getLastError 関数」 472 ページと「getLastErrorMsg 関数」 472 ページ
- QAManagerFactory：「getLastError 関数」 492 ページと「getLastErrorMsg 関数」 493 ページ

Java 例外

QAException クラスは、QAnywhere クライアント・アプリケーションの例外をカプセル化します。QAnywhere 例外をキャッチした後で、QAException ErrorCode プロパティと Message プロパティを使用して、エラー・コードとエラー・メッセージを特定できます。

メッセージ・リスナで QAException がスローされ、メッセージ・リスナでキャッチされなかった場合は、その内容が QAManager ログ・ファイルに記録されます。キャッチされなかった QAException はログに記録されるだけなので、すべての例外がメッセージ・リスナ内または例外リスナによって処理されるようにすることをおすすめします。こうすることで、例外は適切に処理されます。

メッセージ・リスナと例外リスナの詳細については、次の項を参照してください。

- 「QAMessageListener インタフェース」 624 ページ
- 「QAMessageListener2 インタフェース」 625 ページ
- 「QAException クラス」 559 ページ

ログ・ファイルの詳細については、「QAnywhere Manager の設定プロパティ」 69 ページを参照してください。

QAException がスローされると、現在のトランザクションはロール・バックされます。QATransactionalManager を使用するメッセージ・リスナでこれが発生すると、QAException がスローされたときに処理されていたメッセージはキューに戻されます。これにより、メッセージが再度受信されることになります。無限ループを回避するため、メッセージ・ストア・プロパティ ias_MaxDeliveryAttempts を使用できます。

プロパティ ias_MaxDeliveryAttempts が、mgr.SetIntStoreProperty("ias_MaxDeliveryAttempts", 5) のように QAnywhere アプリケーションによって正の整数 *n* に設定されると、QAnywhere クライアントは、メッセージのステータスを受信不可にする前に、受信確認されていないメッセージを最大 *n* 回まで受信するように試みます。プロパティ ias_MaxDeliveryAttempts が設定されていないか、負の値に設定されると、QAnywhere クライアントは、制限なく、メッセージの受信を試みます。

詳細については、次の項を参照してください。

- 「ErrorCode プロパティ」 292 ページ
- 「事前に定義されたクライアント・メッセージ・ストア・プロパティ」 230 ページ

エラー・コード

QAnywhere のエラー・コードを次の表に示します。

LastError の値	説明
0	エラーはありません。
1000	初期化エラーです。
1001	終了エラーです。
1002	クライアント・プロパティ・ファイルにアクセスできません。
1003	送信先が指定されていません。
1004	関数が実装されていません。
1005	読み取り専用モードであるため、メッセージに書き込みできません。
1006	クライアント・メッセージ・ストアにメッセージを格納するときにエラーが発生しました。
1007	クライアント・メッセージ・ストアからメッセージを取得するときにエラーが発生しました。
1008	バックグラウンド・スレッドを初期化するときにエラーが発生しました。
1009	メッセージ・ストアへの接続をオープンするときにエラーが発生しました。
1010	クライアント・プロパティ・ファイル内に無効なプロパティが存在します。
1011	ログ・ファイルをオープンするときにエラーが発生しました。
1012	予期しないところでメッセージの終わりに到達しました。
1013	メッセージ・ストアが大き過ぎて、デバイス上のディスクの空き領域に収まりません。
1014	メッセージ・ストアがメッセージング用に初期化されていません。
1015	キューの深さを取得中にエラーが発生しました。
1016	メッセージ・ストア ID が設定されていない場合は、QAManagerBase.getQueueDepth を使用できません。
1017	フィルタが ALL の場合は、指定された送信先で QAManagerBase.getQueueDepth を使用できません。
1018	メッセージのキャンセル中にエラーが発生しました。
1019	メッセージのキャンセル中にエラーが発生しました。送信済みのメッセージはキャンセルできません。
1020	メッセージの受信確認中にエラーが発生しました。
1021	QAManager がオープンされていません。
1022	QAManager はすでにオープンされています。

LastError の値	説明
1023	指定されたセレクタに構文エラーがあります。
1024	タイムスタンプが許容範囲外です。

QAnywhere の停止

メッセージの送信と受信を完了したら、次のいずれかの手順に従って QAnywhere メッセージング・システムを停止します。

♦ **QAnywhere を停止するには、次の手順に従います (.NET の場合)。**

- QAnywhere Manager を停止し、クローズします。

```
mgr.Stop();
mgr.Close();
```

♦ **QAnywhere を停止するには、次の手順に従います (C++)。**

1. QAnywhere Manager をクローズします。

```
mgr->stop();
mgr->close();
```

2. ファクトリを終了します。

```
QAnywhereFactory_term();
```

これで、アプリケーションのメッセージング部分が停止します。

♦ **QAnywhere を停止するには、次の手順に従います (Java の場合)。**

- QAnywhere Manager を停止し、クローズします。

```
mgr.stop();
mgr.close();
```

参照

- .NET：「Stop メソッド」 337 ページ
- C++：「stop 関数」 488 ページ
- Java：「stop メソッド」 601 ページ
- SQL：SQL API では、QAnywhere を停止できません。

QAnywhere アプリケーションの配備

「QAnywhere アプリケーションの配備」 『Mobile Link - サーバ管理』を参照してください。

第 5 章

サーバ管理要求

目次

サーバ管理要求の概要

QAnywhere クライアント・アプリケーションは、「**サーバ管理要求**」という特殊なメッセージをサーバに送信できます。サーバ管理要求は、XML フォーマットの内容を含み、QAnywhere システム・キューに送信されます。特殊な認証文字列が必要です。サーバ管理要求は、次のようなさまざまな機能を実行できます。

- コネクタと Web サービスの開始と停止

  「コネクタを開く」 114 ページと「コネクタを閉じる」 114 ページを参照してください。

- コネクタのステータスのモニタリング

  「コネクタのモニタ」 115 ページを参照してください。

- クライアント転送ルールの設定と更新

  「サーバ管理要求での転送ルールの指定」 124 ページを参照してください。

- メッセージ・ステータスのモニタリング

  「QAnywhere のモニタ」 128 ページを参照してください。

- サーバ上に置かれているクライアント・メッセージ・ストア・プロパティの設定、更新、削除、クエリ

  「サーバ管理要求でのサーバ・プロパティの設定」 122 ページを参照してください。

- メッセージのキャンセル

  「メッセージのキャンセル」 109 ページを参照してください。

- アクティブなクライアント、メッセージ・ストア・プロパティ、メッセージのクエリ

サーバ管理要求のアドレス指定

デフォルトでは、サーバ管理要求を ianywhere.server¥system に送信する必要があります。このアドレスのクライアント ID の部分を変更するには、ianywhere.qa.server.id プロパティを設定してからサーバを再起動します。たとえば、ianywhere.qa.server.id プロパティが myServer に設定されていると、サーバ管理要求は myServer¥system に送信されます。

ianywhere.qa.server.id プロパティの設定の詳細については、「サーバ・プロパティ」 237 ページを参照してください。

QAnywhere メッセージのアドレス指定の詳細については、「QAnywhere メッセージの送信」 73 ページを参照してください。

システム・キューの詳細については、「システム・キュー」 58 ページを参照してください。

サーバ管理要求の認証

メッセージ文字列プロパティの ias_ServerPassword は、サーバのパスワードを指定します。サーバのパスワードは、ianywhere.qa.server.password.e プロパティを使用して設定されます。このプロパティが設定されていない場合、パスワードは QAnywhere です。

サーバのパスワードはテキストとして転送されます。サーバのパスワードを必要とするサーバ管理要求を送信する場合は、暗号化された通信ストリームを使用してください。

ianywhere.qa.server.password.e プロパティの詳細については、「サーバ・プロパティ」 237 ページを参照してください。

例

次に、メッセージの詳細要求のサンプルを示します。現在サーバ上に置かれている優先度 9 のすべてのメッセージについて、メッセージ ID、ステータス、送信先アドレスを表示するレポートを 1 つ生成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <MessageDetailsRequest>
    <request>
      <requestId>testRequest</requestId>
      <condition>
        <priority>9</priority>
      </condition>
      <status/>
      <address/>
    </request>
  </MessageDetailsRequest>
</actions>
```

次の例は、C# での記述です。これは、サーバ側のクライアント用転送ルールを設定します。優先度が 4 より大きい場合に、サーバからのメッセージを someClient というクライアントにのみ転送するというルールです。

```
QAManager mgr = …;    // Initialize the QAManager
QAMessage msg = mgr.CreateTextMessage();
msg.SetStringProperty( “ias_ServerPassword”, “QAnywhere” );

// Indenting and newlines are just for readability
msg.Text = “<?xml version="1.0" encoding="UTF-8"?>¥n”
+ “<actions>¥n”
+ “    <SetProperty>¥n”
+ “       <prop>¥n”
+ “          <client>someClient</client>¥n”
+ “          <name>ianywhere.qa.server.rules</name>¥n”
+ “          <value>ias_Priority &gt; 4</value>¥n”
+ “       </prop>¥n”
+ “    </SetProperty>¥n”
+ “    <RestartRules>¥n”
+ “       <client>someClient</client>¥n”
+ “    </RestartRules>¥n”
+ “</actions>¥n”;

mgr.PutMessage( @“ianywhere.server¥system”, msg );
```


サーバ管理要求の作成

サーバ管理要求は XML フォーマットの内容を含んでいます。

> **注意**
> **>** や **<** などの記号は、サーバ管理要求内で使用できません。代わりに、**>** と **<** を使用してください。

各サーバ管理要求は <actions> タグで開始します。

サーバ管理要求には、その種類ごとに独自の XML タグが含まれます。たとえば、コネクタを閉じるには <CloseConnector> タグを使用します。

request タグ

また、ほとんどのサーバ管理要求には、要求を記述する <request> タグを含めることができます。<request> タグ内では、次のサブタグを使用できます。

<request> のサブタグ	説明
<condition>	メッセージをレポートに含めるための条件をグループ化します。<MessageDetailsRequest> と <CancelMessageRequest> のサブタグである <request> でのみ使用できます。
<onEvent>	発生時にサーバにレポートを生成させるイベントを指定します。<ClientStatusRequest> でのみ使用します。1 つまたは複数の <onEvent> タグを含めることができます。タグごとにイベント・タイプを 1 つ指定してください。<onEvent> タグが指定されていない場合は、クライアント・ステータス要求でワンタイム要求が生成されます。それ以外の場合は、指定されたイベントのイベント・リスナが、クライアント・ステータス要求で登録されます。
<persistent>	要求の結果をサーバ・データベース内で永続的にすることを指定します。これにより、サーバが再起動された場合でもレポートが送信されます。<schedule> タグでのみ使用します。
<report>	要求がアクティブになるたびにレポートを送信するように指定します。<CancelMessageRequest> のサブタグである <request> でのみ使用できます。
<requestId>	この要求により生成される各レポートに含まれる要求に対して、ユニークな識別子を指定します。サーバ管理要求で応答またはレポートが生成される場合にのみ使用されます。このフィールドに複数の値を指定すると、一度に複数の要求をアクティブにできます。同じ要求 ID を指定すると、クライアントはアクティブな要求を無効または削除できます。
<replyAddr>	この要求により生成される各レポートの返信アドレスを指定します。このタグが指定されていない場合は、レポートのデフォルトの返信アドレスが送信メッセージの返信アドレスになります。サーバ管理要求で応答またはレポートが生成される場合にのみ使用されます。

<request> のサブタグ	説明
<schedule>	スケジュールに従ってレポートを生成することを指定します。サーバ管理要求で応答またはレポートが生成される場合にのみ使用されます。「サーバ管理要求のスケジュール設定」 106 ページを参照してください。

Condition タグ

MessageDetailsRequest に含めるメッセージをフィルタリングするには、次の <condition> タグのサブタグを使用します。次のサブタグは、<condition> タグ内でいくつでも指定できます。同じタグを複数使用した場合、指定された値は論理 OR 結合されます。これに対して、異なる 2 つのタグを使用した場合は、値は AND 結合されます。

<condition> のサブタグ	説明
<address>	指定されたアドレスに送信されるメッセージを選択します。
<customRule>	ルールに基づいてメッセージを選択します。「カスタム・メッセージ要求」 103 ページを参照してください。
<kind>	バイナリまたはテキストのメッセージをフィルタリングします。 たとえば、**<kind>text</kind>** と指定するとテキスト・メッセージがフィルタリングされ、**<kind>binary</kind>** と指定するとバイナリ・メッセージがフィルタリングされます。
<messageId>	特定のメッセージ ID を持つメッセージを選択します。
<originator>	指定したクライアントから発信されるメッセージを選択します。
<priority>	現在優先度が指定されているメッセージを選択します。
<property>	指定されたメッセージ・プロパティを持つメッセージを選択します。プロパティ名と値を確認するには、**<property>***property-name=property-value***</property>** 構文を使用します。プロパティが存在するかどうかを確認するには、**<property>***property-name***</property>** フォーマットを使用します。
<status>	現在ステータスが指定されているメッセージを選択します。

カスタム・メッセージ要求

より複雑な条件ステートメントを作成するには、<condition> タグ (およびその他のタグ) のサブタグとして <customRule> タグを使用します。このタグではデータをサーバ・ルールとして使用

します。このルールは、サーバ転送ルールで使用されるサーバ・ルールに似ています。このクエリは、転送ルールの条件部分と同じ方法で作成できます。

「条件構文」 244 ページを参照してください。

例

次の条件では、優先度が 4 に設定され、発信者の名前が'%sender%' と一致し、ステータスが 20 以上であるという検索基準を満たすメッセージを選択します。

```
<condition>
   <priority>4</priority>
   <customRule>ias_Originator LIKE '%sender%' AND ias_Status &gt;= 20</customRule>
</condition>
```

サーバ管理要求 DTD

次に、サーバ管理要求の XML 文書の定義をすべて示します。この DTD は、この章で説明するサーバ管理要求タグの概要として示されています。

```
<!-- Set of requests -->

<!ELEMENT actions ((CloseConnector|OpenConnector|RestartRules|SetProperty
   |ClientStatusRequest|MessageDetailsRequest|CancelMessageRequest
   |GetClientList)+)>

<!-- Request for list of all clients -->

<!ELEMENT GetClientList EMPTY>

<!-- Request to close a connector -->

<!ELEMENT CloseConnector (client)>

<!-- Request to open a connector -->

<!ELEMENT OpenConnector (client)>

<!-- Request to restart transmission rules for a client -->

<!ELEMENT RestartRules (client)>

<!-- Request for setting a property -->

<!ELEMENT SetProperty (client,prop)>

<!-- Request for client properties -->

<!ELEMENT GetProperties (client,replyAddr?)>

<!-- Request for the status on a connector -->

<!ELEMENT ClientStatusRequest (request)>

<!-- Request for clients -->

<!ELEMENT MessageDetailsRequest (request)>
<!ELEMENT CancelMessageRequest (request)>
```

```
<!ELEMENT request (requestId?,replyAddr?,schedule*,onEvent*,condition?,
   persistent?,report?,messageId?,status?,priority?,address?,originator?,kind?,
   statusTime?,contentSize?,customRule?,property*)>

<!ELEMENT client (#PCDATA)>

<!ELEMENT prop (name?,value?)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT value (#PCDATA)>

<!ELEMENT replyAddr (#PCDATA)>

<!ELEMENT requestId (#PCDATA)>

<!ELEMENT persistent EMPTY>

<!ELEMENT report EMPTY>

<!ELEMENT schedule ((starttime|between)?,everyhour?,everyminute?,everysecond?,
   ondayofweek*,ondayofmonth*)>

<!ELEMENT between (starttime,endtime)>

<!ELEMENT starttime (#PCDATA)>
<!ELEMENT endtime (#PCDATA)>
<!ELEMENT everyhour (#PCDATA)>
<!ELEMENT everyminute (#PCDATA)>
<!ELEMENT everysecond (#PCDATA)>
<!ELEMENT ondayofweek (#PCDATA)>
<!ELEMENT ondayofmonth (#PCDATA)>

<!ELEMENT onEvent (#PCDATA)>

<!ELEMENT condition ((messageId|status|priority|address|originator|kind|
   customRule|property)+)>

<!ELEMENT messageId (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT transmissionStatus (#PCDATA)>
<!ELEMENT priority (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT originator (#PCDATA)>
<!ELEMENT kind (#PCDATA)>
<!ELEMENT statusTime (#PCDATA)>
<!ELEMENT expires (#PCDATA)>
<!ELEMENT contentSize (#PCDATA)>
<!ELEMENT customRule (#PCDATA)>
<!ELEMENT property (#PCDATA)>

<!-- Reports and response sent back by the server -->

<!-- Report returned as a response to a CancelMessageRequest -->

<!ELEMENT CancelMessageReport (requestId,UTCDatetime,statusDescription,
   messageCount,message*)>

<!-- Report returned as a response to a ClientStatusRequest -->

<!ELEMENT ClientStatusReport (requestId,componentReport)>

<!-- Report returned as a response to a MessageDetailsRequest -->
```



```
<!ELEMENT MessageDetailsReport (requestId,UTCDatetime,statusDescription,
   messageCount,message*)>

<!-- Response to a GetPropertiesRequest -->

<!ELEMENT GetPropertiesResponse (client,prop*)>

<!-- Response to a GetClientList -->

<!ELEMENT GetClientListResponse (client*)>

<!ELEMENT UTCDatetime (#PCDATA)>

<!ELEMENT statusDescription (#PCDATA)>

<!ELEMENT messageCount (#PCDATA)>

<!ELEMENT message ((messageId|status|transmissionStatus|priority|address|originator|kind|
   statusTime|expires|contentSize|property)*)>

<!-- Report on a specific server component (such as a connector) -->

<!ELEMENT componentReport (client,UTCDatetime,statusCode,statusSubcode?,
   statusDescription?,vendorStatusCode?,vendorStatusDescription?)>

<!ELEMENT statusCode (#PCDATA)>
<!ELEMENT statusSubcode (#PCDATA)>
<!ELEMENT vendorStatusCode (#PCDATA)>
<!ELEMENT vendorStatusDescription (#PCDATA)>
```

サーバ管理要求のスケジュール設定

オプションで、スケジュールに従ってサーバ管理要求を実行するように設定できます。サーバ管理要求を実行するスケジュールを定義するには、次の <schedule> タグのサブタグを使用します。

<schedule> タグのサブタグ	説明
<starttime>	サーバがレポート生成を開始する時刻を定義する。次に例を示します。 `<starttime>09:00:00</starttime>`

<between>	<starttime> と <endtime> の 2 つのサブタグを持ち、サーバがレポートを生成する間隔を定義する。<starttime> と同じ <schedule> タグ内で使用することはできません。次に例を示します。
	
	`<between>`
	`  <starttime>Mon Jan 16 09:00:00 EST 2006</starttime>`
	`  <endtime>Mon Jan 17 09:00:00 EST 2006</endtime>`
	`</between>`

<schedule> タグのサブタグ	説明
<everyhour>	後続のレポートとの間隔を時間単位で定義する。<everyminute> または <everysecond> と同じ <schedule> タグ内で使用することはできません。たとえば、次の要求では、1 月 16 日の午前 9 時から 2 時間ごとにレポートが生成されます。 <schedule> <starttime>09:00:00</starttime> <everyhour>2</everyhour> </schedule>
<everyminute>	後続のレポートとの間隔を分単位で定義する。<everyhour> または <everysecond> と同じ <schedule> タグ内で使用することはできません。 <schedule> <everyminute>10</everyminute> </schedule>
<everysecond>	後続のレポートとの間隔を秒単位で定義する。<everyhour> または <everyminute> と同じ <schedule> タグ内で使用することはできません。 <schedule> <everysecond>45</everysecond> </schedule>
<ondayofweek>	タグごとに、スケジュールを実行する曜日を 1 つ指定する。たとえば、次のスケジュールは月曜日と火曜日に実行します。 <schedule> <ondayofweek>Monday</ondayofweek> <ondayofweek>Tuesday</ondayofweek> </schedule>
<ondayofmonth>	タグごとに、スケジュールを実行する日を 1 つ指定する。たとえば、次のスケジュールは 15 日に実行します。 <schedule> <ondayofmonth>15</ondayofmonth> </schedule>
<startdate>	スケジュールを実行する日付。次に例を示します。 <startdate>Mon Jan 16 2006</startdate>

スケジュールを変更するには、同じ要求 ID を指定して新しいサーバ管理要求を登録します。スケジュールを削除するには、同じ要求 ID を指定してサーバ管理要求を登録し、<schedule>none</schedule> というタグが含まれるようにします。

注意

- <ondayofweek> タグと <ondayofmonth> タグを除き、各タグは <schedule> タグ内で一度だけしか使用できません。
- <between> タグと個々の <starttime> タグは、同じ <schedule> タグ内で使用できません。

- 同じ <schedule> タグ内で使用できるのは、<everysecond>、<everyminute>、<everyhour> のいずれか 1 つだけです。

例

次の例では、各メッセージの ID とステータスも含め、サーバ上のすべてのメッセージについてレポートを作成する永続的なスケジュールを作成します。また、要求 ID dailyMessageStatus に割り当てられている永続的な要求の上書きも行います。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <MessageDetailsRequest>
        <request>
            <replyAddr>myclient¥messageStatusQueue</replyAddr>
            <requestId>dailyMessageStatus</requestId>
            <schedule>
                <everyhour>24</everyhour>
            </schedule>
            <persistent/>
            <messageId/>
            <status/>
        </request>
    </MessageDetailsRequest>
</actions>
```

このレポートは次のようになります。レポートはアドレス myclient¥messageStatusQueue に送信されます。サーバ上には、ステータスが 60 (受信済み) と 1 (保留中) の 2 つのメッセージがあることがわかります。

```
<?xml version="1.0" encoding="UTF-8"?>
<MessageDetailsReport>
    <requestId>dailyMessageStatus</requestId>
    <UTCDatetime>Mon Jan 16 15:03:04 EST 2007</UTCDatetime>
    <statusDescription>Scheduled report</statusDescription>
    <messageCount>2</messageCount>
    <message>
        <messageId>ID:26080b8927f83f9722357eab0a0628eb</messageId>
        <status>60</status>
    </message>
    <message>
        <messageId>ID:fe857fa8-a7d7-4266-985b-a1818a85d1a2</messageId>
        <status>1</status>
    </message>
</MessageDetailsReport>
```

サーバ管理要求でのサーバ・メッセージ・ストアの管理

サーバ管理要求を使用して、サーバ・メッセージ・ストアを管理できます。

サーバ管理要求の使い方の概要、サーバ管理要求の認証方法とスケジュールの設定方法については、「サーバ管理要求の概要」 100 ページを参照してください。

クライアント転送ルールの更新

サーバ側のクライアント転送ルールが変更されたら、対応するクライアントのルールも変更する必要があります。これを行うには、サーバ管理要求で property ianywhere.qa.server.rules プロパティを設定します。

<RestartRules> タグには、変更するクライアント名を指定する <client> タグが 1 つ含まれます。

<RestartRules> タグのサブタグ	説明
<client>	転送ルールを更新するクライアントの名前です。

例

サーバ XML は新しい転送ルール・プロパティを指定した後で、<RestartRules> タグを使用してルール処理を再起動します。たとえば、次の XML では、クライアント cyclient に対応するサーバ側の転送ルールを auto = ias_Priority > 4 に変更します。この XML で、">" が適切にエンコードされていることに注意してください。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <SetProperty>
      <prop>
        <client>myclient</client>
        <name>ianywhere.qa.server.rules</name>
        <value>auto = ias_Priority &gt; 4</value>
      </prop>
   </SetProperty>
   <RestartRules>
   <client>myclient</client>
   </RestartRules>
</actions>
```

メッセージのキャンセル

サーバ・メッセージ・ストア内にあるメッセージをキャンセルするサーバ管理要求を作成できます。ワンタイム・キャンセル要求を作成することも、キャンセル要求のスケジュールを設定して自動的にキャンセルすることもできます。また、オプションで、キャンセルされたメッセージの詳細を示すレポートを生成することもできます。

メッセージは、要求がアクティブになった時点でサーバ上にあり、最終ステータスに至っていない場合にのみキャンセルできます。

<CancelMessageRequest> タグのサブタグ	
<request>	特定の要求に関する情報をグループ化します。複数の <request> タグを指定することは、複数のメッセージ管理要求を別々に送信することと同じです。

<Request> タグのサブタグ	説明
<condition>	キャンセルするメッセージを含めるための条件をグループ化します。「Condition タグ」 103 ページを参照してください。
<persistent>	要求をサーバ・データベース内で永続的にすることを指定する。これにより、サーバが再起動された場合であってもメッセージをキャンセルできます。<schedule> タグでのみ使用します。
<requestId>	この要求により生成される各レポートに含まれる要求に対して、ユニークな識別子を指定します。このフィールドに複数の値を指定すると、一度に複数の要求をアクティブにできます。同じ要求 ID を指定すると、クライアントはアクティブな要求を無効または削除できます。
<replyAddr>	この要求により生成される各レポートの返信アドレスを指定します。このタグが指定されていない場合は、レポートのデフォルトの返信アドレスが送信メッセージの返信アドレスになります。
<report>	要求がアクティブになるたびにレポートを送信します。要求がアクティブになるたびにレポートを送信するには、<request> タグ内に空の <report> タグを指定します。
<schedule>	スケジュールに従ってレポートを生成することを指定します。「サーバ管理要求のスケジュール設定」 106 ページを参照してください。

例

次の要求は、ianywhere.connector.myConnector¥deadqueue というアドレスが指定された、サーバ上にあるメッセージをキャンセルします。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <CancelMessageRequest>
      <request>
         <requestId>cancelRequest</client>
         <condition>
            <customRule>ias_Address='ianywhere.connector.myConnector¥deadqueue'</customRule>
         </condition>
      </request>
   </CancelMessageRequest>
</actions>
```

メッセージの削除

サーバのクリーンアップ・ポリシーを指定するには、サーバから削除できるメッセージを制御するルールを使用して、特別なクライアント ianywhere.server.deleteRules に ianywhere.qa.server.deleteRules プロパティを設定します。

次の例では、有効期限の切れたキャンセル済みメッセージを削除するように、メッセージのクリーンアップ・ポリシーを変更します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <SetProperty>
   <prop>
      <client>ianywhere.server.deleteRules</client>
      <name>ianywhere.qa.server.deleteRules</name>
     <value>auto = ias_Status in ( ias_ExpiredStatus, ias_CancelledStatus ) and ias_TransmissionStatus
= IAS_TRANSMITTED</value>
   </prop>
   </SetProperty>
   <RestartRules>
   <client>ianywhere.server.deleteRules</client>
   </RestartRules>
</actions>
```

コネクタの管理

サーバ管理要求を使用して、コネクタの作成、設定、削除、起動、停止、モニタを行うことができます。

サーバ管理要求の使い方の概要、サーバ管理要求の認証方法とスケジュールの設定方法については、「サーバ管理要求の概要」 100 ページを参照してください。

参照

- 「JMS コネクタ」 135 ページ
- 「Web サービス・コネクタの設定」 209 ページ

コネクタの作成と設定

コネクタを作成するには、<OpenConnector> を使用します。

例

次の例のサーバ管理要求では、最初に関連プロパティの数を設定します。次に、新しいコネクタのクライアント ID であるクライアント ianywhere.connector.jboss に、これらのプロパティを関連付けます。JMS 固有のプロパティは、ローカル JBOSS JMS サーバに接続されるコネクタが示される方法で設定されます。この後で、コネクタは <OpenConnector> タグを使用して起動されます。JMS クライアントの関連 jar ファイルを指定して Mobile Link サーバを起動していない場合は、コネクタが起動しません。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <SetProperty>
      <prop>
        <client>ianywhere.connector.jboss</client>
        <name>ianywhere.connector.nativeConnection</name>
      <value>ianywhere.message.connector.jms.NativeConnectionJMS</value>
    </prop>
      <prop>
        <client>ianywhere.connector.jboss</client>
        <name>ianywhere.connector.address</name>
      <value>ianywhere.connector.jboss</value>
    </prop>
      <prop>
        <client>ianywhere.connector.jboss</client>
        <name>xjms.jndi.factory</name>
      <value>org.jnp.interfaces.NamingContextFactory</value>
    </prop>
      <prop>
        <client>ianywhere.connector.jboss</client>
        <name>xjms.jndi.url</name>
      <value>jnp://0.0.0.0:1099</value>
    </prop>
      <prop>
        <client>ianywhere.connector.jboss</client>
        <name>xjms.topicFactory</name>
      <value>ConnectionFactory</value>
    </prop>
      <prop>
        <client>ianywhere.connector.jboss</client>
```

```
        <name>xjms.queueFactory</name>
        <value>ConnectionFactory</value>
      </prop>
      <prop>
        <client>ianywhere.connector.jboss</client>
        <name>xjms.receiveDestination</name>
      <value>qanywhere_receive</value>
      </prop>
      <prop>
        <client>ianywhere.connector.jboss</client>
        <name>xjms.deadMessageDestination</name>
      <value>qanywhere_deadMessage</value>
      </prop>
    </SetProperty>
    <OpenConnector>
      <client>ianywhere.connector.jboss</client>
    </OpenConnector>
</actions>
```

コネクタの変更

コネクタを変更するには、コネクタを閉じてから <SetProperty> タグを使用してプロパティを変更し、コネクタを開きます。

例

次の例では、コネクタのログ・レベルを 4 に変更します。ianywhere.connector.jboss という ID を持つコネクタを閉じ、このコネクタのプロパティ logLevel を 4 に変更してから、新しいログ・レベルでコネクタをもう一度開きます。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <CloseConnector>
      <client>ianywhere.connector.jboss</client>
    </CloseConnector>
    <SetProperty>
      <prop>
        <client>ianywhere.connector.jboss</client>
      <name>ianywhere.connector.logLevel</name>
      <value>4</value>
      </prop>
    </SetProperty>
    <OpenConnector>
      <client>ianywhere.connector.jboss</client>
    </OpenConnector>
</actions>
```

コネクタの削除

コネクタを削除するには、クライアント名だけを指定して <SetProperty> を使用します。

例

次の例では、ianywhere.connector.jboss という ID を持つコネクタを閉じます。このコネクタのすべてのプロパティは、<name> タグと <value> タグを指定せずに <SetProperty> タグを使用することで削除されます。


```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <CloseConnector>
      <client>ianywhere.connector.jboss</client>
   </CloseConnector>
   <SetProperty>
      <prop>
        <client>ianywhere.connector.jboss</client>
    </prop>
   </SetProperty>
</actions>
```

コネクタを開く

コネクタを開くには、<OpenConnector> を使用します。

<OpenConnector> タグには、開くコネクタの名前を指定する <client> タグが 1 つ含まれます。

<OpenConnector> タグのサブタグ	説明
<client>	開くコネクタの名前

参照

- 「JMS コネクタ」 135 ページ
- 「Web サービス・コネクタの設定」 209 ページ

例

次の例では、simpleGroup というコネクタを開きます。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <OpenConnector>
    <client>simpleGroup</client>
   </OpenConnector>
</actions>
```

コネクタを閉じる

コネクタを閉じるには、<CloseConnector> タグを使用します。<CloseConnector> には、閉じるコネクタの名前を指定する <client> タグが 1 つ含まれます。

<CloseConnector> タグのサブタグ	説明
<client>	閉じるコネクタの名前

参照

- 「JMS コネクタ」 135 ページ
- 「Web サービス・コネクタの設定」 209 ページ

例

次の例では、simpleGroup というコネクタを閉じます。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <CloseConnector>
    <client>simpleGroup</client>
   </CloseConnector>
</actions>
```

コネクタのモニタ

コネクタの情報を取得するには、クライアント・ステータス要求という特別な種類のサーバ管理要求を作成します。これには <ClientStatusRequest> タグが含まれています。このタグでは、要求の登録に必要な情報を含む 1 つまたは複数の <request> タグを使用します。

クライアント・ステータス要求では、次のような複数の方法でコネクタに関するレポートを取得します。

- ワンタイム要求を作成する。
- コネクタのステータスが変更するたびにレポートが送信される状態変更リスナ (State Change Listner) を登録する。
- コネクタでエラーが発生するたびにレポートが送信されるエラー・リスナを登録する。

レポートを特定の時刻や一定の間隔で送信するようにスケジュール設定することもできます。

ClientStatusRequest タグ

コネクタの情報を取得するには、<ClientStatusRequest> タグを使用します。

クライアント・ステータス要求は、要求の登録に必要なすべての情報を含む 1 つまたは複数の <request> タグから構成されています。

<ClientStatusRequest> タグのサブタグ	
<request>	要求内の情報をグループ化します。

クライアント・ステータス要求の <request> タグ

この要求により生成されるレポートごとに返信アドレスを指定するには、<request> タグ内で、オプションの <replyAddr> タグを使用します。このタグが指定されていない場合は、レポートのデフォルトの返信アドレスが送信メッセージの返信アドレスになります。

各レポートに含まれる要求のラベルを追加するには、オプションの <requestId> タグを使用します。複数の要求を登録する場合や、要求を削除または変更する場合は、特定の要求により生成されたレポートがどれであるかを、この ID で特定できます。

要求のコネクタのリストを指定するには、1 つまたは複数の <client> タグが含まれるようにします。タグごとにコネクタ・アドレスを 1 つ指定してください。ワンタイム要求の場合は、すべて

のコネクタがレポートに含まれます。イベント・リスナ要求の場合は、サーバは各コネクタを受信します。

サーバのダウンタイムを通してイベントの詳細を永久的にするように指定するには、<persistent> タグを指定します。このタグにはデータを指定する必要はありません。<persistent/> または <persistent></persistent> のどちらのフォームを使用することもできます。

オプションで 1 つまたは複数の <onEvent> タグを含めて、イベントのリストを指定できます。<onEvent> タグごとにイベント・タイプを 1 つ指定してください。このタグの指定がない場合は、クライアント・ステータス要求でワンタイム要求が生成されます。それ以外の場合は、指定されたイベントのイベント・リスナが、クライアント・ステータス要求で登録されます。

クライアント・ステータス要求の <request> タグのサブタグ	説明
<client>	1 つまたは複数の <client> タグを含めることができます。タグごとにコネクタ・アドレスを 1 つ指定してください。ワンタイム要求の場合は、列挙されているすべてのコネクタがレポートに含まれます。イベント・リスナ要求の場合は、サーバは各コネクタの受信を開始します。
<onEvent>	発生時にサーバにレポートを生成させるイベントを指定します。1 つまたは複数の <onEvent> タグを含めることができます。タグごとにイベント・タイプを 1 つ指定してください。<onEvent> タグが指定されていない場合は、クライアント・ステータス要求でワンタイム要求が生成されます。それ以外の場合は、指定されたイベントのイベント・リスナが、クライアント・ステータス要求で登録されます。
<persistent>	このクライアント・ステータス要求の詳細情報をサーバ・データベース内で永続的にすることを指定します。
<replyAddr>	この要求により生成される各レポートの返信アドレスを指定します。このタグが指定されていない場合は、レポートのデフォルトの返信アドレスが送信メッセージの返信アドレスになります。
<requestId>	レポートのラベルです。この値は要求のラベルとして使用され、この要求により生成される各レポートに含まれます。複数の要求が登録されており、未処理の要求を削除または変更する場合には、特定の要求により生成されたレポートがどれであるかを、このラベルで特定できます。
<schedule>	「サーバ管理要求のスケジュール設定」 106 ページを参照してください。

<condition> タグ

要求をフィルタリングするには、<condition> タグのサブタグを使用します。<condition> タグ内では、次に示すサブタグをいくつでも使用できます。同じタグを複数使用した場合、値は論理 OR 結合されます。これに対して、異なる 2 つのタグを使用した場合は、値は AND 結合されます。

<condition> タグのサブタグ	説明
<messageId>	特定のメッセージ ID を持つメッセージを選択します。
<status>	現在ステータスが指定されているメッセージを選択します。
<priority>	現在優先度が指定されているメッセージを選択します。
<address>	指定したアドレスに送信されるメッセージを選択します。
<originator>	指定したクライアントから発信されるメッセージを選択します。
<kind>	バイナリまたはテキストのメッセージをフィルタリングします。 たとえば、**<kind>text</kind>** と指定するとテキスト・メッセージがフィルタリングされ、**<kind>binary</kind>** と指定するとバイナリ・メッセージがフィルタリングされます。
<property>	指定されたメッセージ・プロパティを持つメッセージを選択します。プロパティ名と値を確認するには、**<property>***property-name=property-value***</property>** 構文を使用します。プロパティが存在するかどうかを確認するには、**<property>***property-name***</property>** フォーマットを使用します。
<customRule>	ルールに基づいてメッセージを選択します。「カスタム・メッセージ要求」 103 ページを参照してください。

ワンタイム・クライアント・ステータス要求

ワンタイム要求を作成するには、クライアント・ステータス要求から <onEvent> タグを除外します。このようにすると、クライアント・ステータス要求で指定された各コネクタについて、現在のステータス情報を含むレポートが 1 つ生成されます。

次の XML メッセージは、<onEvent> タグが指定されていないワンタイム要求の例です。<ClientStatusRequest> タグで指定された各コネクタについて、現在のステータス情報を含むレポートが 1 つ生成されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <ClientStatusRequest>
      <request>
         <replyAddr>ianywhere.connector.beajms¥q11</replyAddr>
         <requestId>myOneTimeRequest</requestId>
         <client>ianywhere.server</client>
         <client>ianywhere.connector.beajms</client>
```



```
      </request>
    </ClientStatusRequest>
  </actions>
```

イベント発生時のクライアント・ステータス要求

発生時に QAnywhere サーバにステータス・レポートを生成させるイベントを指定するには、クライアント・ステータス要求に 1 つまたは複数の <onEvent> タグが含まれるようにします。ワンタイム要求とは異なり、サーバは要求にすぐに応答しません。代わりにコネクタの受信を開始して、指定されたイベントが発生するまで待機します。いずれかのイベントがトリガされるたびに、イベントを発生させたコネクタの情報を含むレポートが送信されます。

次のイベントは、イベント発生時の要求で使用できます。

イベント	発生するタイミング
open	閉じられているコネクタが開かれたとき
close	開いていたコネクタ、または一時停止状態のコネクタが閉じられたとき
statusChange	コネクタのステータスに変更があったとき。該当するステータスとして、オープンとクローズがあります。
error	予期しないエラーがコネクタからスローされたとき
fatalError	処理できない致命的なエラーがコネクタからスローされたとき
none	このイベントが発生することはありません。これまでのイベント・ウォッチが、コネクタからすべて削除されます。

次の例では、サーバ・コネクタのステータスに変更があるたびに、またはサーバ・コネクタがエラーを生成するたびに、アドレスが ianywhere.connector.beajms¥q11 のコネクタがステータス・レポートに送信されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <ClientStatusRequest>
    <request>
      <replyAddr>ianywhere.connector.beajms¥q11</replyAddr>
      <requestId>myEventRequest</requestId>
      <client>ianywhere.server</client>
      <onEvent>statusChange</onEvent>
      <onEvent>error</onEvent>
    </request>
  </ClientStatusRequest>
</actions>
```

複数の同時要求

サーバ・コネクタを含む任意の数のコネクタについて、返信アドレスごとに独自のイベント・リスナ・セットを指定できます。コネクタにイベント・リスナを追加しても、サーバの他のイベント・リスナが妨害されることはありません (ただし、置換中のイベント・リスナは妨害される可能性があります)。

要求の置換

イベント・リスナがすでに登録されているコネクタに対して、同じ返信アドレスでイベント・リスナを追加すると、古いリスナが新しいリスナで置換されます。たとえば、コネクタ abc の statusChange リスナがアドレス x/y に登録されている場合に、abc のエラー・リスナをアドレス x/y に登録すると、abc は statusChange イベントに応答しなくなります。

同じアドレスに複数のイベントを登録するには、複数の <onEvent> タグを使用して 1 つの要求を作成します。

要求の削除

コネクタのイベント・リスナがアドレスに登録されている場合は、"none" イベントを指定して同じアドレスから別のクライアント・ステータス要求を指定することで、イベント・リスナを削除できます。

次の例では、アドレス ianywhere.connector.beajms¥q11 に登録されているサーバ・コネクタから、すべてのイベント・リスナが削除されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <ClientStatusRequest>
      <request>
         <replyAddr>ianywhere.connector.beajms¥q11</replyAddr>
         <client>ianywhere.server</client>
         <onEvent>none</onEvent>
      </request>
   </ClientStatusRequest>
</actions>
```

永続的なクライアント・ステータス要求

要求の詳細をメッセージ・ストアのグローバル・プロパティに保存するように指定し、サーバの再起動後に要求の詳細を自動的に回復できるようにするには、クライアント・ステータス要求に <persistent> タグが含まれるようにします。永続性の機能は、スケジュール設定されたイベントとイベント・リスナで使用できますが、ワンタイム要求では使用できません。永続的な要求を追加および削除するルールは、通常の要求のルールに類似しています。ただし、スケジュール設定されたイベントとイベント・リスナを別々に追加することはできません。代わりに、クライアントは永続的な要求を追加するときに、特定のコネクタ／返信アドレス・ペアのすべてのイベント・リスナとスケジュールを同じ要求に指定する必要があります。

次の例では、イベント・リスナとスケジュールを ianywhere.connector.myConnector に追加して、これらを永続的にします。また、このコネクタ／返信アドレス・ペアに永続的な要求が既存していた場合は、それらを上書きします。レポートは 30 分ごとに送信されます。コネクタのステータスに変更があった場合にも送信されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <ClientStatusRequest>
      <request>
         <replyAddr>ianywhere.connector.beajms¥q11</replyAddr>
         <client>ianywhere.connector.myConnector</client>
         <onEvent>statusChange</onEvent>
         <schedule>
            <everyminute>30</everyminute>
         </schedule>
         <persistent/>
```



```
      </request>
    </ClientStatusRequest>
  </actions>
```

イベント・リスナの永続性

コネクタが自身のアドレスに登録したイベント・リスナは、コネクタが閉じられた場合でも、サーバが停止するまでサーバ上に存在します。コネクタが再び開かれた場合は、ストアド・イベント・リスナがもう一度アクティブになります。

コネクタのステータス

コネクタのステータスは、次の 2 つのうちのいずれかになります。

- **実行中**　コネクタは着信と送信メッセージを受け入れて処理しています。このステータスの場合は、コネクタ・プロパティ ianywhere.connector.state が 1 になります。
- **非実行中**　コネクタは、着信と送信メッセージの受け入れと処理を行っていません。コネクタのステータスが「実行中」に変わると、コネクタは最初から初期化されます。このステータスの場合は、コネクタ・プロパティ ianywhere.connector.state が 2 になります。

コネクタのステータスの変更方法については、「コネクタの変更」 113 ページを参照してください。

クライアント・ステータス・レポート

サーバは、コネクタがレポートを要求するたび、または登録されているイベントが発生するたびに、クライアント・ステータス・レポートを生成します。クライアント・ステータス・レポートは、メッセージ・プロパティを含まない簡単なテキスト・メッセージで生成されます。

イベントの発生時に取得できる情報の種類に応じて、各レポートには次のいずれかの値が含まれます。

- クライアント (常に表示される)
- UTCDatetime (常に表示される)
- vendorStatusDescription (常に表示される)
- statusCode (常に表示される)
- vendorStatusCode
- statusSubCode
- statusDescription

次に例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<ClientStatusReport>
   <requestId>myRequest</requestId>
   <componentReport>
      <client>ianywhere.server</client>
      <UTCDatetime>Tue May 31 13:53:02 EDT 2005</UTCDatetime>
      <statusCode>Running</statusCode>
      <vendorStatusDescription></vendorStatusDescription>
   </componentReport>
   <componentReport>
```

```
        <client>ianywhere.connector.beajms</client>
        <UTCDatetime>Tue May 31 13:53:02 EDT 2005</UTCDatetime>
        <statusCode>Not running</statusCode>
        <vendorStatusDescription></vendorStatusDescription>
    </componentReport>
</ClientStatusReport>
```

サーバ管理要求でのサーバ・プロパティの設定

<SetProperty> タグには 1 つまたは複数の <prop> タグが含まれています。各タグは、設定するプロパティを指定します。各 prop タグは、<client> タグ、<name> タグ、<value> タグから構成されています。プロパティを削除するには、<value> タグを除外します。

<prop> タグのサブタグ	説明
<client>	サーバ・プロパティを設定するクライアントの名前です。
<name>	設定するプロパティの名前です。
<value>	設定されるプロパティの値です。このタグが含まれていない場合は、プロパティが削除されます。

サーバ管理要求の使い方の概要、サーバ管理要求の認証方法とスケジュールの設定方法については、「サーバ管理要求の概要」 100 ページを参照してください。

例

次のサーバ管理要求は、simpleGroup という送信先エイリアスの ianywhere.qa.member.client3 プロパティを Y に設定して、client3 を simpleGroup に追加します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <SetProperty>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.qa.member.client3</name>
      <value>Y</value>
    </prop>
   </SetProperty>
</actions>
```

次の例では、以下のことを実行します。

- client1 プロパティの myProp1 を作成または変更して値を 3 に設定する。
- client1 プロパティの myProp2 を削除する。
- client2 プロパティの myProp3 の値を "some value" に変更する。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <SetProperty>
   <prop>
      <client>client1</client>
      <name>myProp1</name>
      <value>3</value>
   </prop>
   <prop>
      <client>client1</client>
      <name>myProp2</name>
   </prop>
   <prop>
      <client>client2</client>
      <name>myProp3</name>
```

```
        <value>some value</value>
    </prop>
    </SetProperty>
</actions>
```


サーバ管理要求での転送ルールの指定

サーバ管理要求を使用して、すべてのユーザに適用されるサーバ側のデフォルトの転送ルールを指定したり、クライアントごとに転送ルールを指定したりできます。

サーバのデフォルトの転送ルールを指定するには (サーバの場合)、クライアント ianywhere.server.defaultClient の ianywhere.qa.server.rules プロパティを設定します。クライアントの場合は、ianywhere.qa.server.rules プロパティを使用してサーバの転送ルールを指定します。

サーバ管理要求の使い方の概要、サーバ管理要求の認証方法とスケジュールの設定方法については、「サーバ管理要求の概要」 100 ページを参照してください。

例

次の例では、優先度の高い (優先度が 7 以上) メッセージだけを送信するデフォルト・ルールを作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <SetProperty>
    <prop>
      <client>ianywhere.server.defaultClient</client>
      <name>ianywhere.qa.server.rules</name>
      <value>auto = ias_Priority &gt; 6</value>
    </prop>
  </SetProperty>
  <RestartRules>
    <client>ianywhere.server.defaultClient</client>
  </RestartRules>
</actions>
```

次の例では、内容のサイズが 100 未満で、午前 8 時から午後 6 時の営業時間内に送信する必要のあるメッセージのみを送信するルールを作成し、defaultClient というクライアントに適用します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <SetProperty>
    <prop>
      <client>ianywhere.server.defaultClient</client>
      <name>ianywhere.qa.server.rules</name>
      <value>auto = ias_ContentSize &lt; 100
        or ias_CurrentTime &gt; '8:00:00'
        or ias_CurrentTime &lt; '18:00:00'</value>
    </prop>
  </SetProperty>
  <RestartRules>
    <client>ianywhere.server.defaultClient</client>
  </RestartRules>
</actions>
```

サーバ管理要求を使用した送信先エイリアスの作成

サーバ管理要求を使用して送信先エイリアスを作成したり変更したりできます。

送信先エイリアスの詳細については、「送信先エイリアス」 57 ページを参照してください。

サーバ管理要求の使い方の概要、サーバ管理要求の認証方法とスケジュールの設定方法については、「サーバ管理要求の概要」 100 ページを参照してください。

送信先エイリアスを作成するには、その送信先エイリアスと同じ名前のクライアントにサーバ管理要求を送信します。次のプロパティを指定します。グループは、group、address、nativeConnection の各プロパティで識別されます。グループのメンバは、member プロパティで指定します。

```
<prop>
  <client>simpleGroup</client>
  <name>ianywhere.connector.nativeConnection</name>
  <value>ianywhere.message.connector.group.GroupConnector
  </value>
</prop>
```

プロパティ	説明

ianywhere.qa.group	送信先エイリアスを設定していることを示すには、このプロパティを Y に設定します。次に例を示します。
	
	`<prop>`
	`<client>simpleGroup</client>`
	`<name>ianywhere.qa.group</name>`
	`<value>Y</value>`
	`</prop>`

ianywhere.connector.address	送信先エイリアスのクライアント ID を指定します。次に例を示します。
	
	`<prop>`
	`<client>simpleGroup</client>`
	`<name>ianywhere.connector.address</name>`
	`<value>simpleGroup</value>`
	`</prop>`

ianywhere.connector.nativeConnection	ianywhere.message.connector.group.GroupConnector に設定します。次に例を示します。
	
	`<prop>`
	`<client>simpleGroup</client>`
	`<name>ianywhere.connector.nativeConnection</name>`
	
	`<value>ianywhere.message.connector.group.GroupConnector`
	`</value>`
	`</prop>`

プロパティ	説明
ianywhere.qa.member.*client-name* ¥*queue-name*	メンバを追加する場合は Y、メンバを削除する場合は N を指定します。オプションで配信条件を指定することもできます。「条件構文」 244 ページを参照してください。たとえば、client1 を送信先エイリアス simpleGroup に追加するには、プロパティを次のように設定します。キュー名はオプションです。追加するクライアントごとに、このプロパティを繰り返し設定します。 <prop> <client>simpleGroup</client> <name>ianywhere.qa.member.client1¥queue1</name> <value>Y</value> </prop>

サーバ管理要求の詳細については、「サーバ管理要求の概要」 100 ページを参照してください。

参照

- 「QAnywhere 転送ルールと削除ルール」 241 ページ

例

次のサーバ管理要求では、client1 と client2¥q11 というメンバを持つ送信先エイリアス simpleGroup を作成します。この例では送信先エイリアスを起動して、メッセージの処理をすぐに開始します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <SetProperty>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.qa.group</name>
      <value>Y</value>
    </prop>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.connector.address</name>
      <value>simpleGroup</value>
    </prop>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.connector.nativeConnection</name>
      <value>ianywhere.message.connector.group.GroupConnector</value>
    </prop>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.connector.logLevel</name>
      <value>4</value>
    </prop>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.qa.member.client1</name>
      <value>Y</value>
    </prop>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.qa.member.client2¥q11</name>
      <value>Y</value>
```

```
  </prop>
 </SetProperty>
 <OpenConnector>
  <client>simpleGroup</client>
 </OpenConnector>
</actions>
```

送信先エイリアスのメンバの追加と削除

送信先エイリアスにメンバを追加するには、プロパティでメンバを指定するサーバ管理要求を作成します。グループを再起動してメンバ設定を有効にしてください。

次の例では、メンバ client3 を追加して、グループ simpleGroup を再起動します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <SetProperty>
   <prop>
      <client>simpleGroup</client>
      <name>ianywhere.qa.member.client3</name>
      <value>Y</value>
   </prop>
   </SetProperty>
   <CloseConnector>
   <client>simpleGroup</client>
   </CloseConnector>
   <OpenConnector>
   <client>simpleGroup</client>
   </OpenConnector>
</actions>
```

送信先エイリアスからメンバを削除するには、メンバを削除する必要があることを示すプロパティを持つサーバ管理要求を作成します。グループを再起動してメンバの削除設定を有効にする必要があります。

次の例では、メンバ client3 を削除して、グループ simpleGroup を再起動します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <SetProperty>
   <prop>
      <client>simpleGroup</client>
      <name>ianywhere.qa.member.client3</name>
   </prop>
   </SetProperty>
   <CloseConnector>
   <client>simpleGroup</client>
   </CloseConnector>
   <OpenConnector>
   <client>simpleGroup</client>
   </OpenConnector>
</actions>
```


QAnywhere のモニタ

サーバ管理要求を使用して、メッセージ・セットの情報を取得できます。サーバは情報をコンパイルし、メッセージとしてクライアントに返信します。ワンタイムのメッセージの詳細要求を作成することも、スケジュールを設定してメッセージの詳細要求を自動的に実行することもできます。また、要求を永続的にすることを指定し、サーバが再起動された場合でもメッセージが送信されるようにすることもできます。

サーバ管理要求の使い方の概要、サーバ管理要求の認証方法とスケジュールの設定方法については、「サーバ管理要求の概要」 100 ページを参照してください。

メッセージの詳細要求

メッセージの詳細要求に関するサーバ管理要求を作成するには、<MessageDetailsRequest> タグを使用します。

メッセージの詳細要求には、要求の登録に必要なすべての情報を含む 1 つまたは複数の <request> タグが含まれます。複数の <request> タグを指定することは、複数のメッセージ詳細要求を別々に送信することと同じです。

この要求により生成される各レポートの返信アドレスを指定するには、オプションの <replyAddr> タグを使用します。このタグが指定されていない場合は、レポートのデフォルトの返信アドレスが送信メッセージの返信アドレスになります。

この要求により生成される各レポートに含まれる要求に対して、ユニークな識別子を指定するには、<requestId> タグを使用します。このフィールドに複数の値を指定すると、一度に複数の要求をアクティブにできます。同じ要求 ID を指定すると、クライアントはアクティブな要求を上書きまたは削除します。

レポートに含めるメッセージを特定するには、<condition> タグを指定します。「Condition タグ」 103 ページを参照してください。

詳細リストを指定して、各メッセージのどの詳細情報をレポートに含めるかを特定することもできます。これを行うには、一連の空の詳細要素タグを要求に含めます。

サーバのダウンタイムを通してイベントの詳細を永続的にすることを指定するには、<persistent> タグを使用します。このタグにはデータを指定する必要はありません。<persistent/> または <persistent></persistent> のどちらのフォームを使用することもできます。

スケジュール設定されたレポートの登録に必要な詳細情報をすべて含めるには、<schedule> を使用します。「サーバ管理要求のスケジュール設定」 106 ページを参照してください。

<MessageDetailsRequest> タグのサブタグ	説明
<request>	特定の要求に関する情報をグループ化します。複数の <request> タグを指定することは、メッセージ情報に関する複数のサーバ管理要求を別々に送信することと同じです。以下を参照してください。

Request タグ

<Request> タグのサブタグ	説明
<address>	各メッセージのアドレスを表示します。
<condition>	メッセージをレポートに含めるための条件をグループ化します。「Condition タグ」 103 ページを参照してください。
<contentSize>	各メッセージの内容のサイズを要求します。
<customRule>	「カスタム・メッセージ要求」 103 ページを参照してください。
<expires>	各メッセージの有効期限を要求します。
<kind>	メッセージがテキストであるかバイナリであるかを要求します。
<messageId>	各メッセージのメッセージ ID を要求します。
<originator>	各メッセージの発信者を要求します。
<persistent>	このタグを含めることで、要求の結果をサーバ・データベース内で永続的にすることを示す。これにより、サーバが再起動された場合であってもレポートが送信されます。
<priority>	各メッセージの優先度を要求します。
<property>	すべてのメッセージ・プロパティと各メッセージの値のリストを要求します。
<statusTime>	各メッセージのステータス時間を要求します。
<replyAddr>	この要求により生成される各レポートの返信アドレスを指定します。このタグが指定されていない場合は、レポートのデフォルトの返信アドレスが送信メッセージの返信アドレスになります。
<requestId>	要求のユニークな識別子です。この要求により生成される各レポートに含まれます。このフィールドに複数の値を指定すると、一度に複数の要求をアクティブにできます。同じ要求 ID を指定すると、クライアントはアクティブな要求を無効または削除できます。
<schedule>	スケジュールに従ってレポートを生成することを指定します。<schedule> タグのサブタグは、レポートを実行するスケジュールを特定します。「サーバ管理要求のスケジュール設定」 106 ページを参照してください。
<status>	各メッセージのステータスを要求します。
<transmissionStatus>	各メッセージの転送ステータスを要求します。

MessageDetailsReport タグ

メッセージの詳細レポートは、<MessageDetailsReport> タグを含む XML メッセージです。このレポートは、レポート・ヘッダとそれに続くオプションの <message> タグから構成されます。各レポートのヘッダは、次のタグから構成されます。

<MessageDetailsReport> タグのサブタグ	説明
<message>	レポートの本文は <message> タグのリストから構成されます。<message> タグのサブタグは、選択基準を満たすメッセージごとに特定の詳細情報を表示します。メッセージが選択されていない場合や、元の要求で詳細要素が指定されていない場合は、<message> タグはレポートに含まれません。それ以外の場合は、メッセージごとに独自の <message> タグがあります。
<messageCount>	要求の選択基準を満たすメッセージの数です。
<requestId>	レポートを生成した要求の ID です。
<statusDescription>	レポートが生成された理由の簡単な説明です。
<UTCDateline>	レポートが生成された時刻と日付です。

Message タグ

<message> タグのサブタグ	説明
<address>	メッセージのアドレス。myclient¥myqueue など。
<contentSize>	メッセージのサイズ。テキスト・メッセージの場合は文字数、バイナリ・メッセージの場合はバイト数。
<expires>	メッセージが配信されない場合に期限切れとなる日付と時刻。
<kind>	メッセージがバイナリ (1) であるか、テキスト (2) であるかを示します。
<messageId>	新規メッセージのメッセージ ID。「メッセージ・ヘッダ」 220 ページを参照してください。
<originator>	メッセージの発信者のメッセージ・ストア ID。
<priority>	0 ー 9 の整数で指定したメッセージの優先度。0 は優先度が最も低く、9 は優先度が高いことを表します。
<property>	メッセージのプロパティ。「メッセージ・プロパティ」 223 ページを参照してください。
<status>	メッセージの現在のステータス。ステータス・コードは「事前に定義されたメッセージ・プロパティ」 224 ページで定義されています。
<statusTime>	メッセージが現在のステータスになった時刻。これは現地時刻です。

<message> タグのサブタグ	説明
<transmissionStatus>	メッセージの同期ステータス。次のいずれかの値を取ります。 ◆ 0 - メッセージは、目的の受信側メッセージ・ストアに転送されていない。 ◆ 1 - メッセージは、目的の受信側メッセージ・ストアに転送済み。 ◆ 2 - 受信先および送信元のメッセージ・ストアが同じであるため、転送は不要。 ◆ 3 - メッセージは目的の受信先に転送されたが、まだ受信確認されていない。メッセージ転送が中断された可能性があるため、QAnywhere がメッセージを再転送する可能性があります。

例

次に、メッセージの詳細レポートの例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<MessageDetailsReport>
   <requestId>testReport</requestId>
   <UTCDatetime>Mon Jan 16 15:03:04 EST 2006</UTCDatetime>
   <statusDescription>Scheduled report</statusDescription>
   <messageCount>1</messageCount>
   <message>
      <messageId>ID:26080b8927f83f9722357eab0a0628eb</messageId>
      <status>60</status>
      <property>
         <name>myPropName</name>
         <value>myPropVal</value>
      </property>
   </message>
</MessageDetailsReport>
```

次の <condition> タグは、検索基準の (msgId=ID:144... OR msgId=ID225...)、(status=pending)、(kind=textmessage)、(contains the property 'myProp' with value 'myVal') を満たすメッセージを選択します。

```
<condition>
   <messageId>ID:144d7e44dc2d7e1d</messageId>
   <messageId>ID:22578sd5dsd99s8e</messageId>
   <status>1</status>
   <kind>text</kind>
   <property>myProp=myVal</property>
</condition>
```

ワンタイム要求は、<schedule> タグが指定されていない要求です。ワンタイム要求は、単一レポートを生成する場合に使用され、レポートが送信されるとすぐに削除されます。次の要求では、現在サーバ上に置かれている優先度 9 のすべてのメッセージについて、メッセージ ID、ステータス、送信先アドレスを表示するレポートを 1 つ生成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <MessageDetailsRequest>
      <request>
         <requestId>testRequest</client>
         <condition>
            <priority>9</priority>
```



```
            </condition>
            <messageId/>
            <status/>
            <address/>
        </request>
    </MessageDetailsRequest>
</actions>
```

次のメッセージの詳細要求サンプルでは、メッセージ ID とメッセージ・ステータスを含むレポートを生成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <MessageDetailsRequest>
    <!-- ... -->
    <messageId />
    <status />
  </MessageDetailsRequest>
</actions>
```

QAnywhere クライアントのモニタ

サーバ管理要求を使用して、現在サーバ上にあるクライアントのリストを取得できます。このリストには、サーバに登録されているクライアントが含まれます。リモート・クライアント、開いているコネクタ、送信先エイリアスも含まれます。

サーバ管理要求の使い方の概要、サーバ管理要求の認証方法とスケジュールの設定方法については、「サーバ管理要求の概要」 100 ページを参照してください。

クライアントのリストを取得するには、サーバ管理要求で <GetClientList> タグを使用します。次に例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <GetClientList/>   (or <GetClientList></GetClientList> )
</actions>
```

生成された応答は、要求が含まれているメッセージの返信先アドレスに送信されます。応答には <client> タグのリストが含まれています。タグごとに、サーバに接続されているクライアント名が 1 つ示されています。次に例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<GetClientListResponse>
   <client>ianywhere.server</client>
   <client>ianywhere.connector.myConnector</client>
 <client>myClient</client>
</GetClientListResponse>
```


プロパティのモニタ

サーバ管理要求を使用して、クライアントに設定されているプロパティの内容を確認できます。応答で一覧されるプロパティは、クライアントに設定されているプロパティだけです。デフォルトのプロパティは含まれません。

サーバ管理要求の使い方の概要、サーバ管理要求の認証方法とスケジュールの設定方法については、「サーバ管理要求の概要」 100 ページを参照してください。

クライアントのプロパティのリストを取得するには、サーバ管理要求で <GetProperties> タグを使用します。次に例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <GetProperties>
      <client>ianywhere.connector.myConnector</client>
   </GetProperties>
</actions>
```

生成された応答は、要求が含まれているメッセージの返信先アドレスに送信されます。応答にはクライアント名と <prop> タグのリストが含まれます。タグごとに 1 つのプロパティの詳細が含まれています。次に例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<GetPropertiesResponse>
   <client>ianywhere.connector.myConnector</client>
   <prop>
      <name>ianywhere.connector.logLevel</name>
      <value>4</value>
   </prop>
   <prop>
      <name>ianywhere.connector.state</name>
      <value>2</value>
   </prop>
</GetPropertiesResponse>
```

第 6 章

JMS コネクタ

目次

JMS コネクタの概要

JMS (Java Message Service) API は、Java アプリケーションにメッセージング機能を追加する役割を果たします。メッセージの交換は、QAnywhere クライアント・アプリケーション間だけでなく、JMS インタフェースをサポートする外部メッセージング・システムとの間でも行うことができます。これには、コネクタと呼ばれる特殊なクライアントを使用します。QAnywhere アプリケーションでは、外部メッセージング・システムを、QAnywhere クライアントと同様に動作するように設定します。これには、固有のアドレスと構成を使用します。

ここで説明したアプローチを採用したアーキテクチャの詳細については、「外部メッセージング・システムとのメッセージングのシナリオ」 10 ページを参照してください。

JMS コネクタの設定

以下の手順では、QAnywhere に JMS コネクタを統合するのに必要なタスクの概要について説明します。この手順では、QAnywhere がすでに設定されているものとします。

♦ **QAnywhere アプリケーションと外部 JMS システムの統合の概要**

1. JMS システムの JMS 管理ツールを使用して、JMS キューを作成します。QAnywhere コネクタは、1 つの JMS キューを使用して JMS メッセージを受信します。このキューが存在しない場合、作成する必要があります。

   キューの作成方法の詳細については、ご使用の JMS 製品のマニュアルを参照してください。

2. Sybase Central を開き、サーバ・メッセージ・ストアに接続します。

3. [ファイル] – [New Connector] を選択します。

   [コネクタ] ウィザードが表示されます。

4. JMS が選択されていることを確認し、使用している Web サーバのタイプを選択します。[次へ] をクリックします。

5. [コネクタ名] ページで、次の値を入力します。

   - **コネクタ名**　コネクタのアドレスを指定します。このアドレスは、QAnywhere クライアントで、コネクタを指定するために使用されます。

     「QAnywhere から JMS に送信されるメッセージのアドレス指定」 146 ページを参照してください。

   - **受信側**　JMS からの QAnywhere クライアント宛てのメッセージを受信するコネクタが使用するキュー名を指定します。

6. [JNDI の設定] ページで、次の値を入力します。

   - **JNDI ファクトリ**　外部の JMS JNDI ネームサービスにアクセスするときに使用するファクトリ名を指定します。
   - **ネーム・サービス URL**　JMS JNDI ネームサービスにアクセスするときに使用する URL を指定します。
   - **ユーザ名**　外部の JMS JNDI ネーム・サービスに接続するときに使用する認証名を指定します。
   - **パスワード**　外部の JMS JNDI ネームサービスに接続するときに使用する認証パスワードを指定します。

7. [JMS キューの設定] ページで、次の値を入力します。

   - **キュー・ファクトリ**　外部 JMS プロバイダのキュー・ファクトリ名を指定します。
   - **ユーザ名**　外部 JMS キュー接続に接続するときに使用するユーザ ID を指定します。

- **パスワード**　外部 JMS キュー接続に接続するときに使用するパスワードを指定します。

8. [JMS トピックの設定] ページで、次の値を入力します。

   - **トピック・ファクトリ**　外部 JMS プロバイダのトピック・ファクトリ名を指定します。
   - **ユーザ名**　外部 JMS トピック接続に接続するときに使用するユーザ ID を指定します。
   - **パスワード**　外部 JMS トピック接続に接続するときに使用するパスワードを指定します。

9. [終了] をクリックします。

   mlsrv10 コマンド・ラインでクライアント JAR ファイルを追加するようにプロンプトが表示されます。

10. サーバ・メッセージ・ストアに接続するための接続文字列と -sl java オプションを指定して、Mobile Link サーバを起動します。

    「JMS 統合用 Mobile Link サーバの起動」 140 ページを参照してください。

11. JMS コネクタにその他のオプションを設定するには、作成したコネクタを右クリックし、プロパティを選択します。または、サーバ管理要求を使用することもできます。

    使用可能なプロパティのリストについては、「JMS コネクタ・プロパティ」 141 ページを参照してください。

    サーバ管理要求を使用してコネクタのプロパティを設定する方法については、「コネクタの管理」 112 ページを参照してください。

- **メッセージを送信するには、次の手順に従います。**

1. QAnywhere システム内のアプリケーションから外部メッセージング・システムにメッセージを送信するには、QAnywhere メッセージを作成し、**connector-address¥JMS-queue-name** に送信します。

   「QAnywhere から JMS に送信されるメッセージのアドレス指定」 146 ページを参照してください。

2. 外部メッセージング・システムから QAnywhere システム内のアプリケーションにメッセージを送信するには、次の手順に従います。

   - JMS メッセージを作成します。
   - ias_ToAddress プロパティを QAnywhere の **id¥queue** (*id* はクライアント・メッセージ・ストアの ID、*queue* はアプリケーション・キュー名) に設定します。
   - そのメッセージを JMS キューに登録します。

   「JMS から QAnywhere に送信されるメッセージのアドレス指定」 149 ページを参照してください。

クイック・スタートのためのその他の資料

- *samples-dir¥QAnywhere¥connectors* に、QAnywhere JMS のサンプルがあります。*samples-dir* の詳細については、「サンプル・ディレクトリ」『SQL Anywhere サーバ - データベース管理』を参照してください。

JMS 統合用 Mobile Link サーバの起動

JMS インタフェースをサポートする外部メッセージング・システムとメッセージを交換するには、Mobile Link サーバ (mlsrv10) を起動するときに次のオプションを指定する必要があります。

- **-c *connection-string*** サーバ・メッセージ・ストアに接続します。

  「-c オプション」 『Mobile Link - サーバ管理』を参照してください。

- **-m** QAnywhere のメッセージングに対応します。
- **-sl java (-cp "*jarfile*.jar")** 外部 JMS プロバイダを使用するために必要なクライアント jar ファイルを追加します。

  「-sl java オプション」 『Mobile Link - サーバ管理』を参照してください。

例

次の例では、JMS クライアント・ライブラリ *jmsclient.jar* (現在の作業ディレクトリにあります) と QAnywhere サンプル・データベース (メッセージ・ストア) を指定して、Mobile Link サーバを起動しています。このコマンドは、1 行で入力する必要があります。

```
mlsrv10 -sl java(-cp "jmsclient.jar")
        -m -c "QAnywhere 10.0 Demo" ...
```

JMS コネクタ・プロパティ

JMS コネクタ・プロパティを使用して、JMS システムとの接続情報を指定します。このプロパティにより、BEA WebLogic などのサード・パーティ製 JMS メッセージング・システムや Sybase EAServer に接続するコネクタを設定します。

プロパティの設定や参照を行うには、いくつかの方法があります。

- Sybase Central の [コネクタ] ウィザード

  「JMS コネクタの設定」 137 ページを参照してください。

- Sybase Central の [コネクタ・プロパティ] ダイアログ
- サーバ管理要求

  「コネクタの作成と設定」 112 ページを参照してください。

- ml_qa_global_props Mobile Link システム・テーブル

  「ml_qa_global_props」 『Mobile Link - サーバ管理』を参照してください。

JMS コネクタを設定するには、次のプロパティを使用します。

- **ianywhere.connector.nativeConnection** JMS コネクタを実装する Java クラスを指定します。QAnywhere の内部使用のためのものである場合は、削除や変更はできません。
- **ianywhere.connector.id (旧式)** JMS コネクタをユニークに識別する識別子を指定します。デフォルトは、コネクタ・プロパティ ianywhere.connector.address の値です。
- **ianywhere.connector.address** コネクタのアドレスを指定します。このアドレスは、QAnywhere クライアントで、コネクタを指定するために使用されます。指定したアドレスは、コネクタのサーバ・コンソールに表示されるすべてのログ出力エラー、警告、情報メッセージの先頭に付けられます。

  「QAnywhere から JMS に送信されるメッセージのアドレス指定」 146 ページを参照してください。

  Sybase Central では、このプロパティは [コネクタ] ウィザード、[コネクタ名] ページ、[コネクタ名] フィールドで設定します。
- **ianywhere.connector.incoming.retry.max** コネクタで QAnywhere のメッセージ・ストアへの JMS メッセージの転送をリトライする最大回数を指定します。この回数に達すると、JMS メッセージは ianywhere.connector.jms.deadMessageDestination プロパティの値に再転送されます。デフォルトは -1 であり、コネクタのリトライ回数は無制限です。
- **ianywhere.connector.outgoing.deadMessageAddress** 処理できないメッセージの送信先アドレスを指定します。たとえば、間違った形式の JMS アドレスや不明な JMS アドレスがメッセージに含まれている場合、そのメッセージは受信不可とマーク付けされて、メッセージのコピーがこのアドレスに送信されます。

このアドレスが指定されていない場合は、メッセージは受信不可とマーク付けされますが、メッセージのコピーは送信されません。

Sybase Central では、このプロパティは [コネクタ・プロパティ] ダイアログの [プロパティ] タブで、[新規] をクリックして設定します。

- **ianywhere.connector.logLevel** Mobile Link サーバ コンソールとログ・ファイルに出力されるコネクタ情報の詳細度を指定します。次の詳細度レベルがあります。
  - **1** エラー・メッセージを出力します。
  - **2** エラー・メッセージと警告メッセージを出力します。
  - **3** エラー、警告、情報の各メッセージを出力します。
  - **4** エラー、警告、情報、デバッグのメッセージを出力します。

  Sybase Central では、このプロパティは [コネクタ・プロパティ] ダイアログの [一般] タブにある [ロギング・レベル] セクションで設定します。

  すべてのコネクタに対して、このプロパティを設定することもできます。Sybase Central でこれを行うには、サーバ・メッセージ・ストアに接続し、[このメッセージ・ストアのプロパティの変更] を選択します。[サーバのプロパティ] タブを開きます。

- **ianywhere.connector.compressionLevel** JMS から受信したメッセージのデフォルトのメッセージ圧縮率を指定します。0 ー 9 の整数値で指定します。0 は圧縮なし、9 は最大の圧縮率を表します。

  Sybase Central では、このプロパティは [コネクタ・プロパティ] ダイアログの [一般] タブにある [圧縮レベル] セクションで設定します。

  すべてのコネクタに対して、このプロパティを設定することもできます。Sybase Central でこれを行うには、サーバ・メッセージ・ストアに接続し、[このメッセージ・ストアのプロパティの変更] を選択して、[サーバのプロパティ] タブを開きます。

- **ianywhere.connector.jms.deadMessageDestination** JMS メッセージを QAnywhere メッセージに変換できない場合に、JMS メッセージが送信されるアドレスを指定します。JMS メッセージがサポートされないクラスのインスタンスである場合、JMS メッセージに QAnywhere のアドレスが指定されていない場合、予期しない JMS プロバイダ例外が発生した場合、または予期しない QAnywhere 例外が発生した場合には、JMS メッセージを QAnywhere メッセージに変更できません。

  Sybase Central では、このプロパティは [コネクタ・プロパティ] ダイアログの [JMS] タブにある [その他] セクションで、[Dead メッセージの送信先] フィールドを使用して設定します。

- **ianywhere.connector.outgoing.retry.max** QAnywhere から外部メッセージング・システムへの出力メッセージについての、デフォルトのリトライ回数です。デフォルト値は 5 です。コネクタのリトライ回数を無制限にするには 0 を指定します。

  Sybase Central では、このプロパティは [コネクタ・プロパティ] ダイアログの [プロパティ] タブで、[新規] をクリックして設定します。

- **ianywhere.connector.runtimeError.retry.max** RuntimeException を引き起こすメッセージの処理をコネクタがリトライする回数を指定します。dead になっているメッセージ・キューが指定されている場合は、メッセージはそのキューに登録されます。それ以外の場合は、メッセージは受信不可とマーク付けされてスキップされます。サーバのリトライ回数を無制限にするには、この値に 0 を指定します。
- **ianywhere.connector.startupType** 起動のタイプには、automatic、manual、disabled のいずれかを指定できます。
- **xjms.jndi.authName** 外部の JMS JNDI ネーム・サービスに接続するときに使用する認証名を指定します。

  Sybase Central では、このプロパティは [コネクタ] ウィザード、[JNDI の設定] ページ、[ユーザ名] フィールド、または [コネクタ・プロパティ] ダイアログの [JMS] タブにある [JNDI] セクションで [ユーザ名] フィールドを使用して設定します。
- **xjms.jndi.factory** 外部の JMS JNDI ネーム・サービスにアクセスするときに使用するファクトリ名を指定します。

  Sybase Central では、このプロパティは [コネクタ] ウィザード、[JNDI の設定] ページ、[JNDI ファクトリ] フィールド、または [コネクタ・プロパティ] ダイアログの [JMS] タブにある [JNDI] セクションで [JNDI ファクトリ] フィールドを使用して設定します。
- **xjms.jndi.password.e** 外部の JMS JNDI ネームサービスに接続するときに使用する認証パスワードを指定します。

  Sybase Central では、このプロパティは [コネクタ] ウィザード、[JNDI の設定] ページ、[ネーム・サービス URL] フィールド、または [コネクタ・プロパティ] ダイアログの [JMS] タブにある [JNDI] セクションで [URL] フィールドを使用して設定します。
- **xjms.jndi.url** JMS JNDI ネームサービスにアクセスするときに使用する URL を指定します。

  Sybase Central では、このプロパティは [コネクタ] ウィザード、[JNDI の設定] ページ、[ネーム・サービス URL] フィールド、または [コネクタ・プロパティ] ダイアログの [JMS] タブにある [JNDI] セクションで [URL] フィールドを使用して設定します。
- **xjms.password.e** 外部の JMS プロバイダに接続するときに使用する認証パスワードを指定します。
- **xjms.queueConnectionAuthName** 外部 JMS キュー接続に接続するときに使用するユーザ ID を指定します。

  Sybase Central では、このプロパティは [コネクタ] ウィザード、[JMS キューの設定] ページ、[ユーザ名] フィールド、または [コネクタ・プロパティ] ダイアログの [JMS] タブにある [キュー] セクションで [ユーザ名] フィールドを使用して設定します。
- **xjms.queueConnectionPassword.e** 外部 JMS キュー接続に接続するときに使用するパスワードを指定します。

  Sybase Central では、このプロパティは [コネクタ] ウィザード、[JMS キューの設定] ページ、[パスワード] フィールド、または [コネクタ・プロパティ] ダイアログの [JMS] タブにある [キュー] セクションで [パスワード] フィールドを使用して設定します。

- **xjms.queueFactory**　外部 JMS プロバイダのキュー・ファクトリ名を指定します。

  Sybase Central では、このプロパティは [コネクタ] ウィザード、[JMS キューの設定] ページ、[キュー・ファクトリ] フィールド、または [コネクタ・プロパティ] ダイアログの [JMS] タブにある [キュー] セクションで [キュー・ファクトリ] フィールドを使用して設定します。

- **xjms.receiveDestination**　JMS からの QAnywhere クライアント宛てのメッセージを受信するコネクタが使用するキュー名を指定します。

  Sybase Central では、このプロパティは [コネクタ] ウィザード、[コネクタ名] ページ、[受信側] フィールドで設定します。

- **xjms.topicFactory**　外部 JMS プロバイダのトピック・ファクトリ名を指定します。

  Sybase Central では、このプロパティは [コネクタ] ウィザード、[JMS トピックの設定] ページ、[トピック・ファクトリ] フィールド、または [コネクタ・プロパティ] ダイアログの [JMS] タブにある [トピック] セクションで [トピック・ファクトリ] フィールドを使用して設定します。

- **xjms.topicConnectionAuthName**　外部 JMS トピック接続に接続するときに使用するユーザ ID を指定します。

  Sybase Central では、このプロパティは [コネクタ] ウィザード、[JMS トピックの設定] ページ、[ユーザ名] フィールド、または [コネクタ・プロパティ] ダイアログの [JMS] タブにある [トピック] セクションで [ユーザ名] フィールドを使用して設定します。

- **xjms.topicConnectionPassword.e**　外部 JMS トピック接続に接続するときに使用するパスワードを指定します。

  Sybase Central では、このプロパティは [コネクタ] ウィザード、[JMS トピックの設定] ページ、[パスワード] フィールド、または [コネクタ・プロパティ] ダイアログの [JMS] タブにある [トピック] セクションで [パスワード] フィールドを使用して設定します。

複数のコネクタの設定

JMS メッセージ・システムごとに JMS コネクタを定義すると、QAnywhere から複数の JMS メッセージ・システムに接続できます。コネクタのプロパティのうち、ianywhere.connector.address だけは、設定するコネクタごとにユニークにする必要があります。

ianywhere.connector.address プロパティは、QAnywhere クライアントが JMS システムへのメッセージの宛先アドレスを指定するときに使用する必要があるアドレス・プレフィクスです。

参照

- 「QAnywhere から JMS に送信されるメッセージのアドレス指定」 146 ページ
- 「JMS コネクタ・プロパティ」 141 ページ
- 「コネクタの作成と設定」 112 ページ

QAnywhere から JMS に送信されるメッセージのアドレス指定

QAnywhere クライアントは、アドレスに次の値を設定することで、JMS システムにメッセージを送信できます。

connector-address¥*JMS-queue-name*

connector-address は、コネクタ・プロパティ ianywhere.connector.address の値です。*JMS-queue-name* は、JNDI (Java Naming and Directory Interface) を使用して JMS のキューまたはトピックを検索するときに使用される名前です。

JMS-queue-name に円記号が含まれている場合は、円記号をもう 1 つ追加して、この円記号をエスケープする必要があります。たとえば、ss というコンテキストを持つ qq というキューは、ss¥¥qq と指定します。

```
// C# example
QAMessage msg;
QAManager mgr;
...
mgr.PutMessage( @"ianywhere.connector.wsmqfs¥ss¥¥qq",msg );

// C++ example
QAManagerBase *mgr;
QATextMessage *msg;
...
mgr->putMessage( "ianywhere.connector.easerver¥¥ss¥¥¥¥qq", msg );
```

例

たとえば、ianywhere.connector.address プロパティが ianywhere.connector.easerver に設定されており、JMS キュー名が myqueue である場合には、アドレスを設定するコードは次のようになります。

```
// C# example
QAManagerBase mgr;
QAMessage msg;
// Initialize the manager.
...
msg = mgr.CreateTextMessage();
// Set the message content.
...
mgr.PutMessage(@"ianywhere.connector.easerver¥myqueue", msg );

 // C++ example
QAManagerBase *mgr;
QATextMessage *msg;
// Initialize the manager.
...
msg = mgr.createTextMessage();
// Set the message content.
...
mgr->putMessage( "ianywhere.connector.easerver¥¥myqueue", msg );
```

参照

- 「QAnywhere メッセージ・アドレス」 56 ページ

- 「JMS コネクタ・プロパティ」 141 ページ

QAnywhere メッセージの JMS メッセージへのマッピング

QAnywhere メッセージは、JMS メッセージに必然的にマッピングされます。

QAnywhere メッセージの内容

QAnywhere	JMS	説明
QATextMessage	javax.jms.TextMessage	テキスト・メッセージはユニコード・テキストとしてコピーされる
QABinaryMessage	javax.jms.BytesMessage	バイナリ・メッセージはそのままコピーされる

QAnywhere の組み込みヘッダ

次の表は、組み込みヘッダのマッピングを示します。C++ と JMS では、これらのヘッダに対応するメソッド名があります。たとえば、Address は、QAnywhere では getAddress() または setAddress() に対応し、JMS では getJMSDestination() または setJMSDestination() に対応します。.NET では、これらのヘッダとまったく同じ名前のプロパティが存在します。たとえば、Address は、Address プロパティに対応します。

QAnywhere	JMS	説明
Address	JMSDestination と JMS プロパティ ias_ToAddress	送信先アドレスに円記号が含まれている場合は、円記号をもう 1 つ追加して、この円記号をエスケープする必要がある。 Destination にはアドレスの JMS 情報部分のみがマッピングされる。まれに QAnywhere にメッセージがループバックされることがあるが、そのようなメッセージには QAnywhere アドレスがサフィックスとして追加される。これは ias_ToAddress に格納される。
Expiration	JMSExpiration	
InReplyToID	なし	マッピングされない
MessageID	なし	マッピングされない
Priority	JMSPriority	
Redelivered	なし	マッピングされない
ReplyToAddress	JMS プロパティ ias_ReplyToAddress	JMS プロパティにマッピングされる

QAnywhere	JMS	説明
コネクタの xjms.receiveDestination プロパティの値	JMSReplyTo	ReplyTo には、コネクタが JMS メッセージを受信するときに使用する Destination が設定される
Timestamp	なし	マッピングされない
なし	JMSTimestamp	JMS メッセージが QAnywhere メッセージにマッピングされるとき、QAnywhere メッセージの JMSTimestamp プロパティが、JMS メッセージの JMSTimestamp に設定されます。
Timestamp	なし	QAnywhere メッセージが JMS メッセージにマッピングされるとき、JMS メッセージの JMSTimestamp が、JMS メッセージの作成時刻に設定されます。

QAnywhere のプロパティ

QAnywhere のプロパティは、JMS のプロパティに必然的にマッピングされ、型も維持されます。ただし、1 つだけ例外があります。それは、QAnywhere メッセージの JMSType プロパティは、JMS のヘッダ・プロパティ JMSType にマッピングされることです。

JMS から QAnywhere に送信されるメッセージのアドレス指定

JMS クライアントから QAnywhere クライアントにメッセージを送信するには、JMS メッセージ・プロパティ ias_ToAddress に送信先の QAnywhere アドレスを設定して、コネクタ・プロパティ xjms.receiveDestination に対応する JMS Destination にそのメッセージを送信します。

「QAnywhere メッセージ・アドレス」 56 ページを参照してください。

例

QAnywhere アドレス "qaddr" にメッセージを送信する場合の処理は次のようになります (xjms.receiveDestination のコネクタ設定は "qanywhere_receive" であるとします)。

```
import javax.jms.*;
...
try {
  QueueSession session;
  QueueSender sender;
  TextMessage mgr;
  Queue connectorQueue;
  // Initialize the session.
  ...
  connectorQueue = session.createQueue( "qanywhere_receive" );
  sender = session.createSender( connectorQueue );
  msg = session.createTextMessage();
```

```
  msg.setStringProperty( "ias_ToAddress", "qaddr" );
  // Set the message content.
  ...
  sender.send( msg );
} catch( JMSException e ) {
  // Handle the exception
  ...
}
```

JMS メッセージの QAnywhere メッセージへのマッピング

JMS メッセージは、QAnywhere メッセージに必然的にマッピングされます。

JMS メッセージの内容

JMS	QAnywhere	説明
javax.jms.TextMessage	QATextMessage	テキスト・メッセージはユニコード・テキストとしてコピーされる
javax.jms.BytesMessage	QABinaryMessage	バイナリ・メッセージはそのままコピーされる
javax.jms.StreamMessage	なし	サポートされていない
javax.jms.MapMessage	なし	サポートされていない
javax.jms.ObjectMessage	なし	サポートされていない

JMS の組み込みヘッダ

次の表は、組み込みヘッダのマッピングを示します。C++ と JMS では、これらのヘッダに対応するメソッド名があります。たとえば、Address は、QAnywhere では getAddress() または setAddress() に対応し、JMS では getJMSDestination() または setJMSDestination() に対応します。.NET では、これらのヘッダとまったく同じ名前のプロパティが存在します。たとえば、Address は、Address プロパティに対応します。

JMS	QAnywhere	説明
JMS Destination	なし	JMS Destination には、コネクタ・プロパティ xjms.receiveDestination に指定されているキューを設定する必要がある。
JMS Expiration	Expiration	
JMS CorrelationID	InReplyToID	
JMS MessageID	なし	マッピングされない
JMS Priority	Priority	
JMS Redelivered	なし	マッピングされない

JMS	QAnywhere	説明
JMS ReplyTo とコネクタの ianywhere.connector.address プロパティの値	ReplyToAddress	コネクタ・アドレスと、JMS ReplyTo に指定されている Destination 名が、間に文字 '¥' を挟んで連結される
JMS DeliveryMode	なし	マッピングされない
JMS Type	QAnywhere メッセージ・プロパティ JMSType	
JMS Timestamp	なし	マッピングされない

JMS のプロパティ

JMS のプロパティは、QAnywhere のプロパティに必然的にマッピングされます。型も維持されます。ただし、いくつか例外があります。QAnywhere の Address プロパティは、JMS のメッセージ・プロパティ ias_ToAddress の値を基に設定されます。JMS のメッセージ・プロパティ ias_ReplyToAddress が設定されている場合、その値が、QAnywhere の ReplyToAddress の値に、間に文字 '¥' を挟んでサフィックスとして付加されます。

チュートリアル：JMS コネクタの使用

JMS コネクタを使用すると、JMS メッセージ・システムと QAnywhere を接続できます。このチュートリアルでは、JMS システムと QAnywhere の間でメッセージを送信します。

チュートリアルの概要

このチュートリアルでは JMS コネクタを起動し、JMS クライアントから QAnywhere クライアントにメッセージを送信します。

必要なソフトウェア

このチュートリアルでは、JMS プロバイダにアクセスできることと、JMS プロバイダの設定方法についての基本的な知識があることが必要です。また、JDK バージョン 1.3.1 以降と、JMS プロバイダの JMS クライアントが使用する JAR ファイルも必要です。

レッスン 1：JMS コネクタの起動

♦ **JMS プロバイダを準備するには、次の手順に従います。**

1. JMS サーバを起動します。

   詳細については、JMS サーバのマニュアルを参照してください。

2. JMS サーバ内に、qa_testmessage と qa_receive の 2 つのキューを作成します。キューを作成した後、JMS サーバの再起動が必要になる場合があります。

   詳細については、JMS サーバのマニュアルを参照してください。

♦ **QAnywhere クライアントとサーバ・コンポーネントを起動するには、次の手順に従います。**

1. このチュートリアルで作成されるファイルを保存するディレクトリを作成します。たとえば *c:¥JMSTestMessage* のようになります。このディレクトリに移動します。

2. QAnywhere コネクタを作成します。

   ♦ Sybase Central で、[ファイル] – [新規] – [コネクタ] を選択します。[コネクタ] ウィザードのプロンプトに従って操作します。

   「JMS コネクタの設定」 137 ページを参照してください。

3. Mobile Link サーバをメッセージングに対応した状態で起動します。

   Windows で、[スタート] – [プログラム] – [SQL Anywhere 10] – [Mobile Link] – [メッセージ付き Mobile Link のサンプル] を選択します。

   また、コマンド・プロンプトで *samples-dir¥QAnywhere¥server* に移動し、次のコマンドを入力することもできます。

   ```
   mlsrv10 -m -c "dsn=QAnywhere 10.0 Demo" -sl java(-cp "jarfiles") -vcrs -zu+
   ```

4. QAnywhere Agent を起動します。

   Windows で、[スタート] – [プログラム] – [SQL Anywhere 10] – [QAnywhere] – [Client1 用エージェント・サンプル] を選択します。

5. TestMessage サンプルを起動します。

   Windows で、[スタート] – [プログラム] – [SQL Anywhere 10] – [QAnywhere] – [Client1 用 TestMessage サンプル] を選択します。

♦ Java バージョンの TestMessage クライアントを起動するには、次の手順に従います。

1. コマンド・プロンプトで *Samples¥QAnywhere¥connectors¥JMS¥TestMessage* に移動して、次のように入力します。

   ```
   java -cp .;JMS-client-jar-files ianywhere.message.samples.TestMessage
   ```

   ここで、*JMS-client-jar-files* は、JMS サーバにアクセスするために必要な jar ファイルのリストです。各ファイル名の間はセミコロンで区切ります。詳細については、JMS サーバのマニュアルを参照してください。

   Sybase EAServer の場合、上のコマンドは次のようになります。

   ```
   java -cp .;path¥easclient.jar;path¥easj2ee.jar ianywhere.message.samples.TestMessage
   ```

   ここで、*path* は、jar ファイルが存在するロケーションです。

   注意
   UNIX では、セミコロンではなくコロンを使用します。

2. JMS 用の TestMessage のウィンドウを、画面右側にすでに置かれている Client1 用 TestMessage のウィンドウの下に移動します。

レッスン 2：JMS クライアントから QAnywhere クライアントへのメッセージの送信

♦ JMS クライアントから QAnywhere クライアントにメッセージを送信するには、次の手順に従います。

1. JMS 用の TestMessage のウィンドウで、[Message] – [New] を選択します。

   [New Message] ウィンドウが表示されます。

2. [To] フィールドに、Client1 のクライアント・メッセージ・ストア ID を入力します。

3. [Subject] フィールドと [Massage] フィールドにサンプル・テキストを入力して、[Send] をクリックします。

4. しばらくすると、Client2 用 TestMessage がメッセージを受信したことを知らせるメッセージが、メッセージ・ボックスに表示されます。

チュートリアルのクリーンアップ

TestMessage クライアント、QAnywhere Agent、Mobile Link サーバをそれぞれ停止します。

第 7 章

QAnywhere Agent

目次

qaagent 構文

QAnywhere Agent を使用することで、同一クライアント・デバイス上のすべての QAnywhere アプリケーションのメッセージ送受信を処理できます。

構文

qaagent [*option* ...]

オプション	説明
@*data*	指定された環境変数または設定ファイルからオプションを読み込む。「@data オプション」 158 ページを参照してください。
-c *connection-string*	クライアント・メッセージ・ストアに接続するための接続文字列を指定する。「-c オプション」 159 ページを参照してください。
-id *id*	QAnywhere Agent が接続するクライアント・メッセージ・ストアの ID を指定する。「-id オプション」 163 ページを参照してください。
-idl *download-size*	メッセージ転送中に使用するダウンロードの最大サイズを指定する。「-idl オプション」 164 ページを参照してください。
-iu*upload-size*	メッセージ転送中に使用するアップロードの最大サイズを指定する。「-iu オプション」 165 ページを参照してください。
-lp *number*	Listener が Mobile Link サーバからの通知を受信するポートを指定する。デフォルトは 5001 秒です。「-lp オプション」 166 ページを参照してください。
qaagent -mp *password*	Mobile Link ユーザの新しいパスワードを指定します。「-mn オプション」 167 ページを参照してください。
-mp *password*	Mobile Link ユーザの新しいパスワードを指定する。「-mp オプション」 168 ページを参照してください。
-mu *username*	Mobile Link ユーザを指定する。「-mu オプション」 169 ページを参照してください。
-o *logfile*	出力メッセージのログを取るファイルを指定する。「-o オプション」 170 ページを参照してください。
-on *size*	QAnywhere Agent のメッセージ・ログ・ファイルの最大サイズを指定する。ログ・ファイルがこのサイズに達すると、現在のファイルが拡張子 .old を持つ名前に変更され、新しいファイルが作成されます。「-on オプション」 171 ページを参照してください。
-os *size*	QAnywhere Agent のメッセージ・ログ・ファイルの最大サイズを指定する。ログ・ファイルがこのサイズに達すると、新しい名前を持つ新しいログ・ファイルが作成されて使用されます。「-os オプション」 172 ページを参照してください。

オプション	説明
-ot *logfile*	出力メッセージのログを取るファイルを指定する。「-ot オプション」 173 ページを参照してください。
-pc{**+**|**-**}	メッセージ転送用に永続的な接続を有効にする。「-pc オプション」 174 ページを参照してください。
-policy *policy-type*	転送ポリシーを指定する。指定したポリシーは、QAnywhere Agent で使用されます。「-policy オプション」 175 ページを参照してください。
-push *mode*	Push 通知を有効または無効にする。デフォルトは有効です。「-push オプション」 177 ページを参照してください。
-q	QAnywhere Agent をクワイエット・モードで起動する。ウィンドウをシステム・トレイ内に最小化します。「-q オプション」 179 ページを参照してください。
-qi	QAnywhere Agent をクワイエット・モードで起動する。ウィンドウを完全に非表示にします。「-qi オプション」 180 ページを参照してください。
-si	データベースを、クライアント・メッセージ・ストアとして使用できるように初期化する。「-si オプション」 181 ページを参照してください。
-su	クライアント・メッセージ・ストアを現在のバージョンにアップグレードする。dbunload/reload は実行しません。「-su オプション」 183 ページを参照してください。
-sur	クライアント・メッセージ・ストアを現在のバージョンにアップグレードする。クライアント・メッセージ・ストアの dbunload/reload を実行します。「-sur オプション」 184 ページを参照してください。
-v[*levels*]	ログの冗長レベルを指定する。「-v オプション」 185 ページを参照してください。
-x { **http**|**tcpip**|**tls**|**https** } [(*keyword*=*value*;...)]	Mobile Link サーバとの通信に使用するプロトコルのオプションを指定する。「-x オプション」 186 ページを参照してください。
qaagent -xd	QAnywhere Agent で Mobile Link サーバの動的アドレス指定を使用することを指定する。「-xd オプション」 187 ページを参照してください。

参照

- 「QAnywhere Agent の実行」 38 ページ

@data オプション

指定された環境変数または設定ファイルからオプションを読み込みます。

構文

qaagent @{ *filename* | *environment-variable* } ...

備考

このオプションを指定すると、環境変数または設定ファイル内にコマンド・ライン・オプションを記述できます。指定された名前の環境変数と指定された名前の設定ファイルの両方が存在する場合、環境変数が使用されます。

「設定ファイルの使用」『SQL Anywhere サーバ - データベース管理』を参照してください。

設定ファイル内のパスワードなどの情報を保護する場合は、ファイル非表示ユーティリティを使用して、設定ファイルの内容を難読化できます。

Windows CE ではショートカットのコマンド･ラインが 256 文字に制限されているため、このオプションが役に立ちます。

「ファイル非表示ユーティリティ (dbfhide)」『SQL Anywhere サーバ - データベース管理』を参照してください。

Sybase Central での同等機能

Sybase Central 用の QAnywhere プラグインには、[エージェント設定ファイルの作成] というタスクがあります。このタスクを選択すると、ファイル名を入力するように要求されます。次に [プロパティ] ダイアログが表示され、コマンド情報を入力できます。生成されるファイルの拡張子は *.qaa* です。ファイル拡張子 *.qaa* は、Sybase Central の規則に従っています。つまりこのファイルは、@data オプションで作成するファイルと同じです。Sybase Central で作成されたコマンド・ファイルは、@data の設定ファイルとして使用できます。

-c オプション

クライアント・メッセージ・ストアに接続するための接続文字列を指定します。

構文

qaagent -c *connection-string ...*

デフォルト

接続パラメータ	デフォルト値
uid	**ml_qa_user**
pwd	**qanywhere**

備考

接続文字列では、接続パラメータを *keyword=value* の形式で指定します。パラメータとパラメータの間はセミコロンで区切り、スペースは入れないでください。

通常、DSN はクライアント・デバイスで使用されません。ODBC は qaagent で使用されません。

接続パラメータの完全なリストについては、「接続パラメータ」『SQL Anywhere サーバ - データベース管理』を参照してください。

以下に、よく使用する接続パラメータの一部を示します。

- **dbf=*filename*** 指定されたファイル名を持つメッセージ・ストアに接続します。

  「DatabaseFile 接続パラメータ [DBF]」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **dbn=*database-name*** QAnywhere Agent の起動時にクライアント・メッセージ・ストアがすでに開始されている場合は、データベース・ファイル名ではなくデータベース名を指定することでそのクライアント・メッセージ・ストアに接続できます。

  「DatabaseName 接続パラメータ [DBN]」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **eng=*server-name*** すでに起動されているデータベース・サーバを使用する場合は、そのサーバ名をこのオプションで指定します。デフォルト値はデータベース名です。

  「EngineName 接続パラメータ [ENG]」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **uid=*user*** クライアント・メッセージ・ストアに接続するためのデータベース・ユーザ ID を指定します。デフォルト値を変更する場合、このパラメータは必須です。

  「Userid 接続パラメータ [UID]」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **pwd=*password*** データベース・ユーザ ID に対応するパスワードを指定します。デフォルト値を変更する場合、このオプションは必須です。

「Password 接続パラメータ [PWD]」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **dbkey=*key*** クライアント・メッセージ・ストアを強力な暗号化を使用して暗号化している場合は、データベースへのアクセスに必要な暗号化キーを指定します。

  「DatabaseKey 接続パラメータ [DBKEY]」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **start=*startline*** データベース・サーバの開始行を指定します。開始行を指定しなかった場合、デフォルトは Windows CE では start=dbsrv10 -m -gn 5、その他の Windows プラットフォームでは start=dbsrv10 -m になります。-m オプションを指定すると、トランザクション・ログの内容がチェックポイントで削除されます。このオプションを指定することをおすすめします。次の項を参照してください。

  - 「StartLine 接続パラメータ [START]」『SQL Anywhere サーバ - データベース管理』
  - 「-m サーバ・オプション」『SQL Anywhere サーバ - データベース管理』
  - 「-gn サーバ・オプション」『SQL Anywhere サーバ - データベース管理』

参照

- 「接続パラメータ」『SQL Anywhere サーバ - データベース管理』
- 「データベースへの接続」『SQL Anywhere サーバ - データベース管理』

例

```
qaagent -id Device1 -c "DBF=qanyclient.db" -x tcpip(host=hostname) -policy automatic
```

-fd オプション

このオプションが -fr オプションと一緒に指定されている場合は、Mobile Link サーバへの接続を試行する間隔を指定します。

構文

qaagent -fd *seconds ...*

デフォルト

- -fr だけを指定し -fd を指定しない場合は、遅延は 0 です (間隔をあけずに、すぐに再試行が行われます)。
- -fr を指定しないと、デフォルトで再試行が行われません。

備考

このオプションは、qaagent -fr オプションと一緒に使用する必要があります。-fr オプションはプライマリ・サーバへの接続を再試行する回数を指定し、-fd オプションは再試行の間隔を指定します。

通常このオプションは、-x オプションを使用してフェールオーバ Mobile Link サーバを指定する場合に使用します。フェールオーバ Mobile Link サーバが設定されている場合、デフォルトでは、プライマリ・サーバへの接続で障害が発生するとすぐに、QAnywhere Agent は代替サーバへの接続を試行します。-fr オプションを指定すると、QAnywhere Agent は代替サーバに接続する前に、プライマリ・サーバへの接続を再試行します。-fd オプションを併せて使用することで、プライマリ・サーバへの接続を試行する間隔も指定できます。

-fd オプションは 10 秒以下に設定することをおすすめします。

このオプションは、qaagent -xd オプションと一緒に使用することはできません。

参照

- 「-fr オプション」 162 ページ
- 「-x オプション」 186 ページ
- 「フェールオーバ・メカニズムの設定」 49 ページ

-fr オプション

このオプションは、QAnywhere Agent がプライマリ Mobile Link サーバへの接続を再試行する回数を指定します。

構文

qaagent -fr *number-of-retries ...*

デフォルト

0 (QAnywhere Agent はプライマリ Mobile Link サーバへの接続を再試行しません)

備考

デフォルトでは、QAnywhere Agent が Mobile Link サーバに接続できない場合、エラーは発生せずメッセージも送信されません。このオプションは、QAnywhere Agent が Mobile Link サーバへの接続を再試行することを指定します。さらに、QAnywhere Agent が代替サーバへの接続を試行するまで、またはエラーを発行するまで (代替サーバが指定されていない場合) に、Mobile Link サーバへの接続を再試行する回数も指定します。

通常このオプションは、-x オプションを使用してフェールオーバ Mobile Link サーバを指定する場合に使用します。フェールオーバ Mobile Link サーバが設定されている場合、デフォルトでは、プライマリ・サーバへの接続で障害が発生するとすぐに、QAnywhere Agent は代替サーバへの接続を試行します。このオプションを指定すると、QAnywhere Agent は代替サーバに接続する前に、プライマリ・サーバへの接続を再試行します。

-fd オプションを併せて使用することで、プライマリ・サーバへの接続再試行の間隔も指定できます。

このオプションは、qaagent -xd オプションと一緒に使用することはできません。

参照

- 「-fd オプション」 161 ページ
- 「-x オプション」 186 ページ
- 「フェールオーバ・メカニズムの設定」 49 ページ

-id オプション

QAnywhere Agent が接続するクライアント・メッセージ・ストアの ID を指定します。

構文

qaagent -id *id* ...

デフォルト

ID のデフォルト値は、QAnywhere Agent が動作しているデバイスの名前です。デバイス名はユニークではない場合があります。そのような場合には、-id オプションを指定する必要があります。

備考

各クライアント・メッセージ・ストアは、メッセージ・ストア ID と呼ばれるユニークな文字列で識別されます。メッセージ・ストアに最初に接続するときに ID を設定しなかった場合、ID はデフォルトでデバイス名になります。次回以降の接続時には、設定済みのメッセージ・ストア ID を -id オプションとともに指定する必要があります。

メッセージ・ストア ID は Mobile Link リモート ID と対応しています。すべての Mobile Link アプリケーションではリモート・データベースごとにユニークな ID が必要であるため、メッセージ・ストア ID が要求されます。

「Mobile Link ユーザの作成と登録」 『Mobile Link - クライアント管理』を参照してください。

デバイス上で 2 つ目の qaagent インスタンスを起動する場合は、-id オプションを使用してユニークなメッセージ・ストア ID を指定する必要があります。

次の文字は ID の構成文字として使用できません。

- " (二重引用符)
- 制御文字
- ¥¥ (二重円記号)

このほかにも次の制約が適用されます。

- ID は 120 文字以内である必要がある。
- 単一の円記号 (¥) を使用できるのは、エスケープ文字として使用する場合に限られる。
- クライアント・メッセージ・ストア・データベースで quoted_identifier オプションが Off に設定されている場合 (デフォルトの設定ではありません) は、英数字、アンダースコア (_)、アットマーク (@)、シャープ (#)、ドル記号 ($) のみを ID に含めることができる。

参照

- 「Mobile Link ユーザの概要」 『Mobile Link - クライアント管理』
- 「クライアント・メッセージ・ストアの設定」 36 ページ

-idl オプション

インクリメンタル・ダウンロードのサイズを指定します。

構文

qaagent -idl *download-size* [**K** | **M**] ...

デフォルト

-1 (最大ダウンロード・サイズなし)

備考

このオプションは、メッセージ転送のダウンロード部分のサイズをバイト単位で指定します。キロバイトまたはメガバイトの単位を指定するには、それぞれサフィックス K、M を使用します。

QAnywhere Agent の起動時に、このオプションで指定されている値が ias_MaxDownloadSize メッセージ・ストア・プロパティに割り当てられます。このメッセージ・ストア・プロパティによって、ダウンロード・サイズの上限が定義されます。転送がトリガされると、サーバは、すべてのメッセージの合計サイズがこのオプションで設定されている上限に到達するまで、クライアントへの配信対象メッセージをタグ付けします。サーバは、キューに登録されているすべてのメッセージが配信されるまで、メッセージをバッチ送信します。各バッチのメッセージが転送されるたびに転送ルールが再実行されるため、転送中に優先度の高いメッセージがキューに登録された場合は、そのメッセージがキューの先頭に移動されます。

メッセージは分割されないので、配信対象メッセージがキューに登録されている場合、ダウンロードに常に少なくとも 1 つのメッセージが含まれます。したがって、インクリメンタル・ダウンロードのサイズは概算値であり、このサイズよりもはるかに大きいサイズのダウンロード対象のメッセージがある場合は概算値の信頼性は低くなります。

参照

- 「事前に定義されたクライアント・メッセージ・ストア・プロパティ」 230 ページの ias_MaxDownloadSize

-iu オプション

インクリメンタル・アップロードのサイズを指定します。

構文

qaagent -iu *upload-size* [**K** | **M**] ...

デフォルト

256K

備考

このオプションは、メッセージ転送のアップロード部分のサイズをバイト単位で指定します。キロバイトまたはメガバイトの単位を指定するには、それぞれサフィックス K、M を使用します。

QAnywhere Agent の起動時に、このオプションで指定されている値が ias_MaxUploadSize メッセージ・ストア・プロパティに割り当てられます。このメッセージ・ストア・プロパティによって、アップロード・サイズの上限が定義されます。転送がトリガされると、QAnywhere Agent は、すべてのメッセージの合計サイズがこのオプションで設定されている上限に到達するまで、サーバへの配信対象メッセージをタグ付けします。この上限に到達すると、タグ付けされたメッセージがサーバに送信されます。メッセージがサーバで受信され、受信確認がサーバからクライアントに正常に送信された場合は、転送のダウンロード・フェーズが失敗しても、メッセージは問題なく配信されたとみなされます。QAnywhere Agent は、キューに登録されているすべてのメッセージが配信されるまで、メッセージをサーバにバッチ送信します。各バッチのメッセージが転送されるたびに転送ルールが再実行されるため、転送中に優先度の高いメッセージがキューに登録された場合は、そのメッセージがキューの先頭に移動されます。

メッセージは分割されないので、配信対象メッセージがキューに登録されている場合、アップロードに常に少なくとも 1 つのメッセージが含まれます。インクリメンタル・アップロードのサイズは概算値であり、このサイズよりもはるかに大きいサイズのアップロード対象のメッセージがある場合は概算値の信頼性は低くなります。

参照

- 「事前に定義されたクライアント・メッセージ・ストア・プロパティ」 230 ページの ias_MaxUploadSize

-lp オプション

Listener のポートを指定します。

構文

qaagent -lp *number* ...

デフォルト

5001

備考

Listener が Mobile Link サーバからの UDP 通知を受信するポートの番号を指定します。通知は、サーバ上に待機中のメッセージが存在することを QAnywhere Agent に知らせるために使用されます。

UDP リスナ・ポートは、-push disconnected オプションを指定して QAnywhere Agent が起動された場合にのみ確立されます。

参照

- 「Push 通知によるメッセージングのシナリオ」 9 ページ
- 「-push オプション」 177 ページ

-mn オプション

Mobile Link ユーザの新しいパスワードを指定します。

構文

qaagent -mp *password* ...

デフォルト

なし

備考

パスワードを変更する場合に使用します。

参照

- 「Mobile Link ユーザ」 『Mobile Link - クライアント管理』
- 「-mp オプション」 168 ページ
- 「-mu オプション」 169 ページ

-mp オプション

Mobile Link ユーザの Mobile Link パスワードを指定します。

構文

qaagent -mp *password* ...

デフォルト

なし

備考

Mobile Link サーバでユーザ認証が必要な場合は、-mp オプションを使用して Mobile Link パスワードを指定します。

参照

- 「Mobile Link ユーザ」『Mobile Link - クライアント管理』
- 「-mu オプション」 169 ページ

-mu オプション

Mobile Link ユーザを指定します。

構文

qaagent -mu *username* ...

デフォルト

クライアント・メッセージ・ストア ID

備考

Mobile Link ユーザは、Mobile Link サーバで認証を行う場合に使用します。

存在しないユーザ名を指定した場合は、自動的に作成されます。

すべての Mobile Link ユーザ名は、サーバ・メッセージ・ストアに登録する必要があります。「QAnywhere クライアント・ユーザ名の登録」 34 ページを参照してください。

参照

- 「Mobile Link ユーザ」 『Mobile Link - クライアント管理』
- 「-id オプション」 163 ページ
- 「-mp オプション」 168 ページ
- 「リモート ID」 『Mobile Link - クライアント管理』

-o オプション

出力をログ・ファイルに送信します。

構文

qaagent -o *logfile ...*

デフォルト

なし

備考

指定された名前のファイルに出力のログを取ります。ファイルが既に存在している場合は、新しいログ情報がそのファイルに追加されます。SQL Anywhere 同期クライアント (dbmlsync) は、指定された名前に *_sync* というサフィックスを付けたものを名前とするファイルに、出力のログを取ります。Listener ユーティリティ (dblsn) は、指定された名前に *_lsn* というサフィックスを付けたものを名前とするファイルに、出力のログを取ります。

たとえば、ログ・ファイル名 *c:¥tmp¥mylog.out* を指定した場合、qaagent は *c:¥tmp¥mylog.out* に、dbmlsync は *c:¥tmp¥mylog_sync.out* に、dblsn は *c:¥tmp¥mylog_lsn.out* に、それぞれログを取ります。

参照

- 「-ot オプション」 173 ページ
- 「-on オプション」 171 ページ
- 「-os オプション」 172 ページ
- 「-v オプション」 185 ページ

-on オプション

QAnywhere Agent のメッセージ・ログ・ファイルの最大サイズを指定する。ログ・ファイルがこのサイズに達すると、現在のファイルが拡張子 *.old* を持つ名前に変更され、新しいファイルが作成されます。

構文

qaagent -on *size* [**k** | **m**]…

デフォルト

なし

備考

size には、出力ログの最大ファイル・サイズを、バイト単位で指定します。キロバイトまたはメガバイトの単位を指定するには、それぞれサフィックス k、m を使用します。最小のサイズ制限は 10KB です。

ログ・ファイルが指定されたサイズに達すると、QAnywhere Agent は出力ファイルを拡張子 *.old* を持つ名前に変更し、元の名前で新しいファイルを開始します。

注意
.old ファイルがすでに存在する場合は、そのファイルが上書きされます。古いログ・ファイルを削除しないようにするには、代わりに -os オプションを使用します。
このオプションは、-os オプションと同時に使用できません。

参照

- 「-o オプション」 170 ページ
- 「-ot オプション」 173 ページ
- 「-os オプション」 172 ページ
- 「-v オプション」 185 ページ

-os オプション

QAnywhere Agent のメッセージ・ログ・ファイルの最大サイズを指定します。ログ・ファイルがこのサイズに達すると、新しい名前を持つ新しいログ・ファイルが作成されて使用されます。

構文

qaagent -os *size* [**k** | **m**] ⋯

デフォルト

なし

備考

size には、出力メッセージのログを取るファイルの最大サイズを指定します。デフォルトの単位はバイトです。キロバイトまたはメガバイトの単位を指定するには、それぞれサフィックス k、m を使用します。最小のサイズ制限は 10K です。

QAnywhere Agent は、現在のファイル・サイズを確認してから、ファイルに出力メッセージのログを取ります。ログ・メッセージによって、ファイル・サイズが指定したサイズより大きくなると、QAnywhere Agent はメッセージ・ログ・ファイルの名前を *yymmddxx.mls.* に変更します。*xx* は 00 ー 99 の連続する文字で、*yymmdd* は現在の年月日を表します。

このオプションを使用すると、古いメッセージ・ログ・ファイルを削除して、ディスク領域を解放できます。常に最新の出力が -o または -ot で指定したファイルに追加されます。

> **注意**
> このオプションは、-os オプションとは一緒に使用できません。

参照

- 「-o オプション」 170 ページ
- 「-ot オプション」 173 ページ
- 「-on オプション」 171 ページ
- 「-v オプション」 185 ページ

-ot オプション

ログ・ファイルをトランケートし、そのファイルに出力メッセージを追加します。

構文

qaagent -ot *logfile ...*

デフォルト

なし

備考

指定された名前のファイルに出力のログを取ります。ファイルが存在する場合は、まずファイルのサイズが 0 にトランケートされます。SQL Anywhere 同期クライアント (dbmlsync) は、指定された名前に *_sync* というサフィックスを付けたものを名前とするファイルに、出力のログを取ります。Listener ユーティリティ (dblsn) は、指定された名前に *_lsn* というサフィックスを付けたものを名前とするファイルに、出力のログを取ります。

たとえば、ログ・ファイル名 *c:¥tmp¥mylog.out* を指定した場合、qaagent は *c:¥tmp¥mylog.out* に、dbmlsync は *c:¥tmp¥mylog_sync.out* に、dblsn は *c:¥tmp¥mylog_lsn.out* に、それぞれログを取ります。

参照

- 「-o オプション」 170 ページ
- 「-on オプション」 171 ページ
- 「-os オプション」 172 ページ
- 「-v オプション」 185 ページ

-pc オプション

同期と同期の間で、Mobile Link サーバへの永続的な接続を維持します。

構文

qaagent -pc { **+** | **-** } ...

デフォルト

-pc-

備考

ネットワーク・カバレッジが良好で QAnywhere 上のメッセージ・トラフィックが多い場合は、永続的な接続を有効にする (-pc+) と便利です。この場合は、メッセージ転送が行われるたびに TCP/IP 接続の設定と切断を実行するというネットワーク負荷を軽減できます。

クライアント・デバイスにパブリック IP アドレスが設定されており、UDP や SMS でアクセス可能であれば、次のような場合に永続的な接続を無効にする (-pc-) と便利です。

- クライアント・デバイスでダイヤルアップ・ネットワーキングを使用しており、接続時間にかかるコストが懸案である場合。
- QAnywhere 上のメッセージ・トラフィックが少ない場合。永続的な TCP/IP 接続ではネットワーク・サーバのリソースが消費されるため、スケーラビリティに影響を与える可能性があります。
- クライアント・デバイスのネットワーク・カバレッジが安定していない場合。接続可能なときに自動ポリシーを使用してメッセージを転送できます。このような環境で永続的な接続を維持しようとしても、CPU リソースが浪費されるだけで、何の役にも立ちません。

参照

- 「-push オプション」 177 ページ
- 「-pc オプション」 『Mobile Link - クライアント管理』

-policy オプション

メッセージ転送のタイミングを決定するポリシーを指定します。

構文

qaagent -policy *policy-type ...*

policy-type: **ondemand** | **scheduled**[*interval-in-seconds*] | **automatic** | *rules-file*

デフォルト

- デフォルトのポリシー・タイプは **automatic** です。
- デフォルトの scheduled ポリシーの間隔は 900 秒 (15 分) です。

備考

QAnywhere ではポリシーを使用して、メッセージ転送のタイミングを決定します。*policy-type* には、次のいずれかの値を指定できます。

- **ondemand**　QAnywhere クライアント・アプリケーションによって適切なメソッドが呼び出されたときだけ、メッセージが転送されます。

  QAManager PutMessage() メソッドは、メッセージをローカルのキューに登録します。キューに登録されたメッセージは、QAManager TriggerSendReceive() メソッドが呼び出されるまでサーバに転送されません。同様に、サーバ上で送信待機中のメッセージは、クライアントによって TriggerSendReceive() が呼び出されるまで、クライアントに送信されません。

  ondemand ポリシーが使用されている場合は、アプリケーションはサーバから Push 通知を受信するとメッセージを転送します。Push 通知時にはシステム・メッセージが QAnywhere クライアントに配信されます。クライアント・アプリケーションでは、TriggerSendReceive() を呼び出すことによって、このシステム・メッセージに応答できます。

  例については、「システム・キュー」 58 ページを参照してください。

- **scheduled**　指定された間隔でメッセージを転送します。デフォルト値は 900 秒 (15 分) です。

  クライアント／サーバ間のメッセージ転送が、指定された間隔で実行されます。

  QAManager PutMessage() メソッドは、メッセージをローカルのキューに登録します。これらのメッセージは、指定された時間間隔が経過するまで転送されません。サーバ上のキューに登録された配信対象のメッセージも、指定された時間間隔が経過した時点でクライアントに転送されます。

  Push 通知が有効になっている場合、サーバ上のキューに登録された、クライアントを配信先とするメッセージは、次の時間間隔が経過したときに転送されます。

  この間隔を無効にするには、TriggerSendReceive() を使用します。このメソッドを使用すると、時間間隔が経過する前にメッセージを転送できます。

interval 引数 (オプション) には、送受信操作を実行する間隔を秒単位で指定します。たとえば、次のコマンドを実行すると、20 分ごとにメッセージを同期するように QAnywhere Agent がスケジュール設定されます。

```
qaagent.exe -policy scheduled[1200]
```

- **automatic**　次のいずれかのイベントが発生した場合に、メッセージを転送します。

  QAnywhere Agent は、メッセージ・キューをできるかぎり最新の状態に維持しようとします。次のいずれかのイベントが発生すると、クライアントのキューに登録されているメッセージがサーバに配信され、サーバ上のキューに登録されているメッセージがクライアントに配信されます。

  - PutMessage() の呼び出し
  - TriggerSendReceive() の呼び出し
  - Push 通知

    通知の詳細については、「Push 通知によるメッセージングのシナリオ」 9 ページを参照してください。
  - クライアント上のメッセージ・ステータスの変化。たとえば、アプリケーションがローカルのキューからメッセージを取り出すとステータスの変化が発生します。この場合、メッセージ・ステータスは保留から受信済みに変化します。

- ***rules-file***　クライアントの転送ルール・ファイルを指定します。転送ルール・ファイルには、メッセージを転送するタイミングを決定するための複雑なルール・セットを記述できます。

  「クライアント側の転送ルール」 251 ページを参照してください。

参照

- 「クライアントにメッセージを転送するタイミングの決定」 40 ページ
- 「Push 通知によるメッセージングのシナリオ」 9 ページ

-push オプション

Push 通知が有効であるかどうかを指定します。

構文

qaagent -push *mode* ...

mode : **none** | **connected** | **disconnected**

デフォルト

connected

オプション

モード	説明
none	Push 通知は、このエージェントに対して無効です。Listener (dblsn) は起動されません。
connected	Push 通知は、このエージェントに対して有効です。永続的な TCP/IP 接続が使用されます。Listener (dblsn) は qaagent によって起動され、Mobile Link サーバへの永続的な接続を維持しようとします。クライアント・デバイスにパブリック IP アドレスが設定されていない場合や、UDP メッセージを許可しないファイアウォールの背後に Mobile Link サーバが置かれている場合は、このモードを使用すると便利です。これはデフォルトです。
disconnected	Push 通知は、このエージェントに対して有効です。永続的でない UDP 接続が使用されます。Listener (dblsn) は qaagent によって起動されますが、Mobile Link サーバへの永続的な接続を維持しません。代わりに、UDP リスナが Mobile Link から Push 通知を受信します。クライアント・デバイスにパブリック IP アドレスが設定されており、UDP や SMS でアクセス可能であれば、次のような場合にこのモードを使用すると便利です。 ◆ クライアント・デバイスでダイヤルアップ・ネットワーキングを使用しており、接続時間にかかるコストが懸案である場合。 ◆ QAnywhere 上のメッセージ・トラフィックが少ない場合。永続的な TCP/IP 接続ではネットワーク・サーバのリソースが消費されるため、スケーラビリティに影響を与える可能性があります。 ◆ クライアント・デバイスのネットワーク・カバレッジが安定していない場合。接続可能なときに自動ポリシーを使用してメッセージを転送できます。このような環境で永続的な接続を維持しようとしても、CPU リソースが浪費されるだけで、何の役にも立ちません。 「-lp オプション」 166 ページを参照してください。

備考

通知を使用しない場合は、このオプションを none に設定します。通知を無効にした場合、*dblsn.exe* 実行プログラムをクライアントに配備する必要はありません。

通知を使用しない QAnywhere の詳細については、「簡単なメッセージング・シナリオ」 7 ページを参照してください。

UDP を使用している場合、disconnected モードの ActiveSync で Push 通知を使用することはできません。これは、ActiveSync の UDP 実装に制限があるためです。

参照

- 「Push 通知の使用方法」 45 ページ
- 「-pc オプション」 174 ページ
- 「QAnywhere Agent の実行」 38 ページ
- 「Push 通知の通知」 59 ページ

-q オプション

QAnywhere Agent をクワイエット・モードで起動し、ウィンドウをシステム・トレイ内に最小化します。

構文

qaagent -q ...

デフォルト

なし

備考

QAnywhere Agent をクワイエット・モード (-q) で起動すると、メイン・ウィンドウはシステム・トレイ内に最小化されます。また、メッセージ・ストア用のデータベース・サーバが、-qi オプションを指定した状態で起動されます。

参照

- 「-qi オプション」 180 ページ

-qi オプション

QAnywhere Agent をクワイエット・モードで起動し、ウィンドウを完全に非表示にします。

構文

qaagent -qi ...

デフォルト

なし

備考

QAnywhere Agent をクワイエット・モードで起動すると、Windows デスクトップ上ではメイン・ウィンドウがシステム・トレイ内に最小化され、Windows CE 上ではメイン・ウィンドウが非表示になります。また、メッセージ・ストア用のデータベース・サーバが、-qi オプションを指定した状態で起動されます。

Windows CE では、同時実行可能なプロセス数が 32 個に制限されており、この制限に達するとアプリケーションがクローズされます。クワイエット・モードを使用すると、このような状態になるのを回避できるので、一部の Windows CE アプリケーションでは有用です。クワイエット・モードでは、QAnywhere Agent をサービスのように実行できます。

-qi クワイエット・モードで実行中の QAnywhere Agent を停止するには、**qastop** を使用するしかありません。

参照

- 「-q オプション」 179 ページ

-si オプション

データベースを、クライアント・メッセージ・ストアとして使用できるように初期化します。

構文

qaagent -c "*connection-string*" -si ...

デフォルト

なし。このオプションは、クライアント・メッセージ・ストアの初期化のために、一度だけ使用します。

備考

このオプションを使用する前に、SQL Anywhere データベースを作成する必要があります。-si オプションを指定すると、QAnywhere Agent は、データベースを初期化して、QAnywhere システム・テーブルなどのデータベース・オブジェクトを追加します。初期化後、QAnywhere Agent はすぐに終了します。

-si オプションを使用するときは、初期化の対象となるデータベースを、-c オプションで接続文字列として指定する必要があります。-c オプションに指定する接続文字列には、DBA 権限を持つユーザ ID も指定します。ユーザ ID とパスワードを指定しなければ、デフォルト・ユーザ DBA とパスワード SQL が使用されます。

-si オプションを使用すると、クライアント・メッセージ・ストア用に、データベース・ユーザ **ml_qa_user** とパスワード **qanywhere** が作成されます。ユーザ ml_qa_user には、QAnywhere アプリケーションの実行に必要なパーミッションのみが与えられます。このデータベース・ユーザ名とパスワードを変更しないかぎり、qaagent の起動時にパスワードやユーザ ID を -c オプションに指定する必要はありません。変更した場合は、qaagent コマンド・ラインの -c オプションにユーザ ID またはパスワード (あるいは両方) を指定する必要があります。

> **注意**
> デフォルトのパスワードは必ず変更してください。それには、GRANT 文を使用します。「パスワードの変更」 『SQL Anywhere サーバ - データベース管理』を参照してください。

-si オプションで、クライアント・メッセージ・ストアの ID を指定することはできません。ID を割り当てるには、-si を実行するとき、または次回 qaagent を実行するときに -id オプションを指定します。-id オプションを指定しなければ、デフォルトでデバイス名が ID として割り当てられます。

メッセージ・ストアは作成されているが ID が設定されていないという状態の場合、そのメッセージ・ストアに対してローカルな QAnywhere アプリケーション同士ではメッセージを送受信できますが、リモートの QAnywhere アプリケーションとのメッセージ交換はできません。ID が割り当てられた時点で、リモート・メッセージングも可能になります。

参照

- 「クライアント・メッセージ・ストアの設定」 36 ページ
- 「セキュリティ保護されたクライアント・メッセージ・ストアの作成」 190 ページ

例

次のコマンドを実行すると、*qaclient.db* データベースへの接続が実行され、このデータベースが QAnywhere クライアント・メッセージ・ストアとして初期化されます。QAnywhere Agent は、初期化が完了するとすぐに終了します。

```
qaagent -si -c "DBF=qaclient.db"
```

-su オプション

クライアント・メッセージ・ストアを現在のバージョンにアップグレードします。10.0.0 よりも前のメッセージ・ストアからアップグレードする場合は、最初にメッセージ・ストアを手動でアンロードしてから、再ロードする必要があります。

構文

qaagent -su -c "*connection-string*" ...

備考

アンロードと再ロードを実行した後で、カスタム・アクションを実行してから qaagent をアップグレードする場合は、このオプションが便利です。10.0.0 よりも前のメッセージ・ストアからアップグレードする場合に、アンロードと再ロードを自動的に実行するには、-sur オプションを使用します。

この操作はアップグレードが完了すると終了します。

この操作は取り消せません。

参照

- 「-sur オプション」 184 ページ

例

バージョン 9 のデータベースからアップグレードするには、最初にデータベースのアンロードと再ロードを行います。

```
dbunload  -q  -c "UID=dba;PWD=sql;DBF=qanywhere.db" -ar
```

次に、-su オプションを指定して qaagent を実行します。

```
qaagent -q -su -c "UID=dba;PWD=sql;DBF=qanywhere.db"
```


-sur オプション

クライアント・メッセージ・ストアを現在のバージョンにアップグレードします。

構文

qaagent -sur -c "*connection-string*" ...

備考

アップグレードするデータベースを接続文字列に指定してください。-sur オプションを指定すると、メッセージ・ストアのアンロード、再ロード、アップグレードが自動的に実行されます。

バージョン 9 のメッセージ・ストアをバージョン 10 にアップグレードするには、アンロードと再ロードを実行する必要があります。アンロードと再ロードを手動で行う場合は、-su オプションを指定します。たとえば、再ロード後にカスタム・アクションを実行してからアップグレードする必要がある場合は、-su オプションを使用します。

この操作はアップグレードが完了すると終了します。

この操作は取り消せません。

参照

- 「-su オプション」 183 ページ

例

次の例では、qanywhere.db というバージョン 9.0.2 の SQL Anywhere データベースをアンロードと再ロードしてから、QAnywhere バージョン 10 にアップグレードします。

```
qaagent -q -sur -c "UID=dba;PWD=sql;DBF=qanywhere.db"
```

-v オプション

このオプションを使用すると、メッセージ・ログ・ファイルに出力する情報と、QAnywhere Agent コンソールに表示する情報を指定できます。冗長レベルを上げ過ぎるとパフォーマンスに影響する可能性があるので、通常は冗長レベルを上げるのは開発段階だけにしてください。

構文

qaagent -v *levels* ...

デフォルト

最低の冗長レベル

備考

-v オプションは、ログ・ファイルとコンソールに影響します。qaagent コマンド・ラインで -o または -ot を指定すると、メッセージ・ログだけが記録されます。

−v を単独で指定すると、少量の情報のみがログに出力されます。

levels の値は次のとおりです。-vlm のように、一度に 1 つ以上のオプションを指定することもできます。

- **+** すべてのロギング・オプションを有効にします。
- **l** Mobile Link Listener のロギングをすべて表示します。このオプションを指定すると、Mobile Link Listener (dblsn) が冗長レベル -v3 で起動されます。

  詳細については、「Listener 構文」『Mobile Link - サーバ起動同期』の -v オプションを参照してください。
- **m** dbmlsync のロギングをすべて表示します。このオプションを指定すると、SQL Anywhere 同期サーバ (dbmlsync) が冗長レベル -v+ で起動されます。

  詳細については、dbmlsync の「-v オプション」『Mobile Link - クライアント管理』を参照してください。
- **n** ネットワーク・ステータス変化通知をすべて表示します。QAnywhere Agent は、これらの通知を Listener ユーティリティから受信します。
- **p** Push 通知をすべて表示します。QAnywhere Agent は、これらの通知を、Mobile Link サーバ上の Mobile Link Notifier を介して Listener ユーティリティから受信します。
- **q** 転送ルールを表すために使用される SQL を表示します。
- **s** QAnywhere Agent によって開始されたメッセージ同期をすべて表示します。

参照

- 「-o オプション」 170 ページ
- 「-ot オプション」 173 ページ
- 「-on オプション」 171 ページ
- 「-os オプション」 172 ページ

-x オプション

Mobile Link サーバとの通信に使用するネットワーク・プロトコルとプロトコル・オプションを指定します。

構文

qaagent -x *protocol* [**(** *protocol-options*;... **)** ...

protocol: **http**, **tcpip**, **https**, **tls**

protocol-options: *keyword*=*value*

備考

protocol-options の完全なリストについては、「Mobile Link クライアントのネットワーク・プロトコル・オプション」『Mobile Link - クライアント管理』を参照してください。

-x オプションは、Mobile Link サーバが QAnywhere Agent と同じデバイス上に存在しない場合、必須です。

-x オプションは複数個指定できます。これによって、複数の Mobile Link サーバへのフェールオーバを設定できます。フェールオーバを設定すると、QAnywhere Agent は、コマンド・ラインに入力された順番で各 Mobile Link サーバへの接続を試みます。

QAnywhere Agent は Listener を使用する場合もあります。Listener は、クライアントへの転送を待機中のメッセージがサーバ上に存在することを知らせる通知を Mobile Link サーバから受信します。この Listener は最初に指定されている Mobile Link サーバだけを使用し、それ以外のサーバへのフェールオーバは行いません。

参照

- 「Mobile Link クライアントのネットワーク・プロトコル・オプション」『Mobile Link - クライアント管理』
- 「通信ストリームの暗号化」 192 ページ
- 「トランスポート・レイヤ・セキュリティ」『SQL Anywhere サーバ - データベース管理』
- 「フェールオーバ・メカニズムの設定」 49 ページ
- 「-fd オプション」 161 ページ
- 「-fr オプション」 162 ページ

-xd オプション

QAnywhere Agent で Mobile Link サーバの動的アドレス指定を使用することを指定します。

構文

qaagent -xd

備考

-xd を指定すると、QAnywhere Agent では、メッセージ・ストア・プロパティに基づいて Mobile Link サーバのプロトコルとアドレスを特定できます。したがって、単一の Mobile Link サーバのアドレスを動的に特定できます。Mobile Link サーバのアドレスは、QAnywhere Agent が実行されているデバイスでアクティブな現在のネットワークによって異なります。

QAnywhere アプリケーションでは、Mobile Link サーバの通信プロトコルとアドレスを示すメッセージ・ストア・プロパティと、現在アクティブなネットワーク・インタフェースへの関係を初期化する必要があります。モバイル・デバイスのネットワークが切り替わると、QAnywhere Agent でアクティブなネットワークが検出され、Mobile Link サーバの通信プロトコルとアドレスが自動的に調整されます。このとき再起動は必要ありません。

参照

- 「クライアント・メッセージ・ストア・プロパティ」 230 ページ

例

次の例では、デバイスが接続されているネットワークのタイプに基づいて、適切な Mobile Link のアドレスが使用されるようにプロパティを設定しています。たとえば、デバイスが LAN に接続されている場合、適切な LAN アドレスが使用されます。

```
QAManager mgr;
...
mgr.SetStringStoreProperty( "LAN.CommunicationAddress", "host=1.2.3.4;port=10997" );
mgr.SetStringStoreProperty( "LAN.CommunicationType", "tcpip" );
mgr.SetStringStoreProperty( "WAN.CommunicationAddress", "host=5.6.7.8;port=7777" );
mgr.SetStringStoreProperty( "WAN.CommunicationType", "tcpip" );
mgr.SetStringStoreProperty( "EL3C589 Ethernet Adapter.type", "LAN" );
mgr.SetStringStoreProperty( "Sierra Wireless AirCard 555 Adapter.type", "WAN" );
```


第 8 章

セキュリティ保護されたメッセージング・アプリケーションの作成

目次

セキュリティ保護されたクライアント・メッセージ・ストアの作成

クライアント・メッセージ・ストアをセキュリティで保護するには、次の方法があります。

- デフォルトのパスワードを変更する。

  「クライアント・メッセージ・ストアのパスワード管理」 190 ページを参照してください。

- メッセージ・ストアの内容を暗号化する。

  「クライアント・メッセージ・ストアの暗号化」 191 ページを参照してください。

例

まず、暗号化キーを使用して SQL Anywhere データベースを作成します。

```
dbinit mystore.db -i -s -ek some_phrase
```

小型デバイスの場合は、-i オプションと -s オプションが最適です。強力な暗号化を使用する場合は、-ek オプションで暗号化キーを指定します。「初期化ユーティリティ (dbinit)」『SQL Anywhere サーバ - データベース管理』を参照してください。

次に、データベースをクライアント・メッセージ・ストアとして初期化します。

```
qaagent -id mystore -si -c "dbf=mystore.db;dbkey=some_phrase"
```

次に、DBA 権限を持つ新規のリモート・ユーザを作成し、そのユーザのパスワードを設定します。デフォルトの QAnywhere ユーザを削除し、デフォルトの DBA ユーザのパスワードを変更します。ユーザ DBA、パスワード SQL でログインし、次の SQL 文を実行します。

```
GRANT CONNECT TO secure_user IDENTIFIED BY secure_password
GRANT MEMBERSHIP IN GROUP ml_qa_user_group TO secure_user
GRANT REMOTE dba TO secure_user
REVOKE CONNECT FROM ml_qa_user
GRANT CONNECT TO dba IDENTIFIED BY new_dba_password
COMMIT
```

注意
すべての QAnywhere ユーザが ml_qa_user_group に属しており、REMOTE DBA 権限を持っていることが必要です。

次に、セキュリティ保護された DBA ユーザとして、QAnywhere Agent を起動します。

```
qaagent -id mystore -c "dbf=mystore.db;dbkey=some_phrase;uid=secure_user;pwd=secure_password"
```

クライアント・メッセージ・ストアのパスワード管理

メッセージ・ストア用に作成されたデフォルトのユーザ ID のパスワードは変更しなければなりません。すべての SQL Anywhere データベースには、デフォルトのユーザ ID DBA とパスワー

ド SQL が作成されます。また、qaagent -si オプションを指定すると、デフォルトのユーザ ID ml_qa_user とデフォルトのパスワード qanywhere が作成されます。これらのパスワードを変更するには、GRANT 文を使用します。

「パスワードの変更」『SQL Anywhere サーバ - データベース管理』を参照してください。

クライアント・メッセージ・ストアの暗号化

次のコマンドを使用すると、クライアント・メッセージ・ストアを作成時に暗号化できます。

```
dbinit -i -s -ek encryption-key database-file
```

小型デバイスにデータベースを作成する場合は、-i オプションと -s オプションが最適です。暗号化キーを使用してメッセージ・ストアを初期化した場合は、暗号化されたメッセージ・ストアに対してデータベース・サーバを起動するために、その暗号化キーが必要になります。

暗号化されたメッセージ・ストアに対して QAnywhere Agent を起動するには、次のコマンドを使用して、暗号化キーを指定します。QAnywhere Agent は、与えられた暗号化キーを使用して、暗号化されたメッセージ・ストアに対して自動的にデータベース・サーバを起動します。

```
qaagent -c "DBF=database-file;DBKEY=encryption-key"
```

これで、アプリケーションは、QAnywhere API を介して、暗号化されたメッセージ・ストアにアクセスできるようになります。メッセージ・ストアを管理するためのデータベース・サーバはすでに実行されているため、アプリケーションが暗号化キーを指定する必要はありません。

QAnywhere Agent が実行されていないときに、アプリケーションが暗号化されたメッセージ・ストアにアクセスする必要がある場合、QAnywhere API は、QAnywhere Manager 初期化ファイルに指定された接続パラメータを使用して自動的にデータベース・サーバを起動します。暗号化されたメッセージ・ストアに対してデータベース・サーバを起動するには、次のように、データベース接続パラメータ内に暗号化キーを指定する必要があります。

```
CONNECT_PARAMS=DBF=database-file;DBKEY=encryption-key
```

参照

- 「データベースの暗号化」『SQL Anywhere サーバ - データベース管理』
- 「初期化ユーティリティ (dbinit)」『SQL Anywhere サーバ - データベース管理』
- QAnywhere Agent 「-c オプション」 159 ページ

通信ストリームの暗号化

qaagent -x オプションを使用すると、QAnywhere Agent が Mobile Link サーバと通信するときに使用するセキュリティ保護された通信ストリームを指定できます。また、サーバ側証明書を使用したサーバ認証を実装し、高度な暗号を使用して通信ストリームを暗号化できます。

「-x オプション」 186 ページを参照してください。

Mobile Link サーバのトランスポート・レイヤ・セキュリティの設定も行う必要があります。デジタル証明書の作成と Mobile Link サーバの設定の詳細については、「Mobile Link クライアント／サーバ通信の暗号化」『SQL Anywhere サーバ - データベース管理』を参照してください。

別途ライセンスが必要な必須コンポーネント
ECC 暗号化と FIPS 承認の暗号化には、別途ライセンスが必要です。強力な暗号化テクノロジはすべて、輸出規制対象品目です。
「別途ライセンスが必要なコンポーネント」『SQL Anywhere 10 - 紹介』を参照してください。

例

次の例では、QAnywhere Agent と Mobile Link サーバとの間でセキュリティ保護された通信ストリームを確立する方法を示します。この例では、SQL Anywhere セキュリティ・オプションと一緒にインストールされるサンプルの証明書を使用しています。

RSA を使用してセキュリティ保護された TCP/IP：

```
mlsrv10 -x tls(tls_type=rsa;certificate=rsaserver.crt;certificate_password=test)
qaagent -x tls(tls_type=rsa;trusted_certificates=rsaroot.crt)
```

ECC を使用してセキュリティ保護された TCP/IP：

```
mlsrv10 -x tls(tls_type=ecc;certificate=sample.crt;certificate_password=tJ1#m6+W)
qaagent -x tls(tls_type=ecc;trusted_certificates=eccroot.crt)
```

HTTPS を使用してセキュリティ保護された HTTP (HTTPS では RSA 証明書のみをサポート)：

```
mlsrv10 -x https(certificate=rsaserver.crt;certificate_password=test)
qaagent -x https(trusted_certificates=rsaroot.crt)
```

Mobile Link に対するパスワード認証の使用

リモート・デバイスとサーバの間でセキュリティ保護された通信ストリームを確立したら、デバイスのユーザを認証してサーバと通信できるようにするのが普通です。

これを行うには、クライアント・メッセージ・ストア用の Mobile Link ユーザ名を作成し、サーバ・メッセージ・ストアに登録します。

参照

- 「-mu オプション」 169 ページ
- 「-mp オプション」 168 ページ
- 「Mobile Link ユーザ」『Mobile Link - クライアント管理』

第 9 章

モバイル Web サービス

目次

モバイル Web サービスの概要

Web サービスは、アプリケーションの機能を公開するための一般的な方法になっています。Web サービスを使用すると、さまざまな企業のリソース間で、より良い相互運用性を確保できます。これらのサービスは、モバイル・アプリケーションの機能を広げ、開発プロセスを簡素化します。

モバイル環境に Web サービスを実装することは、取り組みがいのある仕事になることがあります。なぜなら、接続を利用できなかったり (または中断されたり)、無線通信環境やデバイスに起因する制限事項があったりするためです。たとえば、モバイル・アプリケーションを使用するユーザは、オフラインの状態で Web サービスに対する要求を作成したり、オフラインに変更するときに応答を取得したりできます。また、IT 管理者は、モバイル・アプリケーションが使用しているネットワーク接続のタイプ (GPRS、802.11、クレードルなど) に基づいて Web サービス応答のサイズを制限するルールを指定できます。

QAnywhere は、QAnywhere の蓄積転送メッセージング・アーキテクチャを活用する、モバイルに最適化された非同期 Web サービスを使用して、これらの課題に取り組みます。QAnywhere のモバイル Web サービスを使用すると、モバイル・アプリケーションでは、オフラインの状態でも Web サービス要求を作成し、後で転送できるように要求をキューイングできます。要求は、QAnywhere メッセージとして配信されます。サーバ側の Web サービス・コネクタが要求を作成し、Web サービスから要求を受け取って、クライアントに対してメッセージとして応答が返されます。QAnywhere 転送ルールは、さまざまなパラメータ (使用されるネットワーク、要求／応答のサイズ) に基づいて、送信される要求や応答を制御できます。転送ルールで制御することにより、モバイル・アプリケーションが実証済みのテクノロジや単純なプログラミング・モデルを使用してさまざまな Web サービスを利用できるような、洗練された柔軟なアーキテクチャが構築されます。

開発の観点から見ると、接続された環境と同程度に Web サービスのプロキシ・クラスを扱うことができます。QAnywhere は、転送、認証、直列化などのあらゆるものに対応します。WSDL ドキュメントを取得し、モバイル・アプリケーションが Web サービスの起動に使用できる特定のクラス・プロキシ (.NET または Java) を生成することを目的に、WSDL コンパイラが提供されます。これらのクラスは、基本の QAnywhere インフラストラクチャを使用して、要求を送信し、応答を受信します。オブジェクトのメソッド呼び出しが行われると、SOAP 要求が自動的に生成され、メッセージとしてサーバに配信されます。サーバ側では、コネクタが Web サービス要求を作成し、メッセージとして結果を返します。

参照

- 「モバイル Web サービス」 『SQL Anywhere 10 - 紹介』

モバイル Web サービスの設定

次の手順には、モバイル Web サービスの設定に必要なタスクの概要が記載されています。

♦ モバイル Web サービス設定の概要

1. サーバ・メッセージ・ストアがない場合は設定します。

「サーバ・メッセージ・ストアの設定」 32 ページを参照してください。

2. -m オプションを指定して Mobile Link サーバを起動し、サーバ・メッセージ・ストアへの接続を開始します。

   「QAnywhere サーバの起動」 33 ページを参照してください。

3. クライアント・メッセージ・ストアがない場合は設定します。このメッセージ・ストアは、メッセージを一時的に格納するための SQL Anywhere データベースです。

   「クライアント・メッセージ・ストアの設定」 36 ページを参照してください。

4. QAnywhere WSDL コンパイラを実行し、アプリケーションで使用するクラスを作成します。

   「QAnywhere WSDL コンパイラの実行」 198 ページを参照してください。

5. 各クライアントで、WSDL コンパイラによって生成されたクラスを使用する Web サービス・クライアント・アプリケーションを記述します。

   「モバイル Web サービス・アプリケーションの記述」 199 ページを参照してください。

6. Web サービス・コネクタを作成します。

   「Web サービス・コネクタの設定」 209 ページを参照してください。

7. クライアントごとに、ローカルのクライアント・メッセージ・ストアに接続する QAnywhere Agent (qaagent) を起動します。

   「QAnywhere Agent の実行」 38 ページを参照してください。

クイック・スタートのためのその他の資料

- 仮想 Web サービスを使用する簡単な例については、「モバイル Web サービスの例」 212 ページを参照してください。
- フル機能を備えたモバイルの Web サービスのサンプル・アプリケーションが、*samples-dir¥QAnywhere¥MobileWebServices* にインストールされています。*samples-dir* の詳細については、「サンプル・ディレクトリ」『SQL Anywhere サーバ - データベース管理』を参照してください。このサンプルは、モバイル Web サービスを使用して非同期の Web サービス要求を作成する方法を示します。Java と C# の両方のサンプルが提供されています。

QAnywhere WSDL コンパイラの実行

Web サービスを記述する WSDL ファイルを受けて、QAnywhere WSDL コンパイラは、アプリケーションに含まれる Java または C# のプロキシ クラスのセットを生成します。これらのクラスは、メソッド呼び出しとして Web サービス操作を公開します。生成されるクラスは以下のとおりです。

- メインのサービス・バインディング・クラス (このクラスは、モバイル Web サービス・ランタイムの WSBase を継承します)
- プロキシ・クラス (WSDL ファイルに指定された複合型ごとに作成されます)

作成されるプロキシ・クラスの詳細については、次の項を参照してください。

- .NET：「iAnywhere.QAnywhere.WS ネームスペース (.NET 1.0)」 371 ページ
- Java：「ianywhere.qanywhere.ws package」 641 ページ

WSDL コンパイラは、HTTP と HTTPS を介して WSDL 1.1 と SOAP 1.1 をサポートします。

構文

wsdlc -l *programming-language wsdl-file* [*options*]

パラメータ

programming-language: **cs** | **java**

wsdl-file：Web サービスを記述する WSDL ファイルの名前

オプション	説明
-h	ヘルプ画面を表示する
-v	冗長情報を表示する
-o *output-directory*	生成ファイルの出力ディレクトリを指定する
-d	デバッグ情報を表示する
-n	ネームスペースを指定する (C# 出力のみ)
-p	ネームスペースを指定する (Java 出力のみ)

モバイル Web サービス・アプリケーションの記述

アプリケーションは、Web サービス要求を QAnywhere に送信します。QAnywhere は、受信した要求を、Mobile Link サーバのモバイル Web サービス・コネクタに送信します。コネクタは、受信した要求を Web サービスに送信します。または、Web サービスが利用可能になるまで、その要求をキューに格納します。QAnywhere は、要求を受け取ると、アプリケーションに通知します。または、アプリケーションが利用可能になるまで、その要求をキューに格納します。

.NET モバイル Web サービス・アプリケーションの設定

QAnywhere で .NET を使用する前に、Visual Studio .NET プロジェクトに対して以下の変更を加える必要があります。

- QAnywhere .NET DLL とモバイル Web サービス .NET DLL への参照を追加します。これにより、Visual Studio.NET に対して、QAnywhere .NET API とモバイル Web サービス .NET API のコードを見つけるためにインクルードする DLL を通知することになります。
- ソース・コードに、QAnywhere .NET API クラスとモバイル Web サービス .NET API クラスを参照する行を追加します。QAnywhere .NET API を使用するには、データ・プロバイダを参照する行もソース・コードに追加する必要があります。C# では、Visual Basic .NET 用とは異なる行を追加します。

上記の手順について、以下に詳しく説明します。

♦ Visual Studio .NET のプロジェクトに QAnywhere .NET API とモバイル Web サービス API への参照を追加するには、次の手順に従います。

1. Visual Studio .NET を起動し、プロジェクトを開きます。
2. [ソリューション エクスプローラ] ウィンドウで、[参照設定] フォルダを右クリックし、ポップアップ・メニューから [参照の追加] を選択します。

   [参照の追加] ダイアログが表示されます。
3. [.NET] タブで、[参照] をクリックして *iAnywhere.QAnywhere.Client.dll* と *iAnywhere.QAnywhere.WS.dll* を見つけます。これらのファイルは、SQL Anywhere のインストール・ディレクトリを基準とした相対ディレクトリに格納されています。

   - .NET Framework 1.1：*¥Assembly¥v1*
   - .NET Framework 2.0：*¥Assembly¥v2*
   - .NET Compact Framework 1.0：*ce¥Assembly¥v1*
   - .NET Compact Framework 2.0：*ce¥Assembly¥v2*

   使用環境の適切なディレクトリから、それぞれの DLL を選択し、[開く] をクリックします。
4. DLL がプロジェクトに追加されたことを確認するには、[参照の追加] ダイアログを開き、[.NET] タブを開きます。[選択されたコンポーネント] リストに、

iAnywhere.QAnywhere.Client.dll と *iAnywhere.QAnywhere.WS.dll* が表示されます。[OK] をクリックします。

ソース・コードのデータ・プロバイダ・クラスを参照する

♦ **コード内で QAnywhere .NET API とモバイル Web サービス API のクラスを参照するには、次の手順に従います。**

1. Visual Studio .NET を起動し、プロジェクトを開きます。
2. C# を使用している場合は、ファイルの先頭にある using ディレクティブのリストに次の行を追加します。

```
using iAnywhere.QAnywhere.Client;
using iAnywhere.QAnywhere.WS;
```

3. Visual Basic .NET を使用している場合は、ファイルの先頭にある imports リストに次の行を追加します。

```
Imports iAnywhere.QAnywhere.Client
Imports iAnywhere.QAnywhere.WS
```

Imports 行は必須ではありません。しかし、この行を追加すると、QAnywhere とモバイル Web サービスのクラスの省略形を使用できるようになります。この行を追加しなくても、完全に修飾されたクラス名をコードで使用できます。たとえば、次のコードは長形式を使用しています。

```
iAnywhere.QAnywhere.Client.QAManager
mgr =
 new iAnywhere.QAnywhere.Client.QAManagerFactory.Instance.CreateQAManager(
"qa_manager.props" );
```

次のコードは省略形を使用しています。

```
QAManager mgr = QAManagerFactory.Instance.CreateQAManager(
 "qa_manager.props" );
```

♦ **.NET 用に QAnywhere とモバイル Web サービスを初期化するには、次の手順を実行します。**

1. 前の手順の説明に従って、iAnywhere.QAnywhere.Client と iAnywhere.QAnywhere.WS ネームスペースをインクルードします。

```
using iAnywhere.QAnywhere.Client;
using iAnywhere.QAnywhere.WS;
```

2. QAManager オブジェクトを作成します。

たとえば、デフォルトの QAManager オブジェクトを作成するには、パラメータに null を指定して、CreateQAManager を呼び出します。

```
QAManager mgr;
mgr = QAManagerFactory.Instance.CreateQAManager( null );
```

ヒント
同時実行性の利点を最大限に生かすために、マルチスレッド・アプリケーションでは、スレッドごとに QAManager を作成する必要があります。「マルチスレッド QAManager」 68 ページを参照してください。

QAManagerFactory の詳細については、「QAManagerFactory クラス」 339 ページを参照してください。

または、プロパティ・ファイルを使用して、カスタマイズされた QAManager オブジェクトを作成することもできます。次のように、CreateQAManager メソッドの引数としてプロパティ・ファイルを指定します。

```
mgr = QAManagerFactory.Instance.CreateQAManager(
  "qa_mgr.props" );
```

qa_mgr.props は、リモート・デバイス上に存在するプロパティ・ファイルの名前です。

3. QAManager オブジェクトを初期化します。次に例を示します。

```
mgr.Open(
  AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
```

open メソッドの引数には、メッセージの受信確認方法を示す受信確認モードを指定します。これは、IMPLICIT_ACKNOWLEDGEMENT または EXPLICIT_ACKNOWLEDGEMENT のどちらかにする必要があります。

モバイル Web サービスによって使用される QAnywhere メッセージは、モバイル Web サービス・アプリケーションにアクセスできません。EXPLICIT_ACKNOWLEDGEMENT モードで QAManager を使用する場合は、WSResult の Acknowledge メソッドを使用して、Web サービス要求の結果を含む QAnywhere メッセージの受信確認を行います。このメソッドは、アプリケーションが応答を正常に処理したことを示します。

受信確認モードの詳細については、次の項を参照してください。

- WSBase 「SetQAManager メソッド」 376 ページ
- wsResult 「Acknowledge モード」 388 ページ

QAManager を作成する代わりに、QATransactionalManager を作成できます。「トランザクション志向メッセージングの実装 (.NET クライアントの場合)」 75 ページを参照してください。

4. サービス・バインディング・クラスのインスタンスを作成します。

モバイル Web サービスの WSDL コンパイラは、Web サービスを定義する WSDL ドキュメントからサービス・バインディング・クラスを生成します。

QAManager は、Web サービス・バインディング・クラスのインスタンスによって、Web サービス要求を作成する処理でメッセージ操作を行うために使用されます。QAnywhere を介して Web サービス要求を送信するためのコネクタ・アドレスを設定します。これは、サービス・バインディング・クラスのプロパティ WS_CONNECTOR_ADDRESS を設定することによって行うことができます。接続先 Web サービスの URL を使用して、各 QAnywhere Web

サービス・コネクタを設定します。複数の URL に配置される Web サービスを必要とするアプリケーションの場合は、各 URL についてコネクタを設定します。

次に例を示します。

```
CurrencyConverterSoap service = new CurrencyConverterSoap( )
service.SetQAManager(mgr);
service.setProperty(
  "WS_CONNECTOR_ADDRESS",
  "ianywhere.connector.currencyconvertor¥¥");
```

アドレス末尾の ¥¥ は、必ず含めてください。

参照

- 「iAnywhere.QAnywhere.WS ネームスペース (.NET 1.0)」 371 ページ
- 「iAnywhere.QAnywhere.Client ネームスペース (.NET 1.0)」 262 ページ

例

モバイル Web サービスを初期化するには、QAManager を作成し、サービス・バインディング・クラスのインスタンスを作成する必要があります。次に例を示します。

```
// QAnywhere initialization
      QAManager mgr = QAManagerFactory.Instance.CreateQAManager( null );
      mgr.SetProperty( "CONNECT_PARAMS",
"eng=qanywhere;dbf=qanywhere.db;uid=ml_qa_user;pwd=qanywhere" );
      mgr.Open( AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT );
      mgr.Start();

      // Instantiate the web service proxy
      CurrencyConvertorSoap service = new CurrencyConvertorSoap();
      service.SetQAManager( mgr );
    service.SetProperty( "WS_CONNECTOR_ADDRESS", "ianywhere.connector.currencyconvertor¥¥" );
```

Java モバイル Web サービス・アプリケーションの設定

Java でモバイル Web サービス・アプリケーションを作成するには、次の初期化処理を実行する必要があります。

♦ Java 用に QAnywhere とモバイル Web サービスを初期化するには、次の手順を実行します。

1. 次のファイルのロケーションをクラスパスに追加します。デフォルトでは、これらのファイルは *install-dir¥java* に配置されています。

   - *qaclient.jar*
   - *iawsrt.jar*
   - *jaxrpc.jar*

2. ianywhere.qanywhere.client と ianywhere.qanywhere.ws パッケージをインポートします。

   ```
   import ianywhere.qanywhere.client.*;
   import ianywhere.qanywhere.ws.*;
   ```

3. QAManager オブジェクトを作成します。


```
QAManager mgr;
mgr = QAManagerFactory.getInstance().createQAManager(null);
```

createQAManager メソッドにプロパティ・ファイルを指定して、QAManager オブジェクトをカスタマイズすることもできます。

```
mgr = QAManagerFactory.getInstance().createQAManager("qa_mgr.props.");
```

ヒント
同時実行性の利点を最大限に生かすために、マルチスレッド・アプリケーションでは、スレッドごとに QAManager を作成する必要があります。「マルチスレッド QAManager」 68 ページを参照してください。

4. QAManager オブジェクトを初期化します。

```
mgr.open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
```

open メソッドの引数には、メッセージの受信確認方法を示す受信確認モードを指定します。これは、IMPLICIT_ACKNOWLEDGEMENT または EXPLICIT_ACKNOWLEDGEMENT のどちらかにする必要があります。

モバイル Web サービスによって使用される QAnywhere メッセージは、モバイル Web サービス・アプリケーションにアクセスできません。EXPLICIT_ACKNOWLEDGEMENT モードで QAManager を使用する場合は、WSResult の Acknowledge メソッドを使用して、Web サービス要求の結果を含む QAnywhere メッセージの受信確認を行います。このメソッドは、アプリケーションが応答を正常に処理したことを示します。

受信確認モードの詳細については、次の項を参照してください。

- WSBase 「setQAManager メソッド」 644 ページ
- WSResult 「acknowledge メソッド」 651 ページ

QAManager を作成する代わりに、QATransactionalManager を作成できます。「トランザクション志向メッセージングの実装 (Java クライアントの場合)」 78 ページを参照してください。

5. サービス・バインディング・クラスのインスタンスを作成します。

モバイル Web サービスの WSDL コンパイラは、Web サービスを定義する WSDL ドキュメントからサービス・バインディング・クラスを生成します。

Web サービス要求を作成する処理において、QAManager は、Web サービス・バインディング・クラスのインスタンスによって、メッセージ操作を行うために使用されます。QAnywhere を介して Web サービス要求を送信するためのコネクタ・アドレスを設定します。これは、サービス・バインディング・クラスの WS_CONNECTOR_ADDRESS を設定することによって行うことができます。接続先 Web サービスの URL を使用して、各 QAnywhere Web サービス・コネクタを設定します。複数の URL に配置される Web サービスを必要とするアプリケーションの場合は、QAnywhere コネクタはサービスの URL ごとに設定する必要があります。

次に例を示します。

```
CurrencyConverterSoap service = new CurrencyConverterSoap( );
service.setQAManager(mgr);
service.setProperty("WS_CONNECTOR_ADDRESS", "ianywhere.connector.currencyconvertor¥
¥");
```

アドレス末尾の ¥¥ は、必ず含めてください。

参照

- 「ianywhere.qanywhere.ws package」 641 ページ
- 「ianywhere.qanywhere.client パッケージ」 532 ページ

例

モバイル Web サービスを初期化するには、QAManager を作成し、サービス・バインディング・クラスのインスタンスを作成する必要があります。次に例を示します。

```
// QAnywhere initialization
    Properties props = new Properties();
    props.put( "CONNECT_PARAMS",
"eng=qanywhere;dbf=qanywhere.db;uid=ml_qa_user;pwd=qanywhere" );
    QAManager mgr = QAManagerFactory.getInstance().createQAManager( props );
    mgr.open( AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT );
    mgr.start();

    // Instantiate the web service proxy
    CurrencyConvertorSoap service = new CurrencyConvertorSoap();
    service.setQAManager( mgr );
    service.setProperty( "WS_CONNECTOR_ADDRESS", "ianywhere.connector.currencyconvertor¥¥" );
```

サービス・バインディング・クラスの複数のインスタンス

サービス・バインディング・クラスのインスタンスは、QAManager ごとに作成する必要があります。サービス・バインディング・クラスの複数のインスタンスを使用するモバイル Web サービス・アプリケーションである場合は、SetServiceID メソッドを使用してサービス ID を設定することが重要です。次に例を示します。

```
service1.SetServiceID("1")
service2.SetServiceID("2")
```

サービス ID は、サービス名を組み合わせて、Web サービス応答の受信に使用するキュー名を形成します。指定するサービスの各インスタンスには、ユニークなサービス ID を設定することが重要です。これにより、特定のインスタンスが、サービスの別のインスタンスによって作成された要求に対する応答を受け取ることはありません。サービス ID が設定されていないと、デフォルトの "" になります。キュー名では複数のアプリケーションにわたって一時的にメッセージ・ストア内のメッセージを保持するため、サービス ID は、同じサービスを使用する複数のアプリケーションが互いに競合しないようにするためにも重要です。

モバイル Web サービス・アプリケーションのコンパイルと実行

ランタイム・ライブラリ

Java のランタイム・ライブラリは *iawsrt.jar* です。SQL Anywhere インストール・ディレクトリの *java* サブディレクトリに格納されています。

C# のランタイム・ライブラリは *iAnywhere.QAnywhere.WS.dll* です。SQL Anywhere のインストール・ディレクトリを基準として、次の相対ディレクトリに格納されています。

- .NET Framework 1.1：*¥Assembly¥v1*
- .NET Framework 2.0：*¥Assembly¥v2*
- .NET Compact Framework 1.0：*ce¥Assembly¥v1*
- .NET Compact Framework 2.0：*ce¥Assembly¥v2*

次の項では、モバイル Web サービス・アプリケーションをコンパイルして実行するために必要なファイルについて説明します。

必要なランタイム・ライブラリ (Java)

次のファイル (SQL Anywhere 10 インストール・ディレクトリの *java* サブディレクトリにあります) をクラスパスに含めます。

- *jaxrpc.jar*
- *qaclient.jar*
- *iawsrt.jar*

必要なランタイム・ライブラリ (.NET)

SQL Anywhere 10 をインストールすると、次のファイルは Global Assembly Cache に自動的に登録されます。

- *iAnywhere.QAnywhere.Client.dll*
- *iAnywhere.QAnywhere.WS.dll*

モバイル Web サービスの停止

モバイル Web サービス・アプリケーションは、QAManager を閉じることによって、順次停止されます。次に例を示します。

```
// QAnywhere finalization in C#:
mgr.Stop();
mgr.Close();

// QAnywhere finalization in Java:
mgr.stop();
mgr.close();
```

Web サービス要求の作成

モバイル Web サービス・アプリケーションで Web サービス要求を作成するには、2 つの基本的な方法があります。

- **同期**　「同期の Web サービス要求」 206 ページを参照してください。
- **非同期**　「非同期の Web サービス要求」 206 ページを参照してください。

同期の Web サービス要求

同期の Web サービス要求は、アプリケーションがネットワークに接続されている場合に使用します。この方法では、サービス・バインディング・クラスでメソッドの呼び出しを行ったときに、Web サービス要求が作成されます。そして、サーバからの Web サービス応答を受信した場合にのみ、結果が返されます。

例

次の例では、USD から CAD への為替レートを取得する要求を作成します。

```
//C#
double r = service.ConversionRate( Currency.USD, Currency.CAD );

// Java
double r = service.conversionRate( NET.webserviceX.Currency.USD, NET.webserviceX.Currency.CAD );
```

非同期の Web サービス要求

非同期の Web サービス要求は、モバイル Web サービス・アプリケーションがまれにしかネットワークに接続しない場合に有用です。この方法では、サービス・バインディング・クラスでメソッドの呼び出しを行ったときに Web サービス要求が作成され、作成された要求は出力キューに置かれます。メソッドは WSResult を返します。この WSResult は、応答のステータスについて後から問い合わせるときに使用できます。アプリケーションを再起動した後でも使用できます。

次の例では、USD から CAD への為替レートを取得する非同期の要求を作成します。

```
// C#
WSResult r = service.AsyncConversionRate( Currency.USD, Currency.CAD );

// Get the request ID.  Save it for later use if necessary.
string reqID = r.GetRequestID();

// Later: get the response for the specified request ID
WSResult r = service.GetResult( reqID );
if( r.GetStatus() == WSStatus.STATUS_RESULT_AVAILABLE ) {
  Console.WriteLine( "The conversion rate is " + r.GetDoubleValue( "ConversionRateResult" ) );
} else {
  Console.WriteLine( "Response not available" );
}
```

```
// Java
WSResult r = service.asyncConversionRate( NET.webserviceX.Currency.USD,
NET.webserviceX.Currency.CAD );

// Get the request ID.  Save it for later use if necessary.
String reqID = r.getRequestID();

// Later: get the response for the specified request ID
WSResult r = service.getResult( reqID );
if( r.getStatus() == WSStatus.STATUS_RESULT_AVAILABLE ) {
   System.out.println( "The conversion rate is " + r.getDoubleValue( "ConversionRateResult" ) );
} else {
   System.out.println( "Response not available" );
}
```

Web サービス要求に対する応答を利用できるときは、WSListener を使用して、非同期のコールバックを取得することもできます。次に例を示します。

```
// C#
// Make a request to get the USD to CAD exchange rate
WSResult r = service.AsyncConversionRate( Currency.USD, Currency.CAD );

// Register a listener for the result
service.SetListener( r.GetRequestID(), new CurrencyConvertorListener() );

// Java
// Make a request to get the USD to CAD exchange rate
WSResult r = service.asyncConversionRate( NET.webserviceX.Currency.USD,
NET.webserviceX.Currency.CAD );

// Register a listener for the result
service.setListener( r.getRequestID(), new CurrencyConvertorListener() );
```

WSListener インタフェースは、非同期イベントを扱うための 2 つのメソッドを定義します。

- **OnResult**　OnResult メソッドは、Web サービス要求に対する応答を処理するために実装されます。Web サービス要求の結果を表す WSResult オブジェクトが渡されます。
- **OnException**　OnException メソッドは、Web サービス要求への応答を処理しているときに発生したエラーを扱うために実装されます。WSException オブジェクトと WSResult オブジェクトが渡されます。WSException オブジェクトには、発生したエラーに関する情報が含まれます。WSResult オブジェクトは、応答に対応する要求 ID を取得するために使用されます。

```
// C#
class CurrencyConvertorListener : WSListener
{
  public CurrencyConvertorListener() {
  }

  public void OnResult( WSResult r ) {
   try {
       USDToCAD._statusMessage = "USD to CAD currency exchange rate: " + r.GetDoubleValue
( "ConversionRateResult" );
   } catch( Exception exc ) {
      USDToCAD._statusMessage = "Request " + r.GetRequestID() + " failed: " + exc.Message;
   }
  }

  public void OnException( WSException exc, WSResult r ) {
   USDToCAD._statusMessage = "Request " + r.GetRequestID() + " failed: " + exc.Message;
```



```
    }
}

// Java
private class CurrencyConvertorListener implements WSListener
{
  public CurrencyConvertorListener() {
  }

  public void onResult( WSResult r ) {
   try {
        USDToCAD._statusMessage = "USD to CAD currency exchange rate: " + r.getDoubleValue
( "ConversionRateResult" );
   } catch( Exception exc ) {
        USDToCAD._statusMessage = "Request " + r.getRequestID() + " failed: " + exc.getMessage();
   }
   }

   public void onException( WSException exc, WSResult r ) {
        USDToCAD._statusMessage = "Request " + r.getRequestID() + " failed: " + exc.getMessage();
   }
}
```

Web サービス・コネクタの設定

Web サービス・コネクタは、特定のアドレスに送信された QAnywhere メッセージを受信し、メッセージが到着したときに Web サービス呼び出しを作成します。Web サービス応答は、QAnywhere メッセージとして送信元のクライアントに送り返されます。Web サービス・コネクタに送信されるすべてのメッセージは、QAnywhere WSDL コンパイラによって生成されたプロキシ・クラスを使用して作成される必要があります。

♦ **Web サービス・コネクタを作成するには、次の手順に従います。**

1. Sybase Central を開き、サーバ・メッセージ・ストアに接続します。
2. [ファイル] – [New Connector] を選択します。

   [コネクタ] ウィザードが表示されます。
3. [コネクタ・タイプ] ページで、[Web サービス] を選択し、[次へ] をクリックします。
4. [コネクタ名] ページで、コネクタ名を入力します。

   これはコネクタのアドレスです。このアドレスは、QAnywhere クライアントで、コネクタを指定するために使用されます。プロパティ ianywhere.connector.address に設定されます。
5. [通信パラメータ] ページで、URL を入力します。

   これは、Web サービスが配置されている URL です (たとえば、*http://localhost:8080/qanyserv/F2C* のように入力します)。プロパティ webservice.url に設定されます。

   オプションで、タイムアウト時間 (ミリ秒単位) を設定することもできます。タイムアウトを設定すると、指定した時間内に Web サービスが応答しない場合に要求はキャンセルされます。値は、プロパティ webservice.socket.timeout に設定されます。
6. [HTTP パラメータ] ページで、必要に応じて次の値を入力します。

   - **HTTP ユーザ名**　Web サービスが HTTP 認証を要求する場合に、このプロパティを使用してユーザ名を指定します。

     値は、プロパティ webservice.http.authName に設定されます。
   - **HTTP パスワード**　Web サービスが HTTP 認証を要求する場合に、このプロパティを使用してパスワードを指定します。

     値は、プロパティ webservice.http.password.e に設定されます。
   - **プロキシ・ホスト名**　HTTP プロキシ経由で Web サービスにアクセスすることが必要である場合に、このプロパティを使用してホスト名を指定します。このプロパティを指定する場合は、webservice.http.proxy.port プロパティも設定する必要があります。

     値は、プロパティ webservice.http.proxy.host に設定されます。
   - **プロキシ・ポート**　プロキシ・サーバに接続するポートです。このプロパティを指定する場合は、webservice.http.proxy.host プロパティも設定する必要があります。

値は、プロパティ webservice.http.proxy.port に設定されます。

- **プロキシ・ユーザ名**　プロキシが認証を要求する場合に使用する、プロキシ・ユーザ名です。このプロパティを指定する場合は、webservice.http.proxy.password.e プロパティも設定する必要があります。

  値は、プロパティ webservice.http.proxy.authName に設定されます。

- **プロキシ・パスワード**　プロキシが認証を要求する場合に使用する、プロキシ・パスワードです。このプロパティを指定する場合は、webservice.http.proxy.authName プロパティも設定する必要があります。

  値は、プロパティ webservice.http.proxy.password.e に設定されます。

7. [終了] をクリックします。
8. Web サービス・コネクタにその他のオプションを設定するには、作成したコネクタを右クリックし、プロパティを選択します。または、サーバ管理要求を使用することもできます。

   使用可能なプロパティのリストについては、「Web サービス・コネクタのプロパティ」 210 ページを参照してください。

   サーバ管理要求の詳細については、「コネクタの管理」 112 ページを参照してください。

Web サービス・コネクタのプロパティ

Web サービス・コネクタのプロパティを使用して、Web サービスとの接続情報を指定します。これらのプロパティは、Sybase Central の [コネクタ] ウィザードで設定できます。

「Web サービス・コネクタの設定」 209 ページを参照してください。

Web サービス・コネクタのプロパティは、Sybase Central の [コネクタ・プロパティ] ダイアログまたは ml_qa_global_props Mobile Link システム・テーブルで参照できます。

[コネクタ・プロパティ] ダイアログを開くには、Sybase Central でコネクタを右クリックし、[プロパティ] を選択します。

Mobile Link システム・テーブル ml_qa_global_props の詳細については、「ml_qa_global_props」『Mobile Link - サーバ管理』を参照してください。

Web サービス・コネクタのプロパティ

- **ianywhere.connector.nativeConnection**　JMS コネクタを実装する Java クラスを指定します。QAnywhere の内部使用のためのものである場合は、削除や変更はできません。
- **ianywhere.connector.id (旧式)**　JMS コネクタをユニークに識別する識別子を指定します。デフォルトは ianywhere.connector.address です。
- **ianywhere.connector.address**　コネクタのアドレスを指定します。このアドレスは、QAnywhere クライアントで、コネクタを指定するために使用されます。指定したアドレスは、コネクタのサーバ・コンソールに表示されるすべてのログ出力エラー、警告、情報メッセージの先頭に付けられます。

Sybase Central では、このプロパティは [コネクタ] ウィザード、[コネクタ名] ページ、[コネクタ名] フィールドで設定します。

- **ianywhere.connector.compressionLevel** Web サービスから受信したメッセージのデフォルトのメッセージ圧縮率です。圧縮率は、0 ー 9 の整数値で指定します。0 は圧縮なし、9 は最大の圧縮率を表します。

  Sybase Central では、このプロパティは [コネクタ・プロパティ] ダイアログの [一般] タブにある [圧縮レベル] セクションで設定します。

- **ianywhere.connector.logLevel** Mobile Link サーバ コンソールとログ・ファイルに出力されるコネクタ情報の詳細度を指定します。次の詳細度レベルがあります。

  - **1** エラー・メッセージを出力します。
  - **2** エラー・メッセージと警告メッセージを出力します。
  - **3** エラー、警告、情報の各メッセージを出力します。
  - **4** エラー、警告、情報、デバッグのメッセージを出力します。

  Sybase Central では、このプロパティは [コネクタ・プロパティ] ダイアログの [一般] タブにある [ロギング・レベル] セクションで設定します。

- **ianywhere.connector.outgoing.retry.max** QAnywhere から外部メッセージング・システムへの出力メッセージについての、デフォルトのリトライ回数です。デフォルト値は 5 です。コネクタのリトライ回数を無制限にするには 0 を指定します。

  Sybase Central では、このプロパティは [コネクタ・プロパティ] ダイアログの [プロパティ] タブで、[新規] をクリックして設定します。

- **ianywhere.connector.startupType** 起動のタイプには、automatic、manual、disabled のいずれかを指定できます。

- **webservice.http.authName** Web サービスが HTTP 認証を要求する場合に、このプロパティを使用してユーザ名を指定します。

- **webservice.http.password.e** Web サービスが HTTP 認証を要求する場合に、このプロパティを使用してパスワードを指定します。

- **webservice.http.proxy.authName** プロキシが認証を要求する場合は、このプロパティを使用してプロキシ・ユーザ名を設定します。このプロパティを指定する場合は、webservice.http.proxy.password.e プロパティも設定する必要があります。

- **webservice.http.proxy.host** HTTP プロキシ経由で Web サービスにアクセスすることが必要である場合に、このプロパティを使用してホスト名を指定します。このプロパティを指定する場合は、webservice.http.proxy.port プロパティも設定する必要があります。

- **webservice.http.proxy.password.e** プロキシが認証を要求する場合は、このプロパティを使用してプロキシ・パスワードを設定します。このプロパティを指定する場合は、webservice.http.proxy.authName プロパティも設定する必要があります。

- **webservice.http.proxy.port** プロキシ・サーバに接続するポートです。このプロパティを指定する場合は、webservice.http.proxy.host プロパティも設定する必要があります。

モバイル Web サービスの例

この例では、モバイル Web サービス・アプリケーションの作成方法を紹介します。この例では、実在しない Web サービスを使用します。参考用に設計されたものであり、実行用ではありません。

フル機能を備えた例については、*samples-dir¥QAnywhere¥MobileWebServices* にインストールされているサンプルを参照してください。*samples-dir* の詳細については、「サンプル・ディレクトリ」『SQL Anywhere サーバ - データベース管理』を参照してください。

Global Weather Web サービス

Global Weather という Web サービスがあると仮定します。次の WSDL ファイル (*globalweather.wsdl*) は、この Web サービスを記述したものです。

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:s="http://www.w3.org/2001/XMLSchema"
                  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
                  xmlns:tns="http://www.myweather.com"
                  targetNamespace="http://www.myweather.com"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

  <wsdl:types>
    <s:schema targetNamespace="http://www.myweather.com">
      <s:element name="GetWeather">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="CityName" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="CountryName" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetWeatherResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetWeatherResult" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </wsdl:types>


  <wsdl:message name="GetWeatherSoapIn">
    <wsdl:part name="parameters" element="tns:GetWeather" />
  </wsdl:message>
  <wsdl:message name="GetWeatherSoapOut">
    <wsdl:part name="parameters" element="tns:GetWeatherResponse" />
  </wsdl:message>

  <wsdl:portType name="GlobalWeatherSoap">
    <wsdl:operation name="GetWeather">
      <wsdl:input message="tns:GetWeatherSoapIn" />
      <wsdl:output message="tns:GetWeatherSoapOut" />
    </wsdl:operation>
  </wsdl:portType>


  <wsdl:binding name="GlobalWeatherSoap" type="tns:GlobalWeatherSoap">
```

```
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
    <wsdl:operation name="GetWeather">
      <soap:operation soapAction="http://www.myweather.com/GetWeather" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>

  <wsdl:service name="GlobalWeather">
    <wsdl:port name="GlobalWeatherSoap" binding="tns:GlobalWeatherSoap">
      <soap:address location="http://www.myweather.com/" />
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

プロキシ・クラスの生成

Global Weather Web サービスにアクセスするモバイル アプリケーションを作成するには、まず、QAnywhere WSDL コンパイラを実行します。コンパイラにより、Global Weather Web サービスの要求を作成するためにアプリケーションで使用できるプロキシ・クラスが生成されます。この例では、アプリケーションは Java で記述されています。

```
wsdlc -l java globalweather.wsdl
```

このコマンドは、プロキシ・クラス *GlobalWeatherSoap.java* を生成します。このプロキシ・クラスは、*com¥myweather* ディレクトリ (現在のディレクトリを基準とした相対ディレクトリ) に格納されます。このプロキシ・クラスは、アプリケーション用のサービス・バインディング・クラスです。次に、*GlobalWeatherSoap.java* の内容を示します。

```
/*
 * GlobalWeatherSoap.java
 *
 * Generated by the iAnywhere WSDL Compiler
 */

package com.myweather;

import ianywhere.qanywhere.ws.*;
import ianywhere.qanywhere.client.QABinaryMessage;
import ianywhere.qanywhere.client.QAException;

public class GlobalWeatherSoap extends ianywhere.qanywhere.ws.WSBase
{

    public GlobalWeatherSoap(String iniFile) throws WSException
    {
        super(iniFile);
        init();
    }

    public GlobalWeatherSoap() throws WSException
    {
        init();
    }

    public void init()
```



```
    {
        setServiceName("GlobalWeather");
    }


    public java.lang.String getWeather(java.lang.String cityName,
                    java.lang.String countryName)  throws QAException,WSException,WSFaultException
    {
        StringBuffer    soapRequest = new StringBuffer();
        QABinaryMessage    qaRequestMsg = null;
        String        responsePartName = "GetWeatherResult";
        java.lang.String    returnValue;

        writeSOAPHeader( soapRequest, "GetWeather", "http://www.myweather.com" );
        soapRequest.append( WSBaseTypeSerializer.serialize("CityName",cityName,"string","http://
www.w3.org/2001/XMLSchema",true,true) );
       soapRequest.append( WSBaseTypeSerializer.serialize("CountryName",countryName,"string","http://
www.w3.org/2001/XMLSchema",true,true) );
        writeSOAPFooter( soapRequest, "GetWeather" );

        qaRequestMsg = createQAMessage( soapRequest.toString(), "http://www.myweather.com/
GetWeather", "GetWeatherResponse" );

        WSResult wsResult = invokeWait( qaRequestMsg );

        returnValue = wsResult.getStringValue(responsePartName);

        return returnValue;
    }


    public WSResult asyncGetWeather(java.lang.String cityName,
                    java.lang.String countryName)  throws QAException,WSException
    {
        StringBuffer    soapRequest = new StringBuffer();
        QABinaryMessage    qaRequestMsg = null;

        writeSOAPHeader( soapRequest, "GetWeather", "http://www.myweather.com" );
        soapRequest.append( WSBaseTypeSerializer.serialize("CityName",cityName,"string","http://
www.w3.org/2001/XMLSchema",true,true) );
       soapRequest.append( WSBaseTypeSerializer.serialize("CountryName",countryName,"string","http://
www.w3.org/2001/XMLSchema",true,true) );
        writeSOAPFooter( soapRequest, "GetWeather" );

        qaRequestMsg = createQAMessage( soapRequest.toString(), "http://www.myweather.com/
GetWeather", "GetWeatherResponse" );

        WSResult wsResult = invoke( qaRequestMsg );

        return wsResult;
    }

}
```

モバイル Web サービス・アプリケーションの記述

次に、Web サービス要求の作成や結果の処理にサービス・バインディング・クラスを使用するアプリケーションを記述します。次に、2 つのアプリケーションを示します。いずれも、オフラインの状態で Web サービス要求を作成し、後で結果を処理します。

最初のアプリケーションである RequestWeather は、Global Weather Web サービスの要求を作成し、その ID を表示します。

```
import ianywhere.qanywhere.client.*;
import ianywhere.qanywhere.ws.*;
import com.myweather.GlobalWeatherSoap;

class RequestWeather
{
   public static void main( String [] args ) {
      try {
         // QAnywhere initialization
         QAManager mgr = QAManagerFactory.getInstance().createQAManager();
         mgr.open( AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT );
         mgr.start();

         // Instantiate the web service proxy
         GlobalWeatherSoap service = new GlobalWeatherSoap();
         service.setQAManager( mgr );
         service.setProperty( "WS_CONNECTOR_ADDRESS", "ianywhere.connector.globalweather¥¥" );



         // Make a request to get weather for Beijing
         WSResult r = service.asyncGetWeather( "Beijing", "China" );

         // Display the request ID so that it can be used by ShowWeather
         System.out.println( "Request ID: " + r.getRequestID() );

         // QAnywhere finalization
         mgr.stop();
         mgr.close();

      } catch( Exception exc ) {
         System.out.println( exc.getMessage() );
      }
   }
}
```

2 番目のアプリケーションである ShowWeather は、指定された要求 ID の気象情報を表示します。

```
import ianywhere.qanywhere.client.*;
import ianywhere.qanywhere.ws.*;
import com.myweather.GlobalWeatherSoap;

class ShowWeather
{
   public static void main( String [] args ) {
      try {
         // QAnywhere initialization
         QAManager mgr = QAManagerFactory.getInstance().createQAManager();
         mgr.open( AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT );
         mgr.start();

         // Instantiate the web service proxy
         GlobalWeatherSoap service = new GlobalWeatherSoap();
         service.setQAManager( mgr );

         // Get the response for the specified request ID
         WSResult r = service.getResult( args[0] );
         if( r.getStatus() == WSStatus.STATUS_RESULT_AVAILABLE ) {
         System.out.println( "The weather is " + r.getStringValue( "GetWeatherResult" ) );
         r.acknowledge();
```

```
            } else {
            System.out.println( "Response not available" );
            }


            // QAnywhere finalization
            mgr.stop();
            mgr.close();

        } catch( Exception exc ) {
            System.out.println( exc.getMessage() );
        }
        }
    }
```

アプリケーションとサービス・バインディング・クラスをコンパイルします。

```
javac -classpath ".;%sqlany10%¥java¥iawsrt.jar;%sqlany10%¥java¥qaclient.jar" com¥myweather
¥GlobalWeatherSoap.java   RequestWeather.java
javac -classpath ".;%sqlany10%¥java¥iawsrt.jar;%sqlany10%¥java¥qaclient.jar" com¥myweather
¥GlobalWeatherSoap.java   ShowWeather.java
```

QAnywhere メッセージ・ストアの作成と QAnywhere Agent の起動

作成したモバイル Web サービス・アプリケーションは、各モバイル・デバイス上にクライアント・メッセージ・ストアを要求します。サーバ・メッセージ・ストアも必要ですが、この例では、QAnywhere のサンプルのサーバ・メッセージ・ストアを使用します。

クライアント・メッセージ・ストアを作成するには、dbinit ユーティリティを使用して SQL Anywhere データベースを作成し、QAnywhere Agent を実行してクライアント・メッセージ・ストアとして設定します。

```
dbinit -i qanywhere.db
qaagent -q -si -c "dbf=qanywhere.db"
```

QAnywhere Agent を起動し、クライアント・メッセージ・ストアに接続します。次のコマンドは、1 行に入力してください。

```
qaagent
  -c "dbf=qanywhere.db;eng=qanywhere;uid=ml_qa_user;pwd=qanywhere"
  -policy automatic
```

Mobile Link サーバを起動します。この例では、サーバ・メッセージ・ストアとして QAnywhere サンプル・データベースを使用します。次のコマンドは、1 行に入力してください。

```
mlsrv10
  -m
  -zu+
  -c "dsn=QAnywhere 10 Demo;uid=ml_server;pwd=sql;start=dbsrv10
      -xs http(port=8080)"
  -v+
  -ot qanyserv.mls
```

これらのコンポーネントの詳細については、次の項を参照してください。

- 「クライアント・メッセージ・ストアの設定」 36 ページ
- 「サーバ・メッセージ・ストアの設定」 32 ページ
- 「QAnywhere Agent の実行」 38 ページ

- 「QAnywhere サーバの起動」 33 ページ

Web サービス・コネクタの作成

GetWeather Web サービスに送信される QAnywhere を受信し、メッセージが到着したときに Web サービス呼び出しを作成して、応答を送信元のクライアントに送り返すための、Web サービス・コネクタを作成する必要があります。

Sybase Central を開き、サーバ・メッセージ・ストアに接続します。Web サービス・コネクタを作成するには、[ファイル] – [New Connector] を選択します。[コネクタ] ウィザードが表示されたら、[Web サービス] を選択します。ウィザードのページで、この例の最初の方で作成したモバイル・アプリケーションに一致するように、次のプロパティを設定します。

- [コネクタ名] ページで、コネクタ名 **ianywhere.connector.globalweather** を入力する
- [通信パラメータ] ページで、URL **http://www.myweather.com/GetWeather** を入力する

第 10 章

QAnywhere のプロパティ

目次

メッセージ・ヘッダとメッセージ・プロパティ

QAnywhere メッセージは、次の要素から構成されています。

- ヘッダ
- プロパティ
- 内容

メッセージ・プロパティは、転送ルールや削除ルールで参照したり、アプリケーションで参照したりできます。

次の項ではメッセージ・ヘッダとプロパティについて説明します。また、ヘッダとプロパティを QAnywhere メッセージに設定する方法についても説明します。

注意

- メッセージ・ヘッダ、メッセージ・プロパティ、メッセージの内容を、メッセージの送信後に変更することはできません。
- メッセージ・ヘッダ、メッセージ・プロパティ、メッセージの内容は、メッセージの受信後に読み込むことができます。QAnywhere SQL API を使用している場合は、コミットまたはロールバックが発生すると、メッセージ・ヘッダ、メッセージ・プロパティ、メッセージの内容が読み込み不可能になります。
- 内容は、すべての API で確認またはコミットが行われると読み込み不可能になります。

メッセージ・ヘッダ

QAnywhere のすべてのメッセージで、同じヘッダ・フィールド・セットがサポートされます。ヘッダ・フィールドには、メッセージを識別してルートを指定するために、クライアントとプロバイダの両方で使用される値が設定されます。

次のメッセージ・ヘッダが事前に定義されています。メッセージ・ヘッダの使い方は、使用するクライアント・アプリケーションの種類によって変わります。

- **メッセージ ID**　読み込み専用。新規メッセージのメッセージ ID。このヘッダの値は、メッセージの送信後にのみ設定されます。次の項を参照してください。
  - .NET API：「MessageID プロパティ」 347 ページ
  - C++ API：「getMessageID 関数」 502 ページと「setMessageID 関数」 512 ページ
  - Java API：「getMessageID メソッド」 613 ページ
  - SQL API：「ml_qa_createmessage」 707 ページと「ml_qa_getmessage」 707 ページ
- **メッセージの作成時刻のタイムスタンプ**　読み込み専用。メッセージの Timestamp ヘッダ・フィールドには、メッセージが作成された時刻が格納されます。これは、協定世界時 (UTC: Coordinated Universal Time) です。この時刻は、メッセージが実際に送信された時刻ではないので注意してください。メッセージの実際の送信時刻は、トランザクションの進行状況やクライアント側のキューに登録されているその他のメッセージの影響で、作成時刻よりも遅れる可能

性があります。このヘッダは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。次の項を参照してください。

- .NET API：「Timestamp プロパティ」 348 ページ
- C++ API：「getTimestamp 関数」 506 ページと「setTimestamp 関数」 515 ページ
- Java API：「getTimestamp メソッド」 617 ページ
- SQL API：「ml_qa_gettimestamp」 681 ページ

- **返信先アドレス**　読み込み／書き込み。VARCHAR(128) 型の返信アドレス。返信アドレスが存在しない場合は NULL。このヘッダは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。次の項を参照してください。

  - .NET API：「ReplyToAddress プロパティ」 348 ページ
  - C++ API：「getReplyToAddress 関数」 504 ページと「setReplyToAddress 関数」 513 ページ
  - Java API：「getReplyToAddress メソッド」 615 ページと「setReplyToAddress メソッド」 623 ページ
  - SQL API：「ml_qa_getreplytoaddress」 681 ページと「ml_qa_setreplytoaddress」 685 ページ

- **メッセージ・アドレス**　読み込み／書き込み。VARCHAR(128) 型の QAnywhere のメッセージ・アドレス。QAnywhere のメッセージ・アドレスは *id¥queue-name* の形式を取ります。このヘッダは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。次の項を参照してください。

  - .NET API：「Address プロパティ」 345 ページ
  - C++ API：「getAddress 関数」 497 ページと「setAddress 関数」 508 ページ
  - Java API：「getAddress メソッド」 608 ページと「setAddress メソッド」 618 ページ
  - SQL API：「ml_qa_getaddress」 676 ページと「ml_qa_getaddress」 676 ページ

- **メッセージの再配信のステータス**　読み込み専用。再配信のステータスを示す BIT 型の値。1 はメッセージが再配信中であることを示し、0 は再配信中ではないことを示します。

  受信されたが受信確認されていないメッセージは、再配信される場合があります。たとえば、メッセージは受信されたものの、メッセージを受信したアプリケーションがメッセージの内容の処理を完了する前にクラッシュした場合などです。このような場合には、メッセージは再配信とマーク付けされて、メッセージの処理が完了していない可能性があることを示す警告が受信側に送信されます。

  たとえば、次のような手順を経てメッセージが受信されるとします。

  1. 非トランザクション志向の QAnywhere Manager を使用するアプリケーションがメッセージを受信します。
  2. このアプリケーションは、T1 というデータベース・テーブルにメッセージの内容とメッセージ ID を書き込み、変更をコミットします。
  3. アプリケーションがメッセージの受信を確認します。

手順 1 と 2、または手順 2 と 3 の間でアプリケーションに障害が発生した場合は、アプリケーションの再起動時にメッセージが再配信されます。

手順 1 と 2 の間で障害が発生した場合は、手順 2 と 3 を実行してメッセージを再配信してください。手順 2 と 3 の間で障害が発生した場合は、メッセージはすでに処理されているので、メッセージの確認のみ必要です。

アプリケーションの障害時に何が起きたかを特定するには、アプリケーションで ml_qa_getredelivered を呼び出して、メッセージがすでに再配信されていたかどうかをチェックします。テーブル T1 で検索する必要があるのは、再配信されるメッセージだけです。アプリケーションで障害が発生することはほとんどないので、この方法は、受信したメッセージのメッセージ ID にアプリケーションからアクセスして、メッセージがテーブル T1 に格納されているかどうかをチェックするよりも効率的です。

このヘッダは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

次の項を参照してください。

- .NET API：「Redelivered プロパティ」 347 ページ
- C++ API：「getRedelivered 関数」 503 ページと「setRedelivered 関数」 513 ページ
- Java API：「getRedelivered メソッド」 615 ページ
- SQL API：「ml_qa_getredelivered」 679 ページ

- **メッセージの有効期限**　読み取り専用。ただし SQL API の場合は読み取り／書き込み。TIMESTAMP 型の有効期限。有効期限がない場合は NULL が返されます。メッセージは、意図した受信者によって指定の時間内に受信されないと期限切れになります。期限切れになったメッセージは、QAnywhere のデフォルトの削除ルールを使って削除できます。このヘッダは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。次の項を参照してください。

  - .NET API：「Expiration プロパティ」 346 ページ
  - C++ API：「getExpiration 関数」 499 ページ
  - Java API：「getExpiration メソッド」 610 ページ
  - SQL API：「ml_qa_getexpiration」 677 ページと「ml_qa_setexpiration」 682 ページ

- **メッセージの優先度**　読み込み／書き込み。QAnywhere API では、10 レベルの優先度が定義されています。0 が最低の優先度、9 が最高の優先度を表します。クライアントは、優先度 0 ー 4 を通常のメッセージ、優先度 5 ー 9 を緊急度の高いメッセージとみなす必要があります。このヘッダは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。次の項を参照してください。

  - .NET API：「Priority プロパティ」 347 ページ
  - C++ API：「getPriority 関数」 503 ページ
  - Java API：「getPriority メソッド」 613 ページ
  - SQL API：「ml_qa_getpriority」 678 ページ

- **どのメッセージへの返信であるかを示すメッセージ ID**　読み込み／書き込み。VARCHAR(128) 型の in-reply-to ID。クライアントは、In-Reply-To ID ヘッダ・フィールドを使用してメッセージ間リンクを設定できます。これは、応答メッセージをそれに対応する要求メッセージとリンクさせる場合によく使用されます。in-reply-to ID は、返信対象メッセージの ID です。このヘッダは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。次の項を参照してください。

  - .NET API：「InReplyToID プロパティ」 346 ページ
  - C++ API：「getInReplyToID 関数」 501 ページ
  - Java API：「getInReplyToID メソッド」 611 ページ
  - SQL API：「ml_qa_getinreplytoid」 677 ページ

一部のメッセージ・ヘッダは転送ルールで使用できます。「ルール・エンジンで定義される変数」 248 ページを参照してください。

参照

- .NET API：「QAMessage のメンバ」 344 ページ
- C++ API：「QAMessage クラス」 494 ページ
- Java API：「QAMessage インタフェース」 606 ページ
- SQL API：「メッセージ・ヘッダ」 676 ページ

メッセージ・プロパティ

各メッセージには、アプリケーションで定義されたプロパティ値をサポートする組み込みの機能があります。これらのメッセージ・プロパティを使用すると、アプリケーションで定義されたメッセージ・フィルタリングを実行できます。

メッセージ・プロパティは名前と値の組み合わせです。オプションでメッセージに挿入して、構造を示すことができます。たとえば、.NET API では、定数 MessageProperties.ORIGINATOR で識別される、事前に定義されたメッセージ・プロパティ ias_Originator は、メッセージを送信したメッセージ・ストアの ID を指定します。メッセージ・プロパティを転送ルールで使用して、メッセージを転送することが妥当であるかどうかを判別することができます。

メッセージ・プロパティには、次の 2 種類があります。

- **事前に定義されたメッセージ・プロパティ**　このメッセージ・プロパティには、必ず ias_ または IAS_ のプレフィクスが付けられています。
- **カスタムのメッセージ・プロパティ**　ユーザが定義したメッセージ・プロパティです。このプロパティに ias_ または IAS_ のプレフィクスを付けることはできません。

事前定義またはカスタムのプロパティのどちらの場合も、get メソッドと set メソッドを使用してメッセージ・ストア・プロパティにアクセスし、プロパティの名前を最初のパラメータとして渡します。

「メッセージ・プロパティの管理」 226 ページを参照してください。

事前に定義されたメッセージ・プロパティ

利便性のためにいくつかのメッセージ・プロパティは事前に定義されています。事前に定義されたプロパティは読み込み可能ですが、設定はできません。事前に定義されたメッセージ・プロパティは次のとおりです。

- **ias_Adapters** ネットワーク・ステータス通知メッセージのプロパティ。Mobile Link サーバへの接続で使用できるネットワーク・アダプタのリストです。これは文字列のリストで、各文字列は縦線で区切られます。
- **ias_DeliveryCount** 整数。メッセージを配信するために、これまでに実行された試行回数を示します。
- **ias_MessageType** 整数。メッセージのタイプを示します。次のいずれかのメッセージ・タイプを取ります。

値	メッセージ・タイプ	説明
0	REGULAR	メッセージに ias_MessageType プロパティが設定されていない場合は、標準メッセージになる。
13	PUSH_NOTIFICATION	Push 通知がサーバから受信されると、PUSH_NOTIFICATION タイプのメッセージがシステム・キューに送信される。「Push 通知の通知」 59 ページを参照してください。
14	NETWORK_STATUS_NOTIFICATION	ネットワーク・ステータスが変化すると、このタイプのメッセージがシステム・キューに送信される。「ネットワーク・ステータス通知」 59 ページを参照してください。

- **ias_RASNames** 文字列。ネットワーク・ステータス通知メッセージのプロパティ。Mobile Link サーバへの接続で使用できる RAS エントリ名のリストです。各文字列は縦棒で区切られます。
- **ias_NetworkStatus** 整数。ネットワーク・ステータス通知メッセージのプロパティ。ネットワーク接続のステータスです。値 1 は 接続済み、0 はそれ以外を意味します。
- **ias_Originator** 文字列。メッセージの発信者のメッセージ・ストア ID。
- **ias_Status** 整数。メッセージの現在のステータス。このプロパティは SQL API ではサポートされていません。次のいずれかの値を取ります。

ステータス・コード	説明
1	保留 – メッセージは送信されたが、まだ受信されていない。
10	受信中 – メッセージは受信中である、または受信されたがまだ確認されていない。
20	最終 – 最終段階のメッセージ・ステータス。

ステータス・コード	説明
30	失効 – 有効期限が切れる前にメッセージが受信されなかった。
40	キャンセル済み – メッセージはキャンセルされた。
50	受信不可 – メッセージに間違った形式があるか、失敗した試行回数が多すぎて配信されなかった。
60	受信済み ー メッセージは受信され、確認済み。

ステータス値には定数があります。次の項を参照してください。

- .NET API：「StatusCodes 列挙」 370 ページ
- C++ API：「StatusCodes クラス」 527 ページ
- Java API：「StatusCodes インタフェース」 636 ページ

- **ias_StatusTime**　メッセージが現在のステータスになった時刻。タイムスタンプには、プラットフォームに合わせたフォーマットが使用されます。これはローカル時間です。C++ API では、Windows と PocketPC プラットフォームのタイムスタンプは SYSTEMTIME フォーマットです。これが FILETIME フォーマットに変換されて、qa_long 値にコピーされます。このプロパティは SQL API ではサポートされていません。

API	このプロパティからの戻り値
.NET	DateTime
C++	文字列
Java	java.util.Date オブジェクト

メッセージ・プロパティの定数

.NET、C++、Java 用の QAnywhere API には、メッセージ・プロパティを指定するための定数があります。次の項を参照してください。

- .NET API：「MessageProperties のメンバ」 265 ページ
- C++ API：「MessageProperties クラス」 424 ページ
- Java API：「MessageProperties インタフェース」 533 ページ

カスタムのメッセージ・プロパティ

QAnywhere では、C++、Java、または .NET の API を使用してメッセージ・プロパティを定義できます。カスタムのメッセージ・プロパティでは、名前と値を組み合わせてオブジェクトに関連付けることができます。次に例を示します。

```
msg.SetStringProperty("Product", "widget");
msg.SetFloatProperty("Price",1.00);
msg.SetIntProperty("Quantity",10);
```


メッセージ・プロパティ名では大文字と小文字を区別しません。文字、数字、アンダースコアを使用できますが、先頭は文字でなければなりません。次の名前は予約語なので、メッセージ・プロパティ名として使用することはできません。

- NULL
- TRUE
- FALSE
- NOT
- AND
- OR
- BETWEEN
- LIKE
- IN
- IS
- ESCAPE
- **ias_** で始まるすべての名前

メッセージ・プロパティの管理

メッセージ・プロパティの管理には、次の QAMessage メソッドを使用できます。

> カスタム・プロパティは取得と設定はできますが、事前に定義されたプロパティは取得するだけで設定はできません。

メッセージ・プロパティ管理用メソッド (.NET)

- Object GetProperty(String name)
- void SetProperty(String name, Object value)
- boolean GetBooleanProperty(String name)
- void SetBooleanProperty(String name, boolean value)
- byte GetByteProperty(String name)
- void SetByteProperty(String name, byte value)
- short GetShortProperty(String name)
- void SetShortProperty(String name, short value)
- int GetIntProperty(String name)
- void SetIntProperty(String name, int value)
- long GetLongProperty(String name)
- void SetLongProperty(String name, long value)
- float GetFloatProperty(String name)
- void SetFloatProperty(String name, float value)
- double GetDoubleProperty(String name)
- void SetDoubleProperty(String name, double value)
- String GetStringProperty(String name)
- void SetStringProperty(String name, String value)
- IEnumerator GetPropertyNames()
- void ClearProperties()

- PropertyType GetPropertyType(string propName)
- bool PropertyExists(string propName)

「QAMessage インタフェース」 343 ページを参照してください。

メッセージ・プロパティ管理用メソッド (C++)

- qa_bool getBooleanProperty(qa_const_string name, qa_bool * value)
- qa_bool setBooleanProperty(qa_const_string name, qa_bool value)
- qa_bool getByteProperty(qa_const_string name, qa_byte * value)
- qa_bool setByteProperty(qa_const_string name, qa_byte value)
- qa_bool getShortProperty(qa_const_string name, qa_short * value)
- qa_bool setShortProperty(qa_const_string name, qa_short value)
- qa_bool getIntProperty(qa_const_string name, qa_int * value)
- qa_bool setIntProperty(qa_const_string name, qa_int value)
- qa_bool getLongProperty(qa_const_string name, qa_long * value)
- qa_bool setLongProperty(qa_const_string name, qa_long value)
- qa_bool getFloatProperty(qa_const_string name, qa_float * value)
- qa_bool setFloatProperty(qa_const_string name, qa_float value)
- qa_bool getDoubleProperty(qa_const_string name, qa_double * value)
- qa_bool setDoubleProperty(qa_const_string name, qa_double value)
- qa_int getStringProperty(qa_const_string name, qa_string value, qa_int len)
- qa_bool setStringProperty(qa_const_string name, qa_const_string value)
- void QAMessage::clearProperties()
- qa_short QAMessage::getPropertyType(qa_const_string name)
- qa_bool QAMessage::propertyExists(qa_const_string name)
-

「QAMessage クラス」 494 ページを参照してください。

メッセージ・プロパティ管理用メソッド (Java)

- void clearProperties()
- boolean getBooleanProperty(String name)
- void setBooleanProperty(String name, boolean value)
- byte getByteProperty(String name)
- void setByteProperty(String name, byte value)
- double getDoubleProperty(String name)
- void setDoubleProperty(String name, double value)
- java.util.Date getExpiration() void setFloatProperty(String name, float value)
- float getFloatProperty(String name)
- int getIntProperty(String name)
- void setIntProperty(String name, int value)
- long getLongProperty(String name)
- void setLongProperty(String name, long value)
- Object getProperty(String name)
- void setProperty(String name, Object value)
- java.util.Enumeration getPropertyNames()
- short getPropertyType(String name)

- short getShortProperty(String name)
- void setShortProperty(String name, short value)
- String getStringProperty(String name)
- void setStringProperty(String name, String value)
- boolean propertyExists(String name)

「QAMessage インタフェース」 606 ページを参照してください。

メッセージ・プロパティ管理用 SQL ストアド・プロシージャ

- ml_qa_getbooleanproperty
- ml_qa_getbyteproperty
- ml_qa_getdoubleproperty
- ml_qa_getfloatproperty
- ml_qa_getintproperty
- ml_qa_getlongproperty
- ml_qa_getpropertynames
- ml_qa_getshortproperty
- ml_qa_getstringproperty
- ml_qa_setbooleanproperty
- ml_qa_setbyteproperty
- ml_qa_setdoubleproperty
- ml_qa_setfloatproperty
- ml_qa_setfloatproperty
- ml_qa_setintproperty
- ml_qa_setlongproperty
- ml_qa_setshortproperty
- ml_qa_setstringproperty

「メッセージ・プロパティ」 686 ページを参照してください。

例

```
// C++ example.
QAManagerFactory factory;
QAManager * mgr = factory->createQAManager( NULL );
mgr->open(AcknowledgementMode::EXPLICIT_ACKNOWLEDGEMENT);
QAMessage * msg = mgr->createTextMessage();
msg->setStringProperty( "tm_Subject", "Some message subject." );
mgr->putMessage( "myqueue", mgr );

// C# example.
QAManager mgr = QAManagerFactory.Instance.CreateQAManager(null);
mgr.Open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
QAMessage msg = mgr.CreateTextMessage();
msg.SetStringProperty( "tm_Subject", "Some message subject." );
mgr.PutMessage( "myqueue", msg );

// Java example
QAManager mgr = QAManagerFactory.getInstance().createQAManager(null);
mgr.open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
QAMessage msg = mgr.createTextMessage();
msg.setStringProperty("tm_Subject", "Some message subject.");
mgr.putMessage("myqueue", mgr);
```

```
-- SQL example
begin
    DECLARE @msgid VARCHAR(128);
    SET @msgid = ml_qa_createmessage();
    CALL ml_qa_setfloatproperty( @msgid, 'myfloatproperty1', -1.3e-5 );
    CALL ml_qa_setfloatproperty( @msgid, 'myfloatproperty2', 1.3e5 );
    CALL ml_qa_putmessage( @msgid, 'clientid¥queuename' );
    COMMIT;
end
```

クライアント・メッセージ・ストア・プロパティ

クライアント・メッセージ・ストア・プロパティには、次の 2 種類があります。

- **事前に定義されたメッセージ・ストア・プロパティ**　このメッセージ・ストア・プロパティには、必ず ias_ または IAS_ のプレフィクスが付けられています。
- **カスタムのメッセージ・ストア・プロパティ**　ユーザが定義したメッセージ・ストア・プロパティです。このプロパティに ias_ または IAS_ のプレフィクスを付けることはできません。

事前定義またはカスタムのプロパティのどちらの場合も、適切なクラスで定義された get メソッドと set メソッドを使用してクライアント・メッセージ・ストア・プロパティにアクセスし、プロパティの名前を最初のパラメータとして渡します。

「クライアント・メッセージ・ストア・プロパティの管理」 234 ページを参照してください。

メッセージ・ストア・プロパティは、転送ルール、削除ルール、メッセージ・セレクタでも使用できます。次の項を参照してください。

- 「QAnywhere 転送ルールと削除ルール」 241 ページ

事前に定義されたクライアント・メッセージ・ストア・プロパティ

利便性のために多数のクライアント・メッセージ・ストア・プロパティが事前に定義されています。事前に定義されたメッセージ・ストア・プロパティは次のとおりです。

- **ias_Adapters**　Mobile Link サーバへの接続で使用できるネットワーク・アダプタのリスト。これは文字列のリストで、各文字列は縦線で区切られます。
- **ias_MaxDeliveryAttempts**　このプロパティが定義されている場合は、メッセージのステータスが「受信不可」に設定される前に、受信確認されていないメッセージを配信できる最大回数を示します。デフォルトでは、このプロパティは定義されていません。これは値 -1 と同等で、クライアント・ライブラリは受信確認されていないメッセージの配信を無制限に試行します。
- **ias_MaxDownloadSize**　ダウンロードのインクリメント・サイズ。QAnywhere では、デフォルトで最大ダウンロード・サイズ -1 (最大値なし) が使用されます。ただし、ダウンロード・サイズの設定に関係なく、QAnywhere でメッセージは分割されません。このプロパティは、qaagent -idl オプションで設定します。「-idl オプション」 164 ページを参照してください。
- **ias_MaxUploadSize**　アップロードのインクリメント・サイズ。デフォルトでは、QAnywhere は 256K のインクリメント・サイズでメッセージをアップロードします。ただし、QAnywhere は設定されているアップロード・サイズに関係なく、インクリメントごとに少なくとも 1 つのメッセージを分割せずに送信します。このプロパティは、qaagent -iu オプションで設定します。「-iu オプション」 165 ページを参照してください。
- **ias_Network**　現在使用されているネットワークに関する情報。このプロパティは読み込み可能ですが、設定はできません。ias_Network は特殊なプロパティです。このプロパティに

は、デバイスで使用されている現在のネットワークに関する情報を提供する多数の組み込み属性が定義されています。次の属性は、QAnywhere によって自動的に設定されます。

- **ias_Network.Adapter**　ネットワーク・カードを使用している場合は、現在のネットワーク・カード名 (Adapter 属性に割り当てられたネットワーク・カードの名前は、そのネットワークとの接続が確立されたとき、Agent ウィンドウに表示されます)。
- **ias_Network.RAS**　現在の RAS エントリ名。
- **ias_Network.IP**　デバイスに割り当てられた現在の IP アドレス。
- **ias_Network.MAC**　現在使用しているネットワーク・カードの MAC アドレス。

- **ias_RASNames**　文字列。Mobile Link サーバへの接続で使用できる RAS エントリ名のリスト。各文字列は縦棒で区切られます。
- **ias_StoreID**　メッセージ・ストア ID。
- **ias_StoreInitialized**　このメッセージ・ストアが QAnywhere メッセージング用に正常に初期化された場合は True。それ以外の場合は False。

  「-si オプション」 181 ページを参照してください。
- **ias_StoreVersion**　QAnywhere で定義されている、このメッセージ・ストアのバージョン番号。

事前に定義されたメッセージ・プロパティの詳細については、次の項を参照してください。

- C++ API：「MessageStoreProperties クラス」 432 ページ
- .NET API：「MessageStoreProperties クラス」 273 ページ
- Java API：「MessageStoreProperties インタフェース」 539 ページ
- SQL API：「メッセージ・ストア・プロパティ」 705 ページ

カスタムのクライアント・メッセージ・ストア・プロパティ

QAnywhere では、QAnywhere C++、Java、SQL、または .NET API を使用して、ユーザ独自のクライアント・メッセージ・ストア・プロパティを定義できます。これらのプロパティは、同じメッセージ・ストアに接続されたアプリケーション間で共有されます。また、サーバ・メッセージ・ストアと同期されるため、このクライアント用のサーバ側の転送ルールでも使用できます。

クライアント・メッセージ・ストア・プロパティ名では大文字と小文字を区別しません。文字、数字、アンダースコアを使用できますが、先頭は文字でなければなりません。次の名前は予約語なので、メッセージ・ストア・プロパティ名として使用することはできません。

- NULL
- TRUE
- FALSE
- NOT
- AND
- OR

- BETWEEN
- LIKE
- IN
- IS
- ESCAPE
- **ias_** で始まるすべての名前

カスタムのクライアント・メッセージ・ストア・プロパティ属性の使用

クライアント・メッセージ・ストア・プロパティには、ユーザが属性を定義できます。属性を定義するには、プロパティ名の後にドットを付け、その後に属性名を追加します。この機能の主な目的は、ユーザのネットワーク情報を転送ルールで使用できるようにすることです。

例

次に、カスタムのクライアント・メッセージ・ストア・プロパティ属性を設定する簡単な例を示します。この例の Object プロパティは、Shape と Color の 2 つの属性を持ちます。Sharp 属性の値は Round、Color 属性の値は Blue です。

```
// C++ example.
mgr->setStringStoreProperty( “Object.Shape”, “Round” );
mgr->setStringStoreProperty( “Object.Color”, “Blue” );

// C# example.
mgr.SetStoreStringProperty( “Object.Shape”, “Round” );
mgr.SetStringStoreProperty( “Object.Color”, “Blue” );

// Java example
mgr.setStringStoreProperty( "Object.Shape", "Round" );
mgr.setStringStoreProperty( "Object.Color", "Blue" );

-- SQL example
BEGIN
   CALL ml_qa_setstoreproperty( 'Object.Shape', 'Round' );
   CALL ml_qa_setstoreproperty( 'Object.Color', 'Blue' );
   COMMIT;
END
```

すべてのクライアント・メッセージ・ストア・プロパティには、最初は値を持たない Type 属性があります。Type 属性の値は、別のプロパティの名前でなければなりません。プロパティの Type 属性を設定すると、そのプロパティは Type 属性の値に割り当てられているプロパティの属性を継承します。次の例の Object プロパティは、Circle プロパティの属性を継承しています。したがって、Object.Shape の値は Round、Object.Color の値は Blue になります。

```
// C++ example
QAManager    qa_manager;
qa_manager->setStoreStringProperty( “Circle.Shape”, “Round” );
qa_manager->setStoreStringProperty( “Circle.Color”, “Blue” );
qa_manager->setStoreStringProperty( “Object.Type”, “Circle” );

// C# example
QAManager    qa_manager;
qa_manager.SetStringStoreProperty( “Circle.Shape”, “Round” );
qa_manager.SetStringStoreProperty( “Circle.Color”, “Blue” );
qa_manager.SetStringStoreProperty( “Object.Type”, “Circle” );
```

```
// Java example
QAManager    qa_manager;
qa_manager.setStringStoreProperty( "Circle.Shape", "Round" );
qa_manager.setStringStoreProperty( "Circle.Color", "Blue" );
qa_manager.setStringStoreProperty( "Object.Type", "Circle" );

-- SQL example
BEGIN
   CALL ml_qa_setstoreproperty( 'Circle.Shape', 'Round' );
   CALL ml_qa_setstoreproperty( 'Circle.Color', 'Blue' );
   CALL ml_qa_setstoreproperty( 'Object.Type', 'Circle');
   COMMIT;
END
```

例

次の C# の例は、メッセージ・ストア・プロパティを使用して、ユーザのネットワーク情報を転送ルールに指定する方法を示しています。

LAN、無線 LAN、無線 WAN の 3 つの接続オプションが用意されている Windows ラップトップがあるとします。LAN 経由でネットワークにアクセスするには、My LAN Card という名前のネットワーク・カードを使用します。無線 LAN 経由でネットワークにアクセスするには、My Wireless LAN Card という名前のネットワーク・カードを使用します。無線 WAN 経由でネットワークにアクセスするには、My Wireless WAN Card という名前のネットワーク・カードを使用します。

ここでは、LAN または無線 LAN を使用して接続している場合は、サーバにすべてのメッセージを送信し、無線 WAN を使用して接続している場合は、優先度の高いメッセージだけを送信するアプリケーションを開発します。優先度の高いメッセージとは、優先度が 7 以上のメッセージであると定義します。

まず、使用しているネットワーク・アダプタの名前を検索します。ネットワーク・アダプタの名前は、カードを装着し、ドライバをインストールした時点で決まります。特定のネットワーク・カードの名前を見つけるには、そのアダプタを介してネットワークに接続し、-vn オプションを指定して qaagent を実行します。QAnywhere Agent によって、次のようにネットワーク・アダプタ名が表示されます。

"Listener thread received message '[netstat] *network-adapter-name* !...'

次に、LAN、WLAN、WWAN の 3 種類のネットワーク・タイプに対応するクライアント・メッセージ・ストア・プロパティを定義し、各プロパティに Cost 属性を割り当てます。Cost 属性は 1 ー 3 の値をとり、そのネットワークを使用したときに発生するコストを表します。1 が最も低いコストを表します。

```
QAManager    qa_manager;
qa_manager.SetStoreProperty( “LAN.Cost”, “1” );
qa_manager.SetStoreProperty( “WLAN.Cost”, “2” );
qa_manager.SetStoreProperty( “WWAN.Cost”, “3” );
```

次に、使用するネットワーク・カードごとに 1 つずつ、全部で 3 つのクライアント・メッセージ・ストア・プロパティを定義します。このプロパティ名はネットワーク・カード名と一致させます。ネットワークの種類を Type 属性に割り当てることで、各プロパティに適切なネットワーク種別を割り当てます。したがって、各プロパティは、それぞれのネットワークの種類の属性を継承します。

```
QAManager    qa_manager;
qa_manager.SetStoreProperty( “My LAN Card.Type”, “LAN” );
```



```
qa_manager.SetStoreProperty( “My Wireless LAN Card.Type”, “WLAN” );
qa_manager.SetStoreProperty( “My Wireless WAN Card.Type”, “WWAN” );
```

ネットワーク接続が確立されると、QAnywhere は、ias_Network プロパティの Adapter 属性に、My LAN Card、My Wireless LAN Card、または My Wireless WAN Card のいずれかを、使用しているネットワークに応じて自動的に設定します。同様に、ias_Network プロパティの Type 属性に、My LAN Card、My Wireless LAN Card、または My Wireless WAN Card のいずれかを自動的に設定して、使用しているネットワークの属性を ias_Network プロパティが継承するようにします。

最後に、次の転送ルールを作成します。

```
automatic=ias_Network.Cost < 3 or ias_Priority >= 7
```

転送ルールの詳細については、「QAnywhere 転送ルールと削除ルール」 241 ページを参照してください。

クライアント・メッセージ・ストア・プロパティの列挙

QAnywhere .NET、C++、Java API では、事前定義とカスタムのメッセージ・ストア・プロパティを列挙できます。

.NET の例

「GetStorePropertyNames メソッド」 321 ページを参照してください。

```
// qaManager is a QAManager instance.
IEnumerator propertyNames = qaManager.GetStorePropertyNames();
```

C++ の例

「beginEnumStorePropertyNames 関数」 462 ページを参照してください。

```
// qaManager is a QAManager instance.
qa_store_property_enum_handle handle = qaManager->beginEnumStorePropertyNames();
qa_char propertyName[256];
if( handle != qa_null ) {
  while( qaManager->nextStorePropertyName( handle, propertyName, 255 ) != -1 ) {
      // Do something with the message store property name.
  }
  // Message store properties cannot be set after
  // the beginEnumStorePropertyNames call
  // and before the endEnumStorePropertyNames call.
  qaManager->endEnumStorePropertyNames(handle);
}
```

Java の例

「getStorePropertyNames メソッド」 590 ページを参照してください。

```
// qaManager is a QAManager instance.
Enumeration propertyNames = qaManager.getStorePropertyNames();
```

クライアント・メッセージ・ストア・プロパティの管理

クライアント・メッセージ・ストアのプロパティは、クライアント・メッセージ・ストアごとにクライアント・アプリケーションで設定できます。

「アプリケーションでのクライアント・メッセージ・ストア・プロパティの管理」 235 ページを参照してください。

クライアント・メッセージ・ストアのプロパティを転送ルールで使用して、クライアントに送信されるメッセージをフィルタできます。また、削除ルールで使用して、追加するメッセージを決定することもできます。

「QAnywhere 転送ルールと削除ルール」 241 ページを参照してください。

クライアント・メッセージ・ストアのプロパティをサーバ管理メッセージで指定し、サーバ・メッセージ・ストアに格納することもできます。

「サーバ管理要求の概要」 100 ページを参照してください。

アプリケーションでのクライアント・メッセージ・ストア・プロパティの管理

次の QAManagerBase メソッドを使用すると、クライアント・メッセージ・ストア・プロパティを取得および設定できます。

クライアント・メッセージ・ストア・プロパティ管理用メソッド (C++)

- qa_bool getBooleanStoreProperty(qa_const_string name, qa_bool * value)
- qa_bool setBooleanStoreProperty(qa_const_string name, qa_bool value)
- qa_bool getByteStoreProperty(qa_const_string name, qa_byte * value)
- qa_bool setByteStoreProperty(qa_const_string name, qa_byte value)
- qa_bool getShortStoreProperty(qa_const_string name, qa_short * value)
- qa_bool setShortStoreProperty(qa_const_string name, qa_short value)
- qa_bool getIntStoreProperty(qa_const_string name, qa_int * value)
- qa_bool setIntStoreProperty(qa_const_string name, qa_int value)
- qa_bool getLongStoreProperty(qa_const_string name, qa_long * value)
- qa_bool setLongStoreProperty(qa_const_string name, qa_long value)
- qa_bool getFloatStoreProperty(qa_const_string name, qa_float * value)
- qa_bool setFloatStoreProperty(qa_const_string name, qa_float value)
- qa_bool getDoubleStoreProperty(qa_const_string name, qa_double * value)
- qa_bool setDoubleStoreProperty(qa_const_string name, qa_double value)
- qa_int getStringStoreProperty(qa_const_string name, qa_string value, qa_int len)
- qa_bool setStringStoreProperty(qa_const_string name, qa_const_string value)
- qa_store_property_enum_handle QAManagerBase::beginEnumStorePropertyNames()
- virtual qa_int QAManagerBase::nextStorePropertyName(qa_store_property_enum_handle h, qa_string buffer, qa_int bufferLen)
- virtual void QAManagerBase::endEnumStorePropertyNames(qa_store_property_enum_handle h)

「QAManagerBase クラス」 460 ページを参照してください。

クライアント・メッセージ・ストア・プロパティ管理用メソッド (C#)

- Object GetStoreProperty(String name)
- void SetStoreProperty(String name, Object value)
- boolean GetBooleanStoreProperty(String name)
- void SetBooleanStoreProperty(String name, boolean value)

- byte GetByteStoreProperty(String name)
- void SetByteStoreProperty(String name, byte value)
- short GetShortStoreProperty(String name)
- void SetShortStoreProperty(String name, short value)
- int GetIntStoreProperty(String name)
- void SetIntStoreProperty(String name, int value)
- long GetLongStoreProperty(String name)
- void SetLongStoreProperty(String name, long value)
- float GetFloatStoreProperty(String name)
- void SetFloatStoreProperty(String name, float value)
- double GetDoubleStoreProperty(String name)
- void SetDoubleStoreProperty(String name, double value)
- String GetStringStoreProperty(String name)
- void SetStringStoreProperty(String name, String value)
- IEnumerator GetStorePropertyNames()

「QAManagerBase インタフェース」 297 ページを参照してください。

クライアント・メッセージ・ストア・プロパティ管理用メソッド (Java)

- boolean getBooleanStoreProperty(String name)
- void setBooleanStoreProperty(String name, boolean value)
- byte getByteStoreProperty(String name)
- void setByteStoreProperty(String name, byte value)
- double getDoubleStoreProperty(String name)
- void setDoubleStoreProperty(String name, double value)
- float getFloatStoreProperty(String name)
- void setFloatStoreProperty(String name, float value)
- int getIntStoreProperty(String name)
- void setIntStoreProperty(String name, int value)
- long getLongStoreProperty(String name)
- void setLongStoreProperty(String name, long value)
- short getShortStoreProperty(String name)
- void setShortStoreProperty(String name, short value)
- void setStringStoreProperty(String name, String value)
- String getStringStoreProperty(String name)
- java.util.Enumeration getStorePropertyNames()

「QAManagerBase インタフェース」 570 ページを参照してください。

クライアント・メッセージ・ストア・プロパティ管理用 SQL ストアド・プロシージャ

- ml_qa_getstoreproperty
- ml_qa_setstoreproperty

「メッセージ・ストア・プロパティ」 705 ページを参照してください。

サーバ・プロパティ

サーバ・プロパティは、Sybase Central またはサーバ管理要求を使用して設定できます。どちらの場合も、サーバ・プロパティはデータベースに格納されます。次の項を参照してください。

- 「サーバ管理要求でのサーバ・プロパティの設定」 122 ページ
- 「Sybase Central でのサーバ・プロパティの設定」 239 ページ

サーバ・プロパティ

- **ianywhere.qa.server.autoRulesEvaluationPeriod** メッセージの転送ルールと持続性ルールを含む、ルールの評価時間 (ミリ秒単位)。一般に、ルールは、サーバ・ストアへのメッセージの転送中に動的に評価されるため、ルール評価時間は、時間の影響を大きく受けるルールのみが対象です。デフォルト値は **60000** (1 分) です。

- **ianywhere.qa.compressionLevel** QAnywhere コネクタが受信する各メッセージに適用される、デフォルトの圧縮量。圧縮は 0 ー 9 の範囲の整数で表されます。0 は圧縮がない状態で、9 は最も圧縮された状態です。デフォルトは **0** です。

  コネクタの圧縮レベルをコネクタ・プロパティ・ファイルにも設定した場合は、その設定が、該当のコネクタに対して優先されます。「JMS コネクタ・プロパティ」 141 ページを参照してください。

- **ianywhere.qa.server.connectorPropertiesFiles**

  推奨されなくなった機能
  Sybase Central の に置き換えられています。

  QAnywhere コネクタの設定を、JMS などの外部のメッセージ・システムに対して指定する、1 つ以上のファイルのリスト。デフォルトは、コネクタなしです。

  「JMS コネクタ」 135 ページを参照してください。

- **ianywhere.qa.server.disableNotifications** このプロパティを true に設定すると、保留中のメッセージのメッセージに関するサーバからの通知が無効になります。これにより、クライアントを待機するメッセージがサーバ上にある場合、クライアントへの通知を開始するように要求するサーバ上の処理は無効になります。ファイアウォールの制限によって通知が不可能である場合など、サーバから通知を送信できないときに、このプロパティを true に設定します。デフォルトは false です。

- **ianywhere.qa.server.logLevel** メッセージング機能のログ・レベル。プロパティ値は、1、2、3、4 のいずれかです。1 は、メッセージ・エラーのみを記録することを指定します。2 は、さらに警告も記録することを指定します。3 は、さらに情報メッセージも記録することを指定します。4 は、Mobile Link サーバで転送される各 QAnywhere メッセージの詳細を含めて、さらに詳細な情報メッセージも記録することを指定します。デフォルトは **2** です。

  これらのログ・メッセージは、Mobile Link サーバ・コンソールに出力されます。mlsrv10 の -o オプションまたは -ot オプショが指定されている場合、メッセージは Mobile Link サーバのログ・ファイルに出力されます。

- **ianywhere.qa.server.id**　サーバ管理要求の送信先アドレスのエージェント部分を指定します。このプロパティが設定されていない場合、この値は ianywhere.server になります。
- **ianywhere.qa.server.password.e**　サーバ管理要求の認証用パスワードを指定します。このプロパティが設定されていない場合、パスワードは QAnywhere です。

  「サーバ管理要求の概要」 100 ページを参照してください。
- **ianywhere.qa.server.scheduleDateFormat**　サーバ側の転送ルールで使用する日付フォーマットを指定します。デフォルトの日付フォーマットは yyyy-MM-dd です。

文字	日付のコンポーネント	例
y	年	1996
M	月	July
d	日	10

- **ianywhere.qa.server.scheduleTimeFormat**　サーバ側の転送ルールで使用する時間フォーマットを指定します。デフォルトの時間フォーマットは HH:mm:ss です。

文字	日付のコンポーネント	例
a	AM/PM マーカ	PM
H	1 日の特定の時刻 (0 ー 23 の値)	0
k	1 日の特定の時刻 (1 ー 24 の値)	24
K	午前／午後の特定の時刻 (0 ー 11 の値)	0
h	午前／午後の特定の時刻 (1 ー 12 の値)	12
m	分	30
s	秒	55

- **ianywhere.qa.server.transmissionRulesFile**

旧式な機能
Sybase Central の に置き換えられています。

メッセージの転送と持続性に関するルールを指定するファイル。デフォルトでは、メッセージのフィルタはなく、メッセージの最終ステータスがメッセージの発信者に転送されると、メッセージは削除されます。

Sybase Central でのサーバ・プロパティの設定

♦ **Sybase Central でサーバ・プロパティを設定するには、次の手順に従います。**

1. Sybase Central を起動します。
   - ♦ [スタート] – [プログラム] – [SQL Anywhere 10] – [Sybase Central] を選択します。
   - ♦ [接続] メニューから、[QAnywhere 10 に接続] を選択します。
   - ♦ ODBC データ・ソース名または ODBC データ・ソース・ファイルを指定し、必要に応じてユーザ ID とパスワードを指定します。[OK] をクリックします。
2. 左ウィンドウ枠の ［サーバ・ストア・タスク] で、[このメッセージ・ストアのプロパティの変更] を選択します。

   選択したメッセージ・ストアの [プロパティ] ダイアログが表示されます。

第 11 章

QAnywhere 転送ルールと削除ルール

目次

ルールの構文

ルールには、スケジュール部分と条件部分があります。スケジュールはイベントが実行されるタイミングを定義し、条件はイベントに含めるメッセージを定義します。たとえば、イベントがメッセージ転送である場合は、転送を実行するタイミングをスケジュールで指定し、転送に含めるメッセージを条件で定義します。イベントがメッセージの削除である場合は、削除を行うタイミングをスケジュールで指定し、削除対象のメッセージを条件で指定します。

ルールの構文

各ルールの形式は次のとおりです。

schedules=*condition*

スケジュール構文

スケジュール構文

schedules : { **AUTOMATIC** | *schedule-spec* ,⋯ }

schedule-spec :
{ **START TIME** *start-time* | **BETWEEN** *start-time* **AND** *end-time* }
[**EVERY** *period* { **HOURS** | **MINUTES** | **SECONDS** }]
[**ON** { **(** *day-of-week*, ⋯ **)** | **(** *day-of-month*, ⋯ **)** }]
[**START DATE** *start-date*]

パラメータ

- **AUTOMATIC**　転送ルールでは、メッセージのステータスが変更されたり、ネットワーク・ステータスに変更があった場合に、ルールが評価されます。削除ルールでは、メッセージ転送が開始されたときに、削除ルール条件を満たすメッセージが削除されます。
- **schedule-spec**　AUTOMATIC 以外のスケジュール仕様には、条件が評価される時刻を指定します。指定された時刻になると、対応する条件が評価されます。
- **START TIME**　イベントがスケジュールされた各日の最初のスケジュール時刻。START DATE を指定した場合、START TIME はその日付を参照します。START DATE を指定しない場合、START TIME の対象となる日付は、現在の日付 (指定された時刻を過ぎていない場合) とそれ以降の各日 (スケジュールに EVERY または ON が含まれる場合) となります。
- **BETWEEN ...AND ...**　1 日のうち、スケジュールされた時刻が発生する範囲。START DATE を指定した場合、スケジュールされた時刻はその日が来るまで発生しません。
- **EVERY**　連続してスケジュールするときのイベント発生の間隔。スケジュールされたイベントは、その日の START TIME より後、または BETWEEN ⋯ AND で指定された範囲内でのみ発生します。
- **ON**　スケジュールされたイベントが発生する日のリスト。EVERY を指定した場合、デフォルトは毎日です。これらは曜日または日付として指定できます。

曜日は、Mon、Tues のようになります。Monday のような完全形も使用できます。使用言語が英語でない場合、クライアントによって接続文字列で要求された言語でない場合、サーバ・ウィンドウに表示される言語でない場合は、完全形を使用します。

日付は、0 ー 31 の整数で指定します。0 は月の末日を表します。

- **START DATE**　スケジュールされたイベントが開始される日付。デフォルトは現在の日付です。

使用方法

指定した条件に対し、複数のスケジュールを作成できます。これにより、複雑なスケジュールを実装できます。

スケジュール仕様の定義に EVERY または ON が含まれる場合、そのスケジュール仕様は反復されます。EVERY と ON のどちらも使用されていない場合、そのスケジュールは最大でも 1 回しか実行されません。反復されないスケジュールを作成する場合に、そのスケジュールの開始時刻が過ぎているときは、エラーになります。

スケジュール済みの時刻になるたびに、対応する条件が評価され、次のスケジュール日時が計算されます。

次のスケジュール時刻を計算するときには、スケジュールが調べられ、現在以降の次のスケジュール時刻が決定されます。スケジュールに毎分と指定されている場合で、条件の評価に 65 秒かかるときは、2 分ごとに実行されます。実行を重複させたい場合は、複数のルールを作成する必要があります。

1. EVERY 句を使用した場合は、次のスケジュールされた時刻が現在の日付にあり、BETWEEN … AND で指定された範囲の終わりより前であるかどうかを調べます。そうであれば、それが次のスケジュールされた時刻となります。
2. 次のスケジュールされた時刻が現在の日付にない場合は、イベントが実行される次の日付を調べます。
3. その日付の START TIME、または BETWEEN … AND で指定された範囲の始まりを確認します。

QAnywhere スケジュール構文は、SQL Anywhere CREATE EVENT スケジュール構文に由来します。

キーワードは、大文字と小文字を区別しません。

参照

- 「CREATE EVENT 文」『SQL Anywhere サーバ - SQL リファレンス』

例

次の例に示すサーバ側の転送ルール・ファイルは、sample_store_id というクライアント・メッセージ・ストア ID で識別されるクライアントに適用されます。この転送ルール・ファイルでは、二重のスケジュールを作成します。まず、優先度の高いメッセージが 1 時間に 1 回送信されます。スケジュールは every 1 hours (1 時間ごと) に設定されており、条件は ias_priority=9 で

す。さらに、午前 8 時から 9 時までの間は、優先度の高いメッセージが 1 分ごとに送信されます。

```
[sample_store_id]
; This rule governs when messages are transmitted to the client
; store with id sample_store_id.
;
   every 1 hours = ias_priority=9
   between 8:00 and 9:00 every 1 minutes = iasPriority=9
```

条件構文

QAnywhere 条件は、SQL の構文に似ています。条件の評価対象は、メッセージ・ストア内のメッセージです。条件は、true、false、unknown のいずれかに評価されます。条件が空の場合は、すべてのメッセージが条件を満たすものと判定されます。条件は、転送ルール、削除ルール、QAnywhere プログラミング API で使用できます。

キーワードと文字列比較では、大文字と小文字を区別しません。

構文

condition :
expression **IS** [**NOT**] **NULL**
| *expression compare expression*
| *expression* [**NOT**] **BETWEEN** *expression* **AND** *expression*
| *expression* [**NOT**] **LIKE** *string* [**ESCAPE** *character*]
| *expression* [**NOT**] **IN (** *string*, ... **)**
| **NOT** *condition*
| *condition* **AND** *condition*
| *condition* **OR** *condition*
| **(** *condition* **)**

compare: **=** |**>** | **<** | **>=** | **<=** | **<>**

expression:
constant
| *rule-variable*
| **-***expression*
| *expression operator expression*
| **(** *expression* **)**
| *rule-function* **(** *expression*, ... **)**

integer: An integer in the range -2**63 to 2**63-1

number: A n umber in scientific notation in the range 2.2250738585072e-308 to 1.79769313486231e+308

string: A sequence of characters enclosed in single quotes. A single quote in a string is represented by two consecutive single quotes.

constant: *integer* | *number* | *string* | **TRUE** | **FALSE**

operator: **+** | **-** | ***** | */*

rule-variable:

「ルール変数」 248 ページを参照してください。

rule-function:

「ルール関数」 246 ページを参照してください。

パラメータ

- **BETWEEN** BETWEEN 条件は、true、false、または unknown として評価できます。NOT キーワードがない場合、*expression* が start expression と end expression の間にあれば、条件は TRUE と評価されます。

  NOT キーワードを使用すると条件の意味が逆になりますが、UNKNOWN は変わりません。

  BETWEEN 条件は、次の 2 つの不等式の組み合わせと等価です。

  [NOT] (*expression* **>=** *start-expression* **AND** *arithmetic-expression* **<=** *end-expr* **)**

  例：

  - age BETWEEN 15 AND 19 は、age >=15 AND age <= 19 と等価です。
  - age NOT BETWEEN 15 AND 19 は、age < 15 OR age > 19 と等価です。

- **IN** IN 条件は、次の規則に従って評価されます。
  - *expression* が null でなく、リスト内の少なくとも 1 つと等しい場合、true です。
  - *expression* が null で値リストが空でない場合、または少なくとも 1 つの値が null で、expression が他の値のいずれとも等しくない場合、unknown です。
  - いずれの値も null でなく、*expression* がリスト内のいずれの値とも等しくない場合、false です。

  NOT キーワードを指定すると、評価結果の true と false が逆になります。

  次に例を示します。

  - Country IN ('UK', 'US', 'France') は、Country の値が 'UK' の場合は true、'Peru' の場合は false になります。これは、次の条件と同義です。

    ```
    ( Country = 'UK' )      ¥
    OR ( Country = 'US' )   ¥
    OR ( Country = 'France' )
    ```

  - Country NOT IN ('UK', 'US', 'France') は、Country の値が 'UK' の場合は false、'Peru' の場合は true になります。これは、次の条件と同義です。

    ```
    NOT (   ( Country = 'UK' )    ¥
        OR ( Country = 'US' )     ¥
        OR ( Country = 'France' ) )
    ```

- **LIKE** LIKE 条件は、true、false、または unknown として評価できます。

  NOT キーワードがない場合、*expression* が *like expression* に一致すれば、条件は TRUE と評価されます。*expression* または *like expression* が NULL 値の場合、この条件は unknown です。

  NOT キーワードを使用すると条件の意味が逆になりますが、UNKNOWN は変わりません。

like expression には、任意の数のワイルドカードを指定できます。次のワイルドカードを使用できます。

ワイルドカード	一致するもの
_ (アンダースコア)	任意の 1 文字
% (パーセント記号)	0 個以上の文字からなる任意の文字列

次に例を示します。

- phone LIKE 12%3 は、phone が '123' または '12993' の場合は true、'1234' の場合は false になります。
- word LIKE 's_d' は、word が 'sad' の場合は true、'said' の場合は false になります。
- phone NOT LIKE '12%3' は、phone が '123' または '12993' のときは false、'1234' のときは true になります。

- **エスケープ文字**　エスケープ文字は、*pattern* 内のワイルドカード文字 (_、%) の特殊な意味をなくすために使用される 1 文字リテラルです。次に例を示します。
  - underscored LIKE '¥_%' ESCAPE '¥' は、underscored が '_myvar' の場合は true、'myvar' の場合は false になります。
- **IS NULL**　IS NULL 条件は、rule-variable が不定の場合は true、それ以外の場合は false に評価されます。NOT キーワードを使用すると条件の意味が逆になります。IS NULL 条件が unknown に評価されることはありません。

ルール関数

転送ルールでは次の関数を使用できます。

構文	説明
DATEADD(*datepart*, *count*, *datetime* **)**	いくつかの日付の単位を日時に追加して作成された日時を返します。*datepart* は、年、四半期、月、週、日、時、分、秒のいずれかです。たとえば、次の例では 2 か月が追加されて、値 2006-07-02 00:00:00.0 が返されます。 DATEADD(month, 2, '2006/05/02')
DATEPART(*datepart*, *date* **)**	日時の値の一部の値を返します。datepart は、年、四半期、月、週、日、通し日数、曜日、時、分、秒のいずれかです。たとえば、次の例では 5 月を数値で取得するため、値 5 が返されます。 DATEPART(month, '2006/05/02')
DATETIME(*string* **)**	文字列値を日時に変換します。文字列のフォーマットは 'yyyy-mm-dd hh:nn:ss' であることが必要です。

構文	説明
LENGTH(*string* **)**	文字列の文字数を返します。
SUBSTRING(*string*, *start*, *length* **)**	文字列の部分文字列を返します。*start* は、返される部分文字列が開始する位置を文字単位で指定します。*length* は、返される部分文字列の長さを文字単位で指定します。

例

次の削除ルールは、最終ステータスになってから 11 日以上経過したメッセージをすべて削除します。

```
every 1 hours = ias_Status >= ias_FinalState
  AND ias_StatusTime < DATEADD( day, -10, ias_CurrentTimestamp )
  AND ias_TransmissionStatus = 1
```

ルール変数

QAnywhere ルール変数は、ルールの条件部分で使用できます。ルール変数として使用できるのは次のとおりです。

- 「メッセージ・プロパティ」 223 ページ
- 「クライアント・メッセージ・ストア・プロパティ」 230 ページ
- 「ルール・エンジンで定義される変数」 248 ページ

ルール変数としてのプロパティの使用

メッセージ・プロパティとメッセージ・ストア・プロパティを転送ルール変数として使用できます。どちらの場合も、事前に定義されたプロパティを使用したり、カスタム・プロパティを作成したりできます。同じ名前のメッセージ・プロパティとメッセージ・ストア・プロパティがある場合は、メッセージ・プロパティが使用されます。この優先順位を無効にするには、次のようにしてプロパティを明示的に参照します。

- メッセージ・ストア・プロパティ名に ias_Store というプレフィクスを付けます。
- メッセージ・プロパティ名に ias_Message というプレフィクスを付けます。

たとえば、次の自動転送ルールでは、urgent というカスタム・メッセージ・プロパティが yes に設定されたすべてのメッセージが選択されます。

```
automatic = ias_Message.urgent = 'yes'
```

次の自動転送ルールでは、カスタム・メッセージ・ストア・プロパティ transmitNow が yes に設定されたときにメッセージが選択されます。

```
automatic = ias_Store.transmitNow = 'yes'
```

ルール・エンジンで定義される変数

次の変数はルール・エンジンで定義されています。

- **ias_Address**　メッセージのアドレス。myclient¥myqueue など。
- **ias_ContentSize**　メッセージのサイズ。テキスト・メッセージの場合は文字数、バイナリ・メッセージの場合はバイト数。
- **ias_ContentType**　次のメッセージ・タイプ。

IAS_TEXT_CONTENT	メッセージの内容は、ユニコード文字で構成される。
IAS_BINARY_CONTENT	メッセージの内容は、未解釈のバイト・シーケンスとして扱われる。

- **ias_CurrentDate**　現在の日付。

次のどちらかの方法で指定された文字列を、ias_currentDate と比較できます。

- 明確に解釈される形式の文字列。
- クライアント・メッセージ・ストア・データベース用に設定された date_format オプションに従った文字列。

  「オプションの設定」『SQL Anywhere サーバ - データベース管理』と「date_format オプション [互換性]」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **ias_CurrentTime**　現在の時刻。

  ias_CurrentTime と比較できるのは、時、分、秒がコロンで区切られた形式 (hh:mm:ss:sss) で指定された文字列です。**am** または **pm** を指定しないと、24 時間表記が使用されます。「time_format オプション [互換性]」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **ias_CurrentTimestamp**　現在のタイムスタンプ (現在の日付と時刻)。「time_format オプション [互換性]」『SQL Anywhere サーバ - データベース管理』を参照してください。
- **ias_Expires**　メッセージが配信されない場合に期限切れとなる日付と時刻。
- **ias_Network**　現在使用中のネットワークに関する情報。ias_Network は特殊な転送ルール変数です。このプロパティには、デバイスで使用されている現在のネットワークに関する情報を提供する多数の組み込み属性が定義されています。
- **ias_Priority**　0 ー 9 の整数で指定したメッセージの優先度。0 は優先度が最も低く、9 は優先度が高いことを表します。
- **ias_Status**　メッセージの現在のステータス。次のいずれかの値を取ります。

IAS_CANCELLED_STATE	メッセージはキャンセル済みです。
IAS_EXPIRED_STATE	メッセージが宛先に受信される前に期限切れになった。
IAS_FINAL_STATE	メッセージが受信されたか、期限切れになった。したがって、>=IAS_FINAL_STATE であれば、メッセージが受信されたか期限切れであることを意味し、<IAS_FINAL_STATE であれば、メッセージが受信もされず、期限切れでもないことを意味します。
IAS_PENDING_STATE	メッセージが宛先に受信されていない。
IAS_RECEIVED_STATE	メッセージが宛先に受信された。
IAS_UNRECEIVABLE_STATE	メッセージに間違った形式があるか、失敗した試行回数が多すぎたため、受信不可のマークが付けられている。

- **ias_TransmissionStatus**　メッセージの同期ステータス。次のいずれかの値を取ります。

IAS_UNTRANSMITTED	メッセージは、目的の受信側メッセージ・ストアに転送されていない。
IAS_TRANSMITTED	メッセージは、目的の受信側メッセージ・ストアに転送済み。

IAS_DO_NOT_TRANSMIT	受信先と送信元のメッセージ・ストアが同じであるため、転送は不要。
IAS_TRANSMITTING	メッセージは目的の受信先に転送されたが、まだ受信確認されていない。メッセージ転送が中断された可能性があるため、QAnywhere がメッセージを再転送する可能性があります。

例

クライアント・ストア・プロパティの作成方法、転送ルールでの使用方法の詳細については、「カスタムのクライアント・メッセージ・ストア・プロパティ属性の使用」 232 ページを参照してください。

メッセージ転送ルール

メッセージ転送とは、クライアント・メッセージ・ストアとサーバ・メッセージ・ストアの間でメッセージを移動するアクションを指します。

メッセージ転送は、QAnywhere Agent と Mobile Link サーバによって次のように処理されます。

- QAnywhere Agent は、QAnywhere クライアント・メッセージ・ストアと接続されており、Mobile Link サーバとの間でメッセージを転送します。
- Mobile Link サーバは、サーバ・メッセージ・ストアと接続されており、QAnywhere Agent からのメッセージを受信して他の QAnywhere Agent に転送します。

メッセージ転送が実行されるのは、クライアント・メッセージ・ストアとサーバ・メッセージ・ストアの間だけです。メッセージ転送は、QAnywhere Agent が Mobile Link サーバに接続されているときしか実行されません。

転送ルールでは、メッセージ転送のタイミングと転送するメッセージを指定できます。メッセージ・ストアからメッセージを削除するタイミングについてデフォルトの動作を使わない場合は、削除ルールで指定できます。

転送ルールは、サーバ側とクライアント側に作成できます。次の項を参照してください。

- 「クライアント側の転送ルール」 251 ページ
- 「サーバ側の転送ルール」 252 ページ

クライアント側の転送ルール

クライアント側の転送ルールは、クライアントからサーバに転送されるメッセージの動作を管理します。クライアント側の転送ルールは QAnywhere Agent によって処理されます。

デフォルトでは、QAnywhere Agent は自動ポリシーを使用します。QAnywhere Agent の転送ポリシーとして転送ルール・ファイルを指定することによって、このデフォルトの動作を変更、カスタマイズできます。

次の qaagent コマンド・ライン (一部) は、QAnywhere Agent 用のルール・ファイルを指定する方法を表しています。

```
qaagent -policy myrules.txt ...
```

転送ルールの作成方法の詳細については、「ルールの構文」 242 ページを参照してください。

ポリシーの詳細については、次の項を参照してください。

- 「クライアントにメッセージを転送するタイミングの決定」 40 ページ
- 「-policy オプション」 175 ページ

クライアント側の削除ルールについては、「クライアント側の削除ルール」 256 ページを参照してください。

転送ルール・ファイルには、次のエントリがあります。

- **ルール**　1 行に記述できるルールは 1 つだけです。

  各ルールは 1 行に入力する必要がありますが、行が次の行に続くことを表す文字として円記号 (¥) を使用できます。

- **コメント**　コメントは、行頭に # または ; を付けて示します。コメント行のすべての文字は無視されます。

「ルールの構文」 242 ページと「条件構文」 244 ページを参照してください。

転送ルール・ファイルには、メッセージ・ストアからメッセージを削除するタイミングも指定できます。

「メッセージの削除ルール」 256 ページを参照してください。

Sybase Central QAnywhere プラグインを使用して、QAnywhere Agent ルール・ファイルを作成することもできます。

例

たとえば、次に示すクライアント側の転送ルール・ファイルでは、営業時間中はサイズが小さく、優先度が高いメッセージだけを転送し、営業時間外は任意のメッセージを転送するように指定しています。このルールは自動的 (automatic) に実行されます。つまり、条件が満たされると、即座にメッセージが転送されます。この例から分かるように、条件には、メッセージからの情報とそれ以外の情報 (現在時刻など) を指定できます。

```
automatic = ( ias_ContentSize < 100000 and ias_Priority > 7 ) ¥
   or datepart(Weekday,Ias_CurrentDate) in ( 1, 7 ) ¥
   or ias_CurrentTime < '8:00:00' or ias_CurrentTime > '18:00:00'
```

サーバ側の転送ルール

サーバ側の転送ルールは、サーバからクライアントに転送されるメッセージの動作を管理します。サーバ側の転送ルールは Mobile Link サーバによって処理されます。これらのルールは、Push 通知を使用している場合も使用していない場合も適用されます。

サーバ側の転送ルールは、次の 3 つの方法で設定できます。

- サーバ管理要求を作成して転送ルールを設定する。

  「サーバ管理要求での転送ルールの指定」 124 ページを参照してください。

- Sybase Central を使用して転送ルールを設定する。

  「Sybase Central での転送ルールの指定」 253 ページを参照してください。

- サーバ側の転送ルール・ファイルを作成して、Mobile Link サーバの起動時に指定する。この方法は使用されなくなりました。

「転送ルール・ファイルによるサーバ側の転送ルールの指定 (旧式)」 253 ページを参照してください。

デフォルト・ルールの設定

特定のメッセージ・ストアや送信先エイリアスに対してサーバ側の転送ルールを指定することも、すべてのクライアントに対してデフォルト・ルールを指定することもできます。明示的な転送ルールを使用するユーザを除き、すべてのユーザがデフォルト・ルールを使用します。

デフォルト・ルールを設定するには、ianywhere.server.defaultClient という特殊なクライアント名を使用します。

Sybase Central での転送ルールの指定

Sybase Central を使用して、転送ルールを作成したり編集したりできます。

♦ **サーバ側のデフォルトの転送ルールを指定するには、次の手順に従います。**

1. Sybase Central を起動します。
   - [スタート] – [プログラム] – [SQL Anywhere 10] – [Sybase Central] を選択します。
   - [接続] メニューから、[QAnywhere 10 に接続] を選択します。
   - ODBC データ・ソース名または ODBC データ・ソース・ファイルを指定し、必要に応じてユーザ ID とパスワードを指定します。[OK] をクリックします。
2. [サーバ・ストア・タスク] で、[このメッセージ・ストアのプロパティの変更] をクリックします。

   [プロパティ] ダイアログが表示されます。
3. [転送ルール] タブを開き、[デフォルトの転送ルールをカスタマイズする] を選択します。
4. [新規] をクリックして、ルールを追加します。

   ルール・エディタが表示されます。
5. 条件をテキスト・フィールドに入力するか、ドロップダウン・リストから [メッセージ変数] または [ステータス定数] を選択して、条件を追加します。
6. [OK] をクリックしてルール・エディタを終了します。

転送ルール・ファイルによるサーバ側の転送ルールの指定 (旧式)

サーバ側の転送ルール・ファイルを作成したら、そのファイル名を QAnywhere メッセージング・プロパティ・ファイル内で ianywhere.qa.server.transmissionRulesFile プロパティによって指定します。

メッセージング・プロパティ・ファイルの詳細については、「-m オプション」 『Mobile Link - サーバ管理』を参照してください。

特定のクライアントに対して転送ルールを指定するには、対象クライアントのクライアント・メッセージ・ストア ID を角カッコで囲んで各セクションの先頭に指定します。

サーバ側のデフォルトの転送ルールを作成して、すべてのユーザに適用することができます。

デフォルトの転送ルールを指定するには、次の行を含むセクションを開始します。

```
[ianywhere.server.defaultClient]
```

新しい転送ルールを有効にするには、Mobile Link サーバを再起動する必要があります。再起動が必要なのは、転送ルール・ファイルで指定された転送ルールを有効にする場合だけです。Sybase Central またはサーバ管理要求を使用して指定されたサーバ側の転送ルールは、指定後すぐに有効になります。

サーバ側の削除ルールの詳細については、「サーバ側の削除ルール」 256 ページを参照してください。

例

次に示したサーバ側の転送ルール・ファイルのセクションでは、優先度の高いメッセージだけを送信するデフォルト・ルールが作成されます。

```
[ianywhere.server.defaultClient]
auto = ias_Priority > 6
```

次の例で示すサーバ側の転送ルール・ファイルでは、sample_store_id というクライアント・メッセージ・ストア ID で識別されるクライアントだけにルールが適用されます。

```
[sample_store_id]
; This rule governs when messages are transmitted to the client
; store with id sample_store_id.
;
;    ias_Priority >= 7
;
; Messages with priority 7 or greater should always be
; transmitted.
;
;    ias_ContentSize < 100
;
; Small messages (messages less than 100 characters or
; bytes in size) should always be transmitted.
;
;    ias_CurrentTime < '8:00am' or ias_CurrentTime > '6:00pm'
;
; Outside of business hours messages should always be
; transmitted

auto = ias_Priority >= 7 or ias_ContentSize < 100 ¥
    or ias_CurrentTime < '8:00:00' or ias_CurrentTime > '18:00:00'
```

次の例では、qanywhere というクライアント・メッセージ・ストア ID で識別されるクライアントだけにルールが適用されます。

```
[qanywhere]
; This rule governs when messages are transmitted to the client
; store with id qanywhere.
;
;    tm_Subject not like '%non-business%'
```



```
;
; Messages with the property tm_Subject set to a value that
; includes the phrase 'non-business' should not be transmitted
;
;    ias_CurrentTime < '8:00:00' or ias_CurrentTime > '18:00:00'
;
; Outside of business hours, messages should always be
; transmitted

auto = tm_Subject not like '%non-business%' ¥
    or ias_CurrentTime < '8:00am' or ias_CurrentTime > '6:00pm'
```


メッセージの削除ルール

削除ルールは、クライアント・メッセージ・ストアとサーバ・メッセージ・ストアに格納されているメッセージの持続性を決定します。

デフォルトの動作

QAnywhere のメッセージは、有効期限を過ぎてもメッセージが送受信されていない場合に期限切れになります。メッセージが期限切れになると、デフォルトの削除ルールで削除されます。メッセージが 1 回以上受信されたが、受信確認されなかった場合は、有効期限を過ぎても再受信できます。

クライアント側の削除ルール

デフォルトでは、メッセージのステータスが受信済み、失効、キャンセル済み、配信不可のいずれかで、最終ステータスがサーバ・メッセージ・ストアに転送済みである場合に、クライアント・メッセージ・ストアからメッセージが削除されます。しかし、デフォルトよりも早くメッセージを削除したい場合もあれば、デフォルトよりも長くメッセージを残したい場合もあります。それには、クライアント側の転送ルール・ファイル内に削除セクションを記述します。削除セクションの先頭に **[system:delete]** を指定する必要があります。

受信確認の詳細については、次の項を参照してください。

- .NET：「AcknowledgementMode 列挙」 262 ページ
- C++：「AcknowledgementMode クラス」 422 ページ
- Java：「AcknowledgementMode インタフェース」 532 ページ

クライアント側の転送ルールの詳細については、「クライアント側の転送ルール」 251 ページを参照してください。

クライアント側の転送ルール・ファイル内の削除ルール・セクションの例を以下に示します。

```
[system:delete]

; This rule governs when messages are deleted from the client
; store.
;
;    start time '1:00:00' on ( 'Sunday' )
;
; Messages are deleted every Sunday at 1:00 A.M.
;
;    ias_Status >= ias_FinalState
;
; Typically, messages are deleted when they reach a final
; state: received, unreceivable, expired, or cancelled.

start time '1:00:00' on ( 'Sunday' ) = ias_Status >= ias_FinalState
```

ias_Status の詳細については、「ルール変数」 248 ページを参照してください。

サーバ側の削除ルール

デフォルトでは、メッセージのステータスが受信済み、失効、キャンセル済み、配信不可のいずれかで、最終ステータスがメッセージの発信者に返送済みである場合に、サーバ・メッセージ・

ストアからメッセージが削除されます。しかし、監査などのためにメッセージを通常よりも長く残したい場合があります。

サーバ側の削除ルールは、サーバ・メッセージ・ストアのすべてのメッセージに適用されます。

サーバ側の転送ルールの詳細については、「サーバ側の転送ルール」 252 ページを参照してください。

ias_Status の詳細については、「ルール変数」 248 ページを参照してください。

パート II. QAnywhere API リファレンス

パート II では、QAnywhere API の参照マニュアルを示します。

第 12 章

QAnywhere .NET API リファレンス

目次

iAnywhere.QAnywhere.Client ネームスペース (.NET 1.0)

AcknowledgementMode 列挙

QAnywhere クライアント・アプリケーションによってどのようにメッセージが確認されるかを示します。

構文

Visual Basic
Public Enum **AcknowledgementMode**

C#
public enum **AcknowledgementMode**

備考

暗黙的または明示的な受信確認モードは、QAManager.Open(AcknowledgementMode) メソッドを使用して、QAManager インスタンスに割り当てられます。

詳細については、「QAnywhere API の初期化」 61 ページを参照してください。

暗黙的な受信確認では、メッセージは、クライアント・アプリケーションで受信されるとすぐに受信確認されます。明示的な受信確認では、いずれかの QAManager 受信確認メソッドを呼び出してください。すべてのステータス変更は、サーバによってクライアント間で伝達されます。

詳細については、「同期的なメッセージ受信」 82 ページと「非同期的なメッセージ受信」 83 ページを参照してください。

メンバ名

メンバ名	説明
EXPLICIT_ACKNOWLEDGEMENT	受信メッセージは、QAManager のいずれかの受信確認メソッドを使用して確認されます。
IMPLICIT_ACKNOWLEDGEMENT	すべてのメッセージは、クライアント・アプリケーションで受信されるとすぐに受信確認されます。メッセージを同期に受信する場合、メッセージは QAManagerBase.GetMessage (string) メソッドが返されるとすぐに受信確認されます。メッセージを非同期に受信する場合、メッセージはイベント処理関数が返されるとすぐに受信確認されます。
TRANSACTIONAL	このモードでは、メッセージの受信確認はトランザクションの一部として行われます。それ以外では行われません。このモードは、QATransactionalManager インスタンスに自動的に割り当てられます。

参照

- 「QAManager インタフェース」 292 ページ
- 「QATransactionalManager インタフェース」 367 ページ

- 「QAManagerBase インタフェース」 297 ページ

ExceptionListener 委任

ExceptionListener 委任の定義です。ExceptionListener は SetExceptionListener メソッドに渡します。

構文

Visual Basic

```
Public Delegate Sub ExceptionListener( _
  ByVal ex As QAException, _
  ByVal msg As QAMessage _
)
```

C#

```
public delegate void ExceptionListener(
  QAException ex,
  QAMessage msg
);
```

パラメータ

- **ex** 発生した例外
- **msg** 受信したメッセージ。またはメッセージが構築できなかった場合は null。

ExceptionListener2 委任

ExceptionListener2 委任の定義です。ExceptionListener2 は SetExceptionListener2 メソッドに渡します。

構文

Visual Basic

```
Public Delegate Sub ExceptionListener2( _
  ByVal mgr As QAManagerBase, _
  ByVal ex As QAException, _
  ByVal msg As QAMessage _
)
```

C#

```
public delegate void ExceptionListener2(
  QAManagerBase mgr,
  QAException ex,
  QAMessage msg
);
```

パラメータ

- **mgr** メッセージを処理した QAManagerBase
- **ex** 発生した例外
- **msg** 受信したメッセージ。またはメッセージが構築できなかった場合は null。

MessageListener 委任

MessageListener 委任の定義です。MessageListener は QAManagerBase.SetMessageListener メソッドに渡します。

構文

Visual Basic
```
Public Delegate Sub MessageListener( _
  ByVal msg As QAMessage _
)
```

C#
```
public delegate void MessageListener(
  QAMessage msg
);
```

パラメータ

- **msg**　受信したメッセージ

参照

- 「MessageListener 委任」 264 ページ
- 「SetMessageListener メソッド」 329 ページ

MessageListener2 委任

MessageListener2 委任の定義です。MessageListener2 は SetMessageListener2 メソッドに渡します。

構文

Visual Basic
```
Public Delegate Sub MessageListener2( _
  ByVal mgr As QAManagerBase, _
  ByVal msg As QAMessage _
)
```

C#
```
public delegate void MessageListener2(
  QAManagerBase mgr,
  QAMessage msg
);
```

パラメータ

- **mgr**　メッセージを受信した QAManagerBase
- **msg**　受信したメッセージ

MessageProperties クラス

標準のメッセージ・プロパティ名を格納するフィールドを提供します。

構文

Visual Basic
Public Class **MessageProperties**

C#
public class **MessageProperties**

備考

MessageProperties クラスは、標準のメッセージ・プロパティ名を提供します。MessageProperties フィールドは、メッセージ・プロパティの取得と設定で使用する QAMessage メソッドに渡すことができます。

詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

- 「MessageProperties のメンバ」 265 ページ
- 「QAMessage インタフェース」 343 ページ
- 「GetIntProperty メソッド」 352 ページ
- 「GetStringProperty メソッド」 356 ページ

MessageProperties のメンバ

パブリックの静的フィールド (共有)

メンバ名	説明
ADAPTER フィールド	"system" キュー・メッセージの場合は、QAnywhere サーバへの接続で使用されているネットワーク・アダプタです。
ADAPTERS フィールド	このプロパティ名は、QAnywhere サーバへの接続で使用できるネットワーク・アダプタのデリミタ付きリストを示します。
DELIVERY_COUNT フィールド	このプロパティ名は、メッセージを配信するために、これまでに実行された試行回数を示します。
IP フィールド	"system" キュー・メッセージの場合は、QAnywhere サーバへの接続で使用されているネットワーク・アダプタの IP アドレスです。
MAC フィールド	"system" キュー・メッセージの場合は、QAnywhere サーバへの接続で使用されているネットワーク・アダプタの MAC アドレスです。
MSG_TYPE フィールド	このプロパティ名は、QAnywhere メッセージに関連付けられている MessageType 値を示します。
NETWORK_STATUS フィールド	このプロパティ名は、ネットワーク接続のステータスを示します。

メンバ名	説明
ORIGINATOR フィールド	このプロパティ名は、メッセージの発信者のメッセージ・ストア ID を示します。
RAS フィールド	"system" キュー・メッセージの場合は、QAnywhere サーバへの接続で使用されている RAS エントリ名です。
RASNAMES フィールド	"system" キュー・メッセージの場合は、QAnywhere サーバに接続するときに使用できる RAS エントリ名のデリミタ付きリストです。
STATUS フィールド	このプロパティ名は、メッセージの現在のステータスを示します。
STATUS_TIME フィールド	このプロパティ名は、メッセージが現在のステータスになった時刻を示します。
TRANSMISSION_STATUS フィールド	このプロパティ名は、メッセージの現在の転送ステータスを示します。

ADAPTER フィールド

"system" キュー・メッセージの場合は、QAnywhere サーバへの接続で使用されているネットワーク・アダプタです。

構文

Visual Basic
Public Shared **ADAPTER** As String

C#
public const string **ADAPTER**;

備考

このフィールドの値は "ias_Network.Adapter" です。

詳細については、「事前に定義されたクライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

MessageProperties.ADAPTER を QAMessage.GetStringProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- 「MessageProperties クラス」 264 ページ
- 「MessageProperties のメンバ」 265 ページ
- 「MessageProperties クラス」 264 ページ
- 「GetStringProperty メソッド」 356 ページ

ADAPTERS フィールド

このプロパティ名は、QAnywhere サーバへの接続で使用できるネットワーク・アダプタのデリミタ付きリストを示します。

構文

Visual Basic
Public Shared **ADAPTERS** As String

C#
public const string **ADAPTERS**;

備考

これは、システム・キュー・メッセージで使用されます。

MessageProperties.ADAPTERS を QAMessage.GetStringProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

詳細については、「事前に定義されたクライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

参照

- 「MessageProperties クラス」 264 ページ
- 「MessageProperties のメンバ」 265 ページ
- 「MessageProperties クラス」 264 ページ
- 「GetStringProperty メソッド」 356 ページ

DELIVERY_COUNT フィールド

このプロパティ名は、メッセージを配信するために、これまでに実行された試行回数を示します。

構文

Visual Basic
Public Shared **DELIVERY_COUNT** As String

C#
public const string **DELIVERY_COUNT**;

備考

このフィールドの値は "ias_DeliveryCount" です。

MessageProperties.DELIVERY_COUNT を QAMessage.GetIntProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- 「MessageProperties クラス」 264 ページ
- 「MessageProperties のメンバ」 265 ページ
- 「MessageProperties クラス」 264 ページ

- 「GetIntProperty メソッド」 352 ページ

IP フィールド

"system" キュー・メッセージの場合は、QAnywhere サーバへの接続で使用されているネットワーク・アダプタの IP アドレスです。

構文

Visual Basic
Public Shared **IP** As String

C#
public const string **IP**;

備考

このフィールドの値は "ias_Network.IP" です。

詳細については、「事前に定義されたクライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

MessageProperties.IP を QAMessage.GetStringProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- 「MessageProperties クラス」 264 ページ
- 「MessageProperties のメンバ」 265 ページ
- 「MessageProperties クラス」 264 ページ
- 「GetStringProperty メソッド」 356 ページ

MAC フィールド

"system" キュー・メッセージの場合は、QAnywhere サーバへの接続で使用されているネットワーク・アダプタの MAC アドレスです。

構文

Visual Basic
Public Shared **MAC** As String

C#
public const string **MAC**;

備考

このフィールドの値は "ias_Network.MAC" です。

詳細については、「事前に定義されたクライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

MessageProperties.MAC を QAMessage.GetStringProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- 「MessageProperties クラス」 264 ページ
- 「MessageProperties のメンバ」 265 ページ
- 「MessageProperties クラス」 264 ページ
- 「GetStringProperty メソッド」 356 ページ

MSG_TYPE フィールド

このプロパティ名は、QAnywhere メッセージに関連付けられている MessageType 値を示します。

構文

Visual Basic
Public Shared **MSG_TYPE** As String

C#
public const string **MSG_TYPE**;

備考

このフィールドの値は "ias_MessageType" です。

MessageProperties.MSG_TYPE を QAMessage.GetIntProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- 「MessageProperties クラス」 264 ページ
- 「MessageProperties のメンバ」 265 ページ
- 「MessageProperties クラス」 264 ページ
- 「MessageType 列挙」 274 ページ
- 「GetIntProperty メソッド」 352 ページ
- 「GetStringProperty メソッド」 356 ページ

NETWORK_STATUS フィールド

このプロパティ名は、ネットワーク接続のステータスを示します。

構文

Visual Basic
Public Shared **NETWORK_STATUS** As String

C#
public const string **NETWORK_STATUS**;

備考

ネットワークがアクセス可能な場合は 1、それ以外の場合は 0 です。

ネットワーク・ステータスは、システム・キュー・メッセージ (ネットワーク・ステータスの変更など) で使用されます。

詳細については、「事前に定義されたクライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

MessageProperties.NETWORK_STATUS を QAMessage.GetIntProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- 「MessageProperties クラス」 264 ページ
- 「MessageProperties のメンバ」 265 ページ
- 「MessageProperties クラス」 264 ページ
- 「GetIntProperty メソッド」 352 ページ

ORIGINATOR フィールド

このプロパティ名は、メッセージの発信者のメッセージ・ストア ID を示します。

構文

Visual Basic
Public Shared **ORIGINATOR** As String

C#
public const string **ORIGINATOR**;

備考

このフィールドの値は "ias_Originator" です。

MessageProperties.ORIGINATOR を QAMessage.GetStringProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- 「MessageProperties クラス」 264 ページ
- 「MessageProperties のメンバ」 265 ページ
- 「MessageProperties クラス」 264 ページ
- 「GetStringProperty メソッド」 356 ページ

RAS フィールド

"system" キュー・メッセージの場合は、QAnywhere サーバへの接続で使用されている RAS エントリ名です。

構文

Visual Basic
Public Shared **RAS** As String

C#
public const string **RAS**;

備考

このフィールドの値は "ias_Network.RAS" です。

詳細については、「事前に定義されたクライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

MessageProperties.RAS を QAMessage.GetStringProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- 「MessageProperties クラス」 264 ページ
- 「MessageProperties のメンバ」 265 ページ
- 「MessageProperties クラス」 264 ページ
- 「GetStringProperty メソッド」 356 ページ

RASNAMES フィールド

"system" キュー・メッセージの場合は、QAnywhere サーバに接続するときに使用できる RAS エントリ名のデリミタ付きリストです。

構文

Visual Basic
Public Shared **RASNAMES** As String

C#
public const string **RASNAMES**;

備考

このフィールドの値は "ias_RASNames" です。

詳細については、「事前に定義されたクライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

MessageProperties.RASNAMES を QAMessage.GetStringProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- 「MessageProperties クラス」 264 ページ
- 「MessageProperties のメンバ」 265 ページ
- 「MessageProperties クラス」 264 ページ
- 「GetStringProperty メソッド」 356 ページ

STATUS フィールド

このプロパティ名は、メッセージの現在のステータスを示します。

構文

Visual Basic
Public Shared **STATUS** As String

C#
public const string **STATUS**;

備考

プロパティ値のリストについては、StatusCodes 列挙を参照してください。

このフィールドの値は "ias_Status" です。

MessageProperties.STATUS を QAMessage.GetIntProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- 「MessageProperties クラス」 264 ページ
- 「MessageProperties のメンバ」 265 ページ
- 「StatusCodes 列挙」 370 ページ
- 「MessageProperties クラス」 264 ページ
- 「GetIntProperty メソッド」 352 ページ

STATUS_TIME フィールド

このプロパティ名は、メッセージが現在のステータスになった時刻を示します。

構文

Visual Basic
Public Shared **STATUS_TIME** As String

C#
public const string **STATUS_TIME**;

備考

これはローカル時間です。STATUS_TIME は、QAMessage.GetProperty に渡されると、DateTime オブジェクトを返します。このフィールドの値は "ias_StatusTime" です。

参照

- 「MessageProperties クラス」 264 ページ
- 「MessageProperties のメンバ」 265 ページ
- 「GetProperty メソッド」 353 ページ
- 「MessageProperties クラス」 264 ページ
- 「GetProperty メソッド」 353 ページ

TRANSMISSION_STATUS フィールド

このプロパティ名は、メッセージの現在の転送ステータスを示します。

構文

Visual Basic
Public Shared **TRANSMISSION_STATUS** As String

C#
public const string **TRANSMISSION_STATUS**;

備考

プロパティ値のリストについては、StatusCodes 列挙を参照してください。

このフィールドの値は "ias_TransmissionStatus" です。

MessageProperties.TRANSMISSION_STATUS を QAMessage.GetIntProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- 「MessageProperties クラス」 264 ページ
- 「MessageProperties のメンバ」 265 ページ
- 「StatusCodes 列挙」 370 ページ
- 「MessageProperties クラス」 264 ページ
- 「GetIntProperty メソッド」 352 ページ

MessageStoreProperties クラス

このクラスは、有用なメッセージ・ストア・プロパティ名の定数値を定義します。

構文

Visual Basic
Public Class **MessageStoreProperties**

C#
public class **MessageStoreProperties**

備考

MessageStoreProperties クラスは、標準のメッセージ・プロパティ名を提供します。MessageProperties フィールドは、事前定義済みまたはカスタムのメッセージ・ストア・プロパティの取得と設定で使用する QAManagerBase メソッドに渡すことができます。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

MessageStoreProperties のメンバ

パブリックの静的フィールド (共有)

メンバ名	説明
MAX_DELIVERY_ATTEMPTS フィールド	このプロパティ名は、メッセージのステータスが StatusCodes.UNRECEIVABLE に設定されるまでに、明示的な受信確認がないままメッセージを受信できる最大回数を示します。このフィールドの値は "ias_MaxDeliveryAttempts" です。

パブリック・コンストラクタ

メンバ名	説明
MessageStoreProperties コンストラクタ	MessageStoreProperties クラスの新しいインスタンスを初期化します。

MessageStoreProperties コンストラクタ

MessageStoreProperties クラスの新しいインスタンスを初期化します。

構文

Visual Basic
Public Sub **New()**

C#
public **MessageStoreProperties();**

MAX_DELIVERY_ATTEMPTS フィールド

このプロパティ名は、メッセージのステータスが StatusCodes.UNRECEIVABLE に設定されるまでに、明示的な受信確認がないままメッセージを受信できる最大回数を示します。このフィールドの値は "ias_MaxDeliveryAttempts" です。

構文

Visual Basic
Public Shared **MAX_DELIVERY_ATTEMPTS** As String

C#
public const string **MAX_DELIVERY_ATTEMPTS**;

MessageType 列挙

MessageProperties.MSG_TYPE メッセージ・プロパティの定数値を定義します。

構文

Visual Basic
Public Enum **MessageType**

C#
public enum **MessageType**

メンバ名

メンバ名	説明
NETWORK_STATUS_NOTIFICATION	QAnywhere クライアント・アプリケーションにネットワーク・ステータスの変更を通知する場合に使用する、QAnywhere システム・メッセージを特定します。
PUSH_NOTIFICATION	QAnywhere クライアント・アプリケーションに Push 通知を通知する場合に使用する、QAnywhere システム・メッセージを特定します。
REGULAR	メッセージ・タイプ・プロパティが存在しない場合、メッセージ・タイプは REGULAR とみなされます。

PropertyType 列挙

QAMessage プロパティ型列挙。C# の型に必然的に対応します。

構文

Visual Basic
Public Enum **PropertyType**

C#
public enum **PropertyType**

メンバ名

メンバ名	説明
BOOLEAN	プロパティ型が boolean であることを示します。
BYTE	プロパティ型が signed byte であることを示します。
DOUBLE	プロパティ型が double であることを示します。
FLOAT	プロパティ型が float であることを示します。
INT	プロパティ型が int であることを示します。
LONG	プロパティ型が long であることを示します。
SHORT	プロパティ型が short であることを示します。
STRING	プロパティ型が文字列型であることを示します。

メンバ名	説明
UNKNOWN	プロパティ型が不定であることを示します。通常はプロパティを認識できないことが原因です。

QABinaryMessage インタフェース

QABinaryMessage オブジェクトは、未解釈のバイト・ストリームが含まれるメッセージの送信に使用します。これは QAMessage クラスを継承したもので、メッセージ本文にバイト・ストリームが追加されます。QABinaryMessage には、メッセージ本文からのバイト・ストリームの読み込み／書き込みを行うためのさまざまな関数があります。

メッセージが最初に作成された時点では、メッセージ本文は書き込み専用モードになっています。メッセージ送信元のクライアントは、メッセージを送信した後、そのメッセージを保持し変更できます。ただし、それによって、送信されたメッセージが変更されることはありません。同じメッセージ・オブジェクトを複数回送信できます。

メッセージが受信されるとプロバイダが QABinaryMessage.Reset() を呼び出します。これによりメッセージ本文が読み込み専用モードになり、メッセージ本文の先頭から値の読み込みが開始されます。

構文

Visual Basic
Public Interface **QABinaryMessage**

C#
public interface **QABinaryMessage**

QABinaryMessage のメンバ

パブリック・プロパティ

メンバ名	説明
BodyLength プロパティ	メッセージ本文のサイズをバイト単位で返します。

パブリック・メソッド

メンバ名	説明
ReadBinary メソッド	QABinaryMessage インスタンスの本文の未読部分から、指定されたサイズのバイト・ストリームを読み込みます。
ReadBoolean メソッド	QABinaryMessage インスタンスのメッセージ本文の未読部分から boolean 値を読み込みます。
ReadChar メソッド	QABianryMessage メッセージ本文の未読部分から char 値を読み込みます。

メンバ名	説明
ReadDouble メソッド	QABianryMessage メッセージ本文の未読部分から double 値を読み込みます。
ReadFloat メソッド	QABianryMessage メッセージ本文の未読部分から float 値を読み込みます。
ReadInt メソッド	QABianryMessage メッセージ本文の未読部分から int 値を読み込みます。
ReadLong メソッド	QABianryMessage メッセージ本文の未読部分から long 値を読み込みます。
ReadSbyte メソッド	QABianryMessage メッセージ本文の未読部分から signed byte 値を読み込みます。
ReadShort メソッド	QABianryMessage メッセージ本文の未読部分から short 値を読み込みます。
ReadString メソッド	QABianryMessage メッセージ本文の未読部分から文字列値を読み込みます。
Reset メソッド	メッセージをリセットして、メッセージ本文の先頭から値の読み込みを開始できるようにします。
WriteBinary メソッド	QABinaryMessage インスタンスのメッセージ本文にバイト配列値を追加します。
WriteBoolean メソッド	QABinaryMessge インスタンスのメッセージ本文に boolean 値を追加します。
WriteChar メソッド	QABinaryMessge インスタンスのメッセージ本文に char 値を追加します。
WriteDouble メソッド	QABinaryMessge インスタンスのメッセージ本文に double 値を追加します。
WriteFloat メソッド	QABinaryMessge インスタンスのメッセージ本文に float 値を追加します。
WriteInt メソッド	QABinaryMessge インスタンスのメッセージ本文に整数値を追加します。
WriteLong メソッド	QABinaryMessge インスタンスのメッセージ本文に long 値を追加します。
WriteSbyte バイト	QABinaryMessge インスタンスのメッセージ本文に signed byte 値を追加します。
WriteShort メソッド	QABinaryMessge インスタンスのメッセージ本文に short 値を追加します。
WriteString メソッド	QABinaryMessge インスタンスのメッセージ本文に文字列値を追加します。

BodyLength プロパティ

メッセージ本文のサイズをバイト単位で返します。

構文

Visual Basic
Public Readonly Property **BodyLength** As Long

C#
public long **BodyLength** {get;}

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ

ReadBinary メソッド

QABinaryMessage インスタンスの本文の未読部分から、指定されたサイズのバイト・ストリームを読み込みます。

構文

Visual Basic
```
Public Function ReadBinary( _
  ByVal bytes As Byte(), _
  ByVal len As Integer _
) As Integer
```

C#
```
public int ReadBinary(
  byte[] bytes,
  int len
);
```

パラメータ

- **bytes**　読み込まれるバイトに含まれるバイト配列
- **len**　読み込むバイト数の最大値

戻り値

メッセージ本文から読み込まれるバイト数

例外

- QAException クラス - 値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ

- 「QABinaryMessage インタフェース」 276 ページ
- 「WriteBinary メソッド」 284 ページ

ReadBoolean メソッド

QABinaryMessage インスタンスのメッセージ本文の未読部分から boolean 値を読み込みます。

構文

Visual Basic
Public Function **ReadBoolean()** As Boolean

C#
public bool **ReadBoolean();**

戻り値

メッセージ本文から読み込まれる boolean 値

例外

- QAException クラス - 値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「WriteBoolean メソッド」 285 ページ

ReadChar メソッド

QABianryMessage メッセージ本文の未読部分から char 値を読み込みます。

構文

Visual Basic
Public Function **ReadChar()** As Char

C#
public char **ReadChar();**

戻り値

メッセージ本文から読み込まれる char 値

例外

- QAException クラス - 値の読み込み時に変換エラーが発生した場合、または読み込める入力が残っていない場合。

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「WriteChar メソッド」 285 ページ

ReadDouble メソッド

QABianryMessage メッセージ本文の未読部分から double 値を読み込みます。

構文

Visual Basic
Public Function **ReadDouble()** As Double

C#
public double **ReadDouble();**

戻り値

メッセージ本文から読み込まれる double 値

例外

- QAException クラス - 値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「WriteDouble メソッド」 286 ページ

ReadFloat メソッド

QABianryMessage メッセージ本文の未読部分から float 値を読み込みます。

構文

Visual Basic
Public Function **ReadFloat()** As Single

C#
public float **ReadFloat();**

戻り値

メッセージ本文から読み込まれる float 値

例外

- QAException クラス - 値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「WriteFloat メソッド」 286 ページ

ReadInt メソッド

QABianryMessage メッセージ本文の未読部分から int 値を読み込みます。

構文

Visual Basic
Public Function **ReadInt()** As Integer

C#
public int **ReadInt();**

戻り値

メッセージ本文から読み込まれる int 値

例外

- QAException クラス - 値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「WriteInt メソッド」 287 ページ

ReadLong メソッド

QABianryMessage メッセージ本文の未読部分から long 値を読み込みます。

構文

Visual Basic
Public Function **ReadLong()** As Long

C#
public long **ReadLong();**

戻り値

メッセージ本文から読み込まれる long 値

例外

- QAException クラス - 値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「WriteLong メソッド」 287 ページ

ReadSbyte メソッド

QABianryMessage メッセージ本文の未読部分から signed byte 値を読み込みます。

構文

Visual Basic
Public Function **ReadSbyte()** As System.SByte

C#
public System.Sbyte **ReadSbyte();**

戻り値

メッセージ本文から読み込まれる signed byte 値

例外

- QAException クラス - 値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「WriteSbyte バイト」 288 ページ

ReadShort メソッド

QABianryMessage メッセージ本文の未読部分から short 値を読み込みます。

構文

Visual Basic
Public Function **ReadShort()** As Short

C#
public short **ReadShort();**

戻り値

メッセージ本文から読み込まれる short 値

例外

- QAException クラス - 値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「WriteShort メソッド」 289 ページ

ReadString メソッド

QABianryMessage メッセージ本文の未読部分から文字列値を読み込みます。

構文

Visual Basic
Public Function **ReadString()** As String

C#
public string **ReadString();**

戻り値

メッセージ本文から読み込まれる文字列値

例外

- QAException クラス - 値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「WriteString メソッド」 289 ページ

Reset メソッド

メッセージをリセットして、メッセージ本文の先頭から値の読み込みを開始できるようにします。

構文

Visual Basic
Public Sub **Reset()**

C#
public void **Reset();**

備考

また、Reset メソッドは、QABinaryMessage のメッセージ本文を読み込み専用モードにします。

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ

WriteBinary メソッド

QABinaryMessage インスタンスのメッセージ本文にバイト配列値を追加します。

構文

Visual Basic
```
Public Sub WriteBinary( _
  ByVal val As Byte(), _
  ByVal offset As Integer, _
  ByVal len As Integer _
)
```

C#
```
public void WriteBinary(
  byte[] val,
  int offset,
  int len
);
```

パラメータ

- **val**　メッセージ本文に書き込むバイト配列値
- **len**　書き込まれるバイト数
- **offset**　バイト配列内でのオフセット (書き込み開始位置)

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「ReadBinary メソッド」 278 ページ

WriteBoolean メソッド

QABinaryMessge インスタンスのメッセージ本文に boolean 値を追加します。

構文

Visual Basic
```
Public Sub WriteBoolean( _
  ByVal val As Boolean _
)
```

C#
```
public void WriteBoolean(
  bool val
);
```

パラメータ

- **val**　メッセージ本文に書き込む boolean 値

備考

boolean 値は 1 バイトの値で示されます。True は 1、false は 0 で示されます。

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「ReadBoolean メソッド」 279 ページ

WriteChar メソッド

QABinaryMessge インスタンスのメッセージ本文に char 値を追加します。

構文

Visual Basic
```
Public Sub WriteChar( _
  ByVal val As Char _
)
```

C#
```
public void WriteChar(
  char val
);
```

パラメータ

- **val**　メッセージ本文に書き込む char 値

備考

char は 2 バイトの値で示され、上位バイトが最初に追加されます。

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「ReadChar メソッド」 279 ページ

WriteDouble メソッド

QABinaryMessge インスタンスのメッセージ本文に double 値を追加します。

構文

Visual Basic
```
Public Sub WriteDouble( _
  ByVal val As Double _
)
```

C#
```
public void WriteDouble(
  double val
);
```

パラメータ

- **val**　メッセージ本文に書き込む double 値

備考

double 値は 8 バイトの long 値に変換されて、上位のバイトから追加されます。

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「ReadDouble メソッド」 280 ページ

WriteFloat メソッド

QABinaryMessge インスタンスのメッセージ本文に float 値を追加します。

構文

Visual Basic
```
Public Sub WriteFloat( _
  ByVal val As Single _
)
```

C#
```
public void WriteFloat(
  float  val
);
```

パラメータ

- **val**　メッセージ本文に書き込む float 値

備考

float パラメータは 4 バイトの整数に変換され、上位のバイトから追加されます。

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「ReadFloat メソッド」 280 ページ

WriteInt メソッド

QABinaryMessge インスタンスのメッセージ本文に整数値を追加します。

構文

Visual Basic

```
Public Sub WriteInt( _
  ByVal val As Integer _
)
```

C#

```
public void WriteInt(
  int val
);
```

パラメータ

- **val**　メッセージ本文に書き込む int 値

備考

整数パラメータは 4 バイトの値で示され、上位のバイトから追加されます。

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「ReadInt メソッド」 281 ページ

WriteLong メソッド

QABinaryMessge インスタンスのメッセージ本文に long 値を追加します。

構文

Visual Basic

```
Public Sub WriteLong( _
```



```
  ByVal val As Long _
)

C#
public void WriteLong(
  long val
);
```

パラメータ

- **val**　メッセージ本文に書き込む long 値

備考

long パラメータは 8 バイトの値で示され、上位のバイトから追加されます。

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「ReadLong メソッド」 281 ページ

WriteSbyte バイト

QABinaryMessge インスタンスのメッセージ本文に signed byte 値を追加します。

構文

```
Visual Basic
Public Sub WriteSbyte( _
  ByVal val As System.SByte _
)

C#
public void WriteSbyte(
  System.Sbyte val
);
```

パラメータ

- **val**　メッセージ本文に書き込む signed byte 値

備考

signed byte 値は 1 バイトの値で示されます。

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「ReadSbyte メソッド」 282 ページ

WriteShort メソッド

QABinaryMessge インスタンスのメッセージ本文に short 値を追加します。

構文

Visual Basic
```
Public Sub WriteShort( _
  ByVal val As Short _
)
```

C#
```
public void WriteShort(
  short val
);
```

パラメータ

- **val**　メッセージ本文に書き込む short 値

備考

short パラメータは 2 バイトの値で示され、上位のバイトから追加されます。

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「ReadShort メソッド」 282 ページ

WriteString メソッド

QABinaryMessge インスタンスのメッセージ本文に文字列値を追加します。

構文

Visual Basic
```
Public Sub WriteString( _
  ByVal val As String _
)
```

C#
```
public void WriteString(
  string  val
);
```

パラメータ

- **val**　メッセージ本文に書き込む文字列値

備考

注意 : 受信側アプリケーションは、WriteString が呼び出されるたびに ReadString メソッドを呼び出す必要があります。

注意 : 文字列の UTF-8 表記は、最大で 32767 バイトまで書き込み可能です。

参照

- 「QABinaryMessage インタフェース」 276 ページ
- 「QABinaryMessage のメンバ」 276 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「ReadString メソッド」 283 ページ

QAException クラス

QAnywhere クライアント・アプリケーションの例外をカプセル化します。QAException クラスを使用して QAnywhere の例外を受信できます。

構文

Visual Basic
```
Public Class QAException
  Inherits ApplicationException
```

C#
```
public class QAException :
  ApplicationException
```

QAException のメンバ

パブリック・コンストラクタ

メンバ名	説明
QAException コンストラクタ	エラー・メッセージ・テキストを提供する QAException インスタンスを作成します。
QAException コンストラクタ	エラー・コードとエラー・メッセージ・テキストを提供する QAException インスタンスを作成します。

パブリック・プロパティ

メンバ名	説明
ErrorCode プロパティ	例外のエラー・コードです。
HelpLink (Exception から継承)	この例外に関連付けられているヘルプ・ファイルへのリンクを取得または設定します。
InnerException (Exception から継承)	現在の例外を発生させた System.Exception インスタンスを取得します。
Message (Exception から継承)	現在の例外を説明するメッセージを取得します。

メンバ名	説明
Source (Exception から継承)	エラーを発生させたアプリケーションまたはオブジェクトの名前を取得または設定します。
StackTrace (Exception から継承)	現在の例外がスローされた時点のコール・スタックのフレームを表す文字列を取得します。
TargetSite (Exception から継承)	現在の例外をスローしたメソッドを取得します。

パブリック・メソッド

メンバ名	説明
GetBaseException (Exception から継承)	派生クラスで上書きされる場合は、後続の 1 つ以上の例外の根本原因となる System.Exception を返します。
GetObjectData (Exception から継承)	派生クラスで上書きされる場合は、System.Runtime.Serialization.SerializationInfo に例外の情報を設定します。
ToString (Exception から継承)	現在の例外を表す文字列を作成して返します。

QAException コンストラクタ

エラー・メッセージ・テキストを提供する QAException インスタンスを作成します。

構文

Visual Basic

```
Overloads Public Sub New( _
  ByVal msg As String _
)
```

C#

```
public QAException(
  string  msg
);
```

パラメータ

- **msg**　例外のテキスト記述

QAException コンストラクタ

エラー・コードとエラー・メッセージ・テキストを提供する QAException インスタンスを作成します。

構文

Visual Basic
```
Overloads Public Sub New( _
  ByVal msg As String, _
  ByVal errCode As Integer _
)
```

C#
```
public QAException(
  string  msg,
  int errCode
);
```

パラメータ

- **msg**　例外のテキスト記述
- **errCode**　エラー・コード

ErrorCode プロパティ

例外のエラー・コードです。

構文

Visual Basic
```
Public Readonly Property ErrorCode As Integer
```

C#
```
public int ErrorCode {get;}
```

QAManager インタフェース

QAManager クラスは QAManagerBase から派生し、非トランザクション志向の QAnywhere メッセージング操作を管理します。

構文

Visual Basic
```
Public Interface QAManager
```

C#
```
public interface QAManager
```

備考

この動作の完全な説明については、QAManagerBase インタフェースを参照してください。

QAManager は、AcknowledgementMode クラスの定義のようにして、明示的に受信を確認するように設定することも、暗黙的に受信を確認するように設定することもできます。トランザクションの一部としてメッセージの受信を確認するには、QATransactionalManager を使用します。QAManager と QATransactionalManager オブジェクトを作成するには、QAManagerFactory を使用します。

参照

- 「QAManager のメンバ」 293 ページ
- 「AcknowledgementMode 列挙」 262 ページ
- 「QATransactionalManager インタフェース」 367 ページ

QAManager のメンバ

パブリック・メソッド

メンバ名	説明
Acknowledge モード	クライアント・アプリケーションが QAnywhere メッセージを正常に受信したことを確認します。
AcknowledgeAll メソッド	クライアント・アプリケーションが QAnywhere メッセージを正常に受信したことを確認します。受信確認されていないメッセージが、すべて受信確認されます。
AcknowledgeUntil メソッド	指定された QAMessage インスタンスと、指定されたメッセージよりも前に受信されて受信確認されていないメッセージについて、すべて受信確認します。
Open メソッド	指定された AcknowledgementMode 値を使用して QAManager をオープンします。
Recover メソッド	受信確認されていないメッセージを、すべて強制的に未受信に戻します。

Acknowledge モード

クライアント・アプリケーションが QAnywhere メッセージを正常に受信したことを確認します。

構文

Visual Basic

```
Public Sub Acknowledge( _
  ByVal msg As QAMessage _
)
```

C#

```
public void Acknowledge(
  QAMessage msg
);
```

パラメータ

- **msg**　受信確認するメッセージ

備考

注意 : QAMessage が受信確認されると、そのステータス・プロパティは StatusCodes.RECEIVED に変わります。QAMessage の MessageProperties.STATUS のメッセージ・プロパティが

StatusCodes.RECEIVED に変わると、デフォルトの削除ルールを使用して削除できるようになります。

削除ルールの詳細については、「メッセージの削除ルール」 256 ページを参照してください。

例外

- QAException クラス - メッセージの受信確認時に問題が発生した場合にスローされます。

参照

- 「QAManager インタフェース」 292 ページ
- 「QAManager のメンバ」 293 ページ
- 「QAManager インタフェース」 292 ページ
- 「AcknowledgeUntil メソッド」 295 ページ
- 「StatusCodes 列挙」 370 ページ
- 「MessageProperties クラス」 264 ページ
- 「AcknowledgeAll メソッド」 294 ページ

AcknowledgeAll メソッド

クライアント・アプリケーションが QAnywhere メッセージを正常に受信したことを確認します。受信確認されていないメッセージが、すべて受信確認されます。

構文

Visual Basic
Public Sub **AcknowledgeAll()**

C#
public void **AcknowledgeAll();**

備考

注意 : QAMessage が受信確認されると、その MessageProperties.STATUS プロパティは StatusCodes.RECEIVED に変わります。QAMessage は、ステータスが StatusCodes.RECEIVED に変わると、デフォルトの削除ルールを使用して削除できるようになります。

削除ルールの詳細については、「メッセージの削除ルール」 256 ページを参照してください。

例外

- QAException クラス - メッセージの受信確認で問題が発生した場合にスローされます。

参照

- 「QAManager インタフェース」 292 ページ
- 「QAManager のメンバ」 293 ページ
- 「QAManager インタフェース」 292 ページ
- 「Acknowledge モード」 293 ページ
- 「AcknowledgeUntil メソッド」 295 ページ
- 「StatusCodes 列挙」 370 ページ
- 「MessageProperties クラス」 264 ページ

AcknowledgeUntil メソッド

指定された QAMessage インスタンスと、指定されたメッセージよりも前に受信されて受信確認されていないメッセージについて、すべて受信確認します。

構文

Visual Basic
```
Public Sub AcknowledgeUntil( _
  ByVal msg As QAMessage _
)
```

C#
```
public void AcknowledgeUntil(
  QAMessage msg
);
```

パラメータ

- **msg** 受信確認するメッセージのうち最新のもの。それよりも以前の受信確認されていないメッセージも、すべて受信確認されます。

備考

注意 : QAMessage が受信確認されると、その MessageProperties.STATUS プロパティは StatusCodes.RECEIVED に変わります。QAMessage は、ステータスが StatusCodes.RECEIVED に変わると、デフォルトの削除ルールを使用して削除できるようになります。

削除ルールの詳細については、「メッセージの削除ルール」 256 ページを参照してください。

例外

- QAException クラス - メッセージの受信確認で問題が発生した場合にスローされます。

参照

- 「QAManager インタフェース」 292 ページ
- 「QAManager のメンバ」 293 ページ
- 「QAManager インタフェース」 292 ページ
- 「Acknowledge モード」 293 ページ
- 「AcknowledgeAll メソッド」 294 ページ
- 「StatusCodes 列挙」 370 ページ
- 「MessageProperties クラス」 264 ページ

Open メソッド

指定された AcknowledgementMode 値を使用して QAManager をオープンします。

構文

Visual Basic
```
Public Sub Open( _
  ByVal mode As AcknowledgementMode _
)
```


C#
public void **Open(**
  AcknowledgementMode *mode*
);

パラメータ

- **mode**　受信確認モード。AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT または AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT のどちらかです。

備考

Open メソッドは、QAManager を作成した後、最初に呼び出す必要があるメソッドです。

例外

- QAException クラス - QAManager インスタンスのオープンで問題がある場合にスローされます。

参照

- 「QAManager インタフェース」 292 ページ
- 「QAManager のメンバ」 293 ページ
- 「QAManager インタフェース」 292 ページ

Recover メソッド

受信確認されていないメッセージを、すべて強制的に未受信に戻します。

構文

Visual Basic
Public Sub **Recover()**

C#
public void **Recover();**

備考

該当するメッセージは、QAManagerBase.GetMessage を使用して再度受信する必要があります。

例外

- QAException クラス - メッセージのステータスを戻すときに問題が発生した場合にスローされます。

参照

- 「QAManager インタフェース」 292 ページ
- 「QAManager のメンバ」 293 ページ
- 「QAManager インタフェース」 292 ページ
- 「GetMessage メソッド」 312 ページ

QAManagerBase インタフェース

このクラスは、QATransactionalManager と QAManager の基本クラスです。前者の派生クラスはトランザクション志向のメッセージングを、後者の派生クラスは非トランザクション志向のメッセージングを管理します。

構文

Visual Basic
Public Interface **QAManagerBase**

C#
public interface **QAManagerBase**

備考

QAManagerBase インスタンスがメッセージを受信できるようにするには、QAManagerBase.Start() メソッドを使用します。QAManagerBase インスタンスは、アプリケーションのスレッドごとに 1 つだけにします。

このクラスのインスタンスを使用して、QAnywhere メッセージの作成と管理を行うことができます。適切な QAMessage インスタンスを作成するには、QAManagerBase.CreateBinaryMessage() メソッドと QAManagerBase.CreateTextMessage() メソッドを使用します。QAMessage インスタンスには、メッセージの内容とプロパティを設定するための、さまざまなメソッドがあります。

QAnywhere メッセージを送信するには、QAManagerBase.PutMessage メソッドを使用して、アドレス指定されたメッセージをローカルのメッセージ・ストア・キューに登録します。メッセージは、転送ポリシーに基づいて QAnywhere Agent によって転送されるか、QAManagerBase.TriggerSendReceive() が呼び出されたときに転送されます。

qaagent 転送ポリシーの詳細については、「クライアントにメッセージを転送するタイミングの決定」 40 ページを参照してください。

QAManagerBase.Close メソッドを使用して QAManagerBase インスタンスがクローズされると、メッセージがメモリから解放されます。

QAException が発生した場合にエラー情報を返すには、QAManagerBase.LastError と QAManagerBase.LastErrorMessage を使用します。エラー情報は、QAException オブジェクトから取得することもできます。

QAManagerBase にも、メッセージ・ストア・プロパティを設定および取得するためのメソッドがあります。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページと「MessageStoreProperties クラス」を参照してください。

参照

- 「QAManagerBase のメンバ」 298 ページ
- 「CreateBinaryMessage メソッド」 307 ページ
- 「TriggerSendReceive メソッド」 338 ページ
- 「Close メソッド」 307 ページ
- 「LastError プロパティ」 301 ページ

- 「LastErrorMessage プロパティ」 301 ページ
- 「QAException クラス」 290 ページ

QAManagerBase のメンバ

パブリック・プロパティ

メンバ名	説明
LastError プロパティ	最後に実行された QAManagerBase メソッドに関連付けられているエラー・コードです。
LastErrorMessage プロパティ	最後に実行された QAManagerBase メソッドに関連付けられているエラー・テキストです。
Mode プロパティ	受信したメッセージの QAManager 受信確認モードを返します。

パブリック・メソッド

メンバ名	説明
BrowseMessages メソッド	メッセージ・ストア内にある、参照可能なメッセージをすべて参照します。
BrowseMessages メソッド	この方法は使用されなくなりました。代わりに BrowseMessagesByQueue(string) メソッドを使用してください。
BrowseMessagesByID メソッド	指定されたメッセージ ID を持つメッセージを参照します。
BrowseMessagesByQueue メソッド	指定されたアドレスに送信され、次以降に取得可能な一連の受信待機中メッセージを参照します。
BrowseMessagesBySelector メソッド	メッセージ・ストアのキューに登録されているメッセージのうち、指定されたセレクタを満たすメッセージを参照します。
CancelMessage メソッド	指定されたメッセージ ID を持つメッセージをキャンセルします。
Close メソッド	QAnywhere メッセージ・システムへの接続をクローズして、QAManagerBase で使用していたリソースをすべて解放します。
CreateBinaryMessage メソッド	QABinaryMessage オブジェクトを作成します。
CreateTextMessage メソッド	QATextMessage オブジェクトを作成します。
GetBooleanStoreProperty メソッド	事前定義済みまたはカスタムのメッセージ・ストア・プロパティの boolean 値を取得します。
GetDoubleStoreProperty メソッド	事前定義済みまたはカスタムのメッセージ・ストア・プロパティの double 値を取得します。

メンバ名	説明
GetFloatStoreProperty メソッド	事前定義済みまたはカスタムのメッセージ・ストア・プロパティの float 値を取得します。
GetIntStoreProperty メソッド	事前定義済みまたはカスタムのメッセージ・ストア・プロパティの int 値を取得します。
GetLongStoreProperty メソッド	事前定義済みまたはカスタムのメッセージ・ストア・プロパティの long 値を取得します。
GetMessage メソッド	指定されたアドレスに送信され、次に取得可能な QAMessage を返します。
GetMessageBySelector メソッド	指定されたアドレスに送信され、かつ指定されたセレクタを満たす、次に取得可能な QAMessage を返します。
GetMessageBySelectorNoWait メソッド	指定されたアドレスに送信され、かつ指定されたセレクタを満たす、次に取得可能な QAMessage を返します。
GetMessageBySelectorTimeout メソッド	指定されたアドレスに送信され、かつ指定されたセレクタを満たす、次に取得可能な QAMessage を返します。
GetMessageNoWait メソッド	指定されたアドレスに送信され、次に取得可能な QAMessage を返します。
GetMessageTimeout メソッド	指定されたアドレスに送信され、次に取得可能な QAMessage を返します。
GetQueueDepth メソッド	指定されたフィルタに基づいて、キューの深さを返します。
GetQueueDepth メソッド	指定されたフィルタに基づいて、すべてのキューの深さの合計を返します。
GetSbyteStoreProperty メソッド	事前定義済みまたはカスタムのメッセージ・ストア・プロパティの signed byte 値を取得します。
GetShortStoreProperty メソッド	事前定義済みまたはカスタムのメッセージ・ストア・プロパティの short 値を取得します。
GetStoreProperty メソッド	メッセージ・ストア・プロパティを示す System.Object を取得します。
GetStorePropertyNames メソッド	メッセージ・ストアのプロパティ名の列挙子を取得します。
GetStringStoreProperty メソッド	事前定義済みまたはカスタムのメッセージ・ストア・プロパティの文字列値を取得します。
PutMessage メソッド	別の QAnywhere クライアントに送信するメッセージを準備します。
PutMessageTimeToLive メソッド	別の QAnywhere クライアントに送信するメッセージを準備します。

メンバ名	説明
SetBooleanStoreProperty メソッド	事前定義済みまたはカスタムのメッセージ・ストア・プロパティを boolean 値に設定します。
SetDoubleStoreProperty メソッド	事前定義済みまたはカスタムのメッセージ・ストア・プロパティを double 値に設定します。
SetExceptionListener メソッド	QAnywhere メッセージを非同期に処理中に QAExceptions を受信するように、ExceptionListener 委任委任を設定します。
SetExceptionListener2 メソッド	QAnywhere メッセージを非同期に処理中に QAExceptions を受信するように、ExceptionListener2 委任委任を設定します。
SetFloatStoreProperty メソッド	事前定義済みまたはカスタムのメッセージ・ストア・プロパティを float 値に設定します。
SetIntStoreProperty メソッド	事前定義済みまたはカスタムのメッセージ・ストア・プロパティを int 値に設定します。
SetLongStoreProperty メソッド	事前定義済みまたはカスタムのメッセージ・ストア・プロパティを long 値に設定します。
SetMessageListener メソッド	QAnywhere メッセージを非同期に受信するように、MessageListener 委任委任を設定します。
SetMessageListener2 メソッド	QAnywhere メッセージを非同期に受信するように、MessageListener2 委任委任を設定します。
SetMessageListenerBySelector メソッド	メッセージ・セレクタを指定し、QAnywhere メッセージを非同期に受信するように、MessageListener 委任委任を設定します。
SetMessageListenerBySelector2 メソッド	メッセージ・セレクタを指定し、QAnywhere メッセージを非同期に受信するように、MessageListener2 委任委任を設定します。
SetProperty メソッド	QAnywhere Manager の設定プロパティをプログラムで設定できるようにします。
SetSbyteStoreProperty メソッド	事前定義済みまたはカスタムのメッセージ・ストア・プロパティを sbyte 値に設定します。
SetShortStoreProperty メソッド	事前定義済みまたはカスタムのメッセージ・ストア・プロパティを short 値に設定します。
SetStoreProperty メソッド	事前定義済みまたはカスタムのメッセージ・ストア・プロパティを System.Object 値に設定します。
SetStringStoreProperty メソッド	事前定義済みまたはカスタムのメッセージ・ストア・プロパティを文字列値に設定します。
Start メソッド	着信メッセージをメッセージ・リスナで受信するための QAManagerBase を開始します。

メンバ名	説明
Stop メソッド	QAManagerBase による着信メッセージの受信を停止します。
TriggerSendReceive メソッド	QAnywhere メッセージ・サーバとの同期処理を発生させて、他のクライアント宛てのメッセージをアップロードし、ローカル・クライアント宛てのメッセージをダウンロードします。

LastError プロパティ

最後に実行された QAManagerBase メソッドに関連付けられているエラー・コードです。

構文

Visual Basic
Public Readonly Property **LastError** As Integer

C#
public int **LastError** {get;}

戻り値

エラー・コード

備考

値が 0 の場合、エラーはありません。このプロパティは、QAException クラスの受信後に取得できます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「QAException クラス」 290 ページ

LastErrorMessage プロパティ

最後に実行された QAManagerBase メソッドに関連付けられているエラー・テキストです。

構文

Visual Basic
Public Readonly Property **LastErrorMessage** As String

C#
public string **LastErrorMessage** {get;}

備考

LastError プロパティが 0 の場合、この値は null になります。このプロパティは、QAException クラスの受信後に取得できます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「QAException クラス」 290 ページ

Mode プロパティ

受信したメッセージの QAManager 受信確認モードを返します。

構文

Visual Basic
Public Readonly Property **Mode** As AcknowledgementMode

C#
public AcknowledgementMode **Mode** {get;}

備考

使用可能な値のリストについては、AcknowledgementMode 列挙を参照してください。AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT と AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT は QAManager インスタンスで使用されます。AcknowledgementMode.TRANSACTIONAL は QATransactionalManager インスタンス用のモードです。

BrowseMessages メソッド

メッセージ・ストア内にある、参照可能なメッセージをすべて参照します。

構文

Visual Basic
Overloads Public Function **BrowseMessages()** As System.Collections.IEnumerator

C#
public System.Collections.IEnumerator **BrowseMessages();**

戻り値

参照可能な一連のメッセージの列挙子

備考

メッセージは単に参照されるだけなので、受信確認はできません。メッセージを参照するとネイティブのリソースが割り当てられるため、参照後は列挙子の Reset() メソッドを呼び出す必要があります。このメソッドを呼び出さないと、この QAManagerBase オブジェクトが解放されないかぎりネイティブのリソースも解放されません。

メッセージを受信して受信確認できるようにする場合は、QAManagerBase.GetMessage を使用します。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「BrowseMessagesByQueue メソッド」 304 ページ
- 「BrowseMessagesByID メソッド」 304 ページ
- 「BrowseMessages メソッド」 303 ページ

BrowseMessages メソッド

この方法は使用されなくなりました。代わりに BrowseMessagesByQueue(string) メソッドを使用してください。

構文

Visual Basic

```
Overloads Public Function BrowseMessages( _
  ByVal address As String _
) As System.Collections.IEnumerator
```

C#

```
public System.Collections.IEnumerator BrowseMessages(
  string  address
);
```

パラメータ

- **address**　メッセージのアドレス

戻り値

参照可能な一連のメッセージの列挙子

備考

指定されたアドレスに送信され、次以降に取得可能な一連の受信待機中メッセージを参照します。address パラメータは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。メッセージは単に参照されるだけなので、受信確認はできません。

メッセージを参照するとネイティブのリソースが割り当てられるため、参照後は列挙子の Reset() メソッドを呼び出す必要があります。このメソッドを呼び出さないと、この QAManagerBase オブジェクトが解放されないかぎりネイティブのリソースも解放されません。

メッセージを受信して受信確認できるようにする場合は、QAManagerBase.GetMessage を使用します。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「BrowseMessagesByQueue メソッド」 304 ページ
- 「BrowseMessagesByID メソッド」 304 ページ
- 「BrowseMessagesBySelector メソッド」 305 ページ
- 「BrowseMessages メソッド」 303 ページ

BrowseMessagesByID メソッド

指定されたメッセージ ID を持つメッセージを参照します。

構文

Visual Basic
```
Public Function BrowseMessagesByID( _
  ByVal msgid As String _
) As System.Collections.IEnumerator
```

C#
```
public System.Collections.IEnumerator BrowseMessagesByID(
  string  msgid
);
```

パラメータ

- **msgid**　メッセージのメッセージ ID

戻り値

0 または 1 のメッセージを含む列挙子

備考

メッセージは単に参照されるだけなので、受信確認はできません。メッセージを参照するとネイティブのリソースが割り当てられるため、参照後は列挙子の Reset() メソッドを呼び出す必要があります。このメソッドを呼び出さないと、この QAManagerBase オブジェクトが解放されないかぎりネイティブのリソースも解放されません。

メッセージを受信して受信確認できるようにする場合は、QAManagerBase.GetMessage を使用します。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「BrowseMessagesByQueue メソッド」 304 ページ
- 「BrowseMessages メソッド」 302 ページ
- 「BrowseMessages メソッド」 303 ページ

BrowseMessagesByQueue メソッド

指定されたアドレスに送信され、次以降に取得可能な一連の受信待機中メッセージを参照します。

構文

Visual Basic
```
Public Function BrowseMessagesByQueue( _
  ByVal address As String _
) As System.Collections.IEnumerator
```

C#
```
public System.Collections.IEnumerator BrowseMessagesByQueue(
  string address
);
```

パラメータ

- **address**　メッセージのアドレス

戻り値

参照可能な一連のメッセージの列挙子

備考

メッセージは単に参照されるだけなので、受信確認はできません。メッセージを参照するとネイティブのリソースが割り当てられるため、参照後は列挙子の Reset() メソッドを呼び出す必要があります。このメソッドを呼び出さないと、この QAManagerBase オブジェクトが解放されないかぎりネイティブのリソースも解放されません。

メッセージを受信して受信確認できるようにする場合は、QAManagerBase.GetMessage を使用します。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「BrowseMessagesByID メソッド」 304 ページ
- 「BrowseMessages メソッド」 302 ページ
- 「BrowseMessages メソッド」 303 ページ

BrowseMessagesBySelector メソッド

メッセージ・ストアのキューに登録されているメッセージのうち、指定されたセレクタを満たすメッセージを参照します。

構文

Visual Basic
```
Public Function BrowseMessagesBySelector( _
  ByVal selector As String _
) As System.Collections.IEnumerator
```

C#
```
public System.Collections.IEnumerator BrowseMessagesBySelector(
  string selector
);
```

パラメータ

- **selector**　セレクタ

戻り値

参照可能な一連のメッセージの列挙子

備考

メッセージは単に参照されるだけなので、受信確認はできません。メッセージを参照するとネイティブのリソースが割り当てられるため、参照後は列挙子の Reset() メソッドを呼び出す必要があります。このメソッドを呼び出さないと、この QAManagerBase オブジェクトが解放されないかぎりネイティブのリソースも解放されません。

メッセージを受信して受信確認できるようにする場合は、QAManagerBase.GetMessage を使用します。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「BrowseMessagesByQueue メソッド」 304 ページ
- 「BrowseMessages メソッド」 302 ページ
- 「BrowseMessages メソッド」 303 ページ
- 「BrowseMessagesByID メソッド」 304 ページ

CancelMessage メソッド

指定されたメッセージ ID を持つメッセージをキャンセルします。

構文

Visual Basic

```
Public Sub CancelMessage( _
  ByVal msgid As String _
)
```

C#

```
public void CancelMessage(
  string  msgid
);
```

パラメータ

- **msgid**　キャンセルするメッセージのメッセージ ID

備考

CancelMessage は、メッセージが転送される前にメッセージをキャンセル済みの状態にします。QAnywhere Agent のデフォルトの削除ルールにより、キャンセル済みメッセージは最終的にメッセージ・ストアから削除されます。

メッセージがすでに最終ステータスになっている場合や、中央のメッセージング・サーバに転送済みである場合は、CancelMessage は失敗します。

削除ルールの詳細については、「メッセージの削除ルール」 256 ページを参照してください。

例外

- QAException クラス - メッセージのキャンセルで問題が発生した場合にスローされます。

Close メソッド

QAnywhere メッセージ・システムへの接続をクローズして、QAManagerBase で使用していたリソースをすべて解放します。

構文

Visual Basic
Public Sub **Close()**

C#
public void **Close();**

備考

いったんクローズした後は、何度この Close() メソッドを呼び出しても無視されます。このメソッドを呼び出して接続をクローズした後に Close() 以外の QAManagerBase メソッドを呼び出すと、QAException になります。この場合は、新しい QAManagerBase インスタンスを作成してからオープンします。

例外

- QAException クラス - QAManagerBase インスタンスのクローズで問題がある場合にスローされます。

CreateBinaryMessage メソッド

QABinaryMessage オブジェクトを作成します。

構文

Visual Basic
Public Function **CreateBinaryMessage()** As QABinaryMessage

C#
public QABinaryMessage **CreateBinaryMessage();**

戻り値

新しい QABinaryMessage インスタンス

備考

QABinaryMessage オブジェクトは、未解釈のバイト・ストリームのメッセージ本文が含まれるメッセージの送信に使用します。

例外

- QAException クラス - メッセージの作成で問題が発生した場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「QABinaryMessage インタフェース」 276 ページ

CreateTextMessage メソッド

QATextMessage オブジェクトを作成します。

構文

Visual Basic
Public Function **CreateTextMessage()** As QATextMessage

C#
public QATextMessage **CreateTextMessage();**

戻り値

新しい QATextMessage インスタンス

備考

QATextMessage のオブジェクトは、文字列のメッセージ本文が含まれるメッセージを送信する場合に使用します。

例外

- QAException クラス - メッセージの作成で問題が発生した場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「QATextMessage インタフェース」 364 ページ

GetBooleanStoreProperty メソッド

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの boolean 値を取得します。

構文

Visual Basic
```
Public Function GetBooleanStoreProperty( _
  ByVal propName As String _
) As Boolean
```

C#
```
public bool GetBooleanStoreProperty(
  string  propName
);
```

パラメータ

- **propName**　事前定義済みまたはカスタムのプロパティ名

戻り値

boolean 型のプロパティの値

備考

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

例外

- QAException クラス - プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「MessageStoreProperties クラス」 273 ページ

GetDoubleStoreProperty メソッド

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの double 値を取得します。

構文

Visual Basic

```
Public Function GetDoubleStoreProperty( _
  ByVal propName As String _
) As Double
```

C#

```
public double GetDoubleStoreProperty(
  string  propName
);
```

パラメータ

- **propName**　事前定義済みまたはカスタムのプロパティ名

戻り値

double 型のプロパティの値

備考

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

例外

- QAException クラス - プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「MessageStoreProperties クラス」 273 ページ

GetFloatStoreProperty メソッド

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの float 値を取得します。

構文

Visual Basic

```
Public Function GetFloatStoreProperty( _
  ByVal propName As String _
) As Single
```

C#

```
public float  GetFloatStoreProperty(
  string  propName
);
```

パラメータ

- **propName**　事前定義済みまたはカスタムのプロパティ名

戻り値

float 型のプロパティの値

備考

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

例外

- QAException クラス - プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「MessageStoreProperties クラス」 273 ページ

GetIntStoreProperty メソッド

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの int 値を取得します。

構文

Visual Basic
```
Public Function GetIntStoreProperty( _
  ByVal propName As String _
) As Integer
```

C#
```
public int GetIntStoreProperty(
  string  propName
);
```

パラメータ

- **propName**　事前定義済みまたはカスタムのプロパティ名

戻り値

int 型のプロパティの値

備考

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

例外

- QAException クラス - プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「MessageStoreProperties クラス」 273 ページ

GetLongStoreProperty メソッド

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの long 値を取得します。

構文

Visual Basic
```
Public Function GetLongStoreProperty( _
  ByVal propName As String _
) As Long
```

C#
public long **GetLongStoreProperty(**
  string *propName*
);

パラメータ

- **propName**　事前定義済みまたはカスタムのプロパティ名

戻り値

long 型のプロパティの値

備考

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

例外

- QAException クラス - プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「MessageStoreProperties クラス」 273 ページ

GetMessage メソッド

指定されたアドレスに送信され、次に取得可能な QAMessage を返します。

構文

Visual Basic
Public Function **GetMessage(_**
  ByVal *address* As String _
) As QAMessage

C#
public QAMessage **GetMessage(**
  string *address*
);

パラメータ

- **address**　メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。

戻り値

該当する次の QAMessage。メッセージが存在しない場合は null。

備考

address パラメータは、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。

該当するメッセージが存在しない場合、新しいメッセージが着信するまでブロックされます。メッセージを同期に受信する場合は、このメソッドを使用します。

メッセージを非同期に受信する (メッセージ・イベント・ハンドラを使用する) 方法の詳細については、「非同期的なメッセージ受信」 83 ページを参照してください。

例外

- QAException クラス - メッセージの取得で問題が発生した場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「QAMessage インタフェース」 343 ページ

GetMessageBySelector メソッド

指定されたアドレスに送信され、かつ指定されたセレクタを満たす、次に取得可能な QAMessage を返します。

構文

Visual Basic

```
Public Function GetMessageBySelector( _
  ByVal address As String, _
  ByVal selector As String _
) As QAMessage
```

C#

```
public QAMessage GetMessageBySelector(
  string  address,
  string  selector
);
```

パラメータ

- **address**　このアドレスは、メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。
- **selector**　セレクタ

戻り値

該当する次の QAMessage。メッセージが存在しない場合は null。

備考

address パラメータは、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。

該当するメッセージが存在しない場合、新しいメッセージが着信するまでブロックされます。メッセージを同期に受信する場合は、このメソッドを使用します。

メッセージを非同期に受信する (メッセージ・イベント・ハンドラを使用する) 方法の詳細については、「非同期的なメッセージ受信」 83 ページを参照してください。

例外

- QAException クラス - メッセージの取得で問題が発生した場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「QAMessage インタフェース」 343 ページ

GetMessageBySelectorNoWait メソッド

指定されたアドレスに送信され、かつ指定されたセレクタを満たす、次に取得可能な QAMessage を返します。

構文

Visual Basic

```
Public Function GetMessageBySelectorNoWait( _
  ByVal address As String, _
  ByVal selector As String _
) As QAMessage
```

C#

```
public QAMessage GetMessageBySelectorNoWait(
  string  address,
  string  selector
);
```

パラメータ

- **address**　メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。
- **selector**　セレクタ

戻り値

次の取得可能なメッセージ。該当するメッセージが存在しない場合は null。

備考

address パラメータは、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージがない場合は、このメソッドがすぐに返されます。メッセージを同期に受信する場合は、このメソッドを使用します。

メッセージを非同期に受信する (メッセージ・イベント・ハンドラを使用する) 方法の詳細については、「非同期的なメッセージ受信」 83 ページを参照してください。

例外

- QAException クラス - メッセージの取得で問題が発生した場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「QAMessage インタフェース」 343 ページ

GetMessageBySelectorTimeout メソッド

指定されたアドレスに送信され、かつ指定されたセレクタを満たす、次に取得可能な QAMessage を返します。

構文

Visual Basic

```
Public Function GetMessageBySelectorTimeout( _
  ByVal address As String, _
  ByVal selector As String, _
  ByVal timeout As Long _
) As QAMessage
```

C#

```
public QAMessage GetMessageBySelectorTimeout(
  string  address,
  string  selector,
  long timeout
);
```

パラメータ

- **address**　メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。
- **selector**　セレクタ
- **timeout**　メッセージ着信を待機する時間 (ミリ秒単位)

戻り値

該当する次の QAMessage。メッセージが存在しない場合は null。

備考

address パラメータは、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージがない場合、このメソッドは指定されたタイムアウト時間だけ待機してから返されます。メッセージを同期に受信する場合は、このメソッドを使用します。

メッセージを非同期に受信する (メッセージ・イベント・ハンドラを使用する) 方法の詳細については、「非同期的なメッセージ受信」 83 ページを参照してください。

例外

- QAException クラス - メッセージの取得で問題が発生した場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「QAMessage インタフェース」 343 ページ

GetMessageNoWait メソッド

指定されたアドレスに送信され、次に取得可能な QAMessage を返します。

構文

Visual Basic

```
Public Function GetMessageNoWait( _
  ByVal address As String _
) As QAMessage
```

C#

```
public QAMessage GetMessageNoWait(
  string  address
);
```

パラメータ

- **address**　このアドレスは、メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。

戻り値

次の取得可能なメッセージ。該当するメッセージが存在しない場合は null。

備考

address パラメータは、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージがない場合は、このメソッドがすぐに返されます。メッセージを同期に受信する場合は、このメソッドを使用します。メッセージを非同期に受信する (メッセージ・イベント・ハンドラを使用する) 方法の詳細については、「非同期的なメッセージ受信」 ページを参照してください。

例外

- QAException クラス - メッセージの取得で問題が発生した場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「QAMessage インタフェース」 343 ページ

GetMessageTimeout メソッド

指定されたアドレスに送信され、次に取得可能な QAMessage を返します。

構文

Visual Basic

```
Public Function GetMessageTimeout( _
  ByVal address As String, _
  ByVal timeout As Long _
) As QAMessage
```

C#

```
public QAMessage GetMessageTimeout(
  string  address,
  long timeout
);
```

パラメータ

- **address**　メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。
- **timeout**　メッセージ着信を待機する時間 (ミリ秒単位)

戻り値

該当する次の QAMessage。メッセージが存在しない場合は null。

備考

address パラメータは、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。

該当するメッセージがない場合、このメソッドは指定されたタイムアウト時間だけ待機してから返されます。メッセージを同期に受信する場合は、このメソッドを使用します。

メッセージを非同期に受信する (メッセージ・イベント・ハンドラを使用する) 方法の詳細については、「非同期的なメッセージ受信」 83 ページを参照してください。

例外

- QAException クラス - メッセージの取得で問題が発生した場合にスローされます。

GetQueueDepth メソッド

指定されたフィルタに基づいて、キューの深さを返します。

構文

Visual Basic

```
Overloads Public Function GetQueueDepth( _
  ByVal address As String, _
  ByVal filter As QueueDepthFilter _
) As Integer
```


C#
```
public int GetQueueDepth(
  string  address,
  QueueDepthFilter filter
);
```

パラメータ

- **filter**　着信メッセージ、送信メッセージ、またはすべてのメッセージを示すフィルタ
- **address**　キュー名

戻り値

メッセージ数

備考

キューの深さは、受信されていないメッセージの数です (たとえば、QAManagerBase.GetMessage を使用)。

例外

- QAException クラス - エラーが発生した場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「QueueDepthFilter 列挙」 369 ページ

GetQueueDepth メソッド

指定されたフィルタに基づいて、すべてのキューの深さの合計を返します。

構文

Visual Basic
```
Overloads Public Function GetQueueDepth( _
  ByVal filter As QueueDepthFilter _
) As Integer
```

C#
```
public int GetQueueDepth(
  QueueDepthFilter filter
);
```

パラメータ

- **filter**　着信メッセージ、送信メッセージ、またはすべてのメッセージを示すフィルタ

戻り値

メッセージ数

備考

キューの深さは、受信されていないメッセージの数です (たとえば、QAManagerBase.GetMessage を使用)。

例外

- QAException クラス - エラーが発生した場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「QueueDepthFilter 列挙」 369 ページ

GetSbyteStoreProperty メソッド

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの signed byte 値を取得します。

構文

Visual Basic

```
Public Function GetSbyteStoreProperty( _
  ByVal propName As String _
) As System.SByte
```

C#

```
public System.Sbyte GetSbyteStoreProperty(
  string  propName
);
```

パラメータ

- **propName**　事前定義済みまたはカスタムのプロパティ名

戻り値

signed byte 型のプロパティの値

備考

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

例外

- QAException クラス - プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「MessageStoreProperties クラス」 273 ページ

GetShortStoreProperty メソッド

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの short 値を取得します。

構文

Visual Basic

```
Public Function GetShortStoreProperty( _
  ByVal propName As String _
) As Short
```

C#

```
public short GetShortStoreProperty(
  string  propName
);
```

パラメータ

- **propName**　事前定義済みまたはカスタムのプロパティ名

戻り値

short 型のプロパティの値

備考

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

例外

- QAException クラス - プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「MessageStoreProperties クラス」 273 ページ

GetStoreProperty メソッド

メッセージ・ストア・プロパティを示す System.Object を取得します。

構文

Visual Basic
```
Public Function GetStoreProperty( _
  ByVal propName As String _
) As Object
```

C#
```
public object GetStoreProperty(
  string  propName
);
```

パラメータ

- **propName**　事前定義済みまたはカスタムのプロパティ名

戻り値

プロパティの値

備考

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

例外

- QAException クラス - 該当するプロパティが存在しない場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「MessageStoreProperties クラス」 273 ページ

GetStorePropertyNames メソッド

メッセージ・ストアのプロパティ名の列挙子を取得します。

構文

Visual Basic
```
Public Function GetStorePropertyNames() As System.Collections.IEnumerator
```

C#
```
public System.Collections.IEnumerator GetStorePropertyNames();
```


戻り値

メッセージ・ストアのプロパティ名の列挙子

備考

クライアント・ストア・プロパティの詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

GetStringStoreProperty メソッド

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの文字列値を取得します。

構文

Visual Basic

```
Public Function GetStringStoreProperty( _
  ByVal propName As String _
) As String
```

C#

```
public string  GetStringStoreProperty(
  string  propName
);
```

パラメータ

- **propName**　事前定義済みまたはカスタムのプロパティ名

戻り値

文字列型のプロパティの値。該当するプロパティが存在しない場合は null。

備考

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「MessageStoreProperties クラス」 273 ページ

PutMessage メソッド

別の QAnywhere クライアントに送信するメッセージを準備します。

構文

Visual Basic
```
Public Sub PutMessage( _
  ByVal address As String, _
  ByVal msg As QAMessage _
)
```

C#
```
public void PutMessage(
  string  address,
  QAMessage msg
);
```

パラメータ

- **address**　送信先のキュー名を指定するメッセージのアドレス
- **msg**　転送用にローカル・メッセージ・ストアに登録するメッセージ

備考

PutMessage メソッドは、メッセージと送信先アドレスをローカル・メッセージ・ストアに挿入します。メッセージが転送されるタイミングは、QAnywhere Agent の転送ポリシーで決まります。

詳細については、「クライアントにメッセージを転送するタイミングの決定」 40 ページを参照してください。

アドレスは 'id¥queue-name' の形式で指定します。'id' は送信先メッセージ・ストアの ID、'queue-name' は送信先の QAnywhere クライアントがメッセージの受信で使用するキューを特定します。

QAnywhere アドレスの詳細については、「QAnywhere メッセージ・アドレス」 56 ページを参照してください。

例外

- QAException クラス - メッセージの登録で問題が発生した場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「PutMessageTimeToLive メソッド」 323 ページ

PutMessageTimeToLive メソッド

別の QAnywhere クライアントに送信するメッセージを準備します。

構文

Visual Basic
```
Public Sub PutMessageTimeToLive( _
  ByVal address As String, _
  ByVal msg As QAMessage, _
  ByVal ttl As Long _
)
```


C#
```
public void PutMessageTimeToLive(
  string  address,
  QAMessage msg,
  long ttl
);
```

パラメータ

- **address**　送信先のキュー名を指定するメッセージのアドレス
- **msg**　登録するメッセージ
- **ttl**　メッセージの有効期限 (ミリ秒単位)。この期限を過ぎても配信されなかったメッセージは期限切れになります。値 0 は、有効期限なしを意味します。

備考

PutMessageTimeToLive メソッドは、メッセージと送信先アドレスをローカル・メッセージ・ストアに挿入します。メッセージが転送されるタイミングは、QAnywhere Agent の転送ポリシーで決まります。ただし、次のメッセージの転送時間が指定された存続時間を超えると、そのメッセージは期限切れになります。

詳細については、「クライアントにメッセージを転送するタイミングの決定」 40 ページを参照してください。

アドレスは 'id¥queue-name' の形式で指定します。'id' は送信先メッセージ・ストアの ID、'queue-name' は送信先の QAnywhere クライアントがメッセージの受信で使用するキューを特定します。

QAnywhere アドレスの詳細については、「QAnywhere メッセージ・アドレス」 56 ページを参照してください。

例外

- QAException クラス - メッセージの登録で問題が発生した場合にスローされます。

SetBooleanStoreProperty メソッド

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを boolean 値に設定します。

構文

Visual Basic
```
Public Sub SetBooleanStoreProperty( _
  ByVal propName As String, _
  ByVal val As Boolean _
)
```

C#
```
public void SetBooleanStoreProperty(
  string  propName,
  bool val
);
```

パラメータ

- **propName**　事前定義済みまたはカスタムのプロパティ名
- **val**　boolean 型のプロパティの値

備考

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「MessageStoreProperties クラス」 273 ページ

SetDoubleStoreProperty メソッド

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを double 値に設定します。

構文

Visual Basic

```
Public Sub SetDoubleStoreProperty( _
  ByVal propName As String, _
  ByVal val As Double _
)
```

C#

```
public void SetDoubleStoreProperty(
  string  propName,
  double val
);
```

パラメータ

- **propName**　事前定義済みまたはカスタムのプロパティ名
- **val**　double 型のプロパティの値

備考

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「MessageStoreProperties クラス」 273 ページ

SetExceptionListener メソッド

QAnywhere メッセージを非同期に処理中に QAExceptions を受信するように、ExceptionListener 委任委任を設定します。

構文

Visual Basic

```
Public Sub SetExceptionListener( _
  ByVal address As String, _
  ByVal listener As ExceptionListener _
)
```

C#

```
public void SetExceptionListener(
  string  address,
  ExceptionListener listener
);
```

パラメータ

- **address**　メッセージのアドレス
- **listener**　登録する例外リスナ

備考

ExceptionListener 委任では、QAException と QAMessage の各パラメータを指定できます。指定したアドレスに対して ExceptionListener と MessageListener を設定できますが、Listener／Listener2 委任を一致させる必要があります。つまり、同じアドレスに対して ExceptionListener と MessageListener2、または ExceptionListener2 と MessageListener を設定することはできません。

詳細については、「非同期的なメッセージ受信」 83 ページを参照してください。

SetExceptionListener2 メソッド

QAnywhere メッセージを非同期に処理中に QAExceptions を受信するように、ExceptionListener2 委任委任を設定します。

構文

Visual Basic

```
Public Sub SetExceptionListener2( _
  ByVal address As String, _
```

```
  ByVal listener As ExceptionListener2 _
)

C#
public void SetExceptionListener2(
  string  address,
  ExceptionListener2 listener
);
```

パラメータ

- **address**　メッセージのアドレス
- **listener**　登録する例外リスナ

備考

ExceptionListener2 委任では、QAManagerBase、QAException、QAMessage の各パラメータを指定できます。指定したアドレスに対して ExceptionListener2 と MessageListener2 を設定できますが、Listener／Listener2 委任を一致させる必要があります。つまり、同じアドレスに対して ExceptionListener と MessageListener2、または ExceptionListener2 と MessageListener を設定することはできません。

詳細については、「非同期的なメッセージ受信」 83 ページを参照してください。

SetFloatStoreProperty メソッド

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを float 値に設定します。

構文

```
Visual Basic
Public Sub SetFloatStoreProperty( _
  ByVal propName As String, _
  ByVal val As Single _
)

C#
public void SetFloatStoreProperty(
  string  propName,
  float  val
);
```

パラメータ

- **propName**　事前定義済みまたはカスタムのプロパティ名
- **val**　float 型のプロパティの値

備考

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「MessageStoreProperties クラス」 273 ページ

SetIntStoreProperty メソッド

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを int 値に設定します。

構文

Visual Basic

```
Public Sub SetIntStoreProperty( _
  ByVal propName As String, _
  ByVal val As Integer _
)
```

C#

```
public void SetIntStoreProperty(
  string  propName,
  int val
);
```

パラメータ

- **propName**　事前定義済みまたはカスタムのプロパティ名
- **val**　int 型のプロパティの値

備考

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「MessageStoreProperties クラス」 273 ページ

SetLongStoreProperty メソッド

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを long 値に設定します。

構文

Visual Basic
```
Public Sub SetLongStoreProperty( _
  ByVal propName As String, _
  ByVal val As Long _
)
```

C#
```
public void SetLongStoreProperty(
  string  propName,
  long val
);
```

パラメータ

- **propName**　事前定義済みまたはカスタムのプロパティ名
- **val**　long 型のプロパティの値

備考

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「MessageStoreProperties クラス」 273 ページ

SetMessageListener メソッド

QAnywhere メッセージを非同期に受信するように、MessageListener 委任委任を設定します。

構文

Visual Basic
```
Public Sub SetMessageListener( _
  ByVal address As String, _
  ByVal listener As MessageListener _
)
```

C#
```
public void SetMessageListener(
```

```
  string  address,
  MessageListener listener
);
```

パラメータ

- **address**　メッセージのアドレス
- **listener**　登録するリスナ

備考

メッセージを非同期に受信する場合は、このメソッドを使用します。

MessageListener 委任では、QAMessage パラメータを 1 つだけ指定できます。

SetMessageListener の address パラメータは、メッセージの受信で使用されるローカル・キュー名を指定します。指定されたキューには、1 つのリスナ委任だけを割り当てることができます。指定したアドレスに対して ExceptionListener と MessageListener を設定できますが、Listener／Listener2 委任を一致させる必要があります。つまり、同じアドレスに対して ExceptionListener と MessageListener2、または ExceptionListener2 と MessageListener を設定することはできません。

Push 通知とネットワーク・ステータス変更も含め、QAnywhere システム・メッセージを受信したい場合は、キュー名に "system" を指定します。

詳細については、「非同期的なメッセージ受信」 83 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「MessageListener 委任」 264 ページ

SetMessageListener2 メソッド

QAnywhere メッセージを非同期に受信するように、MessageListener2 委任委任を設定します。

構文

Visual Basic

```
Public Sub SetMessageListener2( _
  ByVal address As String, _
  ByVal listener As MessageListener2 _
)
```

C#

```
public void SetMessageListener2(
  string  address,
  MessageListener2 listener
);
```

パラメータ

- **address**　メッセージのアドレス

- **listener**　登録するリスナ

備考

メッセージを非同期に受信する場合は、このメソッドを使用します。

MessageListener2 委任では、QAManagerBase と QAMessage の各パラメータを指定できます。

SetMessageListener2 の address パラメータは、メッセージの受信で使用されるローカル・キュー名を指定します。指定されたキューには、1 つのリスナ委任だけを割り当てることができます。指定したアドレスに対して ExceptionListener2 と MessageListener2 を設定できますが、Listener／Listener2 委任を一致させる必要があります。つまり、同じアドレスに対して ExceptionListener と MessageListener2、または ExceptionListener2 と MessageListener を設定することはできません。

Push 通知とネットワーク・ステータス変更も含め、QAnywhere システム・メッセージを受信したい場合は、キュー名に "system" を指定します。

詳細については、「非同期的なメッセージ受信」 83 ページを参照してください。

SetMessageListenerBySelector メソッド

メッセージ・セレクタを指定し、QAnywhere メッセージを非同期に受信するように、MessageListener 委任委任を設定します。

構文

Visual Basic

```
Public Sub SetMessageListenerBySelector( _
  ByVal address As String, _
  ByVal selector As String, _
  ByVal listener As MessageListener _
)
```

C#

```
public void SetMessageListenerBySelector(
  string  address,
  string  selector,
  MessageListener listener
);
```

パラメータ

- **address**　メッセージのアドレス
- **listener**　登録するリスナ
- **selector**　受信されるメッセージをフィルタリングするために使用するセレクタ

備考

メッセージを非同期に受信する場合は、このメソッドを使用します。

MessageListener 委任では、QAMessage パラメータを 1 つだけ指定できます。

SetMessageListener の address パラメータは、メッセージの受信で使用されるローカル・キュー名を指定します。指定されたキューには、1 つのリスナ委任だけを割り当てることができます。selector パラメータは、指定されたアドレスで受信されるメッセージをフィルタリングするために使用するセレクタを指定します。指定したアドレスに対して ExceptionListener と MessageListener を設定できますが、Listener／Listener2 委任を一致させる必要があります。つまり、同じアドレスに対して ExceptionListener と MessageListener2、または ExceptionListener2 と MessageListener を設定することはできません。

Push 通知とネットワーク・ステータス変更も含め、QAnywhere システム・メッセージを受信したい場合は、キュー名に "system" を指定します。

詳細については、「非同期的なメッセージ受信」 83 ページと「システム・キュー」 56 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「MessageListener 委任」 264 ページ

SetMessageListenerBySelector2 メソッド

メッセージ・セレクタを指定し、QAnywhere メッセージを非同期に受信するように、MessageListener2 委任委任を設定します。

構文

Visual Basic

```
Public Sub SetMessageListenerBySelector2( _
  ByVal address As String, _
  ByVal selector As String, _
  ByVal listener As MessageListener2 _
)
```

C#

```
public void SetMessageListenerBySelector2(
  string  address,
  string  selector,
  MessageListener2 listener
);
```

パラメータ

- **address**　メッセージのアドレス
- **listener**　登録するリスナ
- **selector**　受信されるメッセージをフィルタリングするために使用するセレクタ

備考

メッセージを非同期に受信する場合は、このメソッドを使用します。

MessageListener2 委任では、QAMessage パラメータを 1 つだけ指定できます。

SetMessageListener2 の address パラメータは、メッセージの受信で使用されるローカル・キュー名を指定します。指定されたキューには、1 つのリスナ委任だけを割り当てることができます。selector パラメータは、指定されたアドレスで受信されるメッセージをフィルタリングするために使用するセレクタを指定します。指定したアドレスに対して ExceptionListener2 と MessageListener2 を設定できますが、Listener／Listener2 委任を一致させる必要があります。つまり、同じアドレスに対して ExceptionListener と MessageListener2、または ExceptionListener2 と MessageListener を設定することはできません。

Push 通知とネットワーク・ステータス変更も含め、QAnywhere システム・メッセージを受信したい場合は、キュー名に "system" を指定します。

詳細については、「非同期的なメッセージ受信」 83 ページと「システム・キュー」 56 ページを参照してください。

SetProperty メソッド

QAnywhere Manager の設定プロパティをプログラムで設定できるようにします。

構文

Visual Basic

```
Public Sub SetProperty( _
  ByVal name As String, _
  ByVal val As String _
)
```

C#

```
public void SetProperty(
  string  name,
  string  val
);
```

パラメータ

- **name**　QAnywhere Manager の設定プロパティ名
- **val**　QAnywhere Manager の設定プロパティの値

備考

プロパティ名と値を指定してこのメソッドを使用することで、QAnywhere Manager のデフォルトの設定プロパティを無効にできます。プロパティのリストについては、「QAnywhere Manager の設定プロパティ」 69 ページを参照してください。

QAnywhere Manager の設定プロパティは、プロパティ・ファイルと QAManagerFactory.CreateQAManager メソッドを使用して設定することもできます。

詳細については、「QAnywhere Manager の設定プロパティをファイルに設定する」 70 ページを参照してください。*注意：* QAManager.Open または QATransactionalManager.Open() を呼び出す前に、必要なプロパティを設定してください。

例外

- QAException クラス - プロパティの設定で問題が発生した場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「Open メソッド」 295 ページ
- 「Open メソッド」 368 ページ

SetSbyteStoreProperty メソッド

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを sbyte 値に設定します。

構文

Visual Basic

```
Public Sub SetSbyteStoreProperty( _
  ByVal propName As String, _
  ByVal val As System.SByte _
)
```

C#

```
public void SetSbyteStoreProperty(
  string  propName,
  System.Sbyte val
);
```

パラメータ

- **propName**　事前定義済みまたはカスタムのプロパティ名
- **val**　sbyte 型のプロパティの値

備考

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「MessageStoreProperties クラス」 273 ページ

SetShortStoreProperty メソッド

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを short 値に設定します。

構文

Visual Basic
```
Public Sub SetShortStoreProperty( _
  ByVal propName As String, _
  ByVal val As Short _
)
```

C#
```
public void SetShortStoreProperty(
  string  propName,
  short val
);
```

パラメータ

- **propName**　事前定義済みまたはカスタムのプロパティ名
- **val**　short 型のプロパティの値

備考

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「MessageStoreProperties クラス」 273 ページ

SetStoreProperty メソッド

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを System.Object 値に設定します。

構文

Visual Basic
```
Public Sub SetStoreProperty( _
  ByVal propName As String, _
  ByVal val As Object _
)
```

C#
```
public void SetStoreProperty(
  string  propName,
  object val
);
```


パラメータ

- **propName**　事前定義済みまたはカスタムのプロパティ名
- **val**　プロパティの値

備考

プロパティ型は、使用可能ないずれかのプリミティブ型、または文字列型でなければなりません。このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「MessageStoreProperties クラス」 273 ページ

SetStringStoreProperty メソッド

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを文字列値に設定します。

構文

Visual Basic

```
Public Sub SetStringStoreProperty( _
  ByVal propName As String, _
  ByVal val As String _
)
```

C#

```
public void SetStringStoreProperty(
  string  propName,
  string  val
);
```

パラメータ

- **propName**　事前定義済みまたはカスタムのプロパティ名
- **val**　文字列型のプロパティの値

備考

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「MessageStoreProperties クラス」 273 ページ

Start メソッド

着信メッセージをメッセージ・リスナで受信するための QAManagerBase を起動します。

構文

Visual Basic
Public Sub **Start()**

C#
public void **Start();**

備考

メッセージ・リスナ・セットがない場合、たとえばメッセージが GetMessage メソッドで受信される場合などは、QAManagerBase を起動する必要がありません。GetMessage メソッドとメッセージ・リスナは、メッセージ受信用に使用しないでください。非同期 (メッセージ・リスナ) モデルまたは同期 (GetMessage) モデルのどちらかを使用してください。この Start() メソッドを繰り返し呼び出そうとしても、間に QAManagerBase.Stop() 呼び出しを挟まないかぎり 2 回目以降の呼び出しは無視されます。

例外

- QAException クラス - QAManagerBase インスタンスの起動で問題がある場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「Stop メソッド」 337 ページ

Stop メソッド

QAManagerBase による着信メッセージの受信を停止します。

構文

Visual Basic
Public Sub **Stop()**

C#
public void **Stop();**

備考

メッセージは失われません。メッセージは、Manager が再起動されるまで受信されません。この Stop() メソッドを繰り返し呼び出そうとしても、間に QAManagerBase.Start() 呼び出しを挟まないかぎり 2 回目以降の呼び出しは無視されます。

例外

- QAException クラス - QAManagerBase インスタンスの停止で問題がある場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「Start メソッド」 337 ページ

TriggerSendReceive メソッド

QAnywhere メッセージ・サーバとの同期処理を発生させて、他のクライアント宛てのメッセージをアップロードし、ローカル・クライアント宛てのメッセージをダウンロードします。

構文

Visual Basic
Public Sub **TriggerSendReceive()**

C#
public void **TriggerSendReceive();**

備考

QAManagerBase の TriggerSendReceive では、QAnywhere Agent と中央のメッセージング・サーバの間でメッセージの同期処理がただちに行われます。手動の TriggerSendReceive 呼び出しでは、QAnywhere Agent の転送ポリシーとは無関係に、メッセージ転送がただちに行われます。

QAnywhere Agent の転送ポリシーは、メッセージ転送の方法を決定します。たとえば、クライアントが Push 通知を受信した場合や、QAManagerBase.PutMessage メソッドを呼び出してメッセージを送信した場合に、一定の間隔でメッセージ転送が自動的に実行されます。

詳細については、「クライアントにメッセージを転送するタイミングの決定」 40 ページを参照してください。

例外

- QAException クラス - 送信／受信のトリガで問題が発生した場合にスローされます。

参照

- 「QAManagerBase インタフェース」 297 ページ
- 「QAManagerBase のメンバ」 298 ページ
- 「PutMessage メソッド」 322 ページ

QAManagerFactory クラス

このクラスは、QATransactionalManager オブジェクトまたは QAManager オブジェクトを作成するためのファクトリ・クラスです。

構文

Visual Basic
MustInherit Public Class **QAManagerFactory**
  Inherits Component

C#
public abstract class **QAManagerFactory** :
  Component

備考

QAManagerFactory インスタンスは 1 つだけ持つことができます。

QAManagerFactory のメンバ

パブリックの静的プロパティ (共有)

メンバ名	説明
Instance プロパティ	QAManagerFactory のシングルトン・インスタンスです。
InstanceCount プロパティ	ファクトリ・インスタンスの数を示します。

public フィールド

メンバ名	説明
InstanceID フィールド	ファクトリ ID です。

パブリック・プロパティ

メンバ名	説明
LastError プロパティ	最後に実行された QAManagerFactory メソッドに関連付けられているエラー・コードです。
LastErrorMessage プロパティ	最後に実行された QAManagerFactory メソッドに関連付けられているエラー・テキストです。

パブリック・メソッド

メンバ名	説明
CreateQAManager メソッド	指定されたプロパティを持つ新しい QAManager インスタンスを返します。

メンバ名	説明
CreateQATransactionalManager メソッド	指定されたプロパティを持つ新しい QATransactionalManager インスタンスを返します。

InstanceID フィールド

ファクトリ ID です。

構文

Visual Basic
Public**InstanceID** As Integer

C#
public int **InstanceID**;

Instance プロパティ

QAManagerFactory のシングルトン・インスタンスです。

構文

Visual Basic
Public Shared Readonly Property **Instance** As QAManagerFactory

C#
public const QAManagerFactory **Instance** {get;}

例外

- QAException クラス - マネージャ・ファクトリの作成で問題が発生した場合にスローされます。

InstanceCount プロパティ

ファクトリ・インスタンスの数を示します。

構文

Visual Basic
Public Shared Readonly Property **InstanceCount** As Long

C#
public const long **InstanceCount** {get;}

LastError プロパティ

最後に実行された QAManagerFactory メソッドに関連付けられているエラー・コードです。

構文

Visual Basic
Public Readonly Property **LastError** As Integer

C#
public int **LastError** {get;}

戻り値

エラー・コード

備考

値が 0 の場合、エラーはありません。このプロパティは、QAException クラスの受信後に取得できます。

参照

- 「QAManagerFactory クラス」 339 ページ
- 「QAManagerFactory のメンバ」 339 ページ
- 「QAException クラス」 290 ページ

LastErrorMessage プロパティ

最後に実行された QAManagerFactory メソッドに関連付けられているエラー・テキストです。

構文

Visual Basic
Public Readonly Property **LastErrorMessage** As String

C#
public string **LastErrorMessage** {get;}

戻り値

エラー・メッセージ

備考

LastError プロパティが 0 の場合、この値は null になります。このプロパティは、QAException クラスの受信後に取得できます。

参照

- 「QAManagerFactory クラス」 339 ページ
- 「QAManagerFactory のメンバ」 339 ページ
- 「QAException クラス」 290 ページ

CreateQAManager メソッド

指定されたプロパティを持つ新しい QAManager インスタンスを返します。

構文

Visual Basic
Public Function **CreateQAManager(_**
  ByVal *iniFile* As String _
) As QAManager

C#
public QAManager **CreateQAManager(**
  string *iniFile*
);

パラメータ

- **iniFile**　QAManager インスタンスを設定するためのプロパティ・ファイル

戻り値

新しい QAManager インスタンス

備考

プロパティ・ファイル・パラメータが null の場合は、デフォルトのプロパティを使用して QAManager が作成されます。インスタンスの作成後に SetProperty メソッドを使用して、QAnywhere Manager の設定プロパティをプログラムで設定できます。

QAnywhere Manager の設定プロパティのリストについては、「QAnywhere Manager の設定プロパティ」 69 ページを参照してください。

詳細については、「QAnywhere Manager の設定プロパティをファイルに設定する」 70 ページを参照してください。

例外

- QAException クラス - Manager の作成で問題が発生した場合にスローされます。

参照

- 「QAManagerFactory クラス」 339 ページ
- 「QAManagerFactory のメンバ」 339 ページ
- 「QAManager インタフェース」 292 ページ

CreateQATransactionalManager メソッド

指定されたプロパティを持つ新しい QATransactionalManager インスタンスを返します。

構文

Visual Basic
Public Function **CreateQATransactionalManager(_**
  ByVal *iniFile* As String _
) As QATransactionalManager

C#
public QATransactionalManager **CreateQATransactionalManager(**


```
  string iniFile
);
```

パラメータ

- **iniFile**　QATransactionalManager インスタンスを設定するためのプロパティ・ファイル。QATransactionalManager インスタンスをデフォルトの設定で作成する場合は null。

戻り値

設定された QATransactionalManager

備考

プロパティ・ファイル・パラメータが null の場合は、デフォルトのプロパティを使用して QATransactionalManager が作成されます。インスタンスの作成後に SetProperty メソッドを使用して、QAnywhere Manager の設定プロパティをプログラムで設定できます。

QAnywhere Manager の設定プロパティのリストについては、「QAnywhere Manager の設定プロパティ」 69 ページを参照してください。

詳細については、「QAnywhere Manager の設定プロパティをファイルに設定する」 70 ページを参照してください。

例外

- QAException クラス - Manager の作成で問題が発生した場合にスローされます。

参照

- 「QAManagerFactory クラス」 339 ページ
- 「QAManagerFactory のメンバ」 339 ページ
- 「QATransactionalManager インタフェース」 367 ページ

QAMessage インタフェース

メッセージ・プロパティとヘッダ・ファイルを設定するためのインタフェースがあります。

構文

Visual Basic
Public Interface **QAMessage**

C#
public interface **QAMessage**

備考

派生クラスの QABinaryMessage と QATextMessage には、メッセージ本文の読み込み／書き込みを行うための特別なメソッドがあります。事前定義済みまたはカスタムのメッセージ・プロパティを設定するには、QAMessage のメソッドを使用します。

事前定義済みプロパティ名のリストについては、MessageProperties クラスを参照してください。

メッセージ・プロパティとヘッダ・フィールドの設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

- 「QAMessage のメンバ」 344 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「QATextMessage インタフェース」 364 ページ

QAMessage のメンバ

パブリック・プロパティ

メンバ名	説明
Address プロパティ	QAMessage インスタンスの送信先アドレスです。
Expiration プロパティ	メッセージの有効期限を取得します。
InReplyToID プロパティ	どのメッセージへの返信であるかを示すメッセージ ID です
MessageID プロパティ	メッセージのグローバルにユニークな ID です。
Priority プロパティ	メッセージの優先度 (0 から 9) です。
Redelivered プロパティ	受信されたが受信確認されていないメッセージであるかどうかを示します。
ReplyToAddress プロパティ	メッセージの返信先アドレスです。
Timestamp プロパティ	メッセージのタイムスタンプです。

パブリック・メソッド

メンバ名	説明
ClearBody メソッド	メッセージの本文をクリアします。
ClearProperties メソッド	メッセージのすべてのプロパティをクリアします。
GetBooleanProperty メソッド	boolean 型のメッセージ・プロパティを取得します。
GetByteProperty メソッド	byte 型のメッセージ・プロパティを取得します。
GetDoubleProperty メソッド	double 型のメッセージ・プロパティを取得します。
GetFloatProperty メソッド	float 型のメッセージ・プロパティを取得します。
GetIntProperty メソッド	int 型のメッセージ・プロパティを取得します。
GetLongProperty メソッド	long 型のメッセージ・プロパティを取得します。

メンバ名	説明
GetProperty メソッド	メッセージのプロパティを取得します。
GetPropertyNames メソッド	メッセージのプロパティ名の列挙子を取得します。
GetPropertyType メソッド	指定されたプロパティのプロパティ型を返します。
GetSbyteProperty メソッド	signed byte 型のメッセージ・プロパティを取得します。
GetShortProperty メソッド	short 型のメッセージ・プロパティを取得します。
GetStringProperty メソッド	文字列型のメッセージ・プロパティを取得します。
PropertyExists メソッド	指定されたプロパティがこのメッセージに設定されているかどうかを示します。
SetBooleanProperty メソッド	boolean 型のプロパティを設定します。
SetByteProperty メソッド	byte 型のプロパティを設定します。
SetDoubleProperty メソッド	double 型のプロパティを設定します。
SetFloatProperty メソッド	float 型のプロパティを設定します。
SetIntProperty メソッド	int 型のプロパティを設定します。
SetLongProperty メソッド	long 型のプロパティを設定します。
SetProperty メソッド	プロパティを設定します。
SetSbyteProperty メソッド	signed byte 型のプロパティを設定します。
SetShortProperty メソッド	short 型のプロパティを設定します。
SetStringProperty メソッド	文字列型のプロパティを設定します。

Address プロパティ

QAMessage インスタンスの送信先アドレスです。

構文

Visual Basic
Public Property **Address** As String

C#
public string **Address** {get;set;}

備考

このフィールドは、メッセージの送信時には無視されます。送信操作が完了すると、このフィールドには QAManagerBase.PutMessage で指定された送信先アドレスが入ります。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ
- 「PutMessage メソッド」 322 ページ

Expiration プロパティ

メッセージの有効期限を取得します。

構文

Visual Basic
Public Readonly Property **Expiration** As Date

C#
public DateTime **Expiration** {get;}

備考

メッセージの Expiration ヘッダ・フィールドは、メッセージの送信時には空のままです。このフィールドは、send メソッドが完了した後、メッセージの有効期限を保持します。

メッセージの有効期限は、QAManagerBase::PutMessageTimeToLive の有効期限の引数を現在の時間に追加して設定されるため、このプロパティは読み取り専用です。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

InReplyToID プロパティ

どのメッセージへの返信であるかを示すメッセージ ID です。

構文

Visual Basic
Public Property **InReplyToID** As String

C#
public string **InReplyToID** {get;set;}

備考

null の場合もあります。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

MessageID プロパティ

メッセージのグローバルにユニークな ID です。

構文

Visual Basic
Public Readonly Property **MessageID** As String

C#
public string **MessageID** {get;}

備考

このプロパティは、メッセージがキューに登録されるまで null のままです。

QAManagerBase.PutMessage を使用してメッセージが送信されると、MessageID が null になるため無視できます。send メソッドが戻ると、割り当てられた値がここに格納されます。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ
- 「PutMessage メソッド」 322 ページ

Priority プロパティ

メッセージの優先度 (0 から 9) です。

構文

Visual Basic
Public Property **Priority** As Integer

C#
public int **Priority** {get;set;}

備考

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

Redelivered プロパティ

受信されたが受信確認されていないメッセージであるかどうかを示します。

構文

Visual Basic
Public Readonly Property **Redelivered** As Boolean

C#
public bool **Redelivered** {get;}

備考

受信側の QAManager は、受信中のメッセージが以前に受信されたことを検知した場合に、Redelivered プロパティを設定します。

たとえば、AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT でオープンされた QAManager を使用してアプリケーションがメッセージを受信し、メッセージの受信確認を行わずに終了したとします。このアプリケーションは、次の起動時に同じメッセージを再受信し、Redelivered ヘッダを true に設定します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ
- 「QAManager インタフェース」 292 ページ
- 「AcknowledgementMode 列挙」 262 ページ

ReplyToAddress プロパティ

メッセージの返信先アドレスです。

構文

Visual Basic
Public Property **ReplyToAddress** As String

C#
public string **ReplyToAddress** {get;set;}

備考

null の場合もあります。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

Timestamp プロパティ

メッセージのタイムスタンプです。

構文

Visual Basic
Public Readonly Property **Timestamp** As Date

C#
public DateTime **Timestamp** {get;}

備考

メッセージの Timestamp ヘッダ・フィールドには、メッセージが作成された時刻が格納されます。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

ClearBody メソッド

メッセージの本文をクリアします。

構文

Visual Basic
Public Sub **ClearBody()**

C#
public void **ClearBody();**

ClearProperties メソッド

メッセージのすべてのプロパティをクリアします。

構文

Visual Basic
Public Sub **ClearProperties()**

C#
public void **ClearProperties();**

GetBooleanProperty メソッド

boolean 型のメッセージ・プロパティを取得します。

構文

Visual Basic
```
Public Function GetBooleanProperty( _
  ByVal propName As String _
) As Boolean
```

C#
```
public bool GetBooleanProperty(
  string  propName
);
```

パラメータ

- **propName**　プロパティ名

戻り値

プロパティの値

備考

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

例外

- QAException クラス - プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ
- 「MessageProperties クラス」 264 ページ

GetByteProperty メソッド

byte 型のメッセージ・プロパティを取得します。

構文

Visual Basic

```
Public Function GetByteProperty( _
  ByVal propName As String _
) As Byte
```

C#

```
public byte GetByteProperty(
  string  propName
);
```

パラメータ

- **propName**　プロパティ名

戻り値

プロパティの値

備考

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

例外

- QAException クラス - プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- 「QAMessage インタフェース」 343 ページ

- 「QAMessage のメンバ」 344 ページ
- 「MessageProperties クラス」 264 ページ

GetDoubleProperty メソッド

double 型のメッセージ・プロパティを取得します。

構文

Visual Basic
```
Public Function GetDoubleProperty( _
  ByVal propName As String _
) As Double
```

C#
```
public double GetDoubleProperty(
  string  propName
);
```

パラメータ

- **propName**　プロパティ名

戻り値

プロパティの値

備考

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

例外

- QAException クラス - プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ
- 「MessageProperties クラス」 264 ページ

GetFloatProperty メソッド

float 型のメッセージ・プロパティを取得します。

構文

Visual Basic
```
Public Function GetFloatProperty( _
  ByVal propName As String _
) As Single
```


C#
public float **GetFloatProperty(**
  string *propName*
);

パラメータ

- **propName**　プロパティ名

戻り値

プロパティの値

備考

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

例外

- QAException クラス - プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ
- 「MessageProperties クラス」 264 ページ

GetIntProperty メソッド

int 型のメッセージ・プロパティを取得します。

構文

Visual Basic
Public Function **GetIntProperty(_**
  ByVal *propName* As String _
) As Integer

C#
public int **GetIntProperty(**
  string *propName*
);

パラメータ

- **propName**　プロパティ名

戻り値

プロパティの値

備考

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

例外

- QAException クラス - プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ
- 「MessageProperties クラス」 264 ページ

GetLongProperty メソッド

long 型のメッセージ・プロパティを取得します。

構文

Visual Basic
```
Public Function GetLongProperty( _
  ByVal propName As String _
) As Long
```

C#
```
public long GetLongProperty(
  string  propName
);
```

パラメータ

- **propName**　プロパティ名

戻り値

プロパティの値

備考

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

例外

- QAException クラス - プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ
- 「MessageProperties クラス」 264 ページ

GetProperty メソッド

メッセージのプロパティを取得します。

構文

Visual Basic
Public Function **GetProperty(_**
  ByVal *propName* As String _
) As Object

C#
public object **GetProperty(**
  string *propName*
);

パラメータ

- **propName**　プロパティ名

戻り値

プロパティの値

備考

プロパティ型は、使用可能ないずれかのプリミティブ型、文字列型、または DateTime 型でなければなりません。

例外

- QAException クラス - 該当するプロパティが存在しない場合にスローされます。

GetPropertyNames メソッド

メッセージのプロパティ名の列挙子を取得します。

構文

Visual Basic
Public Function **GetPropertyNames()** As System.Collections.IEnumerator

C#
public System.Collections.IEnumerator **GetPropertyNames();**

戻り値

メッセージのプロパティ名の列挙子

GetPropertyType メソッド

指定されたプロパティのプロパティ型を返します。

構文

Visual Basic
Public Function **GetPropertyType(_**
  ByVal *propName* As String _
) As PropertyType

C#
public PropertyType **GetPropertyType(**
  string *propName*
);

パラメータ

- **propName**　プロパティ名

戻り値

プロパティの型

GetSbyteProperty メソッド

signed byte 型のメッセージ・プロパティを取得します。

構文

Visual Basic
Public Function **GetSbyteProperty(_**
  ByVal *propName* As String _
) As System.SByte

C#
public System.Sbyte **GetSbyteProperty(**
  string *propName*
);

パラメータ

- **propName**　プロパティ名

戻り値

プロパティの値

備考

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

例外

- QAException クラス - プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ
- 「MessageProperties クラス」 264 ページ

GetShortProperty メソッド

short 型のメッセージ・プロパティを取得します。

構文

Visual Basic
```
Public Function GetShortProperty( _
  ByVal propName As String _
) As Short
```

C#
```
public short GetShortProperty(
  string  propName
);
```

パラメータ

- **propName**　プロパティ名

戻り値

プロパティの値

備考

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

例外

- QAException クラス - プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ
- 「MessageProperties クラス」 264 ページ

GetStringProperty メソッド

文字列型のメッセージ・プロパティを取得します。

構文

Visual Basic
```
Public Function GetStringProperty( _
  ByVal propName As String _
) As String
```

C#
```
public string  GetStringProperty(
  string  propName
);
```

パラメータ

- **propName**　プロパティ名

戻り値

プロパティの値。該当するプロパティが存在しない場合は null。

備考

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ
- 「MessageProperties クラス」 264 ページ

PropertyExists メソッド

指定されたプロパティがこのメッセージに設定されているかどうかを示します。

構文

Visual Basic

```
Public Function PropertyExists( _
  ByVal propName As String _
) As Boolean
```

C#

```
public bool PropertyExists(
  string  propName
);
```

パラメータ

- **propName**　プロパティ名

戻り値

プロパティが存在する場合は True

SetBooleanProperty メソッド

boolean 型のプロパティを設定します。

構文

Visual Basic

```
Public Sub SetBooleanProperty( _
  ByVal propName As String, _
  ByVal val As Boolean _
)
```

C#
```
public void SetBooleanProperty(
  string  propName,
  bool val
);
```

パラメータ

- **propName**　プロパティ名
- **val**　プロパティの値

備考

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ
- 「MessageProperties クラス」 264 ページ

SetByteProperty メソッド

byte 型のプロパティを設定します。

構文

Visual Basic
```
Public Sub SetByteProperty( _
  ByVal propName As String, _
  ByVal val As Byte _
)
```

C#
```
public void SetByteProperty(
  string  propName,
  byte val
);
```

パラメータ

- **propName**　プロパティ名
- **val**　プロパティの値

備考

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ

- 「MessageProperties クラス」 264 ページ

SetDoubleProperty メソッド

double 型のプロパティを設定します。

構文

Visual Basic
```
Public Sub SetDoubleProperty( _
  ByVal propName As String, _
  ByVal val As Double _
)
```

C#
```
public void SetDoubleProperty(
  string  propName,
  double val
);
```

パラメータ

- **propName**　プロパティ名
- **val**　プロパティの値

備考

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ
- 「MessageProperties クラス」 264 ページ

SetFloatProperty メソッド

float 型のプロパティを設定します。

構文

Visual Basic
```
Public Sub SetFloatProperty( _
  ByVal propName As String, _
  ByVal val As Single _
)
```

C#
```
public void SetFloatProperty(
  string  propName,
  float  val
);
```


パラメータ

- **propName**　プロパティ名
- **val**　プロパティの値

備考

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ
- 「MessageProperties クラス」 264 ページ

SetIntProperty メソッド

int 型のプロパティを設定します。

構文

Visual Basic

```
Public Sub SetIntProperty( _
  ByVal propName As String, _
  ByVal val As Integer _
)
```

C#

```
public void SetIntProperty(
  string  propName,
  int val
);
```

パラメータ

- **propName**　プロパティ名
- **val**　プロパティの値

備考

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ
- 「MessageProperties クラス」 264 ページ

SetLongProperty メソッド

long 型のプロパティを設定します。

構文

Visual Basic
```
Public Sub SetLongProperty( _
  ByVal propName As String, _
  ByVal val As Long _
)
```

C#
```
public void SetLongProperty(
  string  propName,
  long val
);
```

パラメータ

- **propName**　プロパティ名
- **val**　プロパティの値

備考

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ
- 「MessageProperties クラス」 264 ページ

SetProperty メソッド

プロパティを設定します。

構文

Visual Basic
```
Public Sub SetProperty( _
  ByVal propName As String, _
  ByVal val As Object _
)
```

C#
```
public void SetProperty(
  string  propName,
  object val
);
```

パラメータ

- **propName**　プロパティ名

- **val**　プロパティの値

備考

プロパティ型は、使用可能ないずれかのプリミティブ型、または文字列型でなければなりません。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ
- 「MessageProperties クラス」 264 ページ

SetSbyteProperty メソッド

signed byte 型のプロパティを設定します。

構文

Visual Basic

```
Public Sub SetSbyteProperty( _
  ByVal propName As String, _
  ByVal val As System.SByte _
)
```

C#

```
public void SetSbyteProperty(
  string  propName,
  System.Sbyte val
);
```

パラメータ

- **propName**　プロパティ名
- **val**　プロパティの値

備考

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ
- 「MessageProperties クラス」 264 ページ

SetShortProperty メソッド

short 型のプロパティを設定します。

構文

Visual Basic
```
Public Sub SetShortProperty( _
  ByVal propName As String, _
  ByVal val As Short _
)
```

C#
```
public void SetShortProperty(
  string  propName,
  short val
);
```

パラメータ

- **propName**　プロパティ名
- **val**　プロパティの値

備考

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ
- 「MessageProperties クラス」 264 ページ

SetStringProperty メソッド

文字列型のプロパティを設定します。

構文

Visual Basic
```
Public Sub SetStringProperty( _
  ByVal propName As String, _
  ByVal val As String _
)
```

C#
```
public void SetStringProperty(
  string  propName,
  string  val
);
```

パラメータ

- **propName**　プロパティ名

- **val**　プロパティの値

備考

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

- 「QAMessage インタフェース」 343 ページ
- 「QAMessage のメンバ」 344 ページ
- 「MessageProperties クラス」 264 ページ

QATextMessage インタフェース

QATextMessage は QAMessage クラスを継承したもので、メッセージ本文にテキストが追加されます。QATextMessage には、メッセージ本文からのテキストの読み込み／書き込みを行うためのメソッドがあります。

構文

Visual Basic
Public Interface **QATextMessage**

C#
public interface **QATextMessage**

備考

メッセージが最初に作成された時点では、メッセージ本文は書き込み専用モードになっています。メッセージ送信元のクライアントは、メッセージを送信した後、そのメッセージを保持し変更できます。ただし、それによって、送信されたメッセージが変更されることはありません。同じメッセージ・オブジェクトを複数回送信できます。

メッセージが受信されるとプロバイダが QATextMessage.Reset() を呼び出します。これによりメッセージ本文が読み込み専用モードになり、メッセージ本文の先頭から値の読み込みが開始されます。

参照

- 「QATextMessage のメンバ」 364 ページ
- 「QABinaryMessage インタフェース」 276 ページ
- 「QAMessage インタフェース」 343 ページ

QATextMessage のメンバ

パブリック・プロパティ

メンバ名	説明
Text プロパティ	メッセージ・テキストです。

メンバ名	説明
TextLength プロパティ	メッセージの文字数です。

パブリック・メソッド

メンバ名	説明
ReadText メソッド	未読テキストを指定されたバッファ内に読み込みます。
Reset メソッド	メッセージのテキスト位置を先頭にリセットします。
WriteText メソッド	メッセージ・テキストの末尾にテキストを追加します。

Text プロパティ

メッセージ・テキストです。

構文

Visual Basic
Public Property **Text** As String

C#
public string **Text** {get;set;}

備考

メッセージのサイズが、QAManager.MAX_IN_MEMORY_MESSAGE_SIZE で指定された最大サイズを超える場合、このプロパティは null になります。この場合は、QATextMessage.ReadText メソッドを使用してテキストを読み込みます。

QAManager のプロパティの詳細については、「QAnywhere Manager の設定プロパティ」 69 ページを参照してください。

参照

- 「QATextMessage インタフェース」 364 ページ
- 「QATextMessage のメンバ」 364 ページ
- 「ReadText メソッド」 366 ページ

TextLength プロパティ

メッセージの文字数です。

構文

Visual Basic
Public Readonly Property **TextLength** As Long

C#
public long **TextLength** {get;}

ReadText メソッド

未読テキストを指定されたバッファ内に読み込みます。

構文

Visual Basic
```
Public Function ReadText( _
  ByVal buf As System.Text.StringBuilder _
) As Integer
```

C#
```
public int ReadText(
  System.Text.string Builder buf
);
```

パラメータ

- **buf**　テキストの読み込み先バッファ

戻り値

読み込まれた文字数。読み込めるテキストが残っていない場合は -1。

備考

未読テキストが追加された場合、そのテキストを読み込むにはこのメソッドを繰り返し呼び出す必要があります。読み込みは、未読テキストの先頭から実行されます。

Reset メソッド

メッセージのテキスト位置を先頭にリセットします。

構文

Visual Basic
```
Public Sub Reset()
```

C#
```
public void Reset();
```

WriteText メソッド

メッセージ・テキストの末尾にテキストを追加します。

構文

Visual Basic
```
Public Sub WriteText( _
  ByVal val As String _
)
```

C#
```
public void WriteText(
```

```
  string val
);
```

パラメータ

- **val**　追加するテキスト

QATransactionalManager インタフェース

QATransactionalManager クラスは QAManagerBase から派生し、トランザクション志向の QAnywhere メッセージング操作を管理します。

構文

Visual Basic
Public Interface **QATransactionalManager**

C#
public interface **QATransactionalManager**

備考

この動作の完全な説明については、QAManagerBase インタフェースを参照してください。

QATransactionalManager は、トランザクション志向の受信確認でのみ使用できます。QAManagerBase.PutMessage と QAManagerBase.GetMessage のすべての呼び出しをコミットする場合は、QATransactionalManager.Commit() メソッドを使用します。

詳細については、「トランザクション志向メッセージングの実装」 75 ページを参照してください。

参照

- 「QATransactionalManager のメンバ」 367 ページ
- 「QATransactionalManager インタフェース」 367 ページ

QATransactionalManager のメンバ

パブリック・メソッド

メンバ名	説明
Commit メソッド	現在のトランザクションをコミットし、新しいトランザクションを開始します。
Open メソッド	QATransactionalManager インスタンスをオープンします。
Rollback メソッド	現在のトランザクションをロールバックし、新しいトランザクションを開始します。

Commit メソッド

現在のトランザクションをコミットし、新しいトランザクションを開始します。

構文

Visual Basic
Public Sub **Commit()**

C#
public void **Commit();**

備考

このメソッドは、QAManagerBase.PutMessage と QAManagerBase.GetMessage のすべての呼び出しをコミットします。*注意 :* 最初のトランザクションは、QATransactionalManager.Open() の呼び出しで開始されます。

例外

- QAException クラス - メッセージのコミットで問題が発生した場合にスローされます。

参照

- 「QATransactionalManager インタフェース」 367 ページ
- 「QATransactionalManager のメンバ」 367 ページ
- 「QATransactionalManager インタフェース」 367 ページ

Open メソッド

QATransactionalManager インスタンスをオープンします。

構文

Visual Basic
Public Sub **Open()**

C#
public void **Open();**

備考

Open メソッドは、Manager を作成した後、最初に呼び出す必要があるメソッドです。

例外

- QAException クラス - Manager のオープンで問題が発生した場合にスローされます。

参照

- 「QATransactionalManager インタフェース」 367 ページ
- 「QATransactionalManager のメンバ」 367 ページ
- 「QATransactionalManager インタフェース」 367 ページ

Rollback メソッド

現在のトランザクションをロールバックし、新しいトランザクションを開始します。

構文

Visual Basic
Public Sub **Rollback()**

C#
public void **Rollback();**

備考

このメソッドは、コミットされていない QAManagerBase.PutMessage と QAManagerBase.GetMessage のすべての呼び出しをロールバックします。

例外

- QAException クラス - ロールバックで問題が発生した場合にスローされます。

参照

- 「QATransactionalManager インタフェース」 367 ページ
- 「QATransactionalManager のメンバ」 367 ページ
- 「QATransactionalManager インタフェース」 367 ページ

QueueDepthFilter 列挙

QAManagerBase.GetQueueDepth(QueueDepthFilter) と QAManagerBase.GetQueueDepth(string, QueueDepthFilter) のキューの深さのフィルタ値を提供します。

構文

Visual Basic
Public Enum **QueueDepthFilter**

C#
public enum **QueueDepthFilter**

メンバ名

メンバ名	説明
ALL	着信メッセージと送信メッセージの両方をカウントします。
INCOMING	着信メッセージのみカウントします。
OUTGOING	発信メッセージのみカウントします。

参照

- 「GetQueueDepth メソッド」 318 ページ
- 「GetQueueDepth メソッド」 317 ページ

StatusCodes 列挙

この列挙は、メッセージのステータス・コード・セットを定義します。

構文

Visual Basic
Public Enum **StatusCodes**

C#
public enum **StatusCodes**

メンバ名

メンバ名	説明
CANCELLED	メッセージはキャンセル済みです。
EXPIRED	メッセージは有効期限が切れる前に受信されなかったため、期限切れになりました。
FINAL	メッセージは最終ステータスになりました。
LOCAL	メッセージはローカル・メッセージ・ストア宛てに送信され、サーバに送信されません。
PENDING	メッセージは送信されたが、まだ受信されていません。
RECEIVED	メッセージが受信され、受信者によって受信確認されました。
RECEIVING	メッセージは受信中であるか、または受信されたがまだ確認されていません。
TRANSMITTED	メッセージはサーバに送信されました。
TRANSMITTING	メッセージはサーバに送信中です。
UNRECEIVABLE	メッセージは受信不可のマークが付けられています。メッセージに間違った形式があるか、失敗した試行回数が多すぎて配信されませんでした。
UNTRANSMITTED	メッセージはサーバに送信されていません。

iAnywhere.QAnywhere.WS ネームスペース (.NET 1.0)

WSBase クラス

これは、モバイル Web サービスのコンパイラで生成されるメインの Web サービス・プロキシ・クラスの基本クラスです。

構文

Visual Basic
Public Class **WSBase**

C#
public class **WSBase**

WSBase のメンバ

パブリック・コンストラクタ

メンバ名	説明
WSBase コンストラクタ	設定プロパティ・ファイルで指定されたプロパティを使用して、WSBase インスタンスを構築します。
WSBase コンストラクタ	デフォルトのプロパティを使用して WSBase インスタンスを構築します。

パブリック・メソッド

メンバ名	説明
ClearRequestProperties メソッド	この WSBase に対して送信された要求プロパティをすべてクリアします。
GetResult メソッド	Web サービス要求の結果を表す WSResult オブジェクトを取得します。
GetServiceID メソッド	WSBase のこのインスタンスのサービス ID を取得します。
SetListener メソッド	指定された Web サービス要求の結果を受信するリスナを設定します。
SetListener メソッド	WSBase のこのインスタンスが作成した、すべての Web サービス要求を受信するリスナを設定します。
SetProperty メソッド	WSBase のこのインスタンスの設定プロパティを設定します。
SetQAManager メソッド	この Web サービス・クライアントが Web サービス要求を処理するために使用する QAManagerBase を設定します。

メンバ名	説明
SetRequestProperty メソッド	この WSBase が作成した Web サービス要求の要求プロパティを設定します。
SetServiceID メソッド	WSBase のこのインスタンスのユーザ定義 ID を設定します。

WSBase コンストラクタ

設定プロパティ・ファイルで指定されたプロパティを使用して、WSBase インスタンスを構築します。

構文

Visual Basic

```
Overloads Public Sub New( _
  ByVal iniFile As String _
)
```

C#

```
public WSBase(
  string  iniFile
);
```

パラメータ

- **iniFile**　設定プロパティが含まれているファイル

備考

有効な設定プロパティには、次のものがあります。

LOG_FILE：ランタイム情報のログを記録するファイルです。

LOG_LEVEL：ログを記録する情報の冗長レベルを制御する、0 ー 6 の値。6 が最高の冗長レベルです。

WS_CONNECTOR_ADDRESS：Mobile Link サーバにおける Web サービス・コネクタのアドレスです。

デフォルトの WS_CONNECTOR_ADDRESS は "ianywhere.connector.webservices¥¥" です。

例外

- WSException クラス - WSBase の構築で問題がある場合にスローされます。

WSBase コンストラクタ

デフォルトのプロパティを使用して WSBase インスタンスを構築します。

構文

Visual Basic
Overloads Public Sub **New()**

C#
public **WSBase();**

例外

- WSException クラス - WSBase の構築で問題がある場合にスローされます。

ClearRequestProperties メソッド

この WSBase に対して送信された要求プロパティをすべてクリアします。

構文

Visual Basic
Public Sub **ClearRequestProperties()**

C#
public void **ClearRequestProperties();**

GetResult メソッド

Web サービス要求の結果を表す WSResult オブジェクトを取得します。

構文

Visual Basic
```
Public Function GetResult( _
  ByVal requestID As String _
) As iAnywhere.QAnywhere.WS.WSResult
```

C#
```
public iAnywhere.QAnywhere.WS.WSResult GetResult(
  string  requestID
);
```

パラメータ

- **requestID**　Web サービス要求の ID

戻り値

Web サービス要求の結果を表す WSResult インスタンス

参照

- 「WSBase クラス」 371 ページ
- 「WSBase のメンバ」 371 ページ
- 「WSStatus 列挙」 418 ページ

GetServiceID メソッド

WSBase のこのインスタンスのサービス ID を取得します。

構文

Visual Basic
Public Function **GetServiceID()** As String

C#
public string **GetServiceID();**

戻り値

サービス ID

SetListener メソッド

指定された Web サービス要求の結果を受信するリスナを設定します。

構文

Visual Basic
```
Overloads Public Sub SetListener( _
  ByVal requestID As String, _
  ByVal listener As iAnywhere.QAnywhere.WS.WSListener _
)
```

C#
```
public void SetListener(
  string requestID,
  iAnywhere.QAnywhere.WS.WSListener listener
);
```

パラメータ

- **requestID**　結果を受信する Web サービス要求の ID
- **listener**　指定された Web サービス要求が取得可能な場合に呼び出されるリスナ・オブジェクト

備考

一般にリスナは、サービスの asyncXYZ メソッドの結果を取得する場合に使用されます。

リスナを削除するには、リスナとして null を指定し SetListner を呼び出します。

注意： このメソッドは、以前に呼び出された SetListner でリスナを置き換えます。

SetListener メソッド

WSBase のこのインスタンスが作成した、すべての Web サービス要求を受信するリスナを設定します。

構文

Visual Basic
```
Overloads Public Sub SetListener( _
  ByVal listener As iAnywhere.QAnywhere.WS.WSListener _
)
```

C#
```
public void SetListener(
  iAnywhere.QAnywhere.WS.WSListener listener
);
```

パラメータ

- **listener**　Web サービス要求が取得可能な場合に呼び出されるリスナ・オブジェクト

備考

一般にリスナは、サービスの asyncXYZ メソッドの結果を取得する場合に使用されます。

リスナを削除するには、リスナとして null を指定し SetListner を呼び出します。

*注意：*このメソッドは、以前に呼び出された SetListner でリスナを置き換えます。

SetProperty メソッド

WSBase のこのインスタンスの設定プロパティを設定します。

構文

Visual Basic
```
Public Sub SetProperty( _
  ByVal property As String, _
  ByVal val As String _
)
```

C#
```
public void SetProperty(
  string  property,
  string  val
);
```

パラメータ

- **property**　設定するプロパティ名
- **val**　プロパティの値

備考

非同期または同期の Web サービス要求を出す前に、設定プロパティを作成してください。このメソッドは、Web サービス要求が出された後で呼び出された場合は無効です。

有効な設定プロパティには、次のものがあります。

LOG_FILE：ランタイム情報のログを記録するファイルです。

LOG_LEVEL：ログを記録する情報の冗長レベルを制御する、0 ー 6 の値。6 が最高の冗長レベルです。

WS_CONNECTOR_ADDRESS：Mobile Link サーバにおける Web サービス・コネクタのアドレスです。デフォルトは "ianywhere.connector.webservices¥¥" です。

SetQAManager メソッド

この Web サービス・クライアントが Web サービス要求を処理するために使用する QAManagerBase を設定します。

構文

Visual Basic
```
Public Sub SetQAManager( _
  ByVal mgr As QAManagerBase _
)
```

C#
```
public void SetQAManager(
  QAManagerBase mgr
);
```

パラメータ

- **mgr**　使用する QAManagerBase

備考

注意： EXPLICIT_ACKNOWLEDGEMENT QAManager を使用する場合は、WSResult の acknowledge() メソッドを呼び出して、非同期の Web サービス要求の結果を受信確認できます。同期の Web サービス要求の結果は、EXPLICIT_ACKNOWLEDGEMENT QAManager を使用している場合であっても、自動的に受信確認されます。IMPLICIT_ACKNOWLEDGEMENT QAManager を使用する場合は、あらゆる Web サービス要求の結果が自動的に受信確認されます。

SetRequestProperty メソッド

この WSBase が作成した Web サービス要求の要求プロパティを設定します。

構文

Visual Basic
```
Public Sub SetRequestProperty( _
  ByVal name As String, _
  ByVal value As Object _
)
```

C#
```
public void SetRequestProperty(
  string  name,
  object value
);
```

パラメータ

- **name**　設定するプロパティ名
- **value**　プロパティの値

備考

要求プロパティは、この WSBase によって送信される QAMessage ごとに設定されます。この処理は、プロパティがクリアされるまで行われます。要求プロパティをクリアするには、null 値を設定します。メッセージ・プロパティのタイプは、value パラメータのクラスで決まります。たとえば、値が Int32 のインスタンスである場合は、SetIntProperty を使用して QAMessage のプロパティが設定されます。

SetServiceID メソッド

WSBase のこのインスタンスのユーザ定義 ID を設定します。

構文

Visual Basic
```
Public Sub SetServiceID( _
  ByVal serviceID As String _
)
```

C#
```
public void SetServiceID(
  string  serviceID
);
```

パラメータ

- **serviceID**　サービス ID

備考

サービス ID には、WSBase のこのインスタンスにユニークな値を設定します。サービス ID は、Web サービス要求を送受信するためのキュー名を形成する場合に内部で使用されます。このため、前のセッションで作成された Web サービス要求の結果を取得できるように、サービス ID をアプリケーションのセッション間で保持する必要があります。

WSException クラス

このクラスは、Web サービス要求の処理中に発生した例外を表します。

構文

Visual Basic
```
Public Class WSException
  Inherits Exception
```


C#
public class **WSException** :
  Exception

WSException のメンバ

パブリックの静的フィールド (共有)

メンバ名	説明
WS_STATUS_HTTP_ERROR フィールド	Web サービス・コネクタが作成した Web サービス HTTP 要求にエラーがあったことを示すエラー・コードです。
WS_STATUS_HTTP_OK フィールド	Web サービス・コネクタからの Web サービス HTTP 要求が正常に行われたことを示すエラー・コードです。
WS_STATUS_HTTP_RETRIES_EXCEEDED フィールド	HTTP の再試行回数が Web サービス・コネクタを超えたことを示すエラー・コードです。
WS_STATUS_SOAP_PARSE_ERROR フィールド	SOAP 応答または SOAP 要求の解析中に、Web サービス・ランタイムまたは Web サービス・コネクタでエラーがあったことを示すエラー・コードです。

パブリック・コンストラクタ

メンバ名	説明
WSException コンストラクタ	指定されたエラー・メッセージを使用して、新しい例外を構築します。
WSException コンストラクタ	指定されたエラー・メッセージとエラー・コードを使用して、新しい例外を構築します。
WSException コンストラクタ	新しい例外を構築します。

パブリック・プロパティ

メンバ名	説明
ErrorCode プロパティ	この例外に関連付けられているエラー・コード
HelpLink (Exception から継承)	この例外に関連付けられているヘルプ・ファイルへのリンクを取得または設定します。
InnerException (Exception から継承)	現在の例外を発生させた System.Exception インスタンスを取得します。
Message (Exception から継承)	現在の例外を説明するメッセージを取得します。
Source (Exception から継承)	エラーを発生させたアプリケーションまたはオブジェクトの名前を取得または設定します。

メンバ名	説明
StackTrace (Exception から継承)	現在の例外がスローされた時点のコール・スタックのフレームを表す文字列を取得します。
TargetSite (Exception から継承)	現在の例外をスローしたメソッドを取得します。

パブリック・メソッド

メンバ名	説明
GetBaseException (Exception から継承)	派生クラスで上書きされる場合は、後続の 1 つ以上の例外の根本原因となる System.Exception を返します。
GetObjectData (Exception から継承)	派生クラスで上書きされる場合は、System.Runtime.Serialization.SerializationInfo に例外の情報を設定します。
ToString (Exception から継承)	現在の例外を表す文字列を作成して返します。

WSException コンストラクタ

指定されたエラー・メッセージを使用して、新しい例外を構築します。

構文

Visual Basic

```
Overloads Public Sub New( _
  ByVal msg As String _
)
```

C#

```
public WSException(
  string  msg
);
```

パラメータ

- **msg**　エラー・メッセージ

WSException コンストラクタ

指定されたエラー・メッセージとエラー・コードを使用して、新しい例外を構築します。

構文

Visual Basic

```
Overloads Public Sub New( _
  ByVal msg As String, _
  ByVal errorCode As Integer _
)
```


C#
```
public WSException(
  string  msg,
  int errorCode
);
```

パラメータ

- **msg**　エラー・メッセージ
- **errorCode**　エラー・コード

WSException コンストラクタ

新しい例外を構築します。

構文

Visual Basic
```
Overloads Public Sub New( _
  ByVal ex As System.Exception _
)
```

C#
```
public WSException(
  System.Exception ex
);
```

パラメータ

- **ex**　例外

WS_STATUS_HTTP_ERROR フィールド

Web サービス・コネクタが作成した Web サービス HTTP 要求にエラーがあったことを示すエラー・コードです。

構文

Visual Basic
```
Public Shared WS_STATUS_HTTP_ERROR As Integer
```

C#
```
public const int WS_STATUS_HTTP_ERROR;
```

WS_STATUS_HTTP_OK フィールド

Web サービス・コネクタからの Web サービス HTTP 要求が正常に行われたことを示すエラー・コードです。

構文

Visual Basic
Public Shared **WS_STATUS_HTTP_OK** As Integer

C#
public const int **WS_STATUS_HTTP_OK**;

WS_STATUS_HTTP_RETRIES_EXCEEDED フィールド

HTTP の再試行回数が Web サービス・コネクタを超えたことを示すエラー・コードです。

構文

Visual Basic
Public Shared **WS_STATUS_HTTP_RETRIES_EXCEEDED** As Integer

C#
public const int **WS_STATUS_HTTP_RETRIES_EXCEEDED**;

WS_STATUS_SOAP_PARSE_ERROR フィールド

SOAP 応答または SOAP 要求の解析中に、Web サービス・ランタイムまたは Web サービス・コネクタでエラーがあったことを示すエラー・コードです。

構文

Visual Basic
Public Shared **WS_STATUS_SOAP_PARSE_ERROR** As Integer

C#
public const int **WS_STATUS_SOAP_PARSE_ERROR**;

ErrorCode プロパティ

この例外に関連付けられているエラー・コードです。

構文

Visual Basic
Public Property **ErrorCode** As Integer

C#
public int **ErrorCode** {get;set;}

WSFaultException クラス

このクラスは、Web サービス・コネクタからの SOAP Fault 例外を表します。

構文

Visual Basic
Public Class **WSFaultException**
  Inherits WSException

C#
public class **WSFaultException** :
  WSException

WSFaultException のメンバ

パブリック・コンストラクタ

メンバ名	説明
WSFaultException コンストラクタ	指定されたエラー・メッセージを使用して、新しい例外を構築します。

パブリック・プロパティ

メンバ名	説明
ErrorCode プロパティ (WSException から継承)	この例外に関連付けられているエラー・コード
HelpLink (Exception から継承)	この例外に関連付けられているヘルプ・ファイルへのリンクを取得または設定します。
InnerException (Exception から継承)	現在の例外を発生させた System.Exception インスタンスを取得します。
Message (Exception から継承)	現在の例外を説明するメッセージを取得します。
Source (Exception から継承)	エラーを発生させたアプリケーションまたはオブジェクトの名前を取得または設定します。
StackTrace (Exception から継承)	現在の例外がスローされた時点のコール・スタックのフレームを表す文字列を取得します。
TargetSite (Exception から継承)	現在の例外をスローしたメソッドを取得します。

パブリック・メソッド

メンバ名	説明
GetBaseException (Exception から継承)	派生クラスで上書きされる場合は、後続の 1 つ以上の例外の根本原因となる System.Exception を返します。
GetObjectData (Exception から継承)	派生クラスで上書きされる場合は、System.Runtime.Serialization.SerializationInfo に例外の情報を設定します。

メンバ名	説明
ToString (Exception から継承)	現在の例外を表す文字列を作成して返します。

WSFaultException コンストラクタ

指定されたエラー・メッセージを使用して、新しい例外を構築します。

構文

Visual Basic
```
Public Sub New( _
  ByVal msg As String _
)
```

C#
```
public WSFaultException(
  string  msg
);
```

パラメータ

- **msg**　エラー・メッセージ

WSListener インタフェース

このクラスは、Web サービス要求の結果を受信するリスナを表します。

構文

Visual Basic
```
Public Interface WSListener
```

C#
```
public interface WSListener
```

WSListener のメンバ

パブリック・メソッド

メンバ名	説明
OnException メソッド	非同期の Web サービス要求の結果を処理中に例外が発生した場合に呼び出されます。
OnResult メソッド	非同期の Web サービス要求の結果と一緒に呼び出されます。

OnException メソッド

非同期の Web サービス要求の結果を処理中に例外が発生した場合に呼び出されます。

構文

Visual Basic
```
Public Sub OnException( _
  ByVal e As iAnywhere.QAnywhere.WS.WSException, _
  ByVal wsResult As iAnywhere.QAnywhere.WS.WSResult _
)
```

C#
```
public void OnException(
  iAnywhere.QAnywhere.WS.WSException e,
  iAnywhere.QAnywhere.WS.WSResult wsResult
);
```

パラメータ

- **e**　結果の処理中に発生した WSException
- **wsResult**　WSResult。これから要求 ID を取得できる場合があります。この WSResult の値は定義されていません。

OnResult メソッド

非同期の Web サービス要求の結果を指定して呼び出されます。

構文

Visual Basic
```
Public Sub OnResult( _
  ByVal wsResult As iAnywhere.QAnywhere.WS.WSResult _
)
```

C#
```
public void OnResult(
  iAnywhere.QAnywhere.WS.WSResult wsResult
);
```

パラメータ

- **wsResult**　Web サービス要求の結果を記述する WSResult

WSResult クラス

このクラスは、Web サービス要求の結果を表します。

構文

Visual Basic
```
Public Class WSResult
```

C#
public class **WSResult**

備考

WSResult オブジェクトは、次のいずれかの方法で取得されます。

- WSListener.onResult に渡される。

- コンパイラが生成したサービス・プロキシの asyncXYZ メソッドから返される。

- 特定の要求 ID を指定して WSBase.getResult を呼び出して取得される。

WSResult のメンバ

パブリック・メソッド

メンバ名	説明
Acknowledge モード	この WSResult が処理されたことを確認します。
GetArrayValue メソッド	この WSResult から複合型の値の配列を取得します。
GetBoolArrayValue メソッド	この WSResult から bool 値の配列を取得します。
GetBooleanArrayValue メソッド	この WSResult から Boolean 値の配列を取得します。
GetBooleanValue メソッド	この WSResult から Boolean 値を取得します。
GetBoolValue メソッド	この WSResult から bool 値を取得します。
GetByteArrayValue メソッド	この WSResult から byte 値の配列を取得します。
GetByteValue メソッド	この WSResult から byte 値を取得します。
GetCharArrayValue メソッド	この WSResult から char 値の配列を取得します。
GetCharValue メソッド	この WSResult から char 値を取得します。
GetDecimalArrayValue メソッド	この WSResult から 10 進値の配列を取得します。
GetDecimalValue メソッド	この WSResult から 10 進値を取得します。
GetDoubleArrayValue メソッド	この WSResult から double 値の配列を取得します。
GetDoubleValue メソッド	この WSResult から double 値を取得します。
GetErrorMessage メソッド	エラー・メッセージを取得します。
GetFloatArrayValue メソッド	この WSResult から float 値の配列を取得します。
GetFloatValue メソッド	この WSResult から float 値を取得します。

メンバ名	説明
GetInt16ArrayValue メソッド	この WSResult から Int16 値の配列を取得します。
GetInt16Value メソッド	この WSResult から Int16 値を取得します。
GetInt32ArrayValue メソッド	この WSResult から Int32 値の配列を取得します。
GetInt32Value メソッド	この WSResult から Int32 値を取得します。
GetInt64ArrayValue メソッド	この WSResult から Int64 値の配列を取得します。
GetInt64Value メソッド	この WSResult から Int64 値を取得します。
GetIntArrayValue メソッド	この WSResult から int 値の配列を取得します。
GetIntValue メソッド	この WSResult から ｉ nt 値を取得します。
GetLongArrayValue メソッド	この WSResult から long 値の配列を取得します。
GetLongValue メソッド	この WSResult から long 値を取得します。
GetNullableBoolArrayValue メソッド	この WSResult から bool 値の配列を取得します。
GetNullableBoolValue メソッド	この WSResult から bool 値を取得します。
GetNullableDecimalArrayValue メソッド	この WSResult から NullableDecimal 値の配列を取得します。
GetNullableDecimalValue メソッド	この WSResult から NullableDecimal 値を取得します。
GetNullableDoubleArrayValue メソッド	この WSResult から double 値の配列を取得します。
GetNullableDoubleValue メソッド	この WSResult から double 値を取得します。
GetNullableFloatArrayValue メソッド	この WSResult から float 値の配列を取得します。
GetNullableFloatValue メソッド	この WSResult から float 値を取得します。
GetNullableIntArrayValue メソッド	この WSResult から int 値の配列を取得します。
GetNullableIntValue メソッド	この WSResult から ｉ nt 値を取得します。
GetNullableLongArrayValue メソッド	この WSResult から long 値の配列を取得します。

メンバ名	説明
GetNullableLongValue メソッド	この WSResult から Int64 値を取得します。
GetNullableSByteArrayValue メソッド	この WSResult から byte 値の配列を取得します。
GetNullableSByteValue メソッド	この WSResult から byte 値を取得します。
GetNullableShortArrayValue メソッド	この WSResult から short 値の配列を取得します。
GetNullableShortValue メソッド	この WSResult から short 値を取得します。
GetObjectArrayValue メソッド	この WSResult から Object 値の配列を取得します。
GetObjectValue メソッド	この WSResult から object 値を取得します。
GetRequestID メソッド	この WSResult が表す要求 ID を取得します。
GetSByteArrayValue メソッド	この WSResult から sbyte 値の配列を取得します。
GetSByteValue メソッド	この WSResult から sbyte 値を取得します。
GetShortArrayValue メソッド	この WSResult から short 値の配列を取得します。
GetShortValue メソッド	この WSResult から short 値を取得します。
GetSingleArrayValue メソッド	この WSResult から単一値の配列を取得します。
GetSingleValue メソッド	この WSResult から単一値を取得します。
GetStatus メソッド	この WSResult のステータスを取得します。
GetStringArrayValue メソッド	この WSResult から文字列値の配列を取得します。
GetStringValue メソッド	この WSResult から文字列値を取得します。
GetUIntArrayValue メソッド	この WSResult から unsigned int 値の配列を取得します。
GetUIntValue メソッド	この WSResult から unsigned int 値を取得します。
GetULongArrayValue メソッド	この WSResult から unsigned long 値の配列を取得します。
GetULongValue メソッド	この WSResult から unsigned long 値を取得します。
GetUShortArrayValue メソッド	この WSResult から unsigned short 値の配列を取得します。
GetUShortValue メソッド	この WSResult から unsigned short 値を取得します。
GetValue メソッド	この WSResult から複合型の値を取得します。

メンバ名	説明
SetLogger メソッド	デバッグのオン／オフを切り替えます。

Acknowledge モード

この WSResult が処理されたことを確認します。

構文

Visual Basic
Public Sub **Acknowledge()**

C#
public void **Acknowledge();**

備考

このメソッドは、EXPLICIT_ACKNOWLEDGEMENT QAManager が使用されている場合にのみ有効です。

GetArrayValue メソッド

この WSResult から複合型の値の配列を取得します。

構文

Visual Basic
Public Function **GetArrayValue(_**
  ByVal *parentName* As String _
) As iAnywhere.QAnywhere.WS.WSSerializable()

C#
public iAnywhere.QAnywhere.WS.WSSerializable[] **GetArrayValue(**
  string *parentName*
);

パラメータ

- **parentName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetBoolArrayValue メソッド

この WSResult から bool 値の配列を取得します。

構文

Visual Basic
```
Public Function GetBoolArrayValue( _
  ByVal elementName As String _
) As Boolean()
```

C#
```
public bool[] GetBoolArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetBooleanArrayValue メソッド

この WSResult から Boolean 値の配列を取得します。

構文

Visual Basic
```
Public Function GetBooleanArrayValue( _
  ByVal elementName As String _
) As Boolean()
```

C#
```
public bool[] GetBooleanArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetBooleanValue メソッド

この WSResult から Boolean 値を取得します。

構文

Visual Basic
```
Public Function GetBooleanValue( _
  ByVal childName As String _
) As Boolean
```

C#
```
public bool GetBooleanValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetBoolValue メソッド

この WSResult から bool 値を取得します。

構文

Visual Basic
```
Public Function GetBoolValue( _
  ByVal childName As String _
) As Boolean
```

C#
```
public bool GetBoolValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetByteArrayValue メソッド

この WSResult から byte 値の配列を取得します。

構文

Visual Basic
```
Public Function GetByteArrayValue( _
  ByVal elementName As String _
) As Byte()
```

C#
```
public byte[] GetByteArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetByteValue メソッド

この WSResult から byte 値を取得します。

構文

Visual Basic
```
Public Function GetByteValue( _
  ByVal childName As String _
) As Byte
```

C#
```
public byte GetByteValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetCharArrayValue メソッド

この WSResult から char 値の配列を取得します。

構文

Visual Basic
Public Function **GetCharArrayValue(_**
  ByVal *elementName* As String _
) As Char()

C#
public char[] **GetCharArrayValue(**
  string *elementName*
);

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetCharValue メソッド

この WSResult から char 値を取得します。

構文

Visual Basic
Public Function **GetCharValue(_**
  ByVal *childName* As String _
) As Char

C#
public char **GetCharValue(**
  string *childName*
);

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetDecimalArrayValue メソッド

この WSResult から 10 進値の配列を取得します。

構文

Visual Basic
```
Public Function GetDecimalArrayValue( _
  ByVal elementName As String _
) As Decimal()
```

C#
```
public decimal[] GetDecimalArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetDecimalValue メソッド

この WSResult から 10 進値を取得します。

構文

Visual Basic
```
Public Function GetDecimalValue( _
  ByVal childName As String _
) As Decimal
```

C#
```
public decimal GetDecimalValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetDoubleArrayValue メソッド

この WSResult から double 値の配列を取得します。

構文

Visual Basic
```
Public Function GetDoubleArrayValue( _
  ByVal elementName As String _
) As Double()
```

C#
```
public double[] GetDoubleArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetDoubleValue メソッド

この WSResult から double 値を取得します。

構文

Visual Basic
```
Public Function GetDoubleValue( _
  ByVal childName As String _
) As Double
```

C#
```
public double GetDoubleValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetErrorMessage メソッド

エラー・メッセージを取得します。

構文

Visual Basic
Public Function **GetErrorMessage()** As String

C#
public string **GetErrorMessage();**

戻り値

エラー・メッセージ

GetFloatArrayValue メソッド

この WSResult から float 値の配列を取得します。

構文

Visual Basic
```
Public Function GetFloatArrayValue( _
  ByVal elementName As String _
) As Single()
```

C#
```
public float [] GetFloatArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetFloatValue メソッド

この WSResult から float 値を取得します。

構文

Visual Basic
```
Public Function GetFloatValue( _
  ByVal childName As String _
) As Single
```

C#
```
public float  GetFloatValue(
  string  childName
);
```


パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetInt16ArrayValue メソッド

この WSResult から Int16 値の配列を取得します。

構文

Visual Basic
```
Public Function GetInt16ArrayValue( _
  ByVal elementName As String _
) As Short()
```

C#
```
public short[] GetInt16ArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetInt16Value メソッド

この WSResult から Int16 値を取得します。

構文

Visual Basic
```
Public Function GetInt16Value( _
  ByVal childName As String _
) As Short
```

C#
```
public short GetInt16Value(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetInt32ArrayValue メソッド

この WSResult から Int32 値の配列を取得します。

構文

```
Visual Basic
Public Function GetInt32ArrayValue( _
  ByVal elementName As String _
) As Integer()

C#
public int[] GetInt32ArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetInt32Value メソッド

この WSResult から Int32 値を取得します。

構文

```
Visual Basic
Public Function GetInt32Value( _
  ByVal childName As String _
) As Integer

C#
public int GetInt32Value(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetInt64ArrayValue メソッド

この WSResult から Int64 値の配列を取得します。

構文

Visual Basic
```
Public Function GetInt64ArrayValue( _
  ByVal elementName As String _
) As Long()
```

C#
```
public long[] GetInt64ArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetInt64Value メソッド

この WSResult から Int64 値を取得します。

構文

Visual Basic
```
Public Function GetInt64Value( _
  ByVal childName As String _
) As Long
```

C#
```
public long GetInt64Value(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetIntArrayValue メソッド

この WSResult から int 値の配列を取得します。

構文

```
Visual Basic
Public Function GetIntArrayValue( _
  ByVal elementName As String _
) As Integer()

C#
public int[] GetIntArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetIntValue メソッド

この WSResult から i nt 値を取得します。

構文

```
Visual Basic
Public Function GetIntValue( _
  ByVal childName As String _
) As Integer

C#
public int GetIntValue(
  string  childName
);
```


パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetLongArrayValue メソッド

この WSResult から long 値の配列を取得します。

構文

Visual Basic
```
Public Function GetLongArrayValue( _
  ByVal elementName As String _
) As Long()
```

C#
```
public long[] GetLongArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetLongValue メソッド

この WSResult から long 値を取得します。

構文

Visual Basic
```
Public Function GetLongValue( _
  ByVal childName As String _
) As Long
```

C#
```
public long GetLongValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetNullableBoolArrayValue メソッド

この WSResult から bool 値の配列を取得します。

構文

Visual Basic
```
Public Function GetNullableBoolArrayValue( _
  ByVal elementName As String _
) As iAnywhere.QAnywhere.WS.NullableBool()
```

C#
```
public iAnywhere.QAnywhere.WS.NullableBool[] GetNullableBoolArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetNullableBoolValue メソッド

この WSResult から bool 値を取得します。

構文

Visual Basic
```
Public Function GetNullableBoolValue( _
  ByVal childName As String _
) As iAnywhere.QAnywhere.WS.NullableBool
```

C#
```
public iAnywhere.QAnywhere.WS.NullableBool GetNullableBoolValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetNullableDecimalArrayValue メソッド

この WSResult から NullableDecimal 値の配列を取得します。

構文

Visual Basic
```
Public Function GetNullableDecimalArrayValue( _
  ByVal elementName As String _
) As iAnywhere.QAnywhere.WS.NullableDecimal()
```

C#
```
public iAnywhere.QAnywhere.WS.Nullabledecimal[] GetNullableDecimalArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetNullableDecimalValue メソッド

この WSResult から NullableDecimal 値を取得します。

構文

Visual Basic
```
Public Function GetNullableDecimalValue( _
  ByVal childName As String _
) As iAnywhere.QAnywhere.WS.NullableDecimal
```

C#
```
public iAnywhere.QAnywhere.WS.Nullabledecimal GetNullableDecimalValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetNullableDoubleArrayValue メソッド

この WSResult から double 値の配列を取得します。

構文

Visual Basic
```
Public Function GetNullableDoubleArrayValue( _
  ByVal elementName As String _
) As iAnywhere.QAnywhere.WS.NullableDouble()
```

C#
```
public iAnywhere.QAnywhere.WS.Nullabledouble[] GetNullableDoubleArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetNullableDoubleValue メソッド

この WSResult から double 値を取得します。

構文

Visual Basic
```
Public Function GetNullableDoubleValue( _
  ByVal childName As String _
) As iAnywhere.QAnywhere.WS.NullableDouble
```

C#
```
public iAnywhere.QAnywhere.WS.Nullabledouble GetNullableDoubleValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetNullableFloatArrayValue メソッド

この WSResult から float 値の配列を取得します。

構文

Visual Basic
```
Public Function GetNullableFloatArrayValue( _
  ByVal elementName As String _
) As iAnywhere.QAnywhere.WS.NullableFloat()
```

C#
```
public iAnywhere.QAnywhere.WS.NullableFloat[] GetNullableFloatArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetNullableFloatValue メソッド

この WSResult から float 値を取得します。

構文

Visual Basic
```
Public Function GetNullableFloatValue( _
  ByVal childName As String _
) As iAnywhere.QAnywhere.WS.NullableFloat
```

C#
```
public iAnywhere.QAnywhere.WS.NullableFloat GetNullableFloatValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetNullableIntArrayValue メソッド

この WSResult から int 値の配列を取得します。

構文

```
Visual Basic
Public Function GetNullableIntArrayValue( _
  ByVal elementName As String _
) As iAnywhere.QAnywhere.WS.NullableInt()

C#
public iAnywhere.QAnywhere.WS.NullableInt[] GetNullableIntArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetNullableIntValue メソッド

この WSResult から i nt 値を取得します。

構文

```
Visual Basic
Public Function GetNullableIntValue( _
  ByVal childName As String _
) As iAnywhere.QAnywhere.WS.NullableInt

C#
public iAnywhere.QAnywhere.WS.NullableInt GetNullableIntValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetNullableLongArrayValue メソッド

この WSResult から long 値の配列を取得します。

構文

Visual Basic
```
Public Function GetNullableLongArrayValue( _
  ByVal elementName As String _
) As iAnywhere.QAnywhere.WS.NullableLong()
```

C#
```
public iAnywhere.QAnywhere.WS.NullableLong[] GetNullableLongArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetNullableLongValue メソッド

この WSResult から Int64 値を取得します。

構文

Visual Basic
```
Public Function GetNullableLongValue( _
  ByVal childName As String _
) As iAnywhere.QAnywhere.WS.NullableLong
```

C#
```
public iAnywhere.QAnywhere.WS.NullableLong GetNullableLongValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetNullableSByteArrayValue メソッド

この WSResult から byte 値の配列を取得します。

構文

Visual Basic
```
Public Function GetNullableSByteArrayValue( _
  ByVal elementName As String _
) As iAnywhere.QAnywhere.WS.NullableSByte()
```

C#
```
public iAnywhere.QAnywhere.WS.NullableSbyte[] GetNullableSByteArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetNullableSByteValue メソッド

この WSResult から byte 値を取得します。

構文

Visual Basic
```
Public Function GetNullableSByteValue( _
  ByVal childName As String _
) As iAnywhere.QAnywhere.WS.NullableSByte
```

C#
```
public iAnywhere.QAnywhere.WS.NullableSbyte GetNullableSByteValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetNullableShortArrayValue メソッド

この WSResult から short 値の配列を取得します。

構文

Visual Basic
```
Public Function GetNullableShortArrayValue( _
  ByVal elementName As String _
) As iAnywhere.QAnywhere.WS.NullableShort()
```

C#
```
public iAnywhere.QAnywhere.WS.NullableShort[] GetNullableShortArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetNullableShortValue メソッド

この WSResult から short 値を取得します。

構文

Visual Basic
```
Public Function GetNullableShortValue( _
  ByVal childName As String _
) As iAnywhere.QAnywhere.WS.NullableShort
```

C#
```
public iAnywhere.QAnywhere.WS.NullableShort GetNullableShortValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetObjectArrayValue メソッド

この WSResult から Object 値の配列を取得します。

構文

Visual Basic
```
Public Function GetObjectArrayValue( _
  ByVal elementName As String _
) As Object()
```

C#
```
public object[] GetObjectArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetObjectValue メソッド

この WSResult から object 値を取得します。

構文

Visual Basic
```
Public Function GetObjectValue( _
  ByVal childName As String _
) As Object
```

C#
```
public object GetObjectValue(
  string  childName
);
```


パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetRequestID メソッド

この WSResult が表す要求 ID を取得します。

構文

Visual Basic
```
Public Function GetRequestID() As String
```

C#
```
public string  GetRequestID();
```

戻り値

要求 ID

備考

要求の作成時とは異なるアプリケーションで実行される Web サービス要求に対応する WSResult を取得するために、この要求 ID が必要な場合は、アプリケーションの実行と実行の間でこの要求 ID を保持する必要があります。

GetSByteArrayValue メソッド

この WSResult から sbyte 値の配列を取得します。

構文

Visual Basic
```
Public Function GetSByteArrayValue( _
  ByVal elementName As String _
) As System.SByte()
```

C#
```
public System.Sbyte[] GetSByteArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetSByteValue メソッド

この WSResult から sbyte 値を取得します。

構文

Visual Basic
```
Public Function GetSByteValue( _
  ByVal childName As String _
) As System.SByte
```

C#
```
public System.Sbyte GetSByteValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetShortArrayValue メソッド

この WSResult から short 値の配列を取得します。

構文

Visual Basic
```
Public Function GetShortArrayValue( _
  ByVal elementName As String _
) As Short()
```

C#
```
public short[] GetShortArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetShortValue メソッド

この WSResult から short 値を取得します。

構文

```
Visual Basic
Public Function GetShortValue( _
  ByVal childName As String _
) As Short

C#
public short GetShortValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetSingleArrayValue メソッド

この WSResult から単一値の配列を取得します。

構文

```
Visual Basic
Public Function GetSingleArrayValue( _
  ByVal elementName As String _
) As Single()

C#
public float [] GetSingleArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetSingleValue メソッド

この WSResult から単一値を取得します。

構文

Visual Basic
```
Public Function GetSingleValue( _
  ByVal childName As String _
) As Single
```

C#
```
public float  GetSingleValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetStatus メソッド

この WSResult のステータスを取得します。

構文

Visual Basic
```
Public Function GetStatus() As iAnywhere.QAnywhere.WS.WSStatus
```

C#
```
public iAnywhere.QAnywhere.WS.WSStatus GetStatus();
```

戻り値

ステータス・コード

参照

- 「WSResult クラス」 384 ページ
- 「WSResult のメンバ」 385 ページ

- 「WSStatus 列挙」 418 ページ

GetStringArrayValue メソッド

この WSResult から文字列値の配列を取得します。

構文

Visual Basic
```
Public Function GetStringArrayValue( _
  ByVal elementName As String _
) As String()
```

C#
```
public string [] GetStringArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetStringValue メソッド

この WSResult から文字列値を取得します。

構文

Visual Basic
```
Public Function GetStringValue( _
  ByVal childName As String _
) As String
```

C#
```
public string  GetStringValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetUIntArrayValue メソッド

この WSResult から unsigned int 値の配列を取得します。

構文

Visual Basic
```
Public Function GetUIntArrayValue( _
  ByVal elementName As String _
) As UInt32()
```

C#
```
public uint[] GetUIntArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetUIntValue メソッド

この WSResult から unsigned int 値を取得します。

構文

Visual Basic
```
Public Function GetUIntValue( _
  ByVal childName As String _
) As UInt32
```

C#
```
public uint GetUIntValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetULongArrayValue メソッド

この WSResult から unsigned long 値の配列を取得します。

構文

Visual Basic
```
Public Function GetULongArrayValue( _
  ByVal elementName As String _
) As UInt64()
```

C#
```
public ulong[] GetULongArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetULongValue メソッド

この WSResult から unsigned long 値を取得します。

構文

Visual Basic
```
Public Function GetULongValue( _
  ByVal childName As String _
) As UInt64
```

C#
```
public ulong GetULongValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetUShortArrayValue メソッド

この WSResult から unsigned short 値の配列を取得します。

構文

```
Visual Basic
Public Function GetUShortArrayValue( _
  ByVal elementName As String _
) As UInt16()

C#
public ushort[] GetUShortArrayValue(
  string  elementName
);
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetUShortValue メソッド

この WSResult から unsigned short 値を取得します。

構文

```
Visual Basic
Public Function GetUShortValue( _
  ByVal childName As String _
) As UInt16

C#
public ushort GetUShortValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

GetValue メソッド

この WSResult から複合型の値を取得します。

構文

Visual Basic
```
Public Function GetValue( _
  ByVal childName As String _
) As Object
```

C#
```
public object GetValue(
  string  childName
);
```

パラメータ

- **childName**　この値の WSDL ドキュメント内の要素名

戻り値

値

例外

- WSException クラス - 値の取得で問題が発生した場合にスローされます。

SetLogger メソッド

デバッグのオン／オフを切り替えます。

構文

Visual Basic
```
Public Sub SetLogger( _
  ByVal wsLogger As iAnywhere.QAnywhere.WS.WSLogger _
)
```

C#
```
public void SetLogger(
  iAnywhere.QAnywhere.WS.WSLogger wsLogger
);
```

WSStatus 列挙

このクラスは、Web サービス要求のステータス・コードを定義します。

構文

Visual Basic
Public Enum **WSStatus**

C#
public enum **WSStatus**

メンバ名

メンバ名	説明
STATUS_ERROR	要求の処理でエラーがありました。
STATUS_QUEUED	要求はサーバへの配信用キューに登録されました。
STATUS_RESULT_AVAILABLE	要求の結果を取得できます。
STATUS_SUCCESS	要求は正常に処理されました。

第 13 章

QAnywhere C++ API リファレンス

目次

AcknowledgementMode クラス

構文

public **AcknowledgementMode**

備考

QAnywhere クライアント・アプリケーションによってどのようにメッセージが確認されるかを示します。

IMPLICIT_ACKNOWLEDGEMENT モードと EXPLICIT_ACKNOWLEDGEMENT モードは、QAManageropen() メソッドを使用して、QAManager インスタンスに割り当てられます。TRANSACTIONAL モードは、QATransactionalManager インスタンスに暗黙的に割り当てられます。

詳細については、「QAnywhere API の初期化」 61 ページを参照してください。

暗黙的な受信確認モードでは、メッセージは、クライアント・アプリケーションで受信されるとすぐに受信確認されます。明示的な受信確認モードでは、QAManager のいずれかの受信確認メソッドを呼び出してください。トランザクション志向モードでは、QATransactionalManagercommit () メソッドを呼び出して、未確認のすべてのメッセージの受信を確認します。すべてのステータス変更は、サーバによってクライアント間で伝達されます。

詳細については、「同期的なメッセージ受信」 82 ページと「非同期的なメッセージ受信」 83 ページを参照してください。

トランザクション志向メッセージングの場合は、QATransactionalManager を使用します。この場合は、QATransactionalManagercommit メソッドを使用して、トランザクションに属するメッセージの受信を確認します。

QAManagerBase インスタンスのモードは、QAManagerBaseMode プロパティを使用して判別できます。

参照

QAManager クラス

QATransactionalManager クラス

QAManagerBase クラス

メンバ

AcknowledgementMode クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「EXPLICIT_ACKNOWLEDGEMENT 変数」 423 ページ
- 「IMPLICIT_ACKNOWLEDGEMENT 変数」 423 ページ
- 「TRANSACTIONAL 変数」 423 ページ

EXPLICIT_ACKNOWLEDGEMENT 変数

構文

const qa_short **AcknowledgementMode::EXPLICIT_ACKNOWLEDGEMENT**

備考

受信メッセージは、QAManager のいずれかの受信確認メソッドを使用して確認されます。

IMPLICIT_ACKNOWLEDGEMENT 変数

構文

const qa_short **AcknowledgementMode::IMPLICIT_ACKNOWLEDGEMENT**

備考

すべてのメッセージは、クライアント・アプリケーションで受信されるとすぐに受信確認されます。

メッセージを同期に受信する場合、メッセージは QAManagerBasegetMessage メソッドが返されるとすぐに受信確認されます。メッセージを非同期に受信する場合、メッセージはイベント処理関数が返されるとすぐに受信確認されます。

TRANSACTIONAL 変数

構文

const qa_short **AcknowledgementMode::TRANSACTIONAL**

備考

メッセージの受信確認はトランザクションの一部として行われます。

このモードは、QATransactionalManager インスタンスに自動的に割り当てられます。

MessageProperties クラス

構文

public **MessageProperties**

備考

標準のメッセージ・プロパティ名を格納するフィールドを提供します。

MessageProperties クラスは、標準のメッセージ・プロパティ名を提供します。MessageProperties フィールドは、メッセージ・プロパティの取得と設定で使用する QAMessage メソッドに渡すことができます。

詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

```
QATextMessage * t_msg;
```

次の例では、QAMessagegetIntProperty メソッドを使用して MessagePropertiesMSG_TYPE に対応する値を取得します。MessageType 列挙は、整数値の結果を適切なメッセージ・タイプにマッピングします。

```
int msg_type;
t_msg->getIntProperty( MessageProperties::MSG_TYPE, &msg_type)
```

次の例では、MessagePropertiesMSG_TYPE と MessagePropertiesRASNAMES を使用して、それぞれメッセージ・タイプと RAS 名を評価します。

```
void SystemQueueListener::onMessage(QAMessage * msg) {
   QATextMessage *       t_msg;
   TCHAR                 buffer[512];
   int                   len;
   int                   msg_type;

   t_msg = msg->castToTextMessage();
   if( t_msg != NULL ) {
    t_msg->getIntProperty( MessageProperties::MSG_TYPE, &msg_type );

  if( msg_type == MessageType::NETWORK_STATUS_NOTIFICATION ) {

   // get RAS names using MessageProperties::RASNAMES
   len = t_msg->getStringProperty(MessageProperties::RASNAMES,buffer,sizeof(buffer));

   }

  //...
  }
 }
```

参照

QAMessage クラス

メンバ

MessageProperties クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

ADAPTER 変数

構文

const qa_string **MessageProperties::ADAPTER**

備考

このプロパティ名は、QAnywhere サーバへの接続で使用されている、現在アクティブなネットワーク・アダプタを示します。

これは、システム・キュー・メッセージで使用されます。

このフィールドの値は "ias_Network.Adapter" です。

MessagePropertiesADAPTER を最初のパラメータとして QAMessagegetStringProperty メソッドに渡し、関連付けられているメッセージ・プロパティにアクセスします。

詳細については、「メッセージ・プロパティ」 223 ページを参照してください。

参照

getStringProperty 関数

ADAPTERS 変数

構文

const qa_string **MessageProperties::ADAPTERS**

備考

このプロパティ名は、QAnywhere サーバへの接続で使用できるネットワーク・アダプタのデリミタ付きリストを示します。

これは、システム・キュー・メッセージで使用されます。

このフィールドの値は "ias_Adapters" です。

MessagePropertiesADAPTERS を最初のパラメータとして QAMessagegetStringProperty メソッドに渡し、関連付けられているメッセージ・プロパティにアクセスします。

詳細については、「メッセージ・プロパティ」 223 ページを参照してください。 getStringProperty 関数

DELIVERY_COUNT 変数

構文

const qa_string **MessageProperties::DELIVERY_COUNT**

備考

このプロパティ名は、メッセージを配信するために、これまでに実行された試行回数を示します。

このフィールドの値は "ias_DeliveryCount" です。

MessagePropertiesDELIVERY_COUNT を最初のパラメータとして QAMessagesetStringProperty メソッドまたは QAMessagegetStringProperty メソッドに渡し、関連付けられているメッセージ・プロパティにアクセスします。

参照

setStringProperty 関数

getStringProperty 関数

IP 変数

構文

const qa_string **MessageProperties::IP**

備考

このプロパティ名は、QAnywhere サーバへの接続で使用されている、現在アクティブなネットワーク・アダプタの IP アドレスを示します。

これは、システム・キュー・メッセージで使用されます。

このフィールドの値は "ias_Network.IP" です。

MessagePropertiesIP を最初のパラメータとして QAMessagegetStringProperty メソッドに渡し、関連付けられているメッセージ・プロパティにアクセスします。

詳細については、「メッセージ・プロパティ」 223 ページを参照してください。

参照

getStringProperty 関数

MAC 変数

構文

const qa_string **MessageProperties::MAC**

備考

このプロパティ名は、QAnywhere サーバへの接続で使用されている、現在アクティブなネットワーク・アダプタの MAC アドレスを示します。

これは、システム・キュー・メッセージで使用されます。

このフィールドの値は "ias_Network.MAC" です。

MessagePropertiesMAC を最初のパラメータとして QAMessagegetStringProperty メソッドに渡し、関連付けられているメッセージ・プロパティにアクセスします。

詳細については、「メッセージ・プロパティ」 223 ページを参照してください。

参照

getStringProperty 関数

MSG_TYPE 変数

構文

const qa_string **MessageProperties::MSG_TYPE**

備考

このプロパティ名は、QAnywhere メッセージに関連付けられている MessageType 列挙値を示します。

このフィールドの値は "ias_MessageType" です。MessagePropertiesMSG_TYPE を最初のパラメータとして QAMessagesetIntProperty メソッドまたは QAMessagegetIntProperty メソッドに渡し、関連付けられているプロパティを判断します。

参照

MessageType クラス

setIntProperty 関数

getIntProperty 関数

NETWORK_STATUS 変数

構文

const qa_string **MessageProperties::NETWORK_STATUS**

備考

このプロパティ名は、ネットワーク接続のステータスを示します。

このフィールドの値は "ias_NetworkStatus" です。

ネットワークが接続可能な場合、このプロパティの値は 1 です。それ以外の場合は 0 です。ネットワーク・ステータスは、システム・キュー・メッセージ (ネットワーク・ステータスの変更など) で使用されます。

詳細については、「メッセージ・プロパティ」 223 ページを参照してください。

MessagePropertiesNETWORK_STATUS を最初のパラメータとして QAMessagesetStringProperty メソッドまたは QAMessagegetStringProperty メソッドに渡し、関連付けられているメッセージ・プロパティにアクセスします。

参照

setStringProperty 関数

getStringProperty 関数

ORIGINATOR 変数

構文

const qa_string **MessageProperties::ORIGINATOR**

備考

このプロパティ名は、メッセージの発信者のメッセージ・ストア ID を示します。

このフィールドの値は "ias_Originator" です。

MessagePropertiesORIGINATOR を最初のパラメータとして QAMessagesetStringProperty メソッドまたは QAMessagegetStringProperty メソッドに渡し、関連付けられているメッセージ・プロパティにアクセスします。

参照

setStringProperty 関数

getStringProperty 関数

RAS 変数

構文

const qa_string **MessageProperties::RAS**

備考

このプロパティ名は、QAnywhere サーバへの接続で使用されている、現在アクティブな RAS 名を示します。

これは、システム・キュー・メッセージで使用されます。

このフィールドの値は "ias_Network.RAS" です。

MessagePropertiesRAS を最初のパラメータとして QAMessagegetStringProperty メソッドに渡し、関連付けられているメッセージ・プロパティにアクセスします。

詳細については、「メッセージ・プロパティ」 223 ページを参照してください。

参照

getStringProperty 関数

RASNAMES 変数

構文

const qa_string **MessageProperties::RASNAMES**

備考

このプロパティ名は、QAnywhere サーバへの接続で使用できる RAS エントリ名のデリミタ付きリストを示します。

これは、システム・キュー・メッセージで使用されます。

このフィールドの値は "ias_RASNames" です。

詳細については、「メッセージ・プロパティ」 223 ページを参照してください。

MessagePropertiesRASNAMES を最初のパラメータとして QAMessagesetStringProperty メソッドまたは QAMessagegetStringProperty メソッドに渡し、関連付けられているメッセージ・プロパティにアクセスします。

参照

setStringProperty 関数

getStringProperty 関数

setIntProperty 関数

getIntProperty 関数

STATUS 変数

構文

const qa_string **MessageProperties::STATUS**

備考

このプロパティ名は、メッセージの現在のステータスを示します。

値のリストについては、StatusCodes クラスを参照してください。このフィールドの値は "ias_Status" です。

MessagePropertiesSTATUS を最初のパラメータとして QAMessagesetIntProperty メソッドまたは QAMessagegetIntProperty メソッドに渡し、関連付けられているメッセージ・プロパティにアクセスします。

参照

StatusCodes クラス

setIntProperty 関数

getIntProperty 関数

STATUS_TIME 変数

構文

const qa_string **MessageProperties::STATUS_TIME**

備考

このプロパティ名は、メッセージが現在のステータスを受信した時刻を示します。

タイムスタンプには、プラットフォームに合わせたフォーマットが使用されます。Windows/PocketPC プラットフォームのタイムスタンプは SYSTEMTIME フォーマットです。これが FILETIME フォーマットに変換されて、qa_long 値にコピーされます。これはローカル時間です。このフィールドの値は "ias_StatusTime" です。

MessagePropertiesSTATUS_TIME を最初のパラメータとして QAMessagegetLongProperty メソッドに渡し、関連付けられている読み込み専用のメッセージ・プロパティにアクセスします。

参照

getLongProperty 関数

TRANSMISSION_STATUS 変数

構文

const qa_string **MessageProperties::TRANSMISSION_STATUS**

備考

このプロパティ名は、メッセージの現在の転送ステータスを示します。

値のリストについては、StatusCodes クラスを参照してください。

このフィールドの値は "ias_TransmissionStatus" です。

MessagePropertiesTRANSMISSION_STATUS を最初のパラメータとして QAMessagesetIntProperty メソッドまたは QAMessagegetIntProperty メソッドに渡し、関連付けられているメッセージ・プロパティにアクセスします。

参照

StatusCodes クラス

setIntProperty 関数

getIntProperty 関数

MessageStoreProperties クラス

構文

public **MessageStoreProperties**

備考

MessageStoreProperties クラスは、標準のメッセージ・プロパティ名を提供します。

MessageStoreProperties フィールドは、事前定義済みまたはカスタムのメッセージ・ストア・プロパティの取得と設定で使用する QAManagerBase メソッドに渡すことができます。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

メンバ

MessageStoreProperties クラスのすべてのメンバ (継承されたメンバも含む) を以下に示します。

- 「MAX_DELIVERY_ATTEMPTS 変数」 432 ページ

MAX_DELIVERY_ATTEMPTS 変数

構文

const qa_string **MessageStoreProperties::MAX_DELIVERY_ATTEMPTS**

備考

このプロパティ名は、メッセージのステータスが StatusCodesUNRECEIVABLE に設定されるまでに、明示的な受信確認がないままメッセージを受信できる最大回数を示します。

このフィールドの値は "ias_MaxDeliveryAttempts" です。

参照

StatusCodes クラス

MessageType クラス

構文

public **MessageType**

備考

MessagePropertiesMSG_TYPE メッセージ・プロパティの定数値を定義します。

次の例は、QAnywhere システム・メッセージの処理で使用される onSystemMessage メソッドを示します。

メッセージ・タイプは、MessageType.NETWORK_STATUS_NOTIFICATION と比較されます。

```
void SystemQueueListener::onMessage(QAMessage * msg)
{
    QATextMessage *       t_msg;
    TCHAR                 buffer[512];
    int                   len;
    int                   msg_type;

    t_msg = msg->castToTextMessage();
    if( t_msg != NULL ) {
     t_msg->getIntProperty( MessageProperties::MSG_TYPE, &msg_type );
    if( msg_type == MessageType::NETWORK_STATUS_NOTIFICATION ) {

     // get network names using MessageProperties::NETWORK
    len = t_msg->getStringProperty(MessageProperties::NETWORK,buffer,sizeof(buffer));

}

//...
}
}
```

メンバ

MessageType クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「NETWORK_STATUS_NOTIFICATION 変数」 433 ページ
- 「PUSH_NOTIFICATION 変数」 434 ページ
- 「REGULAR 変数」 434 ページ

NETWORK_STATUS_NOTIFICATION 変数

構文

const qa_int **MessageType::NETWORK_STATUS_NOTIFICATION**

備考

QAnywhere クライアント・アプリケーションにネットワーク・ステータスの変更を通知する場合に使用する、QAnywhere システム・メッセージを特定します。

ネットワーク・ステータスの変更は、システム・メッセージを受信するデバイスに適用されます。新しいネットワーク・ステータス情報を特定するには、MessagePropertiesADAPTER、MessagePropertiesNETWORK、MessagePropertiesNETWORK_STATUS の各フィールドを使用します。

詳細については、「事前に定義されたメッセージ・プロパティ」 224 ページを参照してください。

PUSH_NOTIFICATION 変数

構文

const qa_int **MessageType::PUSH_NOTIFICATION**

備考

QAnywhere クライアント・アプリケーションに Push 通知を通知する場合に使用する、QAnywhere システム・メッセージを特定します。

オンデマンドの QAnywhere Agent ポリシーを使用した場合の一般的な応答は、QAManagerBasetriggerSendReceive() メソッドを呼び出し、中央のメッセージ・サーバで待機しているメッセージを受信することです。

詳細については、「事前に定義されたメッセージ・プロパティ」 224 ページを参照してください。

REGULAR 変数

構文

const qa_int **MessageType::REGULAR**

備考

メッセージ・タイプ・プロパティが存在しない場合、メッセージ・タイプは REGULAR とみなされます。

このタイプのメッセージは、メッセージ・システムで特別な取り扱いを受けません。

QABinaryMessage クラス

構文

public **QABinaryMessage**

基本クラス

- 「QAMessage クラス」 494 ページ

備考

QABinaryMessage オブジェクトは、未解釈のバイト・ストリームが含まれるメッセージの送信に使用します。

これは QAMessage クラスを継承したもので、メッセージ本文にバイト・ストリームが追加されます。QABinaryMessage には、メッセージ本文からのバイト・ストリームの読み込み／書き込みを行うためのさまざまなメソッドがあります。

メッセージが最初に作成された時点では、メッセージ本文は書き込み専用になっています。メッセージ送信元のクライアントは、メッセージを送信した後、そのメッセージを保持し変更できます。ただし、それによって、送信されたメッセージが変更されることはありません。同じメッセージ・オブジェクトを複数回送信できます。

メッセージが受信されるとプロバイダが QABinaryMessagereset メソッドを呼び出します。これによりメッセージ本文が読み込み専用モードになり、メッセージ本文の先頭から値の読み込みが開始されます。クライアントが読み込み専用モードのメッセージに書き込もうとすると、COMMON_MSG_NOT_WRITEABLE_ERROR が設定されます。

次の例では QABinaryMessagewriteString メソッドを使用して、QABinaryMessage インスタンスのメッセージ本文に文字列 "Q" と "Anywhere" を書き込みます。

```
// Create a binary message instance.
QABinaryMessage * binary_message;
binary_message = qa_manager->createBinaryMessage();

// Set optional message properties.
binary_message->setReplyToAddress( "my-queue-name" );

// Write to the message body.
binary_message->writeString("Q");
binary_message->writeString("Anywhere");

// Put the message in the local database, ready for sending.

if( !qa_manager->putMessage( "store-id¥¥queue-name", msg ) ) {
        handleError();
    }
```

*注意：*受信が終了すると、最初の QABinaryMessagereadString() の呼び出しから "Q" が返され、2 番目の QABinaryMessagereadString() の呼び出しから "Anywhere" が返されます。

メッセージは QAnywhere Agent から送信されます。

詳細については、「クライアントにメッセージを転送するタイミングの決定」 40 ページと「QAnywhere クライアント・アプリケーションの作成」 51 ページを参照してください。

メンバ

QABinaryMessage クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

getBodyLength 関数

構文

qa_long **QABinaryMessage::getBodyLength()**

備考

メッセージ本文のサイズをバイト単位で返します。

戻り値

メッセージ本文のサイズ (バイト単位)

readBinary 関数

構文

```
qa_int QABinaryMessage::readBinary(
  qa_bytes value,
  qa_int length
)
```

パラメータ

- **value**　データの読み込み先バッファ

- **length**　読み込むバイト数の最大値

備考

QABinaryMessage インスタンスのメッセージ本文の未読部分から、指定されたバイト数を読み込みます。

参照

writeBinary 関数

戻り値

バッファに読み込まれたバイトの合計数。ストリームの終わりに到達したため読み込めるデータが残っていない場合は -1。

readBoolean 関数

構文

```
qa_bool QABinaryMessage::readBoolean(
  qa_bool * value
)
```

パラメータ

- **value**　バイト・メッセージ・ストリームから読み込んだ qa_bool 値の格納先

備考

QABinaryMessage インスタンスのメッセージ本文の未読部分から、boolean 値を読み込みます。

参照

writeBoolean 関数

戻り値

処理が正常終了した場合のみ True

readByte 関数

構文

```
qa_bool QABinaryMessage::readByte(
  qa_byte * value
)
```

パラメータ

- **value**　バイト・メッセージ・ストリームから読み込んだ qa_byte 値の格納先

備考

QABinaryMessage インスタンスのメッセージ本文の未読部分から、符号付き 8 ビット値を読み込みます。

参照

writeByte 関数

戻り値

処理が正常終了した場合のみ True

readChar 関数

構文

```
qa_bool QABinaryMessage::readChar(
  qa_char * value
)
```

パラメータ

- **value**　バイト・メッセージ・ストリームから読み込んだ qa_char 値の格納先

備考

QABinaryMessage インスタンスのメッセージ本文の未読部分から、char 値を読み込みます。

参照

writeChar 関数

戻り値

読み込まれた cahr 値

readDouble 関数

構文

```
qa_bool QABinaryMessage::readDouble(
  qa_double * value
)
```

パラメータ

- **value**　バイト・メッセージ・ストリームから読み込んだ double 値の格納先

備考

QABinaryMessage インスタンスのメッセージ本文の未読部分から、double 値を読み込みます。

参照

writeDouble 関数

戻り値

処理が正常終了した場合のみ True

readFloat 関数

構文

```
qa_bool QABinaryMessage::readFloat(
  qa_float * value
)
```

パラメータ

- **value**　バイト・メッセージ・ストリームから読み込んだ float 値の格納先

備考

QABinaryMessage インスタンスのメッセージ本文の未読部分から、float 値を読み込みます。

参照

writeFloat 関数

戻り値

処理が正常終了した場合のみ True

readInt 関数

構文

```
qa_bool QABinaryMessage::readInt(
  qa_int * value
)
```

パラメータ

- **value**　バイト・メッセージ・ストリームから読み込んだ qa_int 値の格納先

備考

QABinaryMessage インスタンスのメッセージ本文の未読部分から、符号付き 32 ビット整数値を読み込みます。

参照

writeInt 関数

戻り値

処理が正常終了した場合のみ True

readLong 関数

構文

```
qa_bool QABinaryMessage::readLong(
  qa_long * value
)
```

パラメータ

- **value**　バイト・メッセージ・ストリームから読み込んだ long 値の格納先

備考

QABinaryMessage インスタンスのメッセージ本文の未読部分から、符号付き 64 ビット整数値を読み込みます。

参照

writeLong 関数

戻り値

処理が正常終了した場合のみ True

readShort 関数

構文

```
qa_bool QABinaryMessage::readShort(
  qa_short * value
)
```

パラメータ

- **value**　バイト・メッセージ・ストリームから読み込んだ qa_short 値の格納先

備考

QABinaryMessage インスタンスのメッセージ本文の未読部分から、符号付き 16 ビット値を読み込みます。

参照

writeShort 関数

戻り値

処理が正常終了した場合のみ True

readString 関数

構文

```
qa_int QABinaryMessage::readString(
  qa_string dest,
  qa_int maxLen
)
```

パラメータ

- **dest**　バイト・メッセージ・ストリームから読み込んだ qa_string 値の格納先
- **maxLen**　読み込む文字の最大数 (null 終端子も含みます)。

備考

QABinaryMessage インスタンスのメッセージ本文の未読部分から、文字列値を読み込みます。

参照

writeString 関数

戻り値

バッファに読み込まれた null 以外の qa_char の合計数。読み込めるデータが残っていない場合、またはエラーが発生した場合は -1。バッファが小さすぎる場合は -2。

reset 関数

構文

```
void QABinaryMessage::reset()
```

備考

メッセージをリセットして、メッセージ本文の先頭から値の読み込みを開始できるようにします。

また、reset メソッドは、QABinaryMessage のメッセージ本文を読み込み専用モードにします。

writeBinary 関数

構文

```
void QABinaryMessage::writeBinary(
  qa_const_bytes value,
  qa_int offset,
  qa_int length
)
```

パラメータ

- **value**　メッセージ本文に書き込むバイト配列値
- **offset**　バイト配列内でのオフセット (書き込み開始位置)
- **length**　書き込まれるバイト数

備考

QABinaryMessage インスタンスのメッセージ本文にバイト配列値を追加します。

参照

readBinary 関数

writeBoolean 関数

構文

```
void QABinaryMessage::writeBoolean(
  qa_bool value
)
```

パラメータ

- **value**　メッセージ本文に書き込む boolean 値

備考

QABinaryMessge インスタンスのメッセージ本文に boolean 値を追加します。

boolean 値は 1 バイトの値で示されます。True は 1、false は 0 で示されます。

参照

readBoolean 関数

writeByte 関数

構文

```
void QABinaryMessage::writeByte(
  qa_byte value
)
```

パラメータ

- **value**　メッセージ本文に書き込むバイト配列値

備考

QABinaryMessge インスタンスのメッセージ本文に byte 値を追加します。

byte 値は 1 バイトの値で示されます。

参照

readByte 関数

writeChar 関数

構文

```
void QABinaryMessage::writeChar(
  qa_char value
)
```

パラメータ

- **value**　メッセージ本文に書き込む char 値

備考

QABinaryMessge インスタンスのメッセージ本文に char 値を追加します。

char パラメータは 2 バイトの値で示され、上位のバイトから追加されます。

参照

readChar 関数

writeDouble 関数

構文

```
void QABinaryMessage::writeDouble(
  qa_double value
)
```

パラメータ

- **value**　メッセージ本文に書き込む double 値

備考

QABinaryMessge インスタンスのメッセージ本文に double 値を追加します。

double パラメータは 8 バイトの long 値に変換されます。上位のバイトから追加されます。

参照

readDouble 関数

writeFloat 関数

構文

void **QABinaryMessage::writeFloat(**
  qa_float *value*
)

パラメータ

- **value**　メッセージ本文に書き込む float 値

備考

QABinaryMessge インスタンスのメッセージ本文に float 値を追加します。

float パラメータは 4 バイトの整数に変換され、上位のバイトから追加されます。

参照

readFloat 関数

writeInt 関数

構文

void **QABinaryMessage::writeInt(**
  qa_int *value*
)

パラメータ

- **value**　メッセージ本文に書き込む int 値

備考

QABinaryMessge インスタンスのメッセージ本文に整数値を追加します。

整数パラメータは 4 バイトの値で示され、上位のバイトから追加されます。

参照

readInt 関数

writeLong 関数

構文

void **QABinaryMessage::writeLong(**
  qa_long *value*
)

パラメータ

- **value**　メッセージ本文に書き込む long 値

備考

QABinaryMessge インスタンスのメッセージ本文に long 値を追加します。

long パラメータは 8 バイトの値で示され、上位のバイトから追加されます。

参照

readLong 関数

writeShort 関数

構文

```
void QABinaryMessage::writeShort(
  qa_short value
)
```

パラメータ

- **value**　メッセージ本文に書き込む short 値

備考

QABinaryMessge インスタンスのメッセージ本文に short 値を追加します。

short パラメータは 2 バイトの値で示され、上位のバイトから追加されます。

参照

readShort 関数

writeString 関数

構文

```
void QABinaryMessage::writeString(
  qa_const_string value
)
```

パラメータ

- **value**　メッセージ本文に書き込む文字列値

備考

QABinaryMessge インスタンスのメッセージ本文に文字列値を追加します。

注意 : 受信側アプリケーションは、writeString が呼び出されるたびに QABinaryMessagereadString を呼び出す必要があります。

注意 : UTF-8 表記の文字列は、最大で 32767 バイトまで可能です。

参照

readString 関数

~QABinaryMessage 関数

構文

virtual **QABinaryMessage::~QABinaryMessage()**

備考

仮想デストラクタです。

QAError クラス

構文

public **QAError**

備考

このクラスは、QAnywhere クライアント・アプリケーションに関連付けられているエラー定数を定義します。

QAError オブジェクトは、メッセージング操作と対応するエラーを追跡するために、QAManager オブジェクトによって内部的に使用されます。アプリケーション・プログラマが、このクラスをインスタンス化する必要はありません。アプリケーション・プログラマはこれらのエラー定数を使用して、QAManagergetLastError から返されたエラー・コードを解釈します。

```
if (qa_mgr->getLastError() != QAError::QA_NO_ERROR)
{
  // Process error.
}
```

参照

getLastErrorMsg 関数

メンバ

QAError クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「COMMON_ALREADY_OPEN_ERROR 変数」 449 ページ
- 「COMMON_GET_INIT_FILE_ERROR 変数」 449 ページ
- 「COMMON_GETQUEUEDEPTH_ERROR 変数」 449 ページ
- 「COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG 変数」 449 ページ
- 「COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID 変数」 449 ページ
- 「COMMON_INIT_ERROR 変数」 450 ページ
- 「COMMON_INIT_THREAD_ERROR 変数」 450 ページ
- 「COMMON_INVALID_PROPERTY 変数」 450 ページ
- 「COMMON_MSG_ACKNOWLEDGE_ERROR 変数」 450 ページ
- 「COMMON_MSG_CANCEL_ERROR 変数」 450 ページ
- 「COMMON_MSG_CANCEL_ERROR_SENT 変数」 451 ページ
- 「COMMON_MSG_NOT_WRITEABLE_ERROR 変数」 451 ページ
- 「COMMON_MSG_RETRIEVE_ERROR 変数」 451 ページ
- 「COMMON_MSG_STORE_ERROR 変数」 451 ページ
- 「COMMON_MSG_STORE_NOT_INITIALIZED 変数」 451 ページ
- 「COMMON_MSG_STORE_TOO_LARGE 変数」 452 ページ
- 「COMMON_NO_DEST_ERROR 変数」 452 ページ
- 「COMMON_NO_IMPLEMENTATION 変数」 452 ページ
- 「COMMON_NOT_OPEN_ERROR 変数」 452 ページ
- 「COMMON_OPEN_ERROR 変数」 452 ページ
- 「COMMON_OPEN_LOG_FILE_ERROR 変数」 452 ページ

COMMON_ALREADY_OPEN_ERROR 変数

構文

const qa_int **QAError::COMMON_ALREADY_OPEN_ERROR**

備考

QAManager はすでにオープンされています。

COMMON_GETQUEUEDEPTH_ERROR 変数

構文

const qa_int **QAError::COMMON_GETQUEUEDEPTH_ERROR**

備考

キューの深さを取得中にエラーが発生しました。

COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG 変数

構文

const qa_int **QAError::COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG**

備考

フィルタが ALL の場合は、指定された送信先で QAManagerBase.getQueueDepth を使用できません。

COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID 変数

構文

const qa_int **QAError::COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID**

備考

メッセージ・ストア ID が設定されていない場合は、QAManagerBase.getQueueDepth を使用できません。

COMMON_GET_INIT_FILE_ERROR 変数

構文

const qa_int **QAError::COMMON_GET_INIT_FILE_ERROR**

備考

クライアント・プロパティ・ファイルにアクセスできません。

COMMON_INIT_ERROR 変数

構文

const qa_int **QAError::COMMON_INIT_ERROR**

備考

初期化エラーです。

COMMON_INIT_THREAD_ERROR 変数

構文

const qa_int **QAError::COMMON_INIT_THREAD_ERROR**

備考

バックグラウンド・スレッドを初期化するときにエラーが発生しました。

COMMON_INVALID_PROPERTY 変数

構文

const qa_int **QAError::COMMON_INVALID_PROPERTY**

備考

クライアント・プロパティ・ファイル内に無効なプロパティが存在します。

COMMON_MSG_ACKNOWLEDGE_ERROR 変数

構文

const qa_int **QAError::COMMON_MSG_ACKNOWLEDGE_ERROR**

備考

メッセージの受信確認中にエラーが発生しました。

COMMON_MSG_CANCEL_ERROR 変数

構文

const qa_int **QAError::COMMON_MSG_CANCEL_ERROR**

備考

メッセージのキャンセル中にエラーが発生しました。

COMMON_MSG_CANCEL_ERROR_SENT 変数

構文

const qa_int **QAError::COMMON_MSG_CANCEL_ERROR_SENT**

備考

メッセージのキャンセル中にエラーが発生しました。

送信済みのメッセージはキャンセルできません。

COMMON_MSG_NOT_WRITEABLE_ERROR 変数

構文

const qa_int **QAError::COMMON_MSG_NOT_WRITEABLE_ERROR**

備考

読み込み専用モードであるため、メッセージに書き込みできません。

COMMON_MSG_RETRIEVE_ERROR 変数

構文

const qa_int **QAError::COMMON_MSG_RETRIEVE_ERROR**

備考

クライアント・メッセージ・ストアからメッセージを取得するときにエラーが発生しました。

COMMON_MSG_STORE_ERROR 変数

構文

const qa_int **QAError::COMMON_MSG_STORE_ERROR**

備考

クライアント・メッセージ・ストアにメッセージを格納するときにエラーが発生しました。

COMMON_MSG_STORE_NOT_INITIALIZED 変数

構文

const qa_int **QAError::COMMON_MSG_STORE_NOT_INITIALIZED**

備考

メッセージ・ストアがメッセージング用に初期化されていません。

COMMON_MSG_STORE_TOO_LARGE 変数

構文

const qa_int **QAError::COMMON_MSG_STORE_TOO_LARGE**

備考

メッセージ・ストアが大き過ぎて、デバイス上のディスクの空き領域に収まりません。

COMMON_NOT_OPEN_ERROR 変数

構文

const qa_int **QAError::COMMON_NOT_OPEN_ERROR**

備考

QAManager がオープンされていません。

COMMON_NO_DEST_ERROR 変数

構文

const qa_int **QAError::COMMON_NO_DEST_ERROR**

備考

送信先が指定されていません。

COMMON_NO_IMPLEMENTATION 変数

構文

const qa_int **QAError::COMMON_NO_IMPLEMENTATION**

備考

関数が実装されていません。

COMMON_OPEN_ERROR 変数

構文

const qa_int **QAError::COMMON_OPEN_ERROR**

備考

メッセージ・ストアへの接続をオープンするときにエラーが発生しました。

COMMON_OPEN_LOG_FILE_ERROR 変数

構文

const qa_int **QAError::COMMON_OPEN_LOG_FILE_ERROR**

備考

ログ・ファイルをオープンするときにエラーが発生しました。

COMMON_OPEN_MAXTHREADS_ERROR 変数

構文

const qa_int **QAError::COMMON_OPEN_MAXTHREADS_ERROR**

備考

サーバの同時要求の最大数が不充分であるため、QAManager をオープンできません。

詳細については、「-gn サーバ・オプション」 『SQL Anywhere サーバ - データベース管理』を参照してください。

COMMON_SELECTOR_SYNTAX_ERROR 変数

構文

const qa_int **QAError::COMMON_SELECTOR_SYNTAX_ERROR**

備考

指定されたセレクタに構文エラーがあります。

COMMON_TERMINATE_ERROR 変数

構文

const qa_int **QAError::COMMON_TERMINATE_ERROR**

備考

終了エラーです。

COMMON_UNEXPECTED_EOM_ERROR 変数

構文

const qa_int **QAError::COMMON_UNEXPECTED_EOM_ERROR**

備考

予期しないところでメッセージの終わりに到達しました。

COMMON_UNREPRESENTABLE_TIMESTAMP 変数

構文

const qa_int **QAError::COMMON_UNREPRESENTABLE_TIMESTAMP**

備考

タイムスタンプが許容範囲外です。

QA_NO_ERROR 変数

構文

const qa_int **QAError::QA_NO_ERROR**

備考

エラーはありません。

QAManager クラス

構文

public **QAManager**

基本クラス

- 「QAManagerBase クラス」 460 ページ

備考

QAManager クラスは QAManagerBase から派生し、非トランザクション志向の QAnywhere メッセージング操作を管理します。

この動作の完全な説明については、QAManagerBase クラスを参照してください。

QAManager クラスは、AcknowledgementMode 列挙の定義のようにして、明示的に受信を確認するように設定することも、暗黙的に受信を確認するように設定することもできます。トランザクションの一部としてメッセージの受信を確認するには、QATransactionalManager を使用します。

QAManager と QATransactionalManager オブジェクトを作成するには、QAManagerFactory を使用します。

参照

QAManagerFactory クラス

QAManagerBase クラス

QATransactionalManager クラス

AcknowledgementMode クラス

メンバ

QAManager クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「acknowledge 関数」 456 ページ
- 「acknowledgeAll 関数」 457 ページ
- 「acknowledgeUntil 関数」 458 ページ
- 「beginEnumStorePropertyNames 関数」 462 ページ
- 「browseClose 関数」 462 ページ
- 「browseMessages 関数」 462 ページ
- 「browseMessagesByID 関数」 463 ページ
- 「browseMessagesByQueue 関数」 464 ページ
- 「browseMessagesBySelector 関数」 464 ページ
- 「browseNextMessage 関数」 465 ページ
- 「cancelMessage 関数」 466 ページ
- 「close 関数」 466 ページ
- 「createBinaryMessage 関数」 467 ページ
- 「createTextMessage 関数」 467 ページ
- 「deleteMessage 関数」 468 ページ

acknowledge 関数

構文

```
qa_bool QAManager::acknowledge(
  QAMessage * msg
)
```

パラメータ

- **msg**　受信確認するメッセージ

備考

クライアント・アプリケーションが QAnywhere メッセージを正常に受信したことを確認します。

注意 : QAMessage が受信確認されると、その MessagePropertiesSTATUS プロパティが StatusCodes.RECEIVED に変わります。QAMessage は、ステータスが StatusCodes.RECEIVED に変わると、デフォルトの削除ルールを使用して削除できるようになります。

削除ルールの詳細については、「メッセージの削除ルール」 256 ページを参照してください。

参照

acknowledgeAll 関数

acknowledgeUntil 関数

戻り値

処理が正常終了した場合のみ True

acknowledgeAll 関数

構文

qa_bool **QAManager::acknowledgeAll()**

備考

クライアント・アプリケーションが、受信確認されていない QAnywhere メッセージを、すべて正常に受信したことを確認します。

注意 : QAMessage が受信確認されると、その MessagePropertiesSTATUS プロパティが StatusCodes.RECEIVED に変わります。QAMessage は、ステータスが StatusCodes.RECEIVED に変わると、デフォルトの削除ルールを使用して削除できるようになります。

削除ルールの詳細については、「メッセージの削除ルール」 256 ページを参照してください。

参照

acknowledge 関数

acknowledgeUntil 関数

戻り値

処理が正常終了した場合のみ True

acknowledgeUntil 関数

構文

```
qa_bool QAManager::acknowledgeUntil(
  QAMessage * msg
)
```

パラメータ

- **msg**　受信確認するメッセージのうち最新のもの。それ以前の受信確認されていないメッセージも、すべて受信確認されます。

備考

指定された QAMessage インスタンスと、指定されたメッセージよりも前に受信されて受信確認されていないメッセージについて、すべて受信確認します。

注意： QAMessage が受信確認されると、その MessagePropertiesSTATUS プロパティが StatusCodes.RECEIVED に変わります。QAMessage は、ステータスが StatusCodes.RECEIVED に変わると、デフォルトの削除ルールを使用して削除できるようになります。

削除ルールの詳細については、「メッセージの削除ルール」 256 ページを参照してください。

参照

acknowledge 関数

acknowledgeAll 関数

戻り値

処理が正常終了した場合のみ True

open 関数

構文

```
qa_bool QAManager::open(
  qa_short mode
)
```

パラメータ

- **mode**　受信確認モード

備考

指定された AcknowledgementMode 値を使用して QAManager をオープンします。

open メソッドは、QAManager を作成した後、最初に呼び出す必要があるメソッドです。

参照

AcknowledgementMode クラス

戻り値

処理が正常終了した場合のみ True

recover 関数

構文

qa_bool **QAManager::recover()**

備考

受信確認されていないメッセージを、すべて強制的に未受信に戻します。

つまり、該当するメッセージは、QAManagergetMessage() を使用して再度受信する必要があります。

戻り値

処理が正常終了した場合のみ True

QAManagerBase クラス

構文

public **QAManagerBase**

派生クラス

- 「QAManager クラス」 455 ページ
- 「QATransactionalManager クラス」 521 ページ

備考

このクラスは、QATransactionalManager と QAManager の基本クラスです。前者の派生クラスはトランザクション志向のメッセージングを、後者の派生クラスは非トランザクション志向のメッセージングを管理します。

QAManagerBase インスタンスがメッセージを受信できるようにするには、QAManagerBasestart() メソッドを使用します。QAManagerBase インスタンスは、アプリケーションのスレッドごとに 1 つだけにします。

このクラスのインスタンスを使用して、QAnywhere メッセージの作成と管理を行うことができます。適切な QAMessage インスタンスを作成するには、QAManagerBasecreateBinaryMessage() メソッドと QAManagerBasecreateTextMessage() メソッドを使用します。QAMessage インスタンスには、メッセージの内容とプロパティを設定するための、さまざまなメソッドがあります。QAnywhere メッセージを送信するには、QAManagerputMessage() を使用して、アドレス指定されたメッセージをローカルのメッセージ・ストア・キューに登録します。メッセージは、転送ポリシーに基づいて QAnywhere Agent によって転送されるか、QAManagerBasetriggerSendReceive() が呼び出されたときに転送されます。

qaagent 転送ポリシーの詳細については、「クライアントにメッセージを転送するタイミングの決定」 40 ページを参照してください。

QAManagerBaseclose() を使用して QAManagerBase インスタンスがクローズされると、メッセージがメモリから解放されます。

QAException が発生した場合にエラー情報を返すには、QAManagerBasegetLastError と QAManagerBasegetLastErrorMessage を使用します。QAManagerBase にも、メッセージ・ストア・プロパティを設定および取得するためのメソッドがあります。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページと MessageStoreProperties クラスを参照してください。

参照

QATransactionalManager クラス

QAManager クラス

メンバ

QAManagerBase クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「stop 関数」 488 ページ
- 「triggerSendReceive 関数」 488 ページ

beginEnumStorePropertyNames 関数

構文

qa_store_property_enum_handle **QAManagerBase::beginEnumStorePropertyNames()**

備考

メッセージ・ストア・プロパティ名の列挙を開始します。

このメソッドから返されるハンドルは、QAManagerBasenextStorePropertyName に渡されます。このメソッドと QAManagerBasenextStorePropertyName を使用することで、このメソッドが呼び出されたときのメッセージ・ストア・プロパティ名を列挙できます。
QAManagerBasebeginEnumStorePropertyNames の呼び出しと QAManagerBaseendEnumStorePropertyNames の呼び出しの間は、メッセージ・ストア・プロパティを設定できません。

参照

nextStorePropertyName 関数

beginEnumStorePropertyNames 関数

endEnumStorePropertyNames 関数

戻り値

QAManagerBasenextStorePropertyName に渡されるハンドル

browseClose 関数

構文

void **QAManagerBase::browseClose(**
  qa_browse_handle *handle*
)

パラメータ

- **handle**　参照を開始するいずれかの操作から返されたハンドル

備考

参照操作に関連付けられているリソースを解放します。

browseMessages 関数

構文

qa_browse_handle **QAManagerBase::browseMessages()**

備考

メッセージ・ストアのキューに登録されているメッセージの参照を開始します。

このメソッドから返されるハンドルは、QAManagerBasebrowseNextMessage に渡されます。このメソッドと QAManagerBasebrowseNextMessage を使用することで、このメソッドが呼び出されたときにメッセージ・ストア内に置かれていたメッセージを列挙できます。

メッセージは単に参照されるだけなので、受信確認はできません。メッセージを受信して受信確認できるようにする場合は、QAManagerBasegetMessage を使用します。

参照

browseNextMessage 関数

browseMessagesByQueue 関数

browseMessagesByID 関数

browseClose 関数

戻り値

QAManagerBasebrowseNextMessage に渡されるハンドル

browseMessagesByID 関数

構文

```
qa_browse_handle QAManagerBase::browseMessagesByID(
  qa_const_string msgid
)
```

パラメータ

- **msgid**　メッセージ ID

備考

メッセージ・ストアのキューに登録されているメッセージのうち、指定されたメッセージ ID を持つメッセージの参照を開始します。

このメソッドから返されるハンドルは、QAManagerBasebrowseNextMessage に渡されます。このメソッドと QAManagerBasebrowseNextMessage を使用することで、このメソッドが呼び出されたときにメッセージ・ストア内に置かれていたメッセージを列挙できます。

メッセージは単に参照されるだけなので、受信確認はできません。メッセージを受信して受信確認できるようにする場合は、QAManagerBasegetMessage を使用します。

参照

browseNextMessage 関数

browseMessagesByQueue 関数

browseMessages 関数

browseClose 関数

戻り値

browseNextMessage に渡されるハンドル

browseMessagesByQueue 関数

構文

```
qa_browse_handle QAManagerBase::browseMessagesByQueue(
  qa_const_string address
)
```

パラメータ

- **address**　参照対象のキュー

備考

メッセージ・ストアのキューに登録されているメッセージのうち、指定されたキューのメッセージの参照を開始します。

このメソッドから返されるハンドルは、QAManagerBasebrowseNextMessage に渡されます。このメソッドと QAManagerBasebrowseNextMessage を使用することで、このメソッドが呼び出されたときにメッセージ・ストア内に置かれていたメッセージを列挙できます。

メッセージは単に参照されるだけなので、受信確認はできません。メッセージを受信して受信確認できるようにする場合は、QAManagerBasegetMessage を使用します。

参照

browseNextMessage 関数

browseMessagesByID 関数

browseMessages 関数

browseClose 関数

戻り値

browseNextMessage に渡されるハンドル

browseMessagesBySelector 関数

構文

```
qa_browse_handle QAManagerBase::browseMessagesBySelector(
  qa_const_string selector
)
```

パラメータ

- **selector**　セレクタ

備考

メッセージ・ストアのキューに登録されているメッセージのうち、指定されたセレクタを満たすメッセージの参照を開始します。

このメソッドから返されるハンドルは、QAManagerBasebrowseNextMessage に渡されます。このメソッドと QAManagerBasebrowseNextMessage を使用することで、このメソッドが呼び出されたときにメッセージ・ストア内に置かれていたメッセージを列挙できます。

メッセージは単に参照されるだけなので、受信確認はできません。

メッセージを受信して受信確認できるようにする場合は、QAManagerBasegetMessage を使用します。

参照

browseNextMessage 関数

browseMessagesByID 関数

browseMessagesByQueue 関数

browseMessages 関数

browseClose 関数

戻り値

browseNextMessage に渡されるハンドル

browseNextMessage 関数

構文

```
QAMessage * QAManagerBase::browseNextMessage(
  qa_browse_handle handle
)
```

パラメータ

- **handle**　参照を開始するいずれかの操作から返されたハンドル

備考

指定された参照操作の次のメッセージを返します。メッセージがない場合は null を返します。

参照対象メッセージのハンドルを取得するには、QAManagerBasebrowseMessages を使用します。または、他の QAManagerBase メソッドを使用して、キューまたはメッセージ ID でメッセージを参照することもできます。

参照

browseMessages 関数

browseMessagesByQueue 関数

browseMessagesByID 関数

browseClose 関数

戻り値

次のメッセージ。メッセージがない場合は qa_null。

cancelMessage 関数

構文

```
qa_bool QAManagerBase::cancelMessage(
  qa_const_string msgid
)
```

パラメータ

- **msgid**　キャンセルするメッセージの ID

備考

指定されたメッセージ ID を持つメッセージをキャンセルします。

cancelMessage メソッドは、メッセージが転送される前にメッセージをキャンセル済みの状態にします。QAnywhere Agent のデフォルトの削除ルールにより、キャンセル済みメッセージは最終的にメッセージ・ストアから削除されます。

メッセージがすでに最終ステータスになっている場合や、中央のメッセージング・サーバに転送済みである場合は、cancelMessage メソッドは失敗します。

削除ルールの詳細については、「メッセージの削除ルール」 256 ページを参照してください。

戻り値

処理が正常終了した場合のみ True

close 関数

構文

```
qa_bool QAManagerBase::close()
```

備考

QAnywhere メッセージ・システムへの接続をクローズして、QAManagerBase で使用していたリソースをすべて解放します。

この後で呼び出される close() は無視されます。いったんクローズした QAManagerBase のインスタンスをもう一度オープンすることはできません。この場合は、新規の QAManagerBase インスタンスを作成してから、オープンします。

参照

open 関数

open 関数

戻り値

処理が正常終了した場合のみ True

createBinaryMessage 関数

構文

QABinaryMessage * **QAManagerBase::createBinaryMessage()**

備考

QABinaryMessage インスタンスを作成します。

QABinaryMessage インスタンスは、未解釈のバイト・ストリームのメッセージ本文が含まれるメッセージの送信に使用します。

参照

QABinaryMessage クラス

戻り値

新しい QABinaryMessage クラス インスタンス

createTextMessage 関数

構文

QATextMessage * **QAManagerBase::createTextMessage()**

備考

QATextMessage インスタンスを作成します。

QATextMessage のオブジェクトは、文字列のメッセージ本文が含まれるメッセージを送信する場合に使用します。

参照

QATextMessage クラス

戻り値

新しい QATextMessage インスタンス

deleteMessage 関数

構文

```
void QAManagerBase::deleteMessage(
  QAMessage * msg
)
```

パラメータ

- **msg**　削除するメッセージ

備考

QAMessage オブジェクトを削除します。

デフォルトでは、QAManagerBasecreateTextMessage または QAManagerBasecreateBinaryMessage で作成されたメッセージは、QAManagerBase をクローズすると自動的に削除されます。このメソッドを使用すると、メッセージを削除するタイミングをより柔軟に管理できます。

endEnumStorePropertyNames 関数

構文

```
void QAManagerBase::endEnumStorePropertyNames(
  qa_store_property_enum_handle h
)
```

パラメータ

- **h**　beginEnumStorePropertyNames から返されるハンドル

備考

メッセージ・ストア・プロパティ名の列挙に関連付けられているリソースを解放します。

参照

beginEnumStorePropertyNames 関数

getAllQueueDepth 関数

構文

```
qa_int QAManagerBase::getAllQueueDepth(
  qa_short filter
)
```

パラメータ

- **filter**　着信メッセージ、送信メッセージ、またはすべてのメッセージを示すフィルタ

備考

指定されたフィルタに基づいて、すべてのキューの深さの合計を返します。

キューの深さは、受信されていないメッセージの数です (たとえば、QAManagerBasegetMessage を使用)。

参照

QueueDepthFilter クラス.

戻り値

メッセージの数。エラーが発生した場合は -1。

getBooleanStoreProperty 関数

構文

```
qa_bool QAManagerBase::getBooleanStoreProperty(
  qa_const_string name,
  qa_bool * value
)
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　boolean 値の格納先

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの boolean 値を取得します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

戻り値

処理が正常終了した場合のみ True

getByteStoreProperty 関数

構文

```
qa_bool QAManagerBase::getByteStoreProperty(
  qa_const_string name,
  qa_byte * value
)
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　byte 値の格納先

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの byte 値を取得します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

戻り値

処理が正常終了した場合のみ True

getDoubleStoreProperty 関数

構文

```
qa_bool QAManagerBase::getDoubleStoreProperty(
  qa_const_string name,
  qa_double * value
)
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　double 値の格納先

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの double 値を取得します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

戻り値

処理が正常終了した場合のみ True

getFloatStoreProperty 関数

構文

```
qa_bool QAManagerBase::getFloatStoreProperty(
  qa_const_string name,
  qa_float * value
)
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　float 値の格納先

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの float 値を取得します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

戻り値

処理が正常終了した場合のみ True

getIntStoreProperty 関数

構文

```
qa_bool QAManagerBase::getIntStoreProperty(
  qa_const_string name,
  qa_int * value
)
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名

- **value**　int 値の格納先

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの int 値を取得します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

戻り値

処理が正常終了した場合のみ True

getLastError 関数

構文

qa_int **QAManagerBase::getLastError()**

備考

最後に実行された QAManagerBase メソッドに関連付けられているエラー・コードです。

0 はエラーがないことを示します。

値のリストについては、QAError クラスを参照してください。

参照

getLastErrorMsg 関数

QAError クラス

戻り値

エラー・コード

getLastErrorMsg 関数

構文

qa_string **QAManagerBase::getLastErrorMsg()**

備考

最後に実行された QAManagerBase メソッドに関連付けられているエラー・テキストです。

QAManagerBasegetLastError が 0 を返した場合、このメソッドは null を返します。このプロパティは、QAError を取得した後で取り出せます。

参照

getLastError 関数

QAError クラス

戻り値

エラー・メッセージ

getLongStoreProperty 関数

構文

```
qa_bool QAManagerBase::getLongStoreProperty(
  qa_const_string name,
  qa_long * value
)
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　long 値の格納先

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの long 値を取得します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

戻り値

処理が正常終了した場合のみ True

getMessage 関数

構文

```
QAMessage * QAManagerBase::getMessage(
  qa_const_string address
)
```


パラメータ

- **address**　送信先

備考

指定されたアドレスに送信され、次に取得可能な QAMessage クラスを返します。

address パラメータは、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージが存在しない場合、新しいメッセージが着信するまでブロックされます。メッセージを同期に受信する場合は、このメソッドを使用します。

メッセージを非同期に受信する (メッセージ・イベント・ハンドラを使用する) 方法の詳細については、「非同期的なメッセージ受信」 83 ページを参照してください。

戻り値

該当する次の QAMessage。メッセージが存在しない場合は null。

getMessageBySelector 関数

構文

```
QAMessage * QAManagerBase::getMessageBySelector(
  qa_const_string address,
  qa_const_string selector
)
```

パラメータ

- **address**　送信先
- **selector**　セレクタ

備考

指定されたアドレスに送信され、かつ指定されたセレクタを満たす、次に取得可能な QAMessage を返します。

address パラメータは、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージが存在しない場合、新しいメッセージが着信するまでブロックされます。メッセージを同期に受信する場合は、このメソッドを使用します。

メッセージを非同期に受信する (メッセージ・イベント・ハンドラを使用する) 方法の詳細については、「非同期的なメッセージ受信」 83 ページを参照してください。

戻り値

該当する次の QAMessage。メッセージが存在しない場合は null。

getMessageBySelectorNoWait 関数

構文

```
QAMessage * QAManagerBase::getMessageBySelectorNoWait(
  qa_const_string address,
  qa_const_string selector
)
```

パラメータ

- **address**　送信先
- **selector**　セレクタ

備考

指定されたアドレスに送信され、かつ指定されたセレクタを満たす、次に取得可能な QAMessage を返します。

address パラメータは、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージがない場合は、このメソッドがすぐに返されます。メッセージを同期に受信する場合は、このメソッドを使用します。

メッセージを非同期に受信する (メッセージ・イベント・ハンドラを使用する) 方法の詳細については、「非同期的なメッセージ受信」 83 ページを参照してください。

戻り値

該当する次のメッセージ。メッセージが存在しない場合は qa_null。

getMessageBySelectorTimeout 関数

構文

```
QAMessage * QAManagerBase::getMessageBySelectorTimeout(
  qa_const_string address,
  qa_const_string selector,
  qa_long timeout
)
```

パラメータ

- **address**　送信先
- **selector**　セレクタ
- **timeout**　最大待ち時間 (ミリ秒)

備考

指定されたアドレスに送信され、かつ指定されたセレクタを満たす、次に取得可能な QAMessage を返します。

address パラメータは、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージがない場合、このメソッドは指定されたタイムアウト時間だけ待機してから返されます。メッセージを同期に受信する場合は、このメソッドを使用します。

メッセージを非同期に受信する (メッセージ・イベント・ハンドラを使用する) 方法の詳細については、「非同期的なメッセージ受信」 83 ページを参照してください。

戻り値

該当する次の QAMessage。メッセージが存在しない場合は null。

getMessageNoWait 関数

構文

```
QAMessage * QAManagerBase::getMessageNoWait(
  qa_const_string address
)
```

パラメータ

- **address**　送信先

備考

指定されたアドレスに送信され、次に取得可能な QAMessage を返します。

address パラメータは、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージがない場合は、このメソッドがすぐに返されます。メッセージを同期に受信する場合は、このメソッドを使用します。

メッセージを非同期に受信する (メッセージ・イベント・ハンドラを使用する) 方法の詳細については、「非同期的なメッセージ受信」 83 ページを参照してください。

戻り値

該当する次のメッセージ。メッセージが存在しない場合は qa_null。

getMessageTimeout 関数

構文

```
QAMessage * QAManagerBase::getMessageTimeout(
  qa_const_string address,
  qa_long timeout
)
```

パラメータ

- **address**　送信先
- **timeout**　最大待ち時間 (ミリ秒)

備考

指定されたアドレスに送信され、次に取得可能な QAMessage を返します。

address パラメータは、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージがない場合、このメソッドは指定されたタイムアウト時間だけ待機してから返されます。メッセージを同期に受信する場合は、このメソッドを使用します。

メッセージを非同期に受信する (メッセージ・イベント・ハンドラを使用する) 方法の詳細については、「非同期的なメッセージ受信」 83 ページを参照してください。

戻り値

該当する次の QAMessage。メッセージが存在しない場合は null。

getMode 関数

構文

qa_short **QAManagerBase::getMode()**

備考

受信したメッセージの QAManager 受信確認モードを返します。

値のリストについては、AcknowledgementMode クラスを参照してください。

AcknowledgementModeEXPLICIT_ACKNOWLEDGEMENT と AcknowledgementModeIMPLICIT_ACKNOWLEDGEMENT は QAManager インスタンスで使用されます。AcknowledgementModeTRANSACTIONAL は QATransactionalManager インスタンス用のモードです。

参照

AcknowledgementMode クラス

戻り値

受信確認モード

getQueueDepth 関数

構文

```
qa_int QAManagerBase::getQueueDepth(
  qa_const_string address,
  qa_short filter
)
```

パラメータ

- **address**　キュー名

- **filter**　着信メッセージ、送信メッセージ、またはすべてのメッセージを示すフィルタ

備考

指定されたフィルタに基づいて、キューの深さを返します。

キューの深さは、受信されていないメッセージの数です (たとえば、QAManagerBasegetMessage を使用)。

参照

QueueDepthFilter クラス

戻り値

キューに登録されているメッセージの数。エラーが発生した場合は -1。

getShortStoreProperty 関数

構文

```
qa_bool QAManagerBase::getShortStoreProperty(
  qa_const_string name,
  qa_short * value
)
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　short 値の格納先

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの short 値を取得します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

戻り値

処理が正常終了した場合のみ True

getStringStoreProperty 関数

構文

```
qa_int QAManagerBase::getStringStoreProperty(
  qa_const_string name,
  qa_string address,
  qa_int maxlen
)
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **address**　qa_string 値の格納先
- **maxlen**　取得した値からコピーする qa_char の最大数 (null 終端子も含む)

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの文字列値を取得します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

戻り値

実際にコピーされた null 以外の qa_char の数。エラーが発生した場合は -1。

nextStorePropertyName 関数

構文

```
qa_int QAManagerBase::nextStorePropertyName(
  qa_store_property_enum_handle h,
  qa_string buffer,
  qa_int bufferLen
)
```

パラメータ

- **h**　beginEnumStorePropertyNames から返されるハンドル
- **buffer**　プロパティ名が書き込まれるバッファ
- **bufferLen**　プロパティ名を格納するバッファの長さ。この長さには、null 終端子のスペースも含める必要があります。

備考

指定された列挙のメッセージ・ストア・プロパティ名を返します。

プロパティ名がない場合は -1 を返します。

参照

beginEnumStorePropertyNames 関数

戻り値

プロパティ名の長さ。プロパティ名がない場合は -1。プロパティ名。

putMessage 関数

構文

```
qa_bool QAManagerBase::putMessage(
  qa_const_string address,
  QAMessage * msg
)
```

パラメータ

- **address**　送信先
- **msg**　メッセージ

備考

指定された送信先に送られるようにメッセージをキューに登録します。

戻り値

処理が正常終了した場合のみ True

putMessageTimeToLive 関数

構文

```
qa_bool QAManagerBase::putMessageTimeToLive(
  qa_const_string address,
  QAMessage * msg,
  qa_long ttl
)
```

パラメータ

- **address**　送信先
- **msg**　メッセージ
- **ttl**　存続時間 (ミリ秒)

備考

指定された送信先に送られるようにメッセージをキューに登録します。指定された存続時間 (ミリ秒) が設定されます。

戻り値

処理が正常終了した場合のみ True

setBooleanStoreProperty 関数

構文

```
qa_bool QAManagerBase::setBooleanStoreProperty(
  qa_const_string name,
  qa_bool value
)
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　プロパティの qa_bool 値

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを boolean 値に設定します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

戻り値

処理が正常終了した場合のみ True

setByteStoreProperty 関数

構文

```
qa_bool QAManagerBase::setByteStoreProperty(
  qa_const_string name,
  qa_byte value
)
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名

- **value**　プロパティの qa_byte 値

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを byte 値に設定します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

戻り値

処理が正常終了した場合のみ True

setDoubleStoreProperty 関数

構文

```
qa_bool QAManagerBase::setDoubleStoreProperty(
  qa_const_string name,
  qa_double value
)
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　プロパティの qa_double 値

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを double 値に設定します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

戻り値

処理が正常終了した場合のみ True

setFloatStoreProperty 関数

構文

```
qa_bool QAManagerBase::setFloatStoreProperty(
  qa_const_string name,
  qa_float value
)
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　プロパティの qa_float 値

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを float 値に設定します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

戻り値

処理が正常終了した場合のみ True

setIntStoreProperty 関数

構文

```
qa_bool QAManagerBase::setIntStoreProperty(
  qa_const_string name,
  qa_int value
)
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　プロパティの qa_int 値

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを int 値に設定します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

戻り値

処理が正常終了した場合のみ True

setLongStoreProperty 関数

構文

```
qa_bool QAManagerBase::setLongStoreProperty(
  qa_const_string name,
  qa_long value
)
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　プロパティの qa_long 値

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを long 値に設定します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

戻り値

処理が正常終了した場合のみ True

setMessageListener 関数

構文

```
void QAManagerBase::setMessageListener(
  qa_const_string address,
  QAMessageListener * listener
)
```

パラメータ

- **address**　リスナに適用される送信先アドレス

- **listener**　送信先アドレスに関連付けられるメッセージ・リスナ

備考

QAnywhere メッセージを非同期に受信するようにメッセージ・リスナ・クラスを設定します。

リスナは、QAMessageListener インタフェースで定義される唯一のメソッドである QAMessageListeneronMessage を実装するクラスのインスタンスです。QAMessageListeneronMessage は、QAMessage パラメータを 1 つだけ指定できます。

setMessageListener の address パラメータは、メッセージの受信で使用されるローカル・キュー名を指定します。指定されたキューには、1 つのリスナだけを割り当てることができます。

Push 通知とネットワーク・ステータス変更も含め、QAnywhere システム・メッセージを受信したい場合は、キュー名に "system" を指定します。メッセージを非同期に受信する場合は、このメソッドを使用します。

詳細については、「非同期的なメッセージ受信」 83 ページと「システム・キュー」 56 ページを参照してください。

setMessageListenerBySelector 関数

構文

```
void QAManagerBase::setMessageListenerBySelector(
  qa_const_string address,
  qa_const_string selector,
  QAMessageListener * listener
)
```

パラメータ

- **address**　リスナに適用される送信先アドレス
- **selector**　受信されるメッセージをフィルタリングするために使用するセレクタ
- **listener**　送信先アドレスに関連付けられるメッセージ・リスナ

備考

メッセージ・セレクタを指定し、QAnywhere メッセージを非同期に受信するように、メッセージ・リスナ・クラスを設定します。

リスナは、QAMessageListener インタフェースで定義される唯一のメソッドである QAMessageListeneronMessage を実装するクラスのインスタンスです。QAMessageListeneronMessage は、QAMessage パラメータを 1 つだけ指定できます。

setMessageListener の address パラメータは、メッセージの受信で使用されるローカル・キュー名を指定します。指定されたキューには、1 つのリスナだけを割り当てることができます。selector パラメータは、指定されたアドレスで受信されるメッセージをフィルタリングするために使用するセレクタを指定します。

Push 通知とネットワーク・ステータス変更も含め、QAnywhere システム・メッセージを受信したい場合は、キュー名に "system" を指定します。メッセージを非同期に受信する場合は、このメソッドを使用します。

詳細については、「非同期的なメッセージ受信」 83 ページと「システム・キュー」 56 ページを参照してください。

setProperty 関数

構文

```
qa_bool QAManagerBase::setProperty(
  qa_const_string name,
  qa_const_string value
)
```

パラメータ

- **name** QAnywhere Manager の事前定義済みまたはカスタムの設定プロパティ名
- **value** QAnywhere Manager の設定プロパティの値

備考

QAnywhere Manager の設定プロパティをプログラムで設定できるようにします。

プロパティ名と値を指定してこのメソッドを使用することで、QAnywhere Manager のデフォルトの設定プロパティを無効にできます。

QAnywhere Manager の設定プロパティのリストについては、「QAnywhere Manager の設定プロパティ」 69 ページを参照してください。

QAnywhere Manager の設定プロパティは、プロパティ・ファイルと QAManagerFactorycreateQAManager メソッドを使用して設定することもできます。

詳細については、「QAnywhere Manager の設定プロパティをファイルに設定する」 70 ページを参照してください。

注意 : QAManageropen() または QATransactionalManageropen() を呼び出す前に、必要なプロパティを設定してください。

戻り値

処理が正常終了した場合のみ True

setShortStoreProperty 関数

構文

```
qa_bool QAManagerBase::setShortStoreProperty(
  qa_const_string name,
  qa_short value
)
```


パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　プロパティの qa_short 値

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを short 値に設定します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

戻り値

処理が正常終了した場合のみ True

setStringStoreProperty 関数

構文

```
qa_bool QAManagerBase::setStringStoreProperty(
  qa_const_string name,
  qa_const_string value
)
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　プロパティの qa_string 値

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを文字列値に設定します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties クラスを参照してください。

詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

戻り値

処理が正常終了した場合のみ True

start 関数

構文

qa_bool **QAManagerBase::start()**

備考

着信メッセージをメッセージ・リスナで受信するための QAManagerBase を起動します。

メッセージ・リスナ・セットがない場合、たとえばメッセージが getMessage メソッドで受信される場合などは、QAManagerBase を起動する必要がありません。getMessage メソッドとメッセージ・リスナは、メッセージを受信するために併用しないでください。非同期 (メッセージ・リスナ) モデルまたは同期 (getMessage) モデルのどちらかを使用してください。

この start メソッドを繰り返し呼び出そうとしても、間に QAManagerBasestop() 呼び出しを挟まないかぎり 2 回目以降の呼び出しは無視されます。

参照

stop 関数

戻り値

処理が正常終了した場合のみ True

stop 関数

構文

qa_bool **QAManagerBase::stop()**

備考

QAManagerBase による着信メッセージの受信を停止します。

メッセージは失われません。メッセージは、Manager が再起動されるまで受信されません。この stop メソッドを繰り返し呼び出そうとしても、間に QAManagerBasestart() を挟まないかぎり 2 回目以降の呼び出しは無視されます。

参照

start 関数

戻り値

処理が正常終了した場合のみ True

triggerSendReceive 関数

構文

qa_bool **QAManagerBase::triggerSendReceive()**

備考

QAnywhere メッセージ・サーバとの同期処理を発生させて、他のクライアント宛てのメッセージをアップロードし、ローカル・クライアント宛てのメッセージをダウンロードします。

triggerSendReceive を呼び出すと、QAnywhere Agent と中央のメッセージング・サーバ間でメッセージがただちに同期されます。手動の triggerSendReceive 呼び出しでは、QAnywhere Agent の転送ポリシーとは無関係に、メッセージ転送がただちに行われます。QAnywhere Agent の転送ポリシーは、メッセージ転送の方法を決定します。たとえば、クライアントが Push 通知を受信した場合や、QAManagerBaseputMessage を呼び出してメッセージを送信した場合に、一定の間隔でメッセージ転送が自動的に実行されます。

詳細については、「クライアントにメッセージを転送するタイミングの決定」 40 ページを参照してください。

参照

putMessage 関数

戻り値

処理が正常終了した場合のみ True

QAManagerFactory クラス

構文

public **QAManagerFactory**

備考

このクラスは、QATransactionalManager オブジェクトまたは QAManager オブジェクトを作成するためのファクトリ・クラスです。

QAManagerFactory のインスタンスは 1 つだけ持つことができます。

参照

QAManager クラス

QATransactionalManager クラス

メンバ

QAManagerFactory クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

createQAManager 関数

構文

```
QAManager * QAManagerFactory::createQAManager(
  qa_const_string iniFile
)
```

パラメータ

- **iniFile**　プロパティ・ファイルのパス

備考

指定されたプロパティを持つ新しい QAManager インスタンスを返します。

プロパティ・ファイル・パラメータが null の場合は、デフォルトのプロパティを使用して QAManager が作成されます。QAManager の作成後に QAManagersetProperty() メソッドを使用して、QAnywhere Manager のプロパティをプログラムで設定できます。

QAnywhere Manager の設定プロパティのリストについては、「QAnywhere Manager の設定プロパティ」 69 ページを参照してください。

参照

QAManager クラス

戻り値

QAManager インスタンス

createQATransactionalManager 関数

構文

```
QATransactionalManager * QAManagerFactory::createQATransactionalManager(
  qa_const_string iniFile
)
```

パラメータ

- **iniFile**　プロパティ・ファイルのパス

備考

指定されたプロパティを持つ新しい QATransactionalManager インスタンスを返します。

プロパティ・ファイル・パラメータが null の場合は、デフォルトのプロパティを使用して QATransactionalManager が作成されます。QATransactionalManager の作成後に QATransactionalManagersetProperty() メソッドを使用して、QAnywhere Manager のプロパティをプログラムで設定できます。

QAnywhere Manager の設定プロパティのリストについては、「QAnywhere Manager の設定プロパティ」 69 ページを参照してください。

参照

QATransactionalManager クラス

戻り値

QATransactionalManager インスタンス

deleteQAManager 関数

構文

```
void QAManagerFactory::deleteQAManager(
  QAManager * mgr
)
```

パラメータ

- **mgr**　破棄する QAManager インスタンス

備考

QAManager を破棄し、そのリソースを解放します。

作成済みの QAManager は QAnywhereFactory_term() が呼び出された時点ですべて破棄されるので、このメソッドを呼び出す必要はありません。ただし、すぐにリソースを解放する必要がある場合には、このメソッドを使用すると便利です。

詳細については、「QAnywhere の停止」 96 ページを参照してください。

deleteQATransactionalManager 関数

構文

void **QAManagerFactory::deleteQATransactionalManager(**
  QATransactionalManager * *mgr*
)

パラメータ

- **mgr**　破棄する QATransactionalManager インスタンス

備考

QATransactionalManager インスタンスを破棄し、そのリソースを解放します。

作成済みの QATransactionalManager インスタンスは QAnywhereFactory_term() が呼び出された時点ですべて破棄されるので、このメソッドを呼び出す必要はありません。ただし、すぐにリソースを解放する必要がある場合には、このメソッドを使用すると便利です。

詳細については、「QAnywhere の停止」 96 ページを参照してください。

getLastError 関数

構文

qa_int **QAManagerFactory::getLastError()**

備考

最後に実行された QAManagerFactory メソッドに関連付けられているエラー・コード。

0 はエラーがないことを示します。

値のリストについては、QAError クラスを参照してください。

参照

getLastErrorMsg 関数

戻り値

エラー・コード

getLastErrorMsg 関数

構文

qa_string **QAManagerFactory::getLastErrorMsg()**

備考

最後に実行された QAManagerFactory メソッドに関連付けられているエラー・テキスト。

QAManagerFactorygetLastError が 0 を返した場合、このメソッドは null を返します。

このプロパティは、QAError の受信後に取得できます。

参照

getLastError 関数

QAError クラス

戻り値

エラー・メッセージ

QAMessage クラス

構文

public **QAMessage**

派生クラス

- 「QABinaryMessage クラス」 435 ページ
- 「QATextMessage クラス」 517 ページ

備考

QAMessage には、メッセージ・プロパティとヘッダ・ファイルを設定するためのインタフェースがあります。

派生クラスの QABinaryMessage と QATextMessage には、メッセージ本文の読み込み／書き込みを行うための特別なメソッドがあります。事前定義済みまたはカスタムのメッセージ・プロパティを設定するには、QAMessage のメソッドを使用します。

事前定義済みプロパティ名のリストについては、MessageProperties クラスを参照してください。

メッセージ・プロパティとヘッダ・フィールドの設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

メンバ

QAMessage クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「beginEnumPropertyNames 関数」 496 ページ
- 「castToBinaryMessage 関数」 496 ページ
- 「castToTextMessage 関数」 496 ページ
- 「clearProperties 関数」 497 ページ
- 「DEFAULT_PRIORITY 変数」 495 ページ
- 「DEFAULT_TIME_TO_LIVE 変数」 495 ページ
- 「endEnumPropertyNames 関数」 497 ページ
- 「getAddress 関数」 497 ページ
- 「getBooleanProperty 関数」 498 ページ
- 「getByteProperty 関数」 498 ページ
- 「getDoubleProperty 関数」 499 ページ
- 「getExpiration 関数」 499 ページ
- 「getFloatProperty 関数」 500 ページ
- 「getInReplyToID 関数」 501 ページ
- 「getIntProperty 関数」 501 ページ
- 「getLongProperty 関数」 501 ページ
- 「getMessageID 関数」 502 ページ
- 「getPriority 関数」 503 ページ
- 「getPropertyType 関数」 503 ページ
- 「getRedelivered 関数」 503 ページ
- 「getReplyToAddress 関数」 504 ページ
- 「getShortProperty 関数」 504 ページ

DEFAULT_PRIORITY 変数

構文

const qa_int **QAMessage::DEFAULT_PRIORITY**

備考

デフォルトのメッセージ優先度です。

この値は 4 です。優先度の値が 0 ー 4 は通常のメッセージ、値 5 ー 9 は緊急度の高いメッセージです。

DEFAULT_TIME_TO_LIVE 変数

構文

const qa_long **QAMessage::DEFAULT_TIME_TO_LIVE**

備考

デフォルトのメッセージの存続時間です。

この値は 0 で、メッセージの有効期限がないことを示します。

beginEnumPropertyNames 関数

構文

qa_property_enum_handle **QAMessage::beginEnumPropertyNames()**

備考

メッセージ・プロパティ名の列挙を開始します。

このメソッドから返されるハンドルは、nextPropertyName に渡されます。このメソッドと nextPropertyName を使用して、このメソッドが呼び出されたときのメッセージ・ストア・プロパティ名を列挙できます。メッセージ・プロパティは、beginEnumPropertyNames と endEnumPropertyNames の間では設定できません。

戻り値

nextPropertyName に渡されるハンドル

castToBinaryMessage 関数

構文

QABinaryMessage * **QAMessage::castToBinaryMessage()**

備考

QAMessage を QABinaryMessage にキャストします。

変換演算子を使用して、この QAMessage を QABinaryMessage に変換することもできます。

変換演算子を使用して QAMessage を QABinaryMessage に変換するには、次のようにします。

```
QAMessage *msg;
QABinaryMessage *bmsg;
...
bmsg = (QABinaryMessage *)(*msg);
```

戻り値

QABinaryMessage へのポインタ。処理後のメッセージが QABinaryMessage のインスタンスではない場合は NULL。

castToTextMessage 関数

構文

QATextMessage * **QAMessage::castToTextMessage()**

備考

QAMessage を QATextMessage にキャストします。

変換演算子を使用して、QAMessage を QATextMessage に変換することもできます。

たとえば、変換演算子を使用して QAMessage を QATextMessage に変換するには、次のようにします。

```
QAMessage *msg;
QATextMessage *bmsg;
...
bmsg = (QATextMessage *)(*msg);
```

戻り値

QATextMessage へのポインタ。処理後のメッセージが QATextMessage のインスタンスではない場合は NULL。

clearProperties 関数

構文

void **QAMessage::clearProperties()**

備考

メッセージのプロパティをクリアします。

*注意：*メッセージのヘッダ・フィールドと本文はクリアされません。

endEnumPropertyNames 関数

構文

void **QAMessage::endEnumPropertyNames(**
  qa_property_enum_handle *h*
)

パラメータ

- **h**　beginEnumPropertyNames から返されるハンドル

備考

メッセージ・プロパティ名の列挙に関連付けられているリソースを解放します。

getAddress 関数

構文

qa_const_string **QAMessage::getAddress()**

備考

QAMessage インスタンスの送信先アドレスを取得します。

このフィールドは、メッセージの送信時には無視されます。send メソッドが完了すると、このフィールドには QAManagerBaseputMessage() で指定された送信先アドレスが入ります。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

戻り値

送信先アドレス

getBooleanProperty 関数

構文

```
qa_bool QAMessage::getBooleanProperty(
  qa_const_string name,
  qa_bool * value
)
```

パラメータ

- **name** 取得するプロパティの名前
- **value** qa_bool 値の格納先

備考

指定された名前を持つ qa_bool 型プロパティの値を取得します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

MessageProperties クラス

戻り値

処理が正常終了した場合のみ True

getByteProperty 関数

構文

```
qa_bool QAMessage::getByteProperty(
  qa_const_string name,
  qa_byte * value
)
```

パラメータ

- **name** 取得するプロパティの名前
- **value** qa_byte 値の格納先

備考

指定された名前を持つ qa_byte 型プロパティの値を取得します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

MessageProperties クラス

戻り値

処理が正常終了した場合のみ True

getDoubleProperty 関数

構文

```
qa_bool QAMessage::getDoubleProperty(
  qa_const_string name,
  qa_double * value
)
```

パラメータ

- **name**　取得するプロパティの名前
- **value**　qa_double 値の格納先

備考

指定された名前を持つ qa_double 型プロパティの値を取得します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

MessageProperties クラス

戻り値

処理が正常終了した場合のみ True

getExpiration 関数

構文

```
qa_long QAMessage::getExpiration()
```

備考

メッセージの有効期限を取得します。

メッセージの Expiration ヘッダ・フィールドは、メッセージの送信時には空のままです。send メソッドが完了すると、Espiration ヘッダにはメッセージの有効期限が入ります。

メッセージの有効期限は、QAManagerBaseputMessageTimeToLive の存続時間の引数を現在の時間に追加して設定されるため、このプロパティは読み込み専用です。

有効期限には、プラットフォームに合わせたフォーマットが使用されます。Windows/PocketPC プラットフォームの有効期限は SYSTEMTIME フォーマットです。これが FILETIME フォーマットに変換されて、qa_long 値にコピーされます。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

戻り値

有効期限

参照

getTimestamp 関数

getFloatProperty 関数

構文

```
qa_bool QAMessage::getFloatProperty(
  qa_const_string name,
  qa_float * value
)
```

パラメータ

- **name** 取得するプロパティの名前
- **value** qa_float 値の格納先

備考

指定された名前を持つ qa_float 型プロパティの値を取得します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

MessageProperties クラス

戻り値

処理が正常終了した場合のみ True

getInReplyToID 関数

構文

qa_const_string **QAMessage::getInReplyToID()**

備考

このメッセージが返信対象として指定されているメッセージの ID を取得します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

戻り値

In-Reply-To ID

getIntProperty 関数

構文

```
qa_bool QAMessage::getIntProperty(
  qa_const_string name,
  qa_int * value
)
```

パラメータ

- **name**　取得するプロパティの名前
- **value**　qa_int 値の格納先

備考

指定された名前を持つ qa_int 型プロパティの値を取得します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

MessageProperties クラス

戻り値

処理が正常終了した場合のみ True

getLongProperty 関数

構文

```
qa_bool QAMessage::getLongProperty(
  qa_const_string name,
```



```
  qa_long * value
)
```

パラメータ

- **name**　取得するプロパティの名前
- **value**　qa_long 値の格納先

備考

指定された名前を持つ qa_long 型プロパティの値を取得します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

MessageProperties クラス

戻り値

処理が正常終了した場合のみ True

getMessageID 関数

構文

qa_const_string **QAMessage::getMessageID()**

備考

メッセージ ID を取得します。

MessageID ヘッダ・フィールドには、QAnywhere クライアントによって送信された各メッセージを一意に識別する値が格納されています。

QAManagerBaseputMessage メソッドを使用してメッセージが送信されると、MessageID ヘッダが null になるため無視できます。send メソッドが戻ると、割り当てられた値がここに格納されます。

MessageID は、履歴レポジトリ内のメッセージを識別するためのユニークなキーとして使用できる qa_string 値です。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

戻り値

メッセージ ID

getPriority 関数

構文

qa_int **QAMessage::getPriority()**

備考

メッセージの優先度を取得します。

QAnywhere クライアント API では、10 レベルの優先度が定義されています。0 が最低の優先度、9 が 最高の優先度を表します。クライアントは、優先度 0 ー 4 を通常のメッセージ、優先度 5 ー 9 を緊急度の高いメッセージとみなす必要があります。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

戻り値

メッセージの優先度

getPropertyType 関数

構文

```
qa_short QAMessage::getPropertyType(
  qa_const_string name
)
```

パラメータ

- **name**　プロパティ名

備考

指定された名前を持つプロパティの型を返します。

プロパティの型には、PROPERTY_TYPE_BOOLEAN、PROPERTY_TYPE_BYTE、PROPERTY_TYPE_SHORT、PROPERTY_TYPE_INT、PROPERTY_TYPE_LONG、PROPERTY_TYPE_FLOAT、PROPERTY_TYPE_DOUBLE、PROPERTY_TYPE_STRING、PROPERTY_TYPE_UNKNOWN があります。

戻り値

プロパティの型

getRedelivered 関数

構文

qa_bool **QAMessage::getRedelivered()**

備考

受信されたが受信確認されていないメッセージであるかどうかを示します。

受信側の QAManager は、受信中のメッセージが以前に受信されたことを検知した場合に、Redelivered ヘッダを設定します。

たとえば、AcknowledgementModeEXPLICIT_ACKNOWLEDGEMENT でオープンされた QAManager クラスを使用してアプリケーションがメッセージを受信し、メッセージの受信確認を行わずに終了したとします。このアプリケーションは、次の起動時に同じメッセージを再受信し、Redelivered ヘッダを true に設定します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

戻り値

メッセージが再配信された場合のみ True

getReplyToAddress 関数

構文

qa_const_string **QAMessage::getReplyToAddress()**

備考

メッセージの返信先アドレスを取得します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

戻り値

返信先アドレス

getShortProperty 関数

構文

```
qa_bool QAMessage::getShortProperty(
  qa_const_string name,
  qa_short * value
)
```

パラメータ

- **name**　取得するプロパティの名前
- **value**　qa_short 値の格納先

備考

指定された名前を持つ qa_short 型プロパティの値を取得します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

MessageProperties クラス

戻り値

処理が正常終了した場合のみ True

getStringProperty 関数

構文

```
qa_int QAMessage::getStringProperty(
  qa_const_string name,
  qa_string dest,
  qa_int maxlen
)
```

パラメータ

- **name**　取得するプロパティの名前
- **dest**　qa_string 値の格納先
- **maxlen**　コピーする値の qa_char の最大数。この値には、null 終端子の qa_char も含まれます。

備考

指定された名前を持つ qa_string 型プロパティの値を取得します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

MessageProperties クラス

戻り値

実際にコピーされた null 以外の qa_char の数。エラーが発生した場合は -1。

getStringProperty 関数

構文

```
qa_int QAMessage::getStringProperty(
  qa_const_string name,
  qa_int offset,
  qa_string dest,
```



```
  qa_int maxlen
)
```

パラメータ

- **name**　取得するプロパティの名前
- **offset**　プロパティ値内でのオフセット (コピー開始位置)
- **dest**　qa_string 値の格納先
- **maxlen**　コピーする値の qa_char の最大数。この値には、null 終端子の qa_char も含まれます。

備考

指定された名前を持つ qa_string 型プロパティの値を、オフセット位置から取得します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

MessageProperties クラス

戻り値

実際にコピーされた null 以外の qa_char の数。エラーが発生した場合は -1。

getTimestamp 関数

構文

qa_long **QAMessage::getTimestamp()**

備考

メッセージのタイムスタンプを取得します。

メッセージの Timestamp ヘッダ・フィールドには、メッセージが作成された時刻が格納されます。これは、協定世界時 (UTC: Coordinated Universal Time) です。

この時刻は、メッセージが実際に送信された時刻ではないので注意してください。メッセージの実際の送信時刻は、トランザクションの進行状況やクライアント側のキューに登録されているその他のメッセージの影響で、作成時刻よりも遅れる可能性があります。タイムスタンプには、プラットフォームに合わせたフォーマットが使用されます。Windows/PocketPC プラットフォームのタイムスタンプは SYSTEMTIME フォーマットです。これが FILETIME フォーマットに変換されて、qa_long 値にコピーされます。

タイムスタンプ ts を SYSTEMTIME に変換してユーザに表示するには、次のコードを実行します。

```
SYSTEMTIME stime;
```

```
FILETIME   ftime;
ULARGE_INTEGER time;
time.QuadPart = ts;
memcpy(&ftime, &time, sizeof(FILETIME));
FileTimeToSystemTime(&ftime, &stime);
```

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

戻り値

メッセージのタイムスタンプ

getTimestampAsString 関数

構文

```
qa_int QAMessage::getTimestampAsString(
  qa_string buffer,
  qa_int bufferLen
)
```

パラメータ

- **buffer**　所定のフォーマットのタイムスタンプが格納されるバッファ
- **bufferLen**　バッファのサイズ

備考

メッセージのタイムスタンプを文字列として取得します。この文字列にはフォーマットがあります。

フォーマットは、"dow, MMM dd, yyyy hh:mm:ss.nnn GMT" です。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

戻り値

バッファに書き込まれた null 以外の qa_char の数

nextPropertyName 関数

構文

```
qa_int QAMessage::nextPropertyName(
  qa_property_enum_handle h,
  qa_string buffer,
  qa_int bufferLen
)
```


パラメータ

- **h**　beginEnumPropertyNames から返されるハンドル
- **buffer**　プロパティ名が書き込まれるバッファ
- **bufferLen**　プロパティ名を格納するバッファの長さ。この長さには、null 終端子のスペースも含める必要があります。

備考

指定された列挙のメッセージ・プロパティ名を返します。該当するプロパティ名がない場合は -1 を返します。

戻り値

プロパティ名の長さ。プロパティ名がない場合は -1。

propertyExists 関数

構文

```
qa_bool QAMessage::propertyExists(
  qa_const_string name
)
```

パラメータ

- **name**　プロパティ名

備考

プロパティ値が存在するかどうかを示します。

戻り値

プロパティが存在する場合のみ True

setAddress 関数

構文

```
void QAMessage::setAddress(
  qa_const_string destination
)
```

パラメータ

- **destination**　送信先アドレス

備考

メッセージの送信先アドレスを設定します。

このメソッドを使用すると、受信したメッセージにすでに設定されている値を変更できます。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

setBooleanProperty 関数

構文

```
void QAMessage::setBooleanProperty(
  qa_const_string name,
  qa_bool value
)
```

パラメータ

- **name**　設定するプロパティの名前
- **value**　プロパティの qa_bool 値

備考

指定された名前を持つ qa_bool 型プロパティを指定された値に設定します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

MessageProperties クラス

setByteProperty 関数

構文

```
void QAMessage::setByteProperty(
  qa_const_string name,
  qa_byte value
)
```

パラメータ

- **name**　設定するプロパティの名前
- **value**　プロパティの qa_byte 値

備考

指定された名前を持つ qa_byte 型プロパティを指定された値に設定します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

MessageProperties クラス

setDoubleProperty 関数

構文

```
void QAMessage::setDoubleProperty(
  qa_const_string name,
  qa_double value
)
```

パラメータ

- **name**　設定するプロパティの名前
- **value**　プロパティの qa_double 値

備考

指定された名前を持つ qa_double 型プロパティを指定された値に設定します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

MessageProperties クラス.

setFloatProperty 関数

構文

```
void QAMessage::setFloatProperty(
  qa_const_string name,
  qa_float value
)
```

パラメータ

- **name**　設定するプロパティの名前
- **value**　プロパティの qa_float 値

備考

指定された名前を持つ qa_float 型プロパティを指定された値に設定します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

MessageProperties クラス.

setInReplyToID 関数

構文

```
void QAMessage::setInReplyToID(
  qa_const_string id
)
```

パラメータ

- **id**　In-Reply-To ID

備考

メッセージの In-Reply-To ID を設定します。

クライアントは、In-Reply-To ID ヘッダ・フィールドを使用してメッセージ間リンクを設定できます。これは、応答メッセージをそれに対応する要求メッセージとリンクさせる場合によく使用されます。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

setIntProperty 関数

構文

```
void QAMessage::setIntProperty(
  qa_const_string name,
  qa_int value
)
```

パラメータ

- **name**　設定するプロパティの名前
- **value**　プロパティの qa_int 値

備考

指定された名前を持つ qa_int 型プロパティを指定された値に設定します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

MessageProperties クラス.

setLongProperty 関数

構文

void **QAMessage::setLongProperty(**
  qa_const_string *name*,
  qa_long *value*
)

パラメータ

- **name**　設定するプロパティの名前
- **value**　プロパティの qa_long 値

備考

指定された名前を持つ qa_long 型プロパティを指定された値に設定します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

MessageProperties クラス.

setMessageID 関数

構文

void **QAMessage::setMessageID(**
  qa_const_string *id*
)

パラメータ

- **id**　メッセージ ID

備考

メッセージ ID を設定します。

このメソッドを使用すると、受信したメッセージにすでに設定されている値を変更できます。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

setPriority 関数

構文

void **QAMessage::setPriority(**
  qa_int *priority*
)

パラメータ

- **priority**　メッセージの優先度

備考

メッセージの優先度レベルを設定します。

このメソッドを使用すると、受信したメッセージにすでに設定されている値を変更できます。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

setRedelivered 関数

構文

```
void QAMessage::setRedelivered(
  qa_bool redelivered
)
```

パラメータ

- **redelivered**　再配信されたかどうかを示す値

備考

メッセージが再配信されたかどうかを示す値を設定します。

このメソッドを使用すると、受信したメッセージにすでに設定されている値を変更できます。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

setReplyToAddress 関数

構文

```
void QAMessage::setReplyToAddress(
  qa_const_string replyTo
)
```

パラメータ

- **replyTo**　返信先アドレス

備考

メッセージの返信先アドレスを設定します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

setShortProperty 関数

構文

```
void QAMessage::setShortProperty(
  qa_const_string name,
  qa_short value
)
```

パラメータ

- **name**　設定するプロパティの名前
- **value**　プロパティの qa_short 値

備考

指定された名前を持つ qa_short 型プロパティを指定された値に設定します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

MessageProperties クラス.

setStringProperty 関数

構文

```
void QAMessage::setStringProperty(
  qa_const_string name,
  qa_const_string value
)
```

パラメータ

- **name**　設定するプロパティの名前
- **value**　プロパティの qa_string 値

備考

指定された名前を持つ qa_string 型プロパティを指定された値に設定します。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

MessageProperties クラス.

setTimestamp 関数

構文

```
void QAMessage::setTimestamp(
  qa_long timestamp
)
```

パラメータ

- **timestamp**　メッセージのタイムスタンプ。これは協定世界時 (UTC: Coordinated Universal Time) です。

備考

メッセージのタイムスタンプを設定します。

このメソッドを使用すると、受信したメッセージにすでに設定されている値を変更できます。

メッセージ・ヘッダとプロパティの取得と設定の詳細については、「メッセージ・ヘッダとメッセージ・プロパティ」 220 ページを参照してください。

参照

getTimestamp 関数

QAMessageListener クラス

構文

public **QAMessageListener**

備考

QAMessageListener オブジェクトは、配信されたメッセージを非同期的に受信するために使用します。

メンバ

QAMessageListener クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「onMessage 関数」 516 ページ
- 「~QAMessageListener 関数」 516 ページ

onMessage 関数

構文

void **QAMessageListener::onMessage(**
  QAMessage * *message*
)

パラメータ

- **message**　リスナに渡されるメッセージ

備考

メッセージをリスナに渡します。

~QAMessageListener 関数

構文

virtual **QAMessageListener::~QAMessageListener()**

備考

仮想デストラクタです。

QATextMessage クラス

構文

public **QATextMessage**

基本クラス

- 「QAMessage クラス」 494 ページ

備考

QATextMessage は QAMessage クラスを継承したもので、メッセージ本文にテキストが追加されます。

QATextMessage には、メッセージ本文からのテキストの読み込み／書き込みを行うためのメソッドがあります。

メッセージが最初に作成された時点では、メッセージ本文は書き込み専用モードになっています。メッセージ送信元のクライアントは、メッセージを送信した後、そのメッセージを保持し変更できます。ただし、それによって、送信されたメッセージが変更されることはありません。同じメッセージ・オブジェクトを複数回送信できます。

メッセージが受信されるとプロバイダが QATextMessagereset を呼び出します。これによりメッセージ本文が読み込み専用モードになり、メッセージ本文の先頭から値の読み込みが開始されます。クライアントが読み込み専用モードのメッセージに書き込もうとすると、COMMON_MSG_NOT_WRITEABLE_ERROR が設定されます。

参照

QABinaryMessage クラス

メンバ

QATextMessage クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「beginEnumPropertyNames 関数」 496 ページ
- 「castToBinaryMessage 関数」 496 ページ
- 「castToTextMessage 関数」 496 ページ
- 「clearProperties 関数」 497 ページ
- 「DEFAULT_PRIORITY 変数」 495 ページ
- 「DEFAULT_TIME_TO_LIVE 変数」 495 ページ
- 「endEnumPropertyNames 関数」 497 ページ
- 「getAddress 関数」 497 ページ
- 「getBooleanProperty 関数」 498 ページ
- 「getByteProperty 関数」 498 ページ
- 「getDoubleProperty 関数」 499 ページ
- 「getExpiration 関数」 499 ページ
- 「getFloatProperty 関数」 500 ページ
- 「getInReplyToID 関数」 501 ページ
- 「getIntProperty 関数」 501 ページ
- 「getLongProperty 関数」 501 ページ

getText 関数

構文

qa_string **QATextMessage::getText()**

備考

メッセージのデータが格納されている文字列を取得します。

デフォルト値は null です。

メッセージのサイズが、QAManagerMAX_IN_MEMORY_MESSAGE_SIZE プロパティで指定された最大サイズを超える場合、この関数は null を返します。この場合は、QATextMessagereadText メソッドを使用してテキストを読み込みます。

QAManager のプロパティの詳細については、「QAnywhere Manager の設定プロパティ」 69 ページを参照してください。

戻り値

メッセージのデータが格納されている文字列

getTextLength 関数

構文

qa_long **QATextMessage::getTextLength()**

備考

テキスト長を返します。

注意 : テキスト長がゼロ以外であるのに getText() が qa_null を返す場合があります。これは、そのテキストが大き過ぎてメモリに収まらないためです。この場合、readText を使用して、テキストを分割して読み込みます。

戻り値

テキスト長

readText 関数

構文

```
qa_int QATextMessage::readText(
  qa_string string,
  qa_int length
)
```

パラメータ

- **string**　テキストの格納先
- **length**　格納先バッファに読み込む qa_char の最大数 (null 終端子も含む)

備考

現在のテキスト位置から、要求された長さのテキストをバッファ内に読み込みます。

戻り値

実際に読み込まれた null 以外の qa_char の数。テキスト・ストリーム全体が読み込まれた場合は -1。

reset 関数

構文

void **QATextMessage::reset()**

備考

現在のテキスト位置を先頭に戻します。

setText 関数

構文

void **QATextMessage::setText(**
  qa_const_string *string*
)

パラメータ

- **string**　設定するメッセージ・データが含まれている文字列

備考

メッセージのデータが格納されている文字列を設定します。

writeText 関数

構文

void **QATextMessage::writeText(**
  qa_const_string *string*,
  qa_int *offset*,
  qa_int *length*
)

パラメータ

- **string**　連結するソース・テキスト
- **offset**　ソース・テキスト内でのオフセット (読み込み開始位置)
- **length**　ソース・テキストから読み込む qa_char の数。

備考

現在のテキストに、テキストを連結します。

~QATextMessage 関数

構文

virtual **QATextMessage::~QATextMessage()**

備考

仮想デストラクタです。

QATransactionalManager クラス

構文

public **QATransactionalManager**

基本クラス

- 「QAManagerBase クラス」 460 ページ

備考

このクラスは、トランザクション志向メッセージング用のマネージャです。

QATransactionalManager クラスは QAManagerBase から派生し、トランザクション志向の QAnywhere メッセージング操作を管理します。

この動作の完全な説明については、QAManagerBase クラスを参照してください。

QATransactionalManager は、トランザクション志向の受信確認でのみ使用できます。QAManagerBaseputMessage() と QAManagerBasegetMessage() のすべての呼び出しをコミットする場合は、QATransactionalManagercommit() メソッドを使用します。

詳細については、「トランザクション志向メッセージングの実装」 75 ページを参照してください。

参照

QATransactionalManager クラス.

メンバ

QATransactionalManager クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「beginEnumStorePropertyNames 関数」 462 ページ
- 「browseClose 関数」 462 ページ
- 「browseMessages 関数」 462 ページ
- 「browseMessagesByID 関数」 463 ページ
- 「browseMessagesByQueue 関数」 464 ページ
- 「browseMessagesBySelector 関数」 464 ページ
- 「browseNextMessage 関数」 465 ページ
- 「cancelMessage 関数」 466 ページ
- 「close 関数」 466 ページ
- 「commit 関数」 522 ページ
- 「createBinaryMessage 関数」 467 ページ
- 「createTextMessage 関数」 467 ページ
- 「deleteMessage 関数」 468 ページ
- 「endEnumStorePropertyNames 関数」 468 ページ
- 「getAllQueueDepth 関数」 468 ページ
- 「getBooleanStoreProperty 関数」 469 ページ
- 「getByteStoreProperty 関数」 470 ページ

commit 関数

構文

qa_bool **QATransactionalManager::commit()**

備考

現在のトランザクションをコミットし、新しいトランザクションを開始します。

このメソッドは、QAManagerBaseputMessage() と QAManagerBasegetMessage() のすべての呼び出しをコミットします。

*注意：*最初のトランザクションは、QATransactionalManageropen() の呼び出しで開始されます。

参照

QATransactionalManager クラス

戻り値

commit 処理が正常終了した場合のみ True

open 関数

構文

qa_bool **QATransactionalManager::open()**

備考

QATransactionalManager インスタンスをオープンします。

open メソッドは、Manager を作成した後、最初に呼び出す必要があるメソッドです。

参照

QATransactionalManager クラス

戻り値

処理が正常終了した場合のみ True

rollback 関数

構文

qa_bool **QATransactionalManager::rollback()**

備考

現在のトランザクションをロールバックし、新しいトランザクションを開始します。

このメソッドは、コミットされていない QAManagerBaseputMessage() と QAManagerBasegetMessage() のすべての呼び出しをロールバックします。

参照

QATransactionalManager クラス

戻り値

open 処理が正常終了した場合のみ True

~QATransactionalManager 関数

構文

virtual **QATransactionalManager::~QATransactionalManager()**

備考

仮想デストラクタです。

QueueDepthFilter クラス

構文

public **QueueDepthFilter**

備考

QAManagerBase のキューの深さ関連メソッドの QueueDepthFilter の値です。

メンバ

QueueDepthFilter クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「ALL 変数」 525 ページ
- 「INCOMING 変数」 525 ページ
- 「OUTGOING 変数」 525 ページ

ALL 変数

構文

const qa_short **QueueDepthFilter::ALL**

備考

着信メッセージと送信メッセージの両方をカウントします。

システム・メッセージと期限切れのメッセージは、キューの深さカウントに計上されません。

INCOMING 変数

構文

const qa_short **QueueDepthFilter::INCOMING**

備考

着信メッセージのみカウントします。

着信メッセージは、発信者がメッセージ・ストアのエージェント ID ではないメッセージです。

OUTGOING 変数

構文

const qa_short **QueueDepthFilter::OUTGOING**

備考

送信メッセージのみカウントします。

送信メッセージは、発信者がメッセージ・ストアのエージェント ID であり、送信先がメッセージ・ストアのエージェント ID ではないメッセージです。

StatusCodes クラス

構文

public **StatusCodes**

備考

このインタフェースは、メッセージのステータス・コード・セットを定義します。

メンバ

StatusCodes クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

CANCELLED 変数

構文

const qa_int **StatusCodes::CANCELLED**

備考

メッセージはキャンセル済みです。

このコードの値は 40 です。このコードは MessagePropertiesSTATUS で使用されます。

EXPIRED 変数

構文

const qa_int **StatusCodes::EXPIRED**

備考

メッセージの有効期限が切れました。メッセージは、有効期限になる前に受信されませんでした。

このコードの値は 30 です。このコードは MessagePropertiesSTATUS で使用されます。

FINAL 変数

構文

const qa_int **StatusCodes::FINAL**

備考

メッセージは最終ステータスになりました。

このコードの値は 20 です。このコードは MessagePropertiesSTATUS で使用されます。

LOCAL 変数

構文

const qa_int **StatusCodes::LOCAL**

備考

メッセージはローカル・メッセージ・ストア宛てに送信され、サーバに送信されません。

このコードの値は 2 です。このコードは MessagePropertiesTRANSMISSION_STATUS で使用されます。

PENDING 変数

構文

const qa_int **StatusCodes::PENDING**

備考

メッセージは送信されたが、まだ受信されていません。

このコードの値は 1 です。このコードは MessagePropertiesSTATUS で使用されます。

RECEIVED 変数

構文

const qa_int **StatusCodes::RECEIVED**

備考

メッセージが受信され、受信者によって受信確認されました。

このコードの値は 60 です。このコードは MessagePropertiesSTATUS で使用されます。

RECEIVING 変数

構文

const qa_int **StatusCodes::RECEIVING**

備考

メッセージは受信中であるか、または受信されたがまだ確認されていません。

このコードの値は 10 です。このコードは MessagePropertiesSTATUS で使用されます。

TRANSMITTED 変数

構文

const qa_int **StatusCodes::TRANSMITTED**

備考

メッセージはサーバに送信されました。

このコードの値は 1 です。このコードは MessagePropertiesTRANSMISSION_STATUS で使用されます。

TRANSMITTING 変数

構文

const qa_int **StatusCodes::TRANSMITTING**

備考

メッセージはサーバに送信中です。

このコードの値は 3 です。このコードは MessagePropertiesTRANSMISSION_STATUS で使用されます。

UNRECEIVABLE 変数

構文

const qa_int **StatusCodes::UNRECEIVABLE**

備考

メッセージは受信不可のマークが付けられています。

メッセージに間違った形式があるか、失敗した試行回数が多すぎて配信されませんでした。

このコードの値は 50 です。このコードは MessagePropertiesSTATUS で使用されます。

UNTRANSMITTED 変数

構文

const qa_int **StatusCodes::UNTRANSMITTED**

備考

メッセージはサーバに送信されていません。

このコードの値は 0 です。このコードは MessagePropertiesTRANSMISSION_STATUS で使用されます。

第 14 章

QAnywhere Java API リファレンス

目次

ianywhere.qanywhere.client パッケージ

AcknowledgementMode インタフェース

構文

public **ianywhere.qanywhere.client.AcknowledgementMode**

備考

QAnywhere クライアント・アプリケーションによってどのようにメッセージが確認されるかを示します。

暗黙的または明示的な受信確認モードは、QAManager.open(short) メソッドを使用して、QAManager インスタンスに割り当てられます。

暗黙的な受信確認では、メッセージは、クライアント・アプリケーションで受信されるとすぐに受信確認されます。明示的な受信確認では、いずれかの QAManager 受信確認メソッドを呼び出してください。すべてのステータス変更は、サーバによってクライアント間で伝達されます。

参照

QAManager インタフェース

QATransactionalManager インタフェース

QAManagerBase インタフェース

メンバ

ianywhere.qanywhere.client.AcknowledgementMode のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「EXPLICIT_ACKNOWLEDGEMENT 変数」 532 ページ
- 「IMPLICIT_ACKNOWLEDGEMENT 変数」 533 ページ
- 「TRANSACTIONAL 変数」 533 ページ

EXPLICIT_ACKNOWLEDGEMENT 変数

構文

final short **ianywhere.qanywhere.client.AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT**

備考

受信メッセージは、QAManager のいずれかの受信確認メソッドを使用して確認されます。

参照

QAManager インタフェース

IMPLICIT_ACKNOWLEDGEMENT 変数

構文

final short **ianywhere.qanywhere.client.AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT**

備考

すべてのメッセージは、クライアント・アプリケーションで受信されるとすぐに受信確認されます。

メッセージを同期に受信する場合、メッセージは QAManagerBase.getMessage(String) メソッドが返されるとすぐに受信確認されます。メッセージを非同期に受信する場合、メッセージはイベント処理メソッドが返されるとすぐに受信確認されます。

参照

getMessage メソッド

TRANSACTIONAL 変数

構文

final short **ianywhere.qanywhere.client.AcknowledgementMode.TRANSACTIONAL**

備考

このモードでは、メッセージの受信確認はトランザクションの一部として行われます。それ以外では行われません。

このモードは、QATransactionalManager インスタンスに自動的に割り当てられます。

参照

QATransactionalManager インタフェース

MessageProperties インタフェース

構文

public **ianywhere.qanywhere.client.MessageProperties**

備考

標準のメッセージ・プロパティ名を格納するフィールドを提供します。

MessageProperties クラスは、標準のメッセージ・プロパティ名を提供します。MessageProperties フィールドは、メッセージ・プロパティの取得と設定で使用する QAMessage メソッドに渡すことができます。

```
QAMessage msg = mgr.createTextMessage();
```


次の例では、QAMessage.getIntProperty(String) メソッドを使用して MessageProperties.MSG_TYPE に対応する値を取得します。MessageType 列挙は、整数値の結果を適切なメッセージ・タイプにマッピングします。

```
int msg_type = t_msg.getIntProperty( MessageProperties.MSG_TYPE );
```

次の例は、QAnywhere システム・メッセージの処理で使用される onSystemMessage(QAMessage) メソッドを示します。

メッセージ・タイプは、MessageProperties.MSG_TYPE 変数と QAMessage.getIntProperty(String) メソッドを使用して評価されます。

```
private void onSystemMessage(QAMessage msg) {
  QATextMessage    t_msg;
  int              msg_type;
  String           network_adapters;
  String           network_names;
  String           network_info;

  t_msg = (QATextMessage)msg;
  if( t_msg != null ) {
     // Evaluate the message type.
     msg_type = (MessageType)t_msg.getIntProperty( MessageProperties.MSG_TYPE );
     if( msg_type == MessageType.NETWORK_STATUS_NOTIFICATION ) {
        // Handle network status notification.
        network_info = "";
        network_adapters = t_msg.getStringProperty( MessageProperties.ADAPTERS );
        if( network_adapters != null && network_adapters.length > 0) {
          network_info += network_adapters;
        }
        network_names = t_msg.getStringProperty( MessageProperties.RASNAMES );

         //...
     }
  }
}
```

メンバ

ianywhere.qanywhere.client.MessageProperties のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「ADAPTER 変数」 535 ページ
- 「ADAPTERS 変数」 535 ページ
- 「DELIVERY_COUNT 変数」 535 ページ
- 「IP 変数」 536 ページ
- 「MAC 変数」 536 ページ
- 「MSG_TYPE 変数」 537 ページ
- 「NETWORK_STATUS 変数」 537 ページ
- 「ORIGINATOR 変数」 537 ページ
- 「RAS 変数」 538 ページ
- 「RASNAMES 変数」 538 ページ
- 「STATUS 変数」 539 ページ
- 「STATUS_TIME 変数」 539 ページ

- 「TRANSMISSION_STATUS 変数」 539 ページ

ADAPTER 変数

構文

final String **ianywhere.qanywhere.client.MessageProperties.ADAPTER**

備考

"system" キュー・メッセージの場合は、QAnywhere サーバへの接続で使用されているネットワーク・アダプタです。

このフィールドの値は "ias_Network.Adapter" です。

MessageProperties.ADAPTER を QAMessage.getStringProperty(String) メソッドに渡して、関連付けられているプロパティにアクセスできます。

このプロパティは読み込み専用です。

参照

MessageProperties インタフェース

getStringProperty メソッド

ADAPTERS 変数

構文

final String **ianywhere.qanywhere.client.MessageProperties.ADAPTERS**

備考

このプロパティ名は、QAnywhere サーバへの接続で使用できるネットワーク・アダプタのデリミタ付きリストを示します。

これは、システム・キュー・メッセージで使用されます。

MessageProperties.ADAPTERS を QAMessage.getStringProperty(String) メソッドに渡して、関連付けられているプロパティにアクセスできます。このプロパティは読み込み専用です。

参照

MessageProperties インタフェース

getStringProperty メソッド

DELIVERY_COUNT 変数

構文

final String **ianywhere.qanywhere.client.MessageProperties.DELIVERY_COUNT**

備考

このプロパティ名は、メッセージを配信するために、これまでに実行された試行回数を示します。

IP 変数

構文

final String **ianywhere.qanywhere.client.MessageProperties.IP**

備考

"system" キュー・メッセージの場合は、QAnywhere サーバへの接続で使用されているネットワーク・アダプタの IP アドレスです。

このフィールドの値は "ias_Network.IP" です。

MessageProperties.IP を QAMessage.getStringProperty(String) メソッドに渡して、関連付けられているプロパティにアクセスできます。

このプロパティは読み込み専用です。

参照

MessageProperties インタフェース

getStringProperty メソッド

MAC 変数

構文

final String **ianywhere.qanywhere.client.MessageProperties.MAC**

備考

"system" キュー・メッセージの場合は、QAnywhere サーバへの接続で使用されているネットワーク・アダプタの MAC アドレスです。

このフィールドの値は "ias_Network.MAC" です。

MessageProperties.MAC を QAMessage.getStringProperty(String) メソッドに渡して、関連付けられているプロパティにアクセスできます。

このプロパティは読み込み専用です。

参照

MessageProperties インタフェース

getStringProperty メソッド

MSG_TYPE 変数

構文

final String **ianywhere.qanywhere.client.MessageProperties.MSG_TYPE**

備考

このプロパティ名は、QAnywhere メッセージに関連付けられている MessageType 列挙値を示します。

このフィールドの値は "ias_MessageType" です。

MessageProperties.MSG_TYPE を QAMessage.getIntProperty(String) メソッドに渡して、関連付けられているプロパティにアクセスできます。

このプロパティは読み込み専用です。

参照

MessageProperties インタフェース

getIntProperty メソッド

NETWORK_STATUS 変数

構文

final String **ianywhere.qanywhere.client.MessageProperties.NETWORK_STATUS**

備考

このプロパティ名は、ネットワーク接続のステータスを示します。

ネットワークがアクセス可能な場合は 1、それ以外の場合は 0 になります。ネットワーク・ステータスは、システム・キュー・メッセージ (ネットワーク・ステータスの変更など) で使用されます。

MessageProperties.NETWORK_STATUS を QAMessage.getIntProperty(String) メソッドに渡して、関連付けられているプロパティにアクセスできます。

このプロパティは読み込み専用です。

参照

MessageProperties インタフェース

getIntProperty メソッド

ORIGINATOR 変数

構文

final String **ianywhere.qanywhere.client.MessageProperties.ORIGINATOR**

備考

このプロパティ名は、メッセージの発信者のメッセージ・ストア ID を示します。

RAS 変数

構文

final String **ianywhere.qanywhere.client.MessageProperties.RAS**

備考

"system" キュー・メッセージの場合は、QAnywhere サーバへの接続で使用されている RAS エントリ名です。

このフィールドの値は "ias_Network.RAS" です。

MessageProperties.RAS を QAMessage.getStringProperty(String) メソッドに渡して、関連付けられているプロパティにアクセスできます。

このプロパティは読み込み専用です。

参照

MessageProperties インタフェース

getStringProperty メソッド

RASNAMES 変数

構文

final String **ianywhere.qanywhere.client.MessageProperties.RASNAMES**

備考

"system" キュー・メッセージの場合は、QAnywhere サーバに接続するときに使用できる RAS エントリ名のデリミタ付きリストです。

このフィールドの値は "ias_RASNames" です。

MessageProperties.RASNAMES を QAMessage.getStringProperty(String) メソッドに渡して、関連付けられているプロパティにアクセスできます。

このプロパティは読み込み専用です。

参照

MessageProperties インタフェース

getStringProperty メソッド

STATUS 変数

構文

final String **ianywhere.qanywhere.client.MessageProperties.STATUS**

備考

このプロパティ名は、メッセージの現在のステータスを示します。

参照

StatusCodes インタフェース

STATUS_TIME 変数

構文

final String **ianywhere.qanywhere.client.MessageProperties.STATUS_TIME**

備考

このプロパティ名は、メッセージが現在のステータスになったと判断された時刻を示します。

MessageProperties.STATUS_TIME を QAMessage.getProperty メソッドに渡すと、java.util.Date インスタンスが返されます。

参照

getProperty メソッド

TRANSMISSION_STATUS 変数

構文

final String **ianywhere.qanywhere.client.MessageProperties.TRANSMISSION_STATUS**

備考

このプロパティ名は、メッセージの現在の転送ステータスを示します。

参照

StatusCodes インタフェース

MessageStoreProperties インタフェース

構文

public **ianywhere.qanywhere.client.MessageStoreProperties**

備考

このクラスは、有用なメッセージ・ストア・プロパティ名の定数値を定義します。

MessageStoreProperties クラスは、標準のメッセージ・プロパティ名を提供します。MessageStoreProperties フィールドは、事前定義済みまたはカスタムのメッセージ・ストア・プロパティの取得と設定で使用する QAManagerBase メソッドに渡すことができます。

参照

QAManagerBase インタフェース

メンバ

ianywhere.qanywhere.client.MessageStoreProperties のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「MAX_DELIVERY_ATTEMPTS 変数」 540 ページ

MAX_DELIVERY_ATTEMPTS 変数

構文

final String **ianywhere.qanywhere.client.MessageStoreProperties.MAX_DELIVERY_ATTEMPTS**

備考

このプロパティ名は、メッセージのステータスが StatusCodes.UNRECEIVABLE に設定されるまでに、受信確認がないままメッセージを受信できる最大回数を示します。

参照

UNRECEIVABLE 変数

MessageType インタフェース

構文

public **ianywhere.qanywhere.client.MessageType**

備考

MessageProperties.MSG_TYPE メッセージ・プロパティの定数値を定義します。

次の例は、QAnywhere システム・メッセージの処理で使用される onSystemMessage(QAMessage) メソッドを示します。メッセージ・タイプは、MessageType.NETWORK_STATUS_NOTIFICATION と比較されます。

```
private void onSystemMessage(QAMessage msg)
{
  QATextMessage    t_msg;
  int              msg_type;
  String           network_adapters;
  String           network_names;
  String           network_info;

  t_msg = (QATextMessage)msg;
  if( t_msg != null )
  {
```



```
        // Evaluate message type.
        msg_type = t_msg.getIntProperty( MessageProperties.MSG_TYPE );
        if( msg_type == MessageType.NETWORK_STATUS_NOTIFICATION )
        {
            // Handle network status notification.

        }
    }
}
```

メンバ

ianywhere.qanywhere.client.MessageType のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「NETWORK_STATUS_NOTIFICATION 変数」 541 ページ
- 「PUSH_NOTIFICATION 変数」 541 ページ
- 「REGULAR 変数」 542 ページ

NETWORK_STATUS_NOTIFICATION 変数

構文

final int **ianywhere.qanywhere.client.MessageType.NETWORK_STATUS_NOTIFICATION**

備考

QAnywhere クライアント・アプリケーションにネットワーク・ステータスの変更を通知する場合に使用する、QAnywhere システム・メッセージを特定します。

ネットワーク・ステータスの変更は、システム・メッセージを受信するデバイスに適用されます。新しいネットワーク・ステータス情報を特定するには、MessageProperties.ADAPTER、MessageProperties.NETWORK、MessageProperties.NETWORK_STATUS の各フィールドを使用します。

PUSH_NOTIFICATION 変数

構文

final int **ianywhere.qanywhere.client.MessageType.PUSH_NOTIFICATION**

備考

QAnywhere クライアント・アプリケーションに Push 通知を通知する場合に使用する、QAnywhere システム・メッセージを特定します。

オンデマンドの QAnywhere Agent ポリシーを使用した場合の一般的な応答は、QAManagerBase.triggerSendReceive() メソッドを呼び出し、中央のメッセージ・サーバで待機しているメッセージを受信することです。

REGULAR 変数

構文

final int **ianywhere.qanywhere.client.MessageType.REGULAR**

備考

メッセージ・タイプ・プロパティが存在しない場合、メッセージ・タイプは REGULAR とみなされます。

このタイプのメッセージは、メッセージ・システムで特別な取り扱いを受けません。

PropertyType インタフェース

構文

public **ianywhere.qanywhere.client.PropertyType**

備考

QAMessage プロパティ型列挙。Java の型に必然的に対応します。

参照

QAMessage インタフェース

メンバ

ianywhere.qanywhere.client.PropertyType のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「PROPERTY_TYPE_BOOLEAN 変数」 542 ページ
- 「PROPERTY_TYPE_BYTE 変数」 543 ページ
- 「PROPERTY_TYPE_DOUBLE 変数」 543 ページ
- 「PROPERTY_TYPE_FLOAT 変数」 543 ページ
- 「PROPERTY_TYPE_INT 変数」 543 ページ
- 「PROPERTY_TYPE_LONG 変数」 543 ページ
- 「PROPERTY_TYPE_SHORT 変数」 543 ページ
- 「PROPERTY_TYPE_STRING 変数」 544 ページ
- 「PROPERTY_TYPE_UNKNOWN 変数」 544 ページ

PROPERTY_TYPE_BOOLEAN 変数

構文

final short **ianywhere.qanywhere.client.PropertyType.PROPERTY_TYPE_BOOLEAN**

備考

プロパティ型が boolean であることを示します。

PROPERTY_TYPE_BYTE 変数

構文

final short **ianywhere.qanywhere.client.PropertyType.PROPERTY_TYPE_BYTE**

備考

プロパティ型が signed byte であることを示します。

PROPERTY_TYPE_DOUBLE 変数

構文

final short **ianywhere.qanywhere.client.PropertyType.PROPERTY_TYPE_DOUBLE**

備考

プロパティ型が double であることを示します。

PROPERTY_TYPE_FLOAT 変数

構文

final short **ianywhere.qanywhere.client.PropertyType.PROPERTY_TYPE_FLOAT**

備考

プロパティ型が float であることを示します。

PROPERTY_TYPE_INT 変数

構文

final short **ianywhere.qanywhere.client.PropertyType.PROPERTY_TYPE_INT**

備考

プロパティ型が int であることを示します。

PROPERTY_TYPE_LONG 変数

構文

final short **ianywhere.qanywhere.client.PropertyType.PROPERTY_TYPE_LONG**

備考

プロパティ型が long であることを示します。

PROPERTY_TYPE_SHORT 変数

構文

final short **ianywhere.qanywhere.client.PropertyType.PROPERTY_TYPE_SHORT**

備考

プロパティ型が short であることを示します。

PROPERTY_TYPE_STRING 変数

構文

final short **ianywhere.qanywhere.client.PropertyType.PROPERTY_TYPE_STRING**

備考

プロパティ型が文字列型であることを示します。

PROPERTY_TYPE_UNKNOWN 変数

構文

final short **ianywhere.qanywhere.client.PropertyType.PROPERTY_TYPE_UNKNOWN**

備考

プロパティ型が不定であることを示します。通常はプロパティを認識できないことが原因です。

QABinaryMessage インタフェース

構文

public **ianywhere.qanywhere.client.QABinaryMessage**

基本クラス

- 「QAMessage インタフェース」 606 ページ

備考

QABinaryMessage オブジェクトは、未解釈のバイト・ストリームが含まれるメッセージの送信に使用します。

QABinaryMessage は QAMessage クラスを継承したもので、メッセージ本文にバイト・ストリームが追加されます。QABinaryMessage には、メッセージ本文からのバイト・ストリームの読み込み／書き込みを行うためのさまざまな関数があります。

メッセージが最初に作成された時点では、メッセージ本文は書き込み専用モードになっています。メッセージ送信元のクライアントは、メッセージを送信した後、そのメッセージを保持し変更できます。ただし、それによって、送信されたメッセージが変更されることはありません。同じメッセージ・オブジェクトを複数回送信できます。

メッセージが受信されるとプロバイダが QABinaryMessage.reset() を呼び出します。これによりメッセージ本文が読み込み専用モードになり、メッセージ本文の先頭から値の読み込みが開始されます。

次の例では QABinaryMessage.writeString(String) を使用して、QABinaryMessage インスタンスのメッセージ本文に文字列 "Q" と "Anywhere" を書き込みます。

```
// Create a binary message instance.
QABinaryMessage binary_message;
binary_message = qa_manager.createBinaryMessage();

// Set optional message properties.
binary_message.setReplyToAddress("my-queue-name");

// Write to the message body.
binary_message.writeString("Q");
binary_message.writeString("Anywhere");

// Put the message in the local database, ready for sending.
try {
   qa_manager.putMessage( "store-id¥¥queue-name", binary_message );
}
catch ( QAException e ) {
   handleError();
}
```

注意 : 受信が終了すると、最初の QABinaryMessage.readString() の呼び出しから "Q" が返され、2 番目の QABinaryMessage.readString() の呼び出しから "Anywhere" が返されます。

メッセージは QAnywhere Agent から送信されます。

参照

QAMessage インタフェース

readString メソッド

メンバ

ianywhere.qanywhere.client.QABinaryMessage のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「clearProperties メソッド」 608 ページ
- 「DEFAULT_PRIORITY 変数」 607 ページ
- 「DEFAULT_TIME_TO_LIVE 変数」 608 ページ
- 「getAddress メソッド」 608 ページ
- 「getBodyLength メソッド」 547 ページ
- 「getBooleanProperty メソッド」 608 ページ
- 「getByteProperty メソッド」 609 ページ
- 「getDoubleProperty メソッド」 610 ページ
- 「getExpiration メソッド」 610 ページ
- 「getFloatProperty メソッド」 611 ページ
- 「getInReplyToID メソッド」 611 ページ
- 「getIntProperty メソッド」 612 ページ
- 「getLongProperty メソッド」 612 ページ
- 「getMessageID メソッド」 613 ページ
- 「getPriority メソッド」 613 ページ
- 「getProperty メソッド」 614 ページ
- 「getPropertyNames メソッド」 614 ページ
- 「getPropertyType メソッド」 614 ページ
- 「getRedelivered メソッド」 615 ページ

getBodyLength メソッド

構文

long **ianywhere.qanywhere.client.QABinaryMessage.getBodyLength()**
throws **QAException**

スロー

- メッセージ本文のサイズの取得で問題が発生した場合にスローされます。

備考

メッセージ本文のサイズをバイト単位で返します。

戻り値

メッセージ本文のサイズ (バイト単位)

readBinary メソッド

構文

int **ianywhere.qanywhere.client.QABinaryMessage.readBinary(**
byte[] *dest*
)
throws **QAException**

パラメータ

- **dest**　読み込まれたバイトを保持するバイト配列

スロー

- 値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

備考

QABinaryMessage インスタンスの本文の未読部分から、指定されたサイズのバイト・ストリームを読み込みます。

参照

writeBinary メソッド

戻り値

メッセージ本文から読み込まれるバイト数

readBinary メソッド

構文

int **ianywhere.qanywhere.client.QABinaryMessage.readBinary(**
  byte[] *dest*,
  int *length*
)
throws **QAException**

パラメータ

- **dest**　読み込まれたバイトを保持するバイト配列
- **length**　読み込むバイト数の最大値

スロー

- 値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

備考

QABinaryMessage インスタンスの本文の未読部分から、指定されたサイズのバイト・ストリームを読み込みます。

参照

writeBinary メソッド

戻り値

メッセージ本文から読み込まれるバイト数

readBoolean メソッド

構文

boolean **ianywhere.qanywhere.client.QABinaryMessage.readBoolean()**
throws **QAException**

スロー

- 値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

備考

QABinaryMessage インスタンスのメッセージ本文の未読部分から boolean 値を読み込みます。

参照

writeBoolean メソッド

戻り値

メッセージ本文から読み込まれる boolean 値

readByte メソッド

構文

byte **ianywhere.qanywhere.client.QABinaryMessage.readByte()**
throws **QAException**

スロー

- 値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

備考

QABianryMessage メッセージ本文の未読部分から signed byte 値を読み込みます。

参照

writeByte メソッド

戻り値

メッセージ本文から読み込まれる signed byte 値

readChar メソッド

構文

char **ianywhere.qanywhere.client.QABinaryMessage.readChar()**
throws **QAException**

スロー

- 値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

備考

QABianryMessage メッセージ本文の未読部分から char 値を読み込みます。

参照

writeChar メソッド

戻り値

メッセージ本文から読み込まれる char 値

readDouble メソッド

構文

double **ianywhere.qanywhere.client.QABinaryMessage.readDouble()**
throws **QAException**

スロー

- 値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

備考

QABianryMessage メッセージ本文の未読部分から double 値を読み込みます。

参照

writeDouble メソッド

戻り値

メッセージ本文から読み込まれる double 値

readFloat メソッド

構文

float **ianywhere.qanywhere.client.QABinaryMessage.readFloat()**
throws **QAException**

スロー

- 値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

備考

QABianryMessage メッセージ本文の未読部分から float 値を読み込みます。

参照

writeFloat メソッド

戻り値

メッセージ本文から読み込まれる float 値

readInt メソッド

構文

int **ianywhere.qanywhere.client.QABinaryMessage.readInt()**
throws **QAException**

スロー

- 値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

備考

QABianryMessage メッセージ本文の未読部分から int 値を読み込みます。

参照

writeInt メソッド

戻り値

メッセージ本文から読み込まれる int 値

readLong メソッド

構文

long **ianywhere.qanywhere.client.QABinaryMessage.readLong()**
throws **QAException**

スロー

- 値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

備考

QABianryMessage メッセージ本文の未読部分から long 値を読み込みます。

参照

writeLong メソッド

戻り値

メッセージ本文から読み込まれる long 値

readShort メソッド

構文

short **ianywhere.qanywhere.client.QABinaryMessage.readShort()**
throws **QAException**

スロー

- 値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

備考

QABianryMessage メッセージ本文の未読部分から short 値を読み込みます。

参照

writeShort メソッド

戻り値

メッセージ本文から読み込まれる short 値

readString メソッド

構文

String **ianywhere.qanywhere.client.QABinaryMessage.readString()**
throws **QAException**

スロー

- 値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

備考

QABianryMessage メッセージ本文の未読部分から文字列値を読み込みます。

参照

writeString メソッド

戻り値

メッセージ本文から読み込まれる文字列値

reset メソッド

構文

void **ianywhere.qanywhere.client.QABinaryMessage.reset()**
throws **QAException**

スロー

- メッセージのリセットで問題が発生した場合にスローされます。

備考

メッセージをリセットして、メッセージ本文の先頭から値の読み込みを開始できるようにします。

また、reset メソッドは、QABinaryMessage のメッセージ本文を読み込み専用モードにします。

writeBinary メソッド

構文

void **ianywhere.qanywhere.client.QABinaryMessage.writeBinary(**
  byte[] *val*
)
throws **QAException**

パラメータ

- **val**　メッセージ本文に書き込むバイト配列値

スロー

- メッセージ本文へのバイト配列の追加で問題が発生した場合にスローされます。

備考

QABinaryMessage インスタンスのメッセージ本文にバイト配列値を追加します。

参照

readBinary メソッド

writeBinary メソッド

構文

void **ianywhere.qanywhere.client.QABinaryMessage.writeBinary(**
  byte[] *val*,
  int *len*
)
throws **QAException**

パラメータ

- **val**　メッセージ本文に書き込むバイト配列値
- **len**　書き込まれるバイト数

スロー

- メッセージ本文へのバイト配列の追加で問題が発生した場合にスローされます。

備考

QABinaryMessage インスタンスのメッセージ本文にバイト配列値を追加します。

参照

readBinary メソッド

writeBinary メソッド

構文

void **ianywhere.qanywhere.client.QABinaryMessage.writeBinary(**
byte[] *val*,
int *offset*,
int *len*
)
throws **QAException**

パラメータ

- **val**　メッセージ本文に書き込むバイト配列値
- **offset**　バイト配列内でのオフセット (書き込み開始位置)
- **len**　書き込まれるバイト数

スロー

- メッセージ本文へのバイト配列の追加で問題が発生した場合にスローされます。

備考

QABinaryMessage インスタンスのメッセージ本文にバイト配列値を追加します。

参照

readBinary メソッド

writeBoolean メソッド

構文

void **ianywhere.qanywhere.client.QABinaryMessage.writeBoolean(**
boolean *val*
)
throws **QAException**

パラメータ

- **val**　メッセージ本文に書き込む boolean 値

スロー

- メッセージ本文への boolean 値の追加で問題が発生した場合にスローされます。

備考

QABinaryMessge インスタンスのメッセージ本文に boolean 値を追加します。

boolean 値は 1 バイトの値で示されます。True は 1、false は 0 で示されます。

参照

readBoolean メソッド

writeByte メソッド

構文

void **ianywhere.qanywhere.client.QABinaryMessage.writeByte(**
byte *val*
)
throws **QAException**

パラメータ

- **val**　メッセージ本文に書き込む signed byte 値

スロー

- メッセージ本文への signed byte 値の追加で問題が発生した場合にスローされます。

備考

QABinaryMessge インスタンスのメッセージ本文に signed byte 値を追加します。

signed byte 値は 1 バイトの値で示されます。

参照

readByte メソッド

writeChar メソッド

構文

void **ianywhere.qanywhere.client.QABinaryMessage.writeChar(**
char *val*
)
throws **QAException**

パラメータ

- **val**　メッセージ本文に書き込む char 値

スロー

- メッセージ本文への char 値の追加で問題が発生した場合にスローされます。

備考

QABinaryMessge インスタンスのメッセージ本文に char 値を追加します。

char は 2 バイトの値で示され、上位バイトから追加されます。

参照

readChar メソッド

writeDouble メソッド

構文

```
void ianywhere.qanywhere.client.QABinaryMessage.writeDouble(
  double val
)
throws QAException
```

パラメータ

- **val**　メッセージ本文に書き込む double 値

スロー

- メッセージ本文への double 値の追加で問題が発生した場合にスローされます。

備考

QABinaryMessge インスタンスのメッセージ本文に double 値を追加します。

double 値は 8 バイトの long 値に変換されて、上位のバイトから追加されます。

参照

readDouble メソッド

writeFloat メソッド

構文

```
void ianywhere.qanywhere.client.QABinaryMessage.writeFloat(
  float val
)
throws QAException
```

パラメータ

- **val**　メッセージ本文に書き込む float 値

スロー

- メッセージ本文への float 値の追加で問題が発生した場合にスローされます。

備考

QABinaryMessge インスタンスのメッセージ本文に float 値を追加します。

float 値は 4 バイトの整数に変換され、上位のバイトから追加されます。

参照

readFloat メソッド

writeInt メソッド

構文

void **ianywhere.qanywhere.client.QABinaryMessage.writeInt(**
  int *val*
)
throws **QAException**

パラメータ

- **val**　メッセージ本文に書き込む int 値

スロー

- メッセージ本文への整数値の追加で問題が発生した場合にスローされます。

備考

QABinaryMessge インスタンスのメッセージ本文に整数値を追加します。

整数パラメータは 4 バイトの値で示され、上位のバイトから追加されます。

参照

readInt メソッド

writeLong メソッド

構文

void **ianywhere.qanywhere.client.QABinaryMessage.writeLong(**
  long *val*
)
throws **QAException**

パラメータ

- **val**　メッセージ本文に書き込む long 値

スロー

- メッセージ本文への long 値の追加で問題が発生した場合にスローされます。

備考

QABinaryMessge インスタンスのメッセージ本文に long 値を追加します。

long パラメータは 8 バイトの値で示され、上位のバイトから追加されます。

参照

readLong メソッド

writeShort メソッド

構文

void **ianywhere.qanywhere.client.QABinaryMessage.writeShort(**
  short *val*
)
throws **QAException**

パラメータ

- **val**　メッセージ本文に書き込む short 値

スロー

- メッセージ本文への short 値の追加で問題が発生した場合にスローされます。

備考

QABinaryMessge インスタンスのメッセージ本文に short 値を追加します。

short パラメータは 2 バイトの値で示され、上位のバイトから追加されます。

参照

readShort メソッド

writeString メソッド

構文

void **ianywhere.qanywhere.client.QABinaryMessage.writeString(**
  String *val*
)
throws **QAException**

パラメータ

- **val**　メッセージ本文に書き込む文字列値

スロー

- メッセージ本文への文字列値の追加で問題が発生した場合にスローされます。

備考

QABinaryMessge インスタンスのメッセージ本文に文字列値を追加します。

注意 : 受信側アプリケーションは、writeString が呼び出されるたびに QABinaryMessage.readString を呼び出す必要があります。

*注意：*文字列の UTF-8 表記は、最大で 32767 バイトまで書き込み可能です。

参照

readString メソッド

QAException クラス

構文

public **ianywhere.qanywhere.client.QAException**

備考

QAnywhere クライアント・アプリケーションの例外をカプセル化します。

QAException クラスを使用して QAnywhere の例外を受信できます。

```
try
{
    _qaManager = QAManagerFactory.getInstance().CreateQAManager();
    _qaManager.open( AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT );
    _qaManager.start();
}
catch( QAException e )
{
    // Handle exception.
    System.err.println("Error code: " + e.getErrorCode() );
    System.err.println("Error message: " + e.getMessage() );
}
}
```

メンバ

ianywhere.qanywhere.client.QAException のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「COMMON_ALREADY_OPEN_ERROR 変数」 560 ページ
- 「COMMON_GET_INIT_FILE_ERROR 変数」 561 ページ
- 「COMMON_GETQUEUEDEPTH_ERROR 変数」 560 ページ
- 「COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG 変数」 560 ページ
- 「COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID 変数」 561 ページ
- 「COMMON_INIT_ERROR 変数」 561 ページ
- 「COMMON_INIT_THREAD_ERROR 変数」 561 ページ
- 「COMMON_INVALID_PROPERTY 変数」 562 ページ
- 「COMMON_MSG_ACKNOWLEDGE_ERROR 変数」 562 ページ
- 「COMMON_MSG_CANCEL_ERROR 変数」 562 ページ
- 「COMMON_MSG_CANCEL_ERROR_SENT 変数」 562 ページ
- 「COMMON_MSG_NOT_WRITEABLE_ERROR 変数」 562 ページ
- 「COMMON_MSG_RETRIEVE_ERROR 変数」 563 ページ
- 「COMMON_MSG_STORE_ERROR 変数」 563 ページ
- 「COMMON_MSG_STORE_NOT_INITIALIZED 変数」 563 ページ

COMMON_ALREADY_OPEN_ERROR 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_ALREADY_OPEN_ERROR**

備考

QAManager はすでにオープンされています。

参照

QAManager インタフェース

COMMON_GETQUEUEDEPTH_ERROR 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_GETQUEUEDEPTH_ERROR**

備考

キューの深さを取得中にエラーが発生しました。

参照

getQueueDepth メソッド

COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG 変数

構文

final int
ianywhere.qanywhere.client.QAException.COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG

備考

フィルタが ALL の場合は、指定された送信先で QAManagerBase.getQueueDepth を使用できません。

参照

getQueueDepth メソッド

COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID 変数

構文

final int
ianywhere.qanywhere.client.QAException.COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID

備考

メッセージ・ストア ID が設定されていない場合は、QAManagerBase.getQueueDepth を使用できません。

参照

getQueueDepth メソッド

COMMON_GET_INIT_FILE_ERROR 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_GET_INIT_FILE_ERROR**

備考

クライアント・プロパティ・ファイルにアクセスできません。

COMMON_INIT_ERROR 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_INIT_ERROR**

備考

初期化エラーです。

COMMON_INIT_THREAD_ERROR 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_INIT_THREAD_ERROR**

備考

バックグラウンド・スレッドを初期化するときにエラーが発生しました。

COMMON_INVALID_PROPERTY 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_INVALID_PROPERTY**

備考

クライアント・プロパティ・ファイル内に無効なプロパティが存在します。

COMMON_MSG_ACKNOWLEDGE_ERROR 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_MSG_ACKNOWLEDGE_ERROR**

備考

メッセージの受信確認中にエラーが発生しました。

COMMON_MSG_CANCEL_ERROR 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_MSG_CANCEL_ERROR**

備考

メッセージのキャンセル中にエラーが発生しました。

COMMON_MSG_CANCEL_ERROR_SENT 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_MSG_CANCEL_ERROR_SENT**

備考

メッセージのキャンセル中にエラーが発生しました。

送信済みのメッセージはキャンセルできません。

COMMON_MSG_NOT_WRITEABLE_ERROR 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_MSG_NOT_WRITEABLE_ERROR**

備考

読み込み専用モードのメッセージには書き込みできません。

COMMON_MSG_RETRIEVE_ERROR 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_MSG_RETRIEVE_ERROR**

備考

クライアント・メッセージ・ストアからメッセージを取得するときにエラーが発生しました。

COMMON_MSG_STORE_ERROR 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_MSG_STORE_ERROR**

備考

クライアント・メッセージ・ストアにメッセージを格納するときにエラーが発生しました。

COMMON_MSG_STORE_NOT_INITIALIZED 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_MSG_STORE_NOT_INITIALIZED**

備考

メッセージ・ストアがメッセージング用に初期化されていません。

COMMON_MSG_STORE_TOO_LARGE 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_MSG_STORE_TOO_LARGE**

備考

メッセージ・ストアが大き過ぎて、デバイス上のディスクの空き領域に収まりません。

COMMON_NOT_OPEN_ERROR 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_NOT_OPEN_ERROR**

備考

QAManager がオープンされていません。

参照

QAManager インタフェース

COMMON_NO_DEST_ERROR 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_NO_DEST_ERROR**

備考

送信先が指定されていません。

COMMON_NO_IMPLEMENTATION 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_NO_IMPLEMENTATION**

備考

メソッドが実装されていません。

COMMON_OPEN_ERROR 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_OPEN_ERROR**

備考

メッセージ・ストアへの接続をオープンするときにエラーが発生しました。

COMMON_OPEN_LOG_FILE_ERROR 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_OPEN_LOG_FILE_ERROR**

備考

ログ・ファイルをオープンするときにエラーが発生しました。

COMMON_SELECTOR_SYNTAX_ERROR 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_SELECTOR_SYNTAX_ERROR**

備考

指定されたセレクタに構文エラーがあります。

COMMON_TERMINATE_ERROR 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_TERMINATE_ERROR**

備考

終了エラーです。

COMMON_UNEXPECTED_EOM_ERROR 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_UNEXPECTED_EOM_ERROR**

備考

予期しないところでメッセージの終わりに到達しました。

COMMON_UNREPRESENTABLE_TIMESTAMP 変数

構文

final int **ianywhere.qanywhere.client.QAException.COMMON_UNREPRESENTABLE_TIMESTAMP**

備考

タイムスタンプが許容範囲外です。

QAException メソッド

構文

ianywhere.qanywhere.client.QAException.QAException(
String *message*,
int *errorCode*
)

パラメータ

- **message** 例外のテキスト記述
- **errorCode** エラー・コード

備考

指定されたエラー・コードとエラー・メッセージ・テキストを使用して、QAException インスタンスを作成します。

QA_NO_ERROR 変数

構文

final int **ianywhere.qanywhere.client.QAException.QA_NO_ERROR**

備考

エラーはありません。

getErrorCode メソッド

構文

int **ianywhere.qanywhere.client.QAException.getErrorCode()**

備考

最後に発生した例外のエラー・コードを返します。

戻り値

最後に発生した例外のエラー・コード

QAManager インタフェース

構文

public **ianywhere.qanywhere.client.QAManager**

基本クラス

- 「QAManagerBase インタフェース」 570 ページ

備考

QAManager は、非トランザクション志向の QAnywhere メッセージング操作を管理します。

これは QAManagerBase から派生します。

この動作の完全な説明については、QAManagerBase インタフェースを参照してください。

QAManager インスタンスは、AcknowledgementMode クラスの定義のようにして、明示的に受信を確認するように設定することも、暗黙的に受信を確認するように設定することもできます。トランザクションの一部としてメッセージの受信を確認するには、QATransactionalManager を使用します。

QAManager と QATransactionalManager オブジェクトを作成するには、QAManagerFactory クラスを使用します。

参照

AcknowledgementMode インタフェース

QAManagerFactory クラス

QATransactionalManager インタフェース

メンバ

ianywhere.qanywhere.client.QAManager のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「acknowledge メソッド」 568 ページ
- 「acknowledgeAll メソッド」 568 ページ

- 「start メソッド」 600 ページ
- 「stop メソッド」 601 ページ
- 「triggerSendReceive メソッド」 601 ページ

acknowledge メソッド

構文

void **ianywhere.qanywhere.client.QAManager.acknowledge(**
  QAMessage *msg*
)
throws **QAException**

パラメータ

- **msg**　受信確認するメッセージ

スロー

- メッセージの受信確認で問題が発生した場合にスローされます。

備考

クライアント・アプリケーションが QAnywhere メッセージを正常に受信したことを確認します。

注意 : QAMessage が受信確認されると、そのステータス・プロパティが StatusCodes.RECEIVED に変わり、デフォルトの削除ルールを使用して削除できるようになります。

参照

RECEIVED 変数

acknowledgeUntil メソッド

acknowledgeAll メソッド

acknowledgeAll メソッド

構文

void **ianywhere.qanywhere.client.QAManager.acknowledgeAll()**
throws **QAException**

スロー

- メッセージの受信確認で問題が発生した場合にスローされます。

備考

クライアント・アプリケーションが QAnywhere メッセージを正常に受信したことを確認します。

受信確認されていないメッセージが、すべて受信確認されます。

注意 : QAMessage が確認されると、そのステータス・プロパティが .StatusCodes.RECEIVED に変わり、デフォルトの削除ルールを使用して削除できるようになります。

参照

RECEIVED 変数

acknowledge メソッド

acknowledgeUntil メソッド

acknowledgeUntil メソッド

構文

void **ianywhere.qanywhere.client.QAManager.acknowledgeUntil(**
  QAMessage *msg*
)
throws **QAException**

パラメータ

- **msg**　受信確認するメッセージのうち最新のもの。それよりも以前の受信確認されていないメッセージも、すべて受信確認されます。

スロー

- メッセージの受信確認で問題が発生した場合にスローされます。

備考

指定された QAMessage インスタンスと、指定されたメッセージよりも前に受信されて受信確認されていないメッセージについて、すべて受信確認します。

注意 : QAMessage が確認されると、そのステータス・プロパティが .StatusCodes.RECEIVED に変わり、デフォルトの削除ルールを使用して削除できるようになります。

参照

QAMessage インタフェース

RECEIVED 変数

acknowledge メソッド

acknowledgeAll メソッド

open メソッド

構文

void **ianywhere.qanywhere.client.QAManager.open(**
  short *mode*

)
throws **QAException**

パラメータ

- **mode**　受信確認モード。AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT または AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT のどちらかです。

スロー

- QAManager インスタンスのオープンで問題がある場合にスローされます。

備考

指定された AcknowledgementMode 値を使用して QAManager をオープンします。

open(short) メソッドは、QAManager を作成した後、最初に呼び出す必要があるメソッドです。

参照

AcknowledgementMode インタフェース

EXPLICIT_ACKNOWLEDGEMENT 変数

IMPLICIT_ACKNOWLEDGEMENT 変数

recover メソッド

構文

void **ianywhere.qanywhere.client.QAManager.recover()**
throws **QAException**

スロー

- メッセージのステータスを戻すときに問題が発生した場合にスローされます。

備考

受信確認されていないメッセージを、すべて強制的に未受信に戻します。

これらのメッセージは、QAManagerBase.getMessage(String) を使用して再受信します。

参照

getMessage メソッド

QAManagerBase インタフェース

構文

public **ianywhere.qanywhere.client.QAManagerBase**

派生クラス

- 「QAManager インタフェース」 566 ページ
- 「QATransactionalManager インタフェース」 632 ページ

備考

このクラスは、QATransactionalManager と QAManager の基本クラスです。前者の派生クラスはトランザクション志向のメッセージングを、後者の派生クラスは非トランザクション志向のメッセージングを管理します。

QAManagerBase インスタンスがメッセージを受信できるようにするには、QAManagerBase.start() メソッドを使用します。特定の QAManagerBase インスタンスを使用できるのは、そのインスタンスを生成したスレッドだけです。

このクラスのインスタンスを使用して、QAnywhere メッセージの作成と管理を行うことができます。適切な QAMessage インスタンスを作成するには、QAManagerBase.createBinaryMessage() メソッドと QAManagerBase.createTextMessage() メソッドを使用します。QAMessage インスタンスには、メッセージの内容とプロパティを設定するための、さまざまなメソッドがあります。QAnywhere メッセージを送信するには、QAManagerBase.putMessage(String, QAMessage) メソッドを使用して、アドレス指定されたメッセージをローカルのメッセージ・ストア・キューに登録します。メッセージは、転送ポリシーに基づいて QAnywhere Agent によって転送されるか、QAManagerBase.triggerSendReceive() が呼び出されたときに転送されます。

QAManagerBase にも、メッセージ・ストア・プロパティを設定および取得するためのメソッドがあります。

参照

QATransactionalManager インタフェース

QAManager インタフェース

メンバ

ianywhere.qanywhere.client.QAManagerBase のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「browseMessages メソッド」 572 ページ
- 「browseMessagesByID メソッド」 573 ページ
- 「browseMessagesByQueue メソッド」 574 ページ
- 「browseMessagesBySelector メソッド」 574 ページ
- 「cancelMessage メソッド」 575 ページ
- 「close メソッド」 576 ページ
- 「createBinaryMessage メソッド」 576 ページ
- 「createTextMessage メソッド」 576 ページ
- 「getBooleanStoreProperty メソッド」 577 ページ
- 「getByteStoreProperty メソッド」 578 ページ
- 「getDoubleStoreProperty メソッド」 578 ページ
- 「getFloatStoreProperty メソッド」 579 ページ
- 「getIntStoreProperty メソッド」 580 ページ
- 「getLongStoreProperty メソッド」 580 ページ

browseMessages メソッド

構文

java.util.Enumeration **ianywhere.qanywhere.client.QAManagerBase.browseMessages()**
throws **QAException**

スロー

- メッセージの参照で問題が発生した場合にスローされます。

備考

メッセージ・ストア内にある、参照可能なメッセージをすべて参照します。

メッセージは単に参照されるだけなので、受信確認はできません。

メッセージを受信して受信確認できるようにする場合は、QAManagerBase.getMessage(String) メソッドを使用します。

参照

browseMessagesByQueue メソッド

browseMessagesByID メソッド

getMessage メソッド

戻り値

参照可能な一連のメッセージの列挙子

browseMessagesByID メソッド

構文

```
java.util.Enumeration ianywhere.qanywhere.client.QAManagerBase.browseMessagesByID(
  String id
)
throws QAException
```

パラメータ

- **id**　メッセージのメッセージ ID

スロー

- メッセージの参照で問題が発生した場合にスローされます。

備考

指定されたメッセージ ID を持つメッセージを参照します。

メッセージは単に参照されるだけなので、受信確認はできません。メッセージを受信して受信確認できるようにする場合は、QAManagerBase.getMessage(String) メソッドを使用します。

参照

browseMessagesByQueue メソッド

browseMessages メソッド

getMessage メソッド

戻り値

0 または 1 のメッセージを含む列挙子

browseMessagesByQueue メソッド

構文

java.util.Enumeration **ianywhere.qanywhere.client.QAManagerBase.browseMessagesByQueue(**
String *address*
)
throws **QAException**

パラメータ

- **address**　メッセージのアドレス

スロー

- メッセージの参照で問題が発生した場合にスローされます。

備考

指定されたアドレスに送信され、取得可能な一連の受信待機中メッセージを参照します。

メッセージは単に参照されるだけなので、受信確認はできません。

メッセージを受信して受信確認できるようにする場合は、QAManagerBase.getMessage(String) メソッドを使用します。

参照

browseMessagesByID メソッド

browseMessages メソッド

getMessage メソッド

戻り値

参照可能な一連のメッセージの列挙子

browseMessagesBySelector メソッド

構文

java.util.Enumeration **ianywhere.qanywhere.client.QAManagerBase.browseMessagesBySelector(**
String *selector*
)
throws **QAException**

パラメータ

- **selector**　セレクタ

スロー

- メッセージの参照で問題が発生した場合にスローされます。

備考

メッセージ・ストアのキューに登録されているメッセージのうち、指定されたセレクタを満たすメッセージを参照します。

メッセージは単に参照されるだけなので、受信確認はできません。メッセージを受信して受信確認できるようにする場合は、QAManagerBase.getMessage(String) を使用します。

参照

browseMessagesByQueue メソッド

browseMessages メソッド

browseMessagesByID メソッド

getMessage メソッド

戻り値

参照可能な一連のメッセージの列挙子

cancelMessage メソッド

構文

```
boolean ianywhere.qanywhere.client.QAManagerBase.cancelMessage(
  String id
)
throws QAException
```

パラメータ

- **id**　キャンセルするメッセージのメッセージ ID

スロー

- メッセージのキャンセルで問題が発生した場合にスローされます。

備考

指定されたメッセージ ID を持つメッセージをキャンセルします。

メッセージが転送される前にメッセージをキャンセル済みの状態にします。

QAnywhere Agent のデフォルトの削除ルールにより、キャンセル済みメッセージは最終的にメッセージ・ストアから削除されます。

メッセージがすでに最終ステータスになっている場合や、中央のメッセージング・サーバに転送済みである場合は、メッセージをキャンセルできません。

close メソッド

構文

void **ianywhere.qanywhere.client.QAManagerBase.close()**
throws **QAException**

スロー

- QAManagerBase インスタンスのクローズで問題がある場合にスローされます。

備考

QAnywhere メッセージ・システムへの接続をクローズして、QAManagerBase で使用していたリソースをすべて解放します。

いったんクローズした後は、何度この close() メソッドを呼び出しても無視されます。このメソッドを呼び出して接続をクローズした後に close() 以外の QAManagerBase メソッドを呼び出すと、QAException になります。この場合は、新しい QAManagerBase インスタンスを作成してからオープンします。

createBinaryMessage メソッド

構文

QABinaryMessage **ianywhere.qanywhere.client.QAManagerBase.createBinaryMessage()**
throws **QAException**

スロー

- メッセージの作成で問題が発生した場合にスローされます。

備考

QABinaryMessage オブジェクトを作成します。

QABinaryMessage オブジェクトは、未解釈のバイト・ストリームのメッセージ本文が含まれるメッセージの送信に使用します。

参照

QABinaryMessage インタフェース

戻り値

新しい QABinaryMessage インスタンス

createTextMessage メソッド

構文

QATextMessage **ianywhere.qanywhere.client.QAManagerBase.createTextMessage()**
throws **QAException**

スロー

- メッセージの作成で問題が発生した場合にスローされます。

備考

QATextMessage オブジェクトを作成します。

QATextMessage のオブジェクトは、文字列のメッセージ本文が含まれるメッセージを送信する場合に使用します。

参照

QATextMessage インタフェース

戻り値

新しい QATextMessage インスタンス

getBooleanStoreProperty メソッド

構文

```
boolean ianywhere.qanywhere.client.QAManagerBase.getBooleanStoreProperty(
  String name
)
throws QAException
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名

スロー

- プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの boolean 値を取得します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties インタフェースを参照してください。

参照

MessageStoreProperties インタフェース

戻り値

boolean 型のプロパティの値

getByteStoreProperty メソッド

構文

byte **ianywhere.qanywhere.client.QAManagerBase.getByteStoreProperty(**
String *name*
)
throws **QAException**

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名

スロー

- プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの signed byte 値を取得します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties インタフェースを参照してください。

参照

MessageStoreProperties インタフェース

戻り値

signed byte 型のプロパティの値

getDoubleStoreProperty メソッド

構文

double **ianywhere.qanywhere.client.QAManagerBase.getDoubleStoreProperty(**
String *name*
)
throws **QAException**

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名

スロー

- プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの double 値を取得します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties インタフェースを参照してください。

参照

MessageStoreProperties インタフェース

戻り値

double 型のプロパティの値

getFloatStoreProperty メソッド

構文

```
float ianywhere.qanywhere.client.QAManagerBase.getFloatStoreProperty(
  String name
)
throws QAException
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名

スロー

- プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの float 値を取得します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties インタフェースを参照してください。

参照

MessageStoreProperties インタフェース

戻り値

float 型のプロパティの値

getIntStoreProperty メソッド

構文

int **ianywhere.qanywhere.client.QAManagerBase.getIntStoreProperty(**
String *name*
)
throws **QAException**

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名

スロー

- プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの int 値を取得します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties インタフェースを参照してください。

参照

MessageStoreProperties インタフェース

戻り値

int 型のプロパティの値

getLongStoreProperty メソッド

構文

long **ianywhere.qanywhere.client.QAManagerBase.getLongStoreProperty(**
String *name*
)
throws **QAException**

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名

スロー

- プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの long 値を取得します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties インタフェースを参照してください。

参照

MessageStoreProperties インタフェース

戻り値

long 型のプロパティの値

getMessage メソッド

構文

```
QAMessage ianywhere.qanywhere.client.QAManagerBase.getMessage(
  String address
)
throws QAException
```

パラメータ

- **address**　このアドレスは、メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。

スロー

- メッセージの取得で問題が発生した場合にスローされます。

備考

指定されたアドレスに送信され、次に取得可能な QAMessage を返します。

address パラメータは、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージが存在しない場合、新しいメッセージが着信するまでブロックされます。メッセージを同期に受信する場合は、このメソッドを使用します。

参照

QAMessage インタフェース

戻り値

該当する次の QAMessage。メッセージが存在しない場合は null。

getMessageBySelector メソッド

構文

QAMessage **ianywhere.qanywhere.client.QAManagerBase.getMessageBySelector(**
  String *address*,
  String *selector*
)
throws **QAException**

パラメータ

- **address**　このアドレスは、メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。
- **selector**　セレクタ

スロー

- メッセージの取得で問題が発生した場合にスローされます。

備考

指定されたアドレスに送信され、かつ指定されたセレクタを満たす、次に取得可能な QAMessage を返します。

address パラメータは、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージが存在しない場合、新しいメッセージが着信するまでブロックされます。

メッセージを同期に受信する場合は、このメソッドを使用します。

参照

QAMessage インタフェース

戻り値

該当する次の QAMessage。メッセージが存在しない場合は null。

getMessageBySelectorNoWait メソッド

構文

QAMessage **ianywhere.qanywhere.client.QAManagerBase.getMessageBySelectorNoWait(**
  String *address*,
  String *selector*
)
throws **QAException**

パラメータ

- **address**　このアドレスは、メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。

- **selector**　セレクタ

スロー

- メッセージの取得で問題が発生した場合にスローされます。

備考

指定されたアドレスに送信され、かつ指定されたセレクタを満たす、次に取得可能な QAMessage を返します。

address パラメータは、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージがない場合は、このメソッドがすぐに返されます。

メッセージを同期に受信する場合は、このメソッドを使用します。

参照

QAMessage インタフェース

戻り値

次の取得可能な QAMessage。メッセージが存在しない場合は null。

getMessageBySelectorTimeout メソッド

構文

```
QAMessage ianywhere.qanywhere.client.QAManagerBase.getMessageBySelectorTimeout(
  String address,
  String selector,
  long timeout
)
throws QAException
```

パラメータ

- **address**　このアドレスは、メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。
- **selector**　セレクタ
- **timeout**　メッセージ着信を待機する時間 (ミリ秒単位)

スロー

- メッセージの取得で問題が発生した場合にスローされます。

備考

指定されたアドレスに送信され、かつ指定されたセレクタを満たす、次に取得可能な QAMessage を返します。

address パラメータは、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージがない場合、このメソッドは指定されたタイムアウト時間だけ待機してから返されます。

メッセージを同期に受信する場合は、このメソッドを使用します。

参照

QAMessage インタフェース

戻り値

次の取得可能な QAMessage。メッセージが存在しない場合は null。

getMessageListener メソッド

構文

```
QAMessageListener ianywhere.qanywhere.client.QAManagerBase.getMessageListener(
  String address
)
throws QAException
```

パラメータ

- **address**　メッセージの受信で使用されるローカルのキュー名、またはシステム

スロー

- リスナの取得で問題が発生した場合にスローされます。

備考

指定されたキューに関連付けられている QAMessageListener を返します。

指定されたキューに関連付けられている QAMessageListener がない場合は null を返します。

参照

QAMessageListener インタフェース

戻り値

リスナ

getMessageListener2 メソッド

構文

```
QAMessageListener2 ianywhere.qanywhere.client.QAManagerBase.getMessageListener2(
  String address
)
throws QAException
```

パラメータ

- **address**　メッセージの受信で使用されるローカルのキュー名、またはシステム

スロー

- リスナの取得で問題が発生した場合にスローされます。

備考

指定されたキューに関連付けられている QAMessageListener2 を返します。

指定されたキューに関連付けられている QAMessageListener2 がない場合は null を返します。

参照

QAMessageListener2 インタフェース

戻り値

リスナ

getMessageNoWait メソッド

構文

```
QAMessage ianywhere.qanywhere.client.QAManagerBase.getMessageNoWait(
  String address
)
throws QAException
```

パラメータ

- **address**　このアドレスは、メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。

スロー

- メッセージの取得で問題が発生した場合にスローされます。

備考

指定されたアドレスに送信され、次に取得可能な QAMessage を返します。

address パラメータは、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージがない場合は、このメソッドがすぐに返されます。

メッセージを同期に受信する場合は、このメソッドを使用します。

参照

QAMessage インタフェース

戻り値

次の取得可能な QAMessage。該当するメッセージが存在しない場合は null。

getMessageTimeout メソッド

構文

```
QAMessage ianywhere.qanywhere.client.QAManagerBase.getMessageTimeout(
  String address,
  long timeout
)
throws QAException
```

パラメータ

- **address**　このアドレスは、メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。
- **timeout**　メッセージ着信を待機する時間 (ミリ秒単位)

スロー

- メッセージの取得で問題が発生した場合にスローされます。

備考

指定されたアドレスに送信され、次に取得可能な QAMessage を返します。

address パラメータは、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージがない場合、このメソッドは指定されたタイムアウト時間だけ待機してから返されます。メッセージを同期に受信する場合は、このメソッドを使用します。

参照

QAMessage インタフェース

戻り値

該当する次の QAMessage。メッセージが存在しない場合は null。

getMode メソッド

構文

```
short ianywhere.qanywhere.client.QAManagerBase.getMode()
throws QAException
```

スロー

- QAManager 受信確認モードの取得で問題が発生した場合にスローされます。

備考

受信したメッセージの QAManager 受信確認モードを返します。

戻り値のリストについては、AcknowledgementMode インタフェースを参照してください。

AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT と AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT は QAManager インスタンスで使用されます。AcknowledgementMode.TRANSACTIONAL は QATransactionalManager インスタンス用のモードです。

参照

EXPLICIT_ACKNOWLEDGEMENT 変数

IMPLICIT_ACKNOWLEDGEMENT 変数

QAManager インタフェース

QATransactionalManager インタフェース

戻り値

受信メッセージの QAManager 受信確認モード

getQueueDepth メソッド

構文

```
int ianywhere.qanywhere.client.QAManagerBase.getQueueDepth(
  short filter
)
throws QAException
```

パラメータ

- **filter**　着信メッセージ、送信メッセージ、またはすべてのメッセージを示すフィルタ

スロー

- エラーが発生した場合にスローされます。

備考

指定されたフィルタに基づいて、すべてのキューの深さの合計を返します。

キューの深さは、受信されていないメッセージの数です (たとえば、QAManagerBase.getMessage (String) メソッドを使用)。

使用可能なフィルタの値のリストについては、QueueDepthFilter インタフェースを参照してください。

参照

getMessage メソッド

戻り値

すべてのキューに登録されているメッセージのうち、指定されたフィルタを満たすメッセージの数

getQueueDepth メソッド

構文

```
int ianywhere.qanywhere.client.QAManagerBase.getQueueDepth(
  String queue,
  short filter
)
throws QAException
```

パラメータ

- **queue**　着信メッセージ、送信メッセージ、またはすべてのメッセージを示すフィルタ
- **filter**　キュー名

スロー

- エラーが発生した場合にスローされます。

備考

指定されたフィルタに基づいて、キューの深さを返します。

キューの深さは、受信されていないメッセージの数です (たとえば、QAManagerBase.getMessage (String) メソッドを使用)。

使用可能なフィルタの値のリストについては、QueueDepthFilter インタフェースを参照してください。

戻り値

メッセージ数

getShortStoreProperty メソッド

構文

```
short ianywhere.qanywhere.client.QAManagerBase.getShortStoreProperty(
  String name
)
throws QAException
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名

スロー

- プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの short 値を取得します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties インタフェースを参照してください。

参照

MessageStoreProperties インタフェース

戻り値

short 型のプロパティの値

getStoreProperty メソッド

構文

```
Object ianywhere.qanywhere.client.QAManagerBase.getStoreProperty(
  String name
)
throws QAException
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名

スロー

- プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

メッセージ・ストア・プロパティを表すオブジェクトを取得します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties インタフェースを参照してください。

参照

MessageStoreProperties インタフェース

戻り値

プロパティの値

getStorePropertyNames メソッド

構文

java.util.Enumeration **ianywhere.qanywhere.client.QAManagerBase.getStorePropertyNames()**
throws **QAException**

スロー

- 列挙子の取得で問題が発生した場合にスローされます。

備考

メッセージ・ストアのプロパティ名の列挙子を取得します。

戻り値

メッセージ・ストアのプロパティ名の列挙子

getStringStoreProperty メソッド

構文

String **ianywhere.qanywhere.client.QAManagerBase.getStringStoreProperty(**
String *name*
)
throws **QAException**

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名

スロー

- 文字列値の取得で問題が発生した場合にスローされます。

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティの文字列値を取得します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties インタフェースを参照してください。

参照

MessageStoreProperties インタフェース

戻り値

文字列型のプロパティ値。該当するプロパティが存在しない場合は null。

putMessage メソッド

構文

```
void ianywhere.qanywhere.client.QAManagerBase.putMessage(
  String address,
  QAMessage msg
)
throws QAException
```

パラメータ

- **address**　送信先のキュー名を指定するメッセージのアドレス
- **msg**　転送用ローカル・メッセージ・ストアに格納するメッセージ

スロー

- メッセージの登録で問題が発生した場合にスローされます。

備考

別の QAnywhere クライアントに送信するメッセージを準備します。

このメソッドは、メッセージと送信先アドレスをローカル・メッセージ・ストアに挿入します。メッセージが転送されるタイミングは、QAnywhere Agent の転送ポリシーで決まります。

アドレスは 'id¥queue-name' の形式で指定します。'id' は送信先メッセージ・ストアの ID、'queue-name' は送信先の QAnywhere クライアントがメッセージの受信で使用するキューを特定します。

参照

putMessageTimeToLive メソッド

putMessageTimeToLive メソッド

構文

```
void ianywhere.qanywhere.client.QAManagerBase.putMessageTimeToLive(
  String address,
  QAMessage msg,
  long ttl
)
throws QAException
```

パラメータ

- **address**　送信先のキュー名を指定するメッセージのアドレス
- **msg**　登録するメッセージ

- **ttl**　メッセージの有効期限 (ミリ秒単位)。この期限を過ぎても配信されなかったメッセージは期限切れになります。値 0 は、有効期限なしを意味します。

スロー

- メッセージの登録で問題が発生した場合にスローされます。

備考

別の QAnywhere クライアントに送信するメッセージを準備します。

このメソッドは、メッセージと送信先アドレスをローカル・メッセージ・ストアに挿入します。メッセージが転送されるタイミングは、QAnywhere Agent の転送ポリシーで決まります。ただし、次のメッセージの転送時間が指定された存続時間を超えると、そのメッセージは期限切れになります。

アドレスは 'id¥queue-name' の形式で指定します。'id' は送信先メッセージ・ストアの ID、'queue-name' は送信先の QAnywhere クライアントがメッセージの受信で使用するキューを特定します。

setBooleanStoreProperty メソッド

構文

```
void ianywhere.qanywhere.client.QAManagerBase.setBooleanStoreProperty(
  String name,
  boolean value
)
throws QAException
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　boolean 型のプロパティの値

スロー

- メッセージ・ストア・プロパティの設定で問題が発生した場合にスローされます。

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを boolean 値に設定します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties インタフェースを参照してください。

参照

MessageStoreProperties インタフェース

setByteStoreProperty メソッド

構文

```
void ianywhere.qanywhere.client.QAManagerBase.setByteStoreProperty(
  String name,
  byte value
)
throws QAException
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　sbyte 型のプロパティの値

スロー

- メッセージ・ストア・プロパティの設定で問題が発生した場合にスローされます。

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを sbyte 値に設定します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties インタフェースを参照してください。

参照

MessageStoreProperties インタフェース

setDoubleStoreProperty メソッド

構文

```
void ianywhere.qanywhere.client.QAManagerBase.setDoubleStoreProperty(
  String name,
  double value
)
throws QAException
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　double 型のプロパティの値

スロー

- メッセージ・ストア・プロパティの設定で問題が発生した場合にスローされます。

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを double 値に設定します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties インタフェースを参照してください。

参照

MessageStoreProperties インタフェース

setFloatStoreProperty メソッド

構文

```
void ianywhere.qanywhere.client.QAManagerBase.setFloatStoreProperty(
  String name,
  float value
)
throws QAException
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　float 型のプロパティの値

スロー

- メッセージ・ストア・プロパティの設定で問題が発生した場合にスローされます。

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを float 値に設定します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。事前定義済みプロパティのリストについては、MessageStoreProperties インタフェースを参照してください。

参照

MessageStoreProperties インタフェース

setIntStoreProperty メソッド

構文

```
void ianywhere.qanywhere.client.QAManagerBase.setIntStoreProperty(
  String name,
  int value
)
throws QAException
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　int 型のプロパティの値

スロー

- メッセージ・ストア・プロパティの設定で問題が発生した場合にスローされます。

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを int 値に設定します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。事前定義済みプロパティのリストについては、MessageStoreProperties インタフェースを参照してください。

参照

MessageStoreProperties インタフェース

setLongStoreProperty メソッド

構文

```
void ianywhere.qanywhere.client.QAManagerBase.setLongStoreProperty(
  String name,
  long value
)
throws QAException
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　long 型のプロパティの値

スロー

- メッセージ・ストア・プロパティの設定で問題が発生した場合にスローされます。

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを long 値に設定します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties インタフェースを参照してください。

参照

MessageStoreProperties インタフェース

setMessageListener メソッド

構文

void **ianywhere.qanywhere.client.QAManagerBase.setMessageListener(**
String *address*,
QAMessageListener *listener*
)
throws **QAException**

パラメータ

- **address**　メッセージの受信で使用されるローカルのキュー名のアドレス、または QAnywhere システム・メッセージを受信するシステムのアドレス
- **listener**　リスナ

スロー

- QAMessageListener オブジェクトの登録で問題が発生した場合にスローされます。

備考

QAMessageListener オブジェクトを登録して QAnywhere メッセージを非同期に受信します。

address パラメータは、メッセージの受信で使用されるローカル・キュー名を指定します。指定されたキューには、1 つのリスナ・オブジェクトだけを割り当てることができます。Push 通知とネットワーク・ステータス変更も含め、QAnywhere システム・メッセージを受信したい場合は、キュー名に "system" を指定します。

メッセージを非同期に受信する場合は、このメソッドを使用します。

参照

QAMessageListener インタフェース

setMessageListener2 メソッド

構文

void **ianywhere.qanywhere.client.QAManagerBase.setMessageListener2(**
String *address*,
QAMessageListener2 *listener*
)
throws **QAException**

パラメータ

- **address**　メッセージの受信で使用されるローカルのキュー名のアドレス、または QAnywhere システム・メッセージを受信するシステムのアドレス
- **listener**　リスナ

スロー

- QAMessageListener2 オブジェクトの登録で問題が発生した場合にスローされます。

備考

QAMessageListener2 オブジェクトを登録して QAnywhere メッセージを非同期に受信します。

address パラメータは、メッセージの受信で使用されるローカル・キュー名を指定します。指定されたキューには、1 つのリスナ・オブジェクトだけを割り当てることができます。Push 通知とネットワーク・ステータス変更も含め、QAnywhere システム・メッセージを受信したい場合は、キュー名に "system" を指定します。

メッセージを非同期に受信する場合は、このメソッドを使用します。

参照

QAMessageListener2 インタフェース

setMessageListenerBySelector メソッド

構文

```
void ianywhere.qanywhere.client.QAManagerBase.setMessageListenerBySelector(
  String address,
  String selector,
  QAMessageListener listener
)
throws QAException
```

パラメータ

- **address**　メッセージの受信で使用されるローカルのキュー名のアドレス、または QAnywhere システム・メッセージを受信するシステムのアドレス
- **selector**　受信されるメッセージをフィルタリングするために使用するセレクタ
- **listener**　リスナ

スロー

- 指定されたキューにリスナ・オブジェクトがすでに割り当てられてるなどの理由により、QAMessageListener オブジェクトの登録で問題が発生した場合にスローされます。

備考

メッセージ・セレクタを指定し、QAMessageListener オブジェクトを登録して QAnywhere メッセージを非同期に受信します。

address パラメータは、メッセージの受信で使用されるローカル・キュー名を指定します。指定されたキューには、1 つのリスナ・オブジェクトだけを割り当てることができます。selector パラメータは、指定されたアドレスで受信されるメッセージをフィルタリングするために使用するセレクタを指定します。Push 通知とネットワーク・ステータス変更も含め、QAnywhere システム・メッセージを受信したい場合は、キュー名に "system" を指定します。

メッセージを非同期に受信する場合は、このメソッドを使用します。

setMessageListenerBySelector2 メソッド

構文

void **ianywhere.qanywhere.client.QAManagerBase.setMessageListenerBySelector2(**
  String *address*,
  String *selector*,
  QAMessageListener2 *listener*
)
throws **QAException**

パラメータ

- **address**　メッセージの受信で使用されるローカルのキュー名のアドレス、または QAnywhere システム・メッセージを受信するシステムのアドレス
- **selector**　受信されるメッセージをフィルタリングするために使用するセレクタ
- **listener**　リスナ

スロー

- QAMessageListener2 オブジェクトの登録で問題が発生した場合にスローされます。

備考

メッセージ・セレクタを指定し、QAMessageListener2 オブジェクトを登録して QAnywhere メッセージを非同期に受信します。

address パラメータは、メッセージの受信で使用されるローカル・キュー名を指定します。指定されたキューには、1 つのリスナ・オブジェクトだけを割り当てることができます。selector パラメータは、指定されたアドレスで受信されるメッセージをフィルタリングするために使用するセレクタを指定します。Push 通知とネットワーク・ステータス変更も含め、QAnywhere システム・メッセージを受信したい場合は、キュー名に "system" を指定します。

メッセージを非同期に受信する場合は、このメソッドを使用します。

参照

QAMessageListener2 インタフェース

setShortStoreProperty メソッド

構文

void **ianywhere.qanywhere.client.QAManagerBase.setShortStoreProperty(**
  String *name*,
  short *value*
)
throws **QAException**

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名

- **value**　short 型のプロパティの値

スロー

- メッセージ・ストア・プロパティの設定で問題が発生した場合にスローされます。

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを short 値に設定します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties インタフェースを参照してください。

参照

MessageStoreProperties インタフェース

setStoreProperty メソッド

構文

```
void ianywhere.qanywhere.client.QAManagerBase.setStoreProperty(
  String name,
  Object value
)
throws QAException
```

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　プロパティの値

スロー

- メッセージ・ストア・プロパティを value パラメータで指定した型の値に設定するときに問題が発生した場合にスローされます。

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを System.Object 値に設定します。

プロパティ型は、使用可能ないずれかのプリミティブ型、または文字列型でなければなりません。このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties インタフェースを参照してください。

参照

MessageStoreProperties インタフェース

setStringStoreProperty メソッド

構文

void **ianywhere.qanywhere.client.QAManagerBase.setStringStoreProperty(**
  String *name*,
  String *value*
)
throws **QAException**

パラメータ

- **name**　事前定義済みまたはカスタムのプロパティ名
- **value**　文字列型のプロパティの値

スロー

- メッセージ・ストア・プロパティを文字列値に設定するときに問題が発生した場合にスローされます。

備考

事前定義済みまたはカスタムのメッセージ・ストア・プロパティを文字列値に設定します。

このメソッドを使用して、事前定義済みまたはユーザ定義のクライアント・ストア・プロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties インタフェースを参照してください。

参照

MessageStoreProperties インタフェース

start メソッド

構文

void **ianywhere.qanywhere.client.QAManagerBase.start()**
throws **QAException**

スロー

- QAManagerBase インスタンスの起動で問題がある場合にスローされます。

備考

着信メッセージを受信するための QAManagerBase を開始します。

このメソッドを繰り返し呼び出そうとしても、間に QAManagerBase.stop() 呼び出しを挟まないかぎり 2 回目以降の呼び出しは無視されます。

参照

stop メソッド

stop メソッド

構文

void **ianywhere.qanywhere.client.QAManagerBase.stop()**
throws **QAException**

スロー

- QAManagerBase インスタンスの停止で問題がある場合にスローされます。

備考

QAManagerBase による着信メッセージの受信を停止します。

メッセージは失われません。メッセージは、Manager が再起動されるまで受信されません。この stop() メソッドを繰り返し呼び出そうとしても、間に QAManagerBase.start() 呼び出しを挟まないかぎり 2 回目以降の呼び出しは無視されます。

参照

start メソッド

triggerSendReceive メソッド

構文

void **ianywhere.qanywhere.client.QAManagerBase.triggerSendReceive()**
throws **QAException**

スロー

- 送信／受信のトリガで問題が発生した場合にスローされます。

備考

QAnywhere メッセージ・サーバとの同期処理を発生させて、他のクライアント宛てのメッセージをアップロードし、ローカル・クライアント宛てのメッセージをダウンロードします。

このメソッドを呼び出すと、QAnywhere Agent と中央のメッセージング・サーバ間でメッセージがただちに同期されます。手動の triggerSendReceive() 呼び出しでは、QAnywhere Agent の転送ポリシーとは無関係に、メッセージ転送がただちに行われます。

QAnywhere Agent の転送ポリシーは、メッセージ転送の方法を決定します。たとえば、クライアントが Push 通知を受信した場合や、QAManagerBase.putMessage() メソッドを呼び出してメッセージを送信した場合に、一定の間隔でメッセージ転送が自動的に実行されます。

参照

putMessage メソッド

QAManagerFactory クラス

構文

public **ianywhere.qanywhere.client.QAManagerFactory**

備考

このクラスは、QATransactionalManager オブジェクトまたは QAManager オブジェクトを作成するためのファクトリ・クラスです。

QAManagerFactory のインスタンスは 1 つだけ持つことができます。

参照

QAManager インタフェース

QATransactionalManager インタフェース

メンバ

ianywhere.qanywhere.client.QAManagerFactory のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「createQAManager メソッド」 602 ページ
- 「createQAManager メソッド」 603 ページ
- 「createQAManager メソッド」 604 ページ
- 「createQATransactionalManager メソッド」 604 ページ
- 「createQATransactionalManager メソッド」 605 ページ
- 「createQATransactionalManager メソッド」 605 ページ
- 「getInstance メソッド」 606 ページ

createQAManager メソッド

構文

abstract QAManager **ianywhere.qanywhere.client.QAManagerFactory.createQAManager(**
  String *iniFile*
)
throws **QAException**

パラメータ

- **iniFile**　QAManager のインスタンスを設定するためのプロパティ・ファイル。QAManager のインスタンスをデフォルトの設定で作成する場合は null。

スロー

- Manager の作成で問題が発生した場合にスローされます。

備考

指定されたプロパティを持つ新しい QAManager インスタンスを返します。

iniFile パラメータが null の場合、QAManager はデフォルトのプロパティで作成されます。インスタンスの作成後に QAManagerBase の各プロパティ設定メソッドを使用して、QAManager プロパティをプログラムで設定できます。

参照

QAManager インタフェース

戻り値

新しい QAManager インスタンス

createQAManager メソッド

構文

```
abstract QAManager ianywhere.qanywhere.client.QAManagerFactory.createQAManager(
  java.util.Hashtable properties
)
throws QAException
```

パラメータ

- **properties**　QAManager のインスタンスを設定するためのハッシュテーブル

スロー

- Manager の作成で問題が発生した場合にスローされます。

備考

指定されたプロパティをハッシュテーブルとして持つ新しい QAManager インスタンスを返します。

参照

QAManager インタフェース

戻り値

新しい QAManager インスタンス

createQAManager メソッド

構文

abstract QAManager **ianywhere.qanywhere.client.QAManagerFactory.createQAManager()**
throws **QAException**

スロー

- Manager の作成で問題が発生した場合にスローされます。

備考

デフォルトのプロパティを持つ新しい QAManager インスタンスを返します。

参照

QAManager インタフェース

戻り値

新しい QAManager インスタンス

createQATransactionalManager メソッド

構文

abstract QATransactionalManager
ianywhere.qanywhere.client.QAManagerFactory.createQATransactionalManager(
String *iniFile*
)
throws **QAException**

パラメータ

- **iniFile** QATransactionalManager のインスタンスを設定するためのプロパティ・ファイル

スロー

- Manager の作成で問題が発生した場合にスローされます。

備考

指定されたプロパティを持つ新しい QATransactionalManager インスタンスを返します。

iniFile パラメータが null の場合、QATransactionalManager はデフォルトのプロパティで作成されます。インスタンスの作成後に QAManagerBase の各プロパティ設定メソッドを使用して、QATransactionalManager のプロパティをプログラムで設定できます。

参照

QATransactionalManager インタフェース

戻り値

設定された QATransactionalManager

createQATransactionalManager メソッド

構文

abstract QATransactionalManager
ianywhere.qanywhere.client.QAManagerFactory.createQATransactionalManager(
java.util.Hashtable *properties*
)
throws **QAException**

パラメータ

- **properties** QATransactionalManager のインスタンスを設定するためのハッシュテーブル

スロー

- Manager の作成で問題が発生した場合にスローされます。

備考

指定されたプロパティを持つ新しい QATransactionalManager インスタンスを返します。

参照

QATransactionalManager インタフェース

戻り値

設定された QATransactionalManager

createQATransactionalManager メソッド

構文

abstract QATransactionalManager
ianywhere.qanywhere.client.QAManagerFactory.createQATransactionalManager()
throws **QAException**

スロー

- Manager の作成で問題が発生した場合にスローされます。

備考

デフォルトのプロパティを持つ新しい QATransactionalManager インスタンスを返します。

参照

QATransactionalManager インタフェース

戻り値

新しい QATransactionalManager

getInstance メソッド

構文

QAManagerFactory **ianywhere.qanywhere.client.QAManagerFactory.getInstance()**
throws **QAException**

スロー

- マネージャ・ファクトリの作成で問題が発生した場合にスローされます。

備考

QAManagerFactory のシングルトン・インスタンスを返します。

戻り値

QAManagerFactory のシングルトン・インスタンス

QAMessage インタフェース

構文

public **ianywhere.qanywhere.client.QAMessage**

派生クラス

- 「QABinaryMessage インタフェース」 544 ページ
- 「QATextMessage インタフェース」 627 ページ

備考

QAMessage には、メッセージ・プロパティとヘッダ・ファイルを設定するためのインタフェースがあります。

派生クラスの QABinaryMessage と QATextMessage には、メッセージ本文の読み込み／書き込みを行うための特別な関数があります。事前定義済みまたはカスタムのメッセージ・プロパティを設定するには、QAMessage の関数を使用します。

事前定義済みプロパティ名のリストについては、MessageProperties インタフェースを参照してください。

参照

QABinaryMessage インタフェース

QATextMessage インタフェース

メンバ

ianywhere.qanywhere.client.QAMessage のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「clearProperties メソッド」 608 ページ

DEFAULT_PRIORITY 変数

構文

final int **ianywhere.qanywhere.client.QAMessage.DEFAULT_PRIORITY**

備考

デフォルトのメッセージ優先度です。

DEFAULT_TIME_TO_LIVE 変数

構文

final long **ianywhere.qanywhere.client.QAMessage.DEFAULT_TIME_TO_LIVE**

備考

デフォルトの存続時間です。

clearProperties メソッド

構文

void **ianywhere.qanywhere.client.QAMessage.clearProperties()**
throws **QAException**

スロー

- メッセージ・プロパティのクリアで問題が発生した場合にスローされます。

備考

メッセージのすべてのプロパティをクリアします。

getAddress メソッド

構文

String **ianywhere.qanywhere.client.QAMessage.getAddress()**
throws **QAException**

スロー

- 送信先アドレスの取得で問題が発生した場合にスローされます。

備考

QAMessage インスタンスの送信先アドレスを返します。

このフィールドは、メッセージの送信時には無視されます。送信操作が完了すると、このフィールドには QAManagerBase.putMessage(String, QAMessage) で指定された送信先アドレスが入ります。

戻り値

QAMessage インスタンスの送信先アドレス

getBooleanProperty メソッド

構文

boolean **ianywhere.qanywhere.client.QAMessage.getBooleanProperty(**
  String *name*
)
throws **QAException**

パラメータ

- **name**　プロパティ名

スロー

- プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

boolean 型のメッセージ・プロパティを取得します。

参照

MessageProperties インタフェース

戻り値

プロパティの値

getByteProperty メソッド

構文

byte **ianywhere.qanywhere.client.QAMessage.getByteProperty(**
  String *name*
)
throws **QAException**

パラメータ

- **name**　プロパティ名

スロー

- プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

signed byte 型のメッセージ・プロパティを取得します。

参照

MessageProperties インタフェース

戻り値

プロパティの値

getDoubleProperty メソッド

構文

double **ianywhere.qanywhere.client.QAMessage.getDoubleProperty(**
  String *name*
)
throws **QAException**

パラメータ

- **name**　プロパティ名

スロー

- プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

double 型のメッセージ・プロパティを取得します。

参照

MessageProperties インタフェース

戻り値

プロパティの値

getExpiration メソッド

構文

java.util.Date **ianywhere.qanywhere.client.QAMessage.getExpiration()**
throws **QAException**

スロー

- 有効期限の取得で問題が発生した場合にスローされます。

備考

メッセージの有効期限の値を返します。メッセージの有効期限がない場合や、まだ送信されていない場合は、null になります。

メッセージの送信時には、有効期限は空のままです。送信操作が終了すると、メッセージの有効期限が入ります。

メッセージの有効期限は、QAManagerBase.putMessageTimeToLive(String, QAMessage, long) の存続時間の引数を現在の時間に追加して設定されるため、このプロパティは読み込み専用です。

参照

putMessageTimeToLive メソッド

戻り値

メッセージの有効期限の値。メッセージの有効期限がない場合や、まだ送信されていない場合は null。

getFloatProperty メソッド

構文

float **ianywhere.qanywhere.client.QAMessage.getFloatProperty(**
String *name*
)
throws **QAException**

パラメータ

- **name**　プロパティ名

スロー

- プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

float 型のメッセージ・プロパティを取得します。

参照

MessageProperties インタフェース

戻り値

プロパティの値

getInReplyToID メソッド

構文

String **ianywhere.qanywhere.client.QAMessage.getInReplyToID()**
throws **QAException**

スロー

- このメッセージの返信先メッセージのメッセージ ID の取得で問題が発生した場合にスローされます。

備考

このメッセージの返信先メッセージのメッセージ ID を返します。

戻り値

このメッセージの返信先メッセージのメッセージ ID。このメッセージが返信しない場合は null。

getIntProperty メソッド

構文

int **ianywhere.qanywhere.client.QAMessage.getIntProperty(**
  String *name*
)
throws **QAException**

パラメータ

- **name**　プロパティ名

スロー

- プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

int 型のメッセージ・プロパティを取得します。

参照

MessageProperties インタフェース

戻り値

プロパティの値

getLongProperty メソッド

構文

long **ianywhere.qanywhere.client.QAMessage.getLongProperty(**
  String *name*
)
throws **QAException**

パラメータ

- **name**　プロパティ名

スロー

- プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

long 型のメッセージ・プロパティを取得します。

参照

MessageProperties インタフェース

戻り値

プロパティの値

getMessageID メソッド

構文

String **ianywhere.qanywhere.client.QAMessage.getMessageID()**
throws **QAException**

スロー

- メッセージ ID の取得で問題が発生した場合にスローされます。

備考

メッセージのグローバルにユニークなメッセージ ID を返します。

このプロパティは、メッセージがキューに登録されるまで null のままです。

QAManagerBase.putMessage(String, QAMessage) を使用してメッセージが送信されると、メッセージ ID が null になるため無視できます。send メソッドが戻ると、割り当てられた値がここに格納されます。

参照

putMessage メソッド

戻り値

メッセージのメッセージ ID。メッセージが登録されていない場合は null。

getPriority メソッド

構文

int **ianywhere.qanywhere.client.QAMessage.getPriority()**
throws **QAException**

スロー

- メッセージの優先度の取得で問題が発生した場合にスローされます。

備考

メッセージの優先度 (0 から 9) を返します。

戻り値

メッセージの優先度

getProperty メソッド

構文

Object **ianywhere.qanywhere.client.QAMessage.getProperty(**
  String *name*
)
throws **QAException**

パラメータ

- **name**　プロパティ名

スロー

- プロパティ値の取得で変換エラーが発生した場合にスローされます。

備考

メッセージのプロパティを取得します。

戻り値

プロパティの値。該当するプロパティが存在しない場合は null。

getPropertyNames メソッド

構文

java.util.Enumeration **ianywhere.qanywhere.client.QAMessage.getPropertyNames()**
throws **QAException**

スロー

- メッセージのプロパティ名の列挙子を取得するときに問題が発生した場合にスローされます。

備考

メッセージのプロパティ名の列挙子を取得します。

戻り値

メッセージのプロパティ名の列挙子

getPropertyType メソッド

構文

short **ianywhere.qanywhere.client.QAMessage.getPropertyType(**
  String *name*
)
throws **QAException**

パラメータ

- **name**　プロパティ名

スロー

- プロパティ型の取得で問題が発生した場合にスローされます。

備考

指定されたプロパティのプロパティ型を返します。

参照

PropertyType インタフェース

戻り値

プロパティ型

getRedelivered メソッド

構文

boolean **ianywhere.qanywhere.client.QAMessage.getRedelivered()**
throws **QAException**

スロー

- 再配信ステータスの取得で問題が発生した場合にスローされます。

備考

受信されたが受信確認されていないメッセージであるかどうかを示します。

受信側の QAManager は、受信中のメッセージが以前に受信されたことを検知した場合に、Redelivered プロパティを設定します。

たとえば、AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT でオープンされた QAManager を使用してアプリケーションがメッセージを受信し、メッセージの受信確認を行わずに終了したとします。このアプリケーションが再起動されて同じメッセージを受信すると、メッセージに再配信のマークが付けられます。

戻り値

受信されたが受信確認されていないメッセージである場合は True

getReplyToAddress メソッド

構文

String **ianywhere.qanywhere.client.QAMessage.getReplyToAddress()**
throws **QAException**

スロー

- 返信アドレスの取得で問題が発生した場合にスローされます。

備考

メッセージの返信アドレスを設定します。

戻り値

このメッセージの返信アドレス。返信アドレスがない場合は null。

getShortProperty メソッド

構文

short **ianywhere.qanywhere.client.QAMessage.getShortProperty(**
  String *name*
)
throws **QAException**

パラメータ

- **name**　プロパティ名

スロー

- プロパティ値の取得で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

short 型のメッセージ・プロパティを取得します。

参照

MessageProperties インタフェース

戻り値

プロパティの値

getStringProperty メソッド

構文

String **ianywhere.qanywhere.client.QAMessage.getStringProperty(**
  String *name*
)
throws **QAException**

パラメータ

- **name**　プロパティ名

スロー

- メッセージ・プロパティの取得で問題が発生した場合にスローされます。

備考

文字列型のメッセージ・プロパティを取得します。

参照

MessageProperties インタフェース

戻り値

プロパティの値。該当するプロパティが存在しない場合は null。

getTimestamp メソッド

構文

java.util.Date **ianywhere.qanywhere.client.QAMessage.getTimestamp()**
throws **QAException**

スロー

- メッセージのタイムスタンプの取得で問題が発生した場合にスローされます。

備考

メッセージの作成時刻を示すタイムスタンプを返します。

戻り値

メッセージのタイムスタンプ。

propertyExists メソッド

構文

boolean **ianywhere.qanywhere.client.QAMessage.propertyExists(**
String *name*
)
throws **QAException**

パラメータ

- **name**　プロパティ名

スロー

- プロパティが設定されているかどうかの確認で問題が発生した場合にスローされます。

備考

指定されたプロパティがこのメッセージに設定されているかどうかを示します。

戻り値

プロパティが存在する場合は True

setAddress メソッド

構文

```
void ianywhere.qanywhere.client.QAMessage.setAddress(
  String dest
)
throws QAException
```

パラメータ

- **dest**　送信先アドレス

スロー

- メッセージの送信先アドレスの設定で問題が発生した場合にスローされます。

備考

メッセージの送信先アドレスを設定します。

setBooleanProperty メソッド

構文

```
void ianywhere.qanywhere.client.QAMessage.setBooleanProperty(
  String name,
  boolean value
)
throws QAException
```

パラメータ

- **name**　プロパティ名
- **value**　プロパティの値

スロー

- プロパティの設定で問題が発生した場合にスローされます。

備考

boolean 型のプロパティを設定します。

参照

MessageProperties インタフェース

setByteProperty メソッド

構文

void **ianywhere.qanywhere.client.QAMessage.setByteProperty(**
String *name*,
byte *value*
)
throws **QAException**

パラメータ

- **name** プロパティ名
- **value** プロパティの値

スロー

- プロパティの設定で問題が発生した場合にスローされます。

備考

signed byte 型のプロパティを設定します。

参照

MessageProperties インタフェース

setDoubleProperty メソッド

構文

void **ianywhere.qanywhere.client.QAMessage.setDoubleProperty(**
String *name*,
double *value*
)
throws **QAException**

パラメータ

- **name** プロパティ名
- **value** プロパティの値

スロー

- プロパティの設定で問題が発生した場合にスローされます。

備考

double 型のプロパティを設定します。

参照

MessageProperties インタフェース

setFloatProperty メソッド

構文

void **ianywhere.qanywhere.client.QAMessage.setFloatProperty(**
  String *name*,
  float *value*
)
throws **QAException**

パラメータ

- **name**　プロパティ名
- **value**　プロパティの値

スロー

- プロパティの設定で問題が発生した場合にスローされます。

備考

float 型のプロパティを設定します。

参照

MessageProperties インタフェース

setInReplyToID メソッド

構文

void **ianywhere.qanywhere.client.QAMessage.setInReplyToID(**
  String *id*
)
throws **QAException**

パラメータ

- **id**　このメッセージの返信先メッセージの ID

スロー

- 返信先 ID の設定で問題が発生した場合にスローされます。

備考

このメッセージの返信対象メッセージを特定する、返信先 ID を設定します。

setIntProperty メソッド

構文

```
void ianywhere.qanywhere.client.QAMessage.setIntProperty(
  String name,
  int value
)
throws QAException
```

パラメータ

- **name**　プロパティ名
- **value**　プロパティの値

スロー

- プロパティの設定で問題が発生した場合にスローされます。

備考

int 型のプロパティを設定します。

参照

MessageProperties インタフェース

setLongProperty メソッド

構文

```
void ianywhere.qanywhere.client.QAMessage.setLongProperty(
  String name,
  long value
)
throws QAException
```

パラメータ

- **name**　プロパティ名
- **value**　プロパティの値

スロー

- プロパティの設定で問題が発生した場合にスローされます。

備考

long 型のプロパティを設定します。

参照

MessageProperties インタフェース

setPriority メソッド

構文

void **ianywhere.qanywhere.client.QAMessage.setPriority(**
  int *priority*
)
throws **QAException**

パラメータ

- **priority**　メッセージの優先度

スロー

- 優先度の設定で問題が発生した場合にスローされます。

備考

メッセージの優先度 (0 から 9) を設定します。

setProperty メソッド

構文

void **ianywhere.qanywhere.client.QAMessage.setProperty(**
  String *name*,
  Object *value*
)
throws **QAException**

パラメータ

- **name**　プロパティ名
- **value**　プロパティの値

スロー

- プロパティの設定で問題が発生した場合にスローされます。

備考

プロパティを設定します。

プロパティ型は、使用可能ないずれかのプリミティブ型、または文字列型でなければなりません。

参照

MessageProperties インタフェース

setReplyToAddress メソッド

構文

void **ianywhere.qanywhere.client.QAMessage.setReplyToAddress(**
String *address*
)
throws **QAException**

パラメータ

- **address**　返信先アドレス

スロー

- 返信アドレスの設定で問題が発生した場合にスローされます。

備考

返信先アドレスを設定します。

setShortProperty メソッド

構文

void **ianywhere.qanywhere.client.QAMessage.setShortProperty(**
String *name*,
short *value*
)
throws **QAException**

パラメータ

- **name**　プロパティ名
- **value**　プロパティの値

スロー

- プロパティの設定で問題が発生した場合にスローされます。

備考

short 型のプロパティを設定します。

参照

MessageProperties インタフェース

setStringProperty メソッド

構文

void **ianywhere.qanywhere.client.QAMessage.setStringProperty(**
  String *name*,
  String *value*
)
throws **QAException**

パラメータ

- **name**　プロパティ名
- **value**　プロパティの値

スロー

- プロパティの設定で問題が発生した場合にスローされます。

備考

文字列型のプロパティを設定します。

参照

MessageProperties インタフェース

QAMessageListener インタフェース

構文

public **ianywhere.qanywhere.client.QAMessageListener**

備考

メッセージを受信するには、QAMangerBase.setMessageListener(String,QAMessageListener) を呼び出して、このインタフェースを実装し、実装を登録します。

参照

setMessageListener メソッド

メンバ

ianywhere.qanywhere.client.QAMessageListener のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「onException メソッド」 625 ページ
- 「onMessage メソッド」 625 ページ

onException メソッド

構文

void **ianywhere.qanywhere.client.QAMessageListener.onException(**
QAException *exception*,
QAMessage *message*
)

パラメータ

- **exception**　発生した例外
- **message**　メッセージが onMessage(QAMessage) に渡された後で例外が発生した場合は、処理されたメッセージ。それ以外の場合は null。

備考

このメソッドは、メッセージの受信中に例外が発生すると呼び出されます。

このメソッドを使用して QAManagerBase インスタンスを自動的に終了することはできません。すべてのメッセージ・リスナの処理が終了するまでは、QAManagerBase.close() メソッドでブロックされるためです。

参照

QAManagerBase インタフェース

close メソッド

onMessage メソッド

構文

void **ianywhere.qanywhere.client.QAMessageListener.onMessage(**
QAMessage *message*
)

パラメータ

- **message**　受信したメッセージ

備考

このメソッドは、メッセージが受信されると呼び出されます。

QAMessageListener2 インタフェース

構文

public **ianywhere.qanywhere.client.QAMessageListener2**

備考

メッセージを受信するには、QAMangerBase を呼び出して、このインタフェースを実装し、実装を登録します。

setMessageListener2(String,QAMessageListener2)

参照

setMessageListener2 メソッド

メンバ

ianywhere.qanywhere.client.QAMessageListener2 のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「onException メソッド」 626 ページ
- 「onMessage メソッド」 627 ページ

onException メソッド

構文

```
void ianywhere.qanywhere.client.QAMessageListener2.onException(
  QAManagerBase mgr,
  QAException exception,
  QAMessage message
)
```

パラメータ

- **mgr**　メッセージを処理した QAManagerBase
- **exception**　発生した例外
- **message**　メッセージが onMessage(QAMessage) に渡された後で例外が発生した場合は、処理されたメッセージ。それ以外の場合は null。

備考

このメソッドは、メッセージの受信中に例外が発生すると呼び出されます。

このメソッドを使用して QAManagerBase インスタンスを自動的に終了することはできません。すべてのメッセージ・リスナの処理が終了するまでは、QAManagerBase.close() メソッドでブロックされるためです。

参照

QAManagerBase インタフェース

close メソッド

onMessage(QAMessage)

onMessage メソッド

構文

void **ianywhere.qanywhere.client.QAMessageListener2.onMessage(**
QAManagerBase *mgr*,
QAMessage *message*
)

パラメータ

- **mgr**　メッセージを受信した QAManagerBase
- **message**　受信したメッセージ

備考

このメソッドは、メッセージが受信されると呼び出されます。

参照

QAManagerBase インタフェース

QATextMessage インタフェース

構文

public **ianywhere.qanywhere.client.QATextMessage**

基本クラス

- 「QAMessage インタフェース」 606 ページ

備考

QATextMessage は QAMessage クラスを継承したもので、メッセージ本文にテキストが追加されます。また、メッセージ本文からのテキストの読み込み／書き込みを行うためのメソッドがあります。

メッセージが最初に作成された時点では、メッセージ本文は書き込み専用モードになっています。メッセージ送信元のクライアントは、メッセージを送信した後、そのメッセージを保持し変更できます。ただし、それによって、送信されたメッセージが変更されることはありません。同じメッセージ・オブジェクトを複数回送信できます。

メッセージが受信されるとプロバイダが QATextMessage.reset() を呼び出します。これによりメッセージ本文が読み込み専用モードになり、メッセージ本文の先頭から値の読み込みが開始されます。

参照

QAmessage

メンバ

ianywhere.qanywhere.client.QATextMessage のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「writeText メソッド」 632 ページ

getText メソッド

構文

String **ianywhere.qanywhere.client.QATextMessage.getText()**
throws **QAException**

スロー

- メッセージ・テキストの取得で問題が発生した場合にスローされます。

備考

メッセージ・テキストを返します。

メッセージ・テキストのサイズが、QAManager.MAX_IN_MEMORY_MESSAGE_SIZE プロパティで指定された最大サイズを超える場合、このメソッドは null を返します。この場合は、QATextMessage.readText(int) メソッドを使用してテキストを読み込みます。

参照

readText メソッド

戻り値

メッセージ・テキストまたは null

getTextLength メソッド

構文

long **ianywhere.qanywhere.client.QATextMessage.getTextLength()**
throws **QAException**

スロー

- メッセージの長さの取得で問題が発生した場合にスローされます。

備考

メッセージの文字数を返します。

戻り値

メッセージの文字数

readText メソッド

構文

String **ianywhere.qanywhere.client.QATextMessage.readText(**
  int *maxLength*
)
throws **QAException**

パラメータ

- **maxLength**　読み込まれる最大文字数

スロー

- 未読テキストの取得で問題が発生した場合にスローされます。

備考

メッセージの未読テキストを返します。

未読テキストが追加された場合、そのテキストを読み込むにはこのメソッドを繰り返し呼び出す必要があります。読み込みは、未読テキストの先頭から実行されます。

戻り値

テキスト

reset メソッド

構文

void **ianywhere.qanywhere.client.QATextMessage.reset()**
throws **QAException**

スロー

- メッセージのテキスト位置のリセットで問題が発生した場合にスローされます。

備考

メッセージのテキスト位置を先頭にリセットします。

setText メソッド

構文

void **ianywhere.qanywhere.client.QATextMessage.setText(**
  String *value*
)
throws **QAException**

パラメータ

- **value**　メッセージ本文に書き込むテキスト

スロー

- メッセージ・テキストの上書きで問題が発生した場合にスローされます。

備考

メッセージ・テキストを上書きします。

writeText メソッド

構文

```
void ianywhere.qanywhere.client.QATextMessage.writeText(
  String value
)
throws QAException
```

パラメータ

- **value**　追加するテキスト

スロー

- メッセージ・テキストの追加で問題が発生した場合にスローされます。

備考

メッセージ・テキストの末尾にテキストを追加します。

writeText メソッド

構文

```
void ianywhere.qanywhere.client.QATextMessage.writeText(
  String value,
  int length
)
throws QAException
```

パラメータ

- **value**　追加するテキスト
- **length**　追加するテキストの文字数

スロー

- メッセージ・テキストの追加で問題が発生した場合にスローされます。

備考

メッセージ・テキストの末尾にテキストを追加します。

writeText メソッド

構文

void **ianywhere.qanywhere.client.QATextMessage.writeText(**
  String *value*,
  int *offset*,
  int *length*
)
throws **QAException**

パラメータ

- **value**　追加するテキスト
- **offset**　追加するテキスト値内でのオフセット
- **length**　追加するテキストの文字数

スロー

- メッセージ・テキストの追加で問題が発生した場合にスローされます。

備考

メッセージ・テキストの末尾にテキストを追加します。

QATransactionalManager インタフェース

構文

public **ianywhere.qanywhere.client.QATransactionalManager**

基本クラス

- 「QAManagerBase インタフェース」 570 ページ

備考

QATransactionalManager クラスは QAManagerBase から派生し、トランザクション志向の QAnywhere メッセージング操作を管理します。

この動作の完全な説明については、QAManagerBase インタフェースを参照してください。

QATransactionalManager インスタンスは、トランザクション志向の受信確認でのみ使用できます。QAManagerBase.putMessage(String, QAMessage) と QAManagerBase.getMessage(String) のすべての呼び出しをコミットする場合は、QATransactionalManager.commit() メソッドを使用します。

参照

commit メソッド

putMessage メソッド

getMessage メソッド

メンバ

ianywhere.qanywhere.client.QATransactionalManager のすべてのメンバ。継承されるすべてのメンバも含まれます。

commit メソッド

構文

void **ianywhere.qanywhere.client.QATransactionalManager.commit()**
throws **QAException**

スロー

- メッセージのコミットで問題が発生した場合にスローされます。

備考

現在のトランザクションをコミットし、新しいトランザクションを開始します。

このメソッドは、QAManagerBase.putMessage(String, QAMessage) と QAManagerBase.getMessage (String) のすべての呼び出しをコミットします。

注意 : 最初のトランザクションは、QATransactionalManager.open() の呼び出しで開始されます。

open メソッド

構文

void **ianywhere.qanywhere.client.QATransactionalManager.open()**
throws **QAException**

スロー

- Manager のオープンで問題が発生した場合にスローされます。

備考

QATransactionalManager インスタンスをオープンします。

これは、Manager を作成した後、最初に呼び出す必要があるメソッドです。

rollback メソッド

構文

void **ianywhere.qanywhere.client.QATransactionalManager.rollback()**
throws **QAException**

スロー

- ロールバックで問題が発生した場合にスローされます。

備考

現在のトランザクションをロールバックし、新しいトランザクションを開始します。

このメソッドは、QAManagerBase.putMessage(String, QAMessage) と QAManagerBase.getMessage (String) のすべてのコミットされていない呼び出しをロールバックします。

QueueDepthFilter インタフェース

構文

public **ianywhere.qanywhere.client.QueueDepthFilter**

備考

QAManagerBase.getQueueDepth(short) と QAManagerBase.getQueueDepth(String, short) のキューの深さのフィルタ値を提供します。

参照

getQueueDepth メソッド

getQueueDepth メソッド

メンバ

ianywhere.qanywhere.client.QueueDepthFilter のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「ALL 変数」 635 ページ
- 「INCOMING 変数」 636 ページ
- 「OUTGOING 変数」 636 ページ

ALL 変数

構文

final short **ianywhere.qanywhere.client.QueueDepthFilter.ALL**

備考

このフィルタは、着信メッセージと送信メッセージの両方を指定します。

システム・メッセージと期限切れのメッセージは、キューの深さカウントに計上されません。

INCOMING 変数

構文

final short **ianywhere.qanywhere.client.QueueDepthFilter.INCOMING**

備考

このフィルタは、着信メッセージのみ指定します。

着信メッセージは、発信者がメッセージ・ストアのエージェント ID と異なるメッセージです。

OUTGOING 変数

構文

final short **ianywhere.qanywhere.client.QueueDepthFilter.OUTGOING**

備考

このフィルタは、送信メッセージのみ指定します。

送信メッセージは、発信者がメッセージ・ストアのエージェント ID であり、送信先がメッセージ・ストアのエージェント ID ではないメッセージです。

StatusCodes インタフェース

構文

public **ianywhere.qanywhere.client.StatusCodes**

備考

このインタフェースは、メッセージのステータス・コード・セットを定義します。

メンバ

ianywhere.qanywhere.client.StatusCodes のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「CANCELLED 変数」 637 ページ
- 「EXPIRED 変数」 637 ページ
- 「FINAL 変数」 637 ページ
- 「LOCAL 変数」 638 ページ
- 「PENDING 変数」 638 ページ
- 「RECEIVED 変数」 638 ページ
- 「RECEIVING 変数」 638 ページ
- 「TRANSMITTED 変数」 639 ページ
- 「TRANSMITTING 変数」 639 ページ
- 「UNRECEIVABLE 変数」 639 ページ
- 「UNTRANSMITTED 変数」 640 ページ

CANCELLED 変数

構文

final int **ianywhere.qanywhere.client.StatusCodes.CANCELLED**

備考

メッセージはキャンセル済みです。

このコードは MessageProperties.STATUS で使用されます。

参照

STATUS 変数

EXPIRED 変数

構文

final int **ianywhere.qanywhere.client.StatusCodes.EXPIRED**

備考

メッセージの有効期限が切れました。メッセージは、有効期限になる前に受信されませんでした。

このコードは MessageProperties.STATUS で使用されます。

参照

STATUS 変数

FINAL 変数

構文

final int **ianywhere.qanywhere.client.StatusCodes.FINAL**

備考

メッセージは最終ステータスになりました。

このコードは MessageProperties.STATUS で使用されます。

参照

STATUS 変数

LOCAL 変数

構文

final int **ianywhere.qanywhere.client.StatusCodes.LOCAL**

備考

メッセージはローカル・メッセージ・ストア宛てに送信され、サーバに送信されません。

このコードは MessageProperties.TRANSMISSION_STATUS で使用されます。

参照

TRANSMISSION_STATUS 変数

PENDING 変数

構文

final int **ianywhere.qanywhere.client.StatusCodes.PENDING**

備考

メッセージは送信されたが、まだ受信されていません。

このコードは MessageProperties.STATUS で使用されます。

参照

STATUS 変数

RECEIVED 変数

構文

final int **ianywhere.qanywhere.client.StatusCodes.RECEIVED**

備考

メッセージが受信され、受信者によって受信確認されました。

このコードは MessageProperties.STATUS で使用されます。

参照

STATUS 変数

RECEIVING 変数

構文

final int **ianywhere.qanywhere.client.StatusCodes.RECEIVING**

備考

メッセージは受信中であるか、または受信されたがまだ確認されていません。

このコードは MessageProperties.STATUS で使用されます。

参照

STATUS 変数

TRANSMITTED 変数

構文

final int **ianywhere.qanywhere.client.StatusCodes.TRANSMITTED**

備考

メッセージはサーバに送信されました。

このコードは MessageProperties.TRANSMISSION_STATUS で使用されます。

参照

TRANSMISSION_STATUS 変数

TRANSMITTING 変数

構文

final int **ianywhere.qanywhere.client.StatusCodes.TRANSMITTING**

備考

メッセージはサーバに送信中です。

このコードは MessageProperties.TRANSMISSION_STATUS で使用されます。

参照

TRANSMISSION_STATUS 変数

UNRECEIVABLE 変数

構文

final int **ianywhere.qanywhere.client.StatusCodes.UNRECEIVABLE**

備考

メッセージは受信不可のマークが付けられています。

メッセージに間違った形式があるか、失敗した試行回数が多すぎて配信されませんでした。

このコードは MessageProperties.STATUS で使用されます。

参照

STATUS 変数

UNTRANSMITTED 変数

構文

final int **ianywhere.qanywhere.client.StatusCodes.UNTRANSMITTED**

備考

メッセージはサーバに送信されていません。

このコードは MessageProperties.TRANSMISSION_STATUS で使用されます。

参照

TRANSMISSION_STATUS 変数

ianywhere.qanywhere.ws package

WSBase クラス

構文

public **ianywhere.qanywhere.ws.WSBase**

備考

これは、モバイル Web サービスのコンパイラで生成されるメインの Web サービス・プロキシ・クラスの基本クラスです。

メンバ

ianywhere.qanywhere.ws.WSBase のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「clearRequestProperties メソッド」 642 ページ
- 「getResult メソッド」 642 ページ
- 「getServiceID メソッド」 643 ページ
- 「setListener メソッド」 643 ページ
- 「setListener メソッド」 643 ページ
- 「setProperty メソッド」 644 ページ
- 「setQAManager メソッド」 644 ページ
- 「setRequestProperty メソッド」 645 ページ
- 「setServiceID メソッド」 645 ページ
- 「WSBase メソッド」 641 ページ
- 「WSBase メソッド」 642 ページ

WSBase メソッド

構文

ianywhere.qanywhere.ws.WSBase.WSBase(
String *iniFile*
)
throws **WSException**

パラメータ

- **iniFile**　設定プロパティが含まれているファイル

スロー

- WSBase の構築で問題がある場合にスローされます。

備考

設定プロパティ・ファイルが指定されたコンストラクタです。

有効な設定プロパティには、次のものがあります。

LOG_FILE：ランタイム情報のログを記録するファイルです。

LOG_LEVEL：ログを記録する情報の冗長レベルを制御する、0 ー 6 の値。6 が最高の冗長レベルです。

WS_CONNECTOR_ADDRESS：Mobile Link サーバにおける Web サービス・コネクタのアドレスです。デフォルトの WS_CONNECTOR_ADDRESS は "ianywhere.connector.webservices¥¥" です。

WSBase メソッド

構文

ianywhere.qanywhere.ws.WSBase.WSBase()
throws **WSException**

スロー

- WSBase の構築で問題がある場合にスローされます。

備考

コンストラクタです。

clearRequestProperties メソッド

構文

void **ianywhere.qanywhere.ws.WSBase.clearRequestProperties()**

備考

この WSBase に対して送信された要求プロパティをすべてクリアします。

getResult メソッド

構文

WSResult **ianywhere.qanywhere.ws.WSBase.getResult(**
String *requestID*
)

パラメータ

- **requestID**　Web サービス要求の ID

備考

Web サービス要求の結果を表す WSResult オブジェクトを取得します。

戻り値

Web サービス要求の結果を表す WSResult インスタンス

参照

WSStatus クラス

getServiceID メソッド

構文

String **ianywhere.qanywhere.ws.WSBase.getServiceID()**

備考

WSBase のこのインスタンスのサービス ID を取得します。

戻り値

サービス ID

setListener メソッド

構文

void **ianywhere.qanywhere.ws.WSBase.setListener(**
String *requestID*,
WSListener *listener*
)

パラメータ

- **requestID**　結果を受信する Web サービス要求の ID
- **listener**　指定された Web サービス要求が取得可能な場合に呼び出されるリスナ・オブジェクト

備考

指定された Web サービス要求の結果を受信するリスナを設定します。

一般にリスナは、サービスの asyncXYZ メソッドの結果を取得する場合に使用されます。

リスナを削除するには、リスナとして null を指定し setListener を呼び出します。

注意： このメソッドは、以前に呼び出された setListener でリスナを置き換えます。

setListener メソッド

構文

void **ianywhere.qanywhere.ws.WSBase.setListener(**
WSListener *listener*
)

パラメータ

- **listener**　Web サービス要求が取得可能な場合に呼び出されるリスナ・オブジェクト

備考

WSBase のこのインスタンスが作成した、すべての Web サービス要求を受信するリスナを設定します。

一般にリスナは、サービスの asyncXYZ メソッドの結果を取得する場合に使用されます。

リスナを削除するには、リスナとして null を指定し setListener を呼び出します。

注意： このメソッドは、以前に呼び出された setListener でリスナを置き換えます。

setProperty メソッド

構文

```
void ianywhere.qanywhere.ws.WSBase.setProperty(
  String property,
  String val
)
```

パラメータ

- **property**　設定するプロパティ名
- **val**　プロパティの値

備考

WSBase のこのインスタンスの設定プロパティを設定します。

設定プロパティは、非同期または同期の Web サービス要求を作成する前に設定してください。Web サービス要求の作成後は、このメソッドは無効になります。

有効な設定プロパティには、次のものがあります。

LOG_FILE：ランタイム情報のログを記録するファイルです。

LOG_LEVEL：ログを記録する情報の冗長レベルを制御する、0 ー 6 の値。6 が最高の冗長レベルです。

WS_CONNECTOR_ADDRESS：Mobile Link サーバにおける Web サービス・コネクタのアドレスです。デフォルトは "ianywhere.connector.webservices¥¥" です。

setQAManager メソッド

構文

```
void ianywhere.qanywhere.ws.WSBase.setQAManager(
  QAManagerBase mgr
)
```

パラメータ

- **mgr**　使用する QAManagerBase

備考

この Web サービス・クライアントが Web サービス要求を処理するために使用する QAManagerBase を設定します。

注意 : EXPLICIT_ACKNOWLEDGEMENT QAManager を使用する場合は、WSResult の acknowledge() メソッドを呼び出して、非同期の Web サービス要求の結果を受信確認できます。同期の Web サービス要求の結果は、EXPLICIT_ACKNOWLEDGEMENT QAManager を使用している場合であっても、自動的に受信確認されます。IMPLICIT_ACKNOWLEDGEMENT QAManager を使用する場合は、あらゆる Web サービス要求の結果が自動的に受信確認されます。

setRequestProperty メソッド

構文

```
void ianywhere.qanywhere.ws.WSBase.setRequestProperty(
  String name,
  Object value
)
```

パラメータ

- **name**　設定するプロパティ名
- **value**　プロパティの値

備考

この WSBase クラスが作成した Web サービス要求の要求プロパティを設定します。

要求プロパティは、この WSBase によって送信される QAMessage ごとに設定されます。この処理は、プロパティがクリアされるまで行われます。要求プロパティをクリアするには、null 値を設定します。メッセージ・プロパティのタイプは、value パラメータのクラスで決まります。たとえば、値が Integer のインスタンスである場合は、setIntProperty を使用して QAMessage のプロパティが設定されます。

setServiceID メソッド

構文

```
void ianywhere.qanywhere.ws.WSBase.setServiceID(
  String serviceID
)
```

パラメータ

- **serviceID**　サービス ID

備考

WSBase のこのインスタンスのユーザ定義 ID を設定します。

サービス ID には、WSBase のこのインスタンスにユニークな値を設定します。サービス ID は、Web サービス要求を送受信するためのキュー名を形成する場合に内部で使用されます。このため、前のセッションで作成された Web サービス要求の結果を取得できるように、サービス ID をアプリケーションのセッション間で保持する必要があります。

WSException クラス

構文

public **ianywhere.qanywhere.ws.WSException**

派生クラス

- 「WSFaultException クラス」 647 ページ

備考

このクラスは、Web サービス要求の処理中に発生した例外を表します。

メンバ

ianywhere.qanywhere.ws.WSException のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「getErrorCode メソッド」 647 ページ
- 「WSException メソッド」 646 ページ
- 「WSException メソッド」 647 ページ
- 「WSException メソッド」 647 ページ

WSException メソッド

構文

ianywhere.qanywhere.ws.WSException.WSException(
String *msg*
)

パラメータ

- **msg**　エラー・メッセージ

備考

指定されたエラー・メッセージを使用して、新しい例外を構築します。

WSException メソッド

構文

```
ianywhere.qanywhere.ws.WSException.WSException(
 String msg,
 int errorCode
)
```

パラメータ

- **msg**　エラー・メッセージ
- **errorCode**　エラー・コード

備考

指定されたエラー・メッセージとエラー・コードを使用して、新しい例外を構築します。

WSException メソッド

構文

```
ianywhere.qanywhere.ws.WSException.WSException(
 Exception exception
)
```

パラメータ

- **exception**　例外

備考

新しい例外を構築します。

getErrorCode メソッド

構文

int **ianywhere.qanywhere.ws.WSException.getErrorCode()**

備考

この例外に関連付けられているエラー・コードを取得します。

戻り値

この例外に関連付けられているエラー・コード

WSFaultException クラス

構文

public **ianywhere.qanywhere.ws.WSFaultException**

基本クラス

- 「WSException クラス」 646 ページ

備考

このクラスは、Web サービス・コネクタからの SOAP Fault 例外を表します。

メンバ

ianywhere.qanywhere.ws.WSFaultException のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「getErrorCode メソッド」 647 ページ
- 「WSException メソッド」 646 ページ
- 「WSException メソッド」 647 ページ
- 「WSException メソッド」 647 ページ
- 「WSFaultException メソッド」 648 ページ

WSFaultException メソッド

構文

```
ianywhere.qanywhere.ws.WSFaultException.WSFaultException(
  String msg
)
```

パラメータ

- **msg**　エラー・メッセージ

備考

指定されたエラー・メッセージを使用して、新しい例外を構築します。

WSListener インタフェース

構文

public **ianywhere.qanywhere.ws.WSListener**

備考

このクラスは、Web サービス要求の結果を受信するリスナを表します。

メンバ

ianywhere.qanywhere.ws.WSListener のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「onException メソッド」 649 ページ
- 「onResult メソッド」 649 ページ

onException メソッド

構文

void **ianywhere.qanywhere.ws.WSListener.onException(**
WSException *e*,
WSResult *wsResult*
)

パラメータ

- **e**　結果の処理中に発生した WSException
- **wsResult**　WSResult。これから要求 ID を取得できる場合があります。この WSResult の値は定義されていません。

備考

非同期の Web サービス要求の結果を処理中に例外が発生した場合に呼び出されます。

参照

WSException クラス

WSResult クラス

onResult メソッド

構文

void **ianywhere.qanywhere.ws.WSListener.onResult(**
WSResult *wsResult*
)

パラメータ

- **wsResult**　Web サービス要求の結果を記述する WSResult

備考

非同期の Web サービス要求の結果を指定して呼び出されます。

参照

WSResult クラス

WSResult クラス

構文

public **ianywhere.qanywhere.ws.WSResult**

備考

このクラスは、Web サービス要求の結果を表します。

- WSListener.onResult に渡されます。
- コンパイラが生成したサービス・プロキシの asyncXYZ メソッドから返されます。
- 特定の要求 ID を指定して WSBase.getResult を呼び出して取得されます。

WSResult オブジェクトは、次のいずれかの方法で取得されます。

メンバ

ianywhere.qanywhere.ws.WSResult のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「acknowledge メソッド」 651 ページ
- 「getArrayValue メソッド」 651 ページ
- 「getBigDecimalArrayValue メソッド」 652 ページ
- 「getBigDecimalValue メソッド」 652 ページ
- 「getBigIntegerArrayValue メソッド」 653 ページ
- 「getBigIntegerValue メソッド」 653 ページ
- 「getBooleanArrayValue メソッド」 654 ページ
- 「getBooleanValue メソッド」 654 ページ
- 「getByteArrayValue メソッド」 655 ページ
- 「getByteValue メソッド」 655 ページ
- 「getCharacterArrayValue メソッド」 656 ページ
- 「getCharacterValue メソッド」 656 ページ
- 「getDoubleArrayValue メソッド」 657 ページ
- 「getDoubleValue メソッド」 657 ページ
- 「getErrorMessage メソッド」 658 ページ
- 「getFloatArrayValue メソッド」 658 ページ
- 「getFloatValue メソッド」 658 ページ
- 「getIntegerArrayValue メソッド」 659 ページ
- 「getIntegerValue メソッド」 659 ページ
- 「getLongArrayValue メソッド」 660 ページ
- 「getLongValue メソッド」 660 ページ
- 「getObjectArrayValue メソッド」 661 ページ
- 「getObjectValue メソッド」 661 ページ
- 「getPrimitiveBooleanArrayValue メソッド」 662 ページ
- 「getPrimitiveBooleanValue メソッド」 662 ページ
- 「getPrimitiveByteArrayValue メソッド」 663 ページ
- 「getPrimitiveByteValue メソッド」 663 ページ
- 「getPrimitiveCharArrayValue メソッド」 664 ページ
- 「getPrimitiveCharValue メソッド」 664 ページ
- 「getPrimitiveDoubleArrayValue メソッド」 665 ページ
- 「getPrimitiveDoubleValue メソッド」 665 ページ
- 「getPrimitiveFloatArrayValue メソッド」 666 ページ
- 「getPrimitiveFloatValue メソッド」 666 ページ
- 「getPrimitiveIntArrayValue メソッド」 667 ページ

acknowledge メソッド

構文

void **ianywhere.qanywhere.ws.WSResult.acknowledge()**

備考

この WSResult が処理されたことを確認します。

このメソッドは、EXPLICIT_ACKNOWLEDGEMENT QAManager が使用されている場合にのみ有効です。

getArrayValue メソッド

構文

```
WSSerializable[] ianywhere.qanywhere.ws.WSResult.getArrayValue(
  String parentName
)
throws WSException
```

パラメータ

- **parentName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から複合型の値の配列を取得します。

戻り値

値

getBigDecimalArrayValue メソッド

構文

BigDecimal[] **ianywhere.qanywhere.ws.WSResult.getBigDecimalArrayValue(**
  String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から BigDecimal 配列値を取得します。

戻り値

値

getBigDecimalValue メソッド

構文

BigDecimal **ianywhere.qanywhere.ws.WSResult.getBigDecimalValue(**
  String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から BigDecimal 値を取得します。

戻り値

値

getBigIntegerArrayValue メソッド

構文

```
BigInteger[] ianywhere.qanywhere.ws.WSResult.getBigIntegerArrayValue(
  String elementName
)
throws WSException
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から BigInteger 配列値を取得します。

戻り値

値

getBigIntegerValue メソッド

構文

```
BigInteger ianywhere.qanywhere.ws.WSResult.getBigIntegerValue(
  String elementName
)
throws WSException
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から BigInteger 値を取得します。

戻り値

値

getBooleanArrayValue メソッド

構文

```
Boolean[] ianywhere.qanywhere.ws.WSResult.getBooleanArrayValue(
  String elementName
)
throws WSException
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から java.lang.Boolean 配列値を取得します。

戻り値

値

getBooleanValue メソッド

構文

```
Boolean ianywhere.qanywhere.ws.WSResult.getBooleanValue(
  String elementName
)
throws WSException
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から java.lang.Boolean 値を取得します。

戻り値

値

getByteArrayValue メソッド

構文

Byte[] **ianywhere.qanywhere.ws.WSResult.getByteArrayValue(**
String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から java.lang.Byte 配列値を取得します。

戻り値

値

getByteValue メソッド

構文

Byte **ianywhere.qanywhere.ws.WSResult.getByteValue(**
String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から java.lang.Byte 値を取得します。

戻り値

値

getCharacterArrayValue メソッド

構文

Character[] **ianywhere.qanywhere.ws.WSResult.getCharacterArrayValue(**
  String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から java.lang.Character 配列値を取得します。

戻り値

値

getCharacterValue メソッド

構文

Character **ianywhere.qanywhere.ws.WSResult.getCharacterValue(**
  String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から java.lang.Character 値を取得します。

戻り値

値

getDoubleArrayValue メソッド

構文

Double[] **ianywhere.qanywhere.ws.WSResult.getDoubleArrayValue(**
String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から java.lang.Double 配列値を取得します。

戻り値

値

getDoubleValue メソッド

構文

Double **ianywhere.qanywhere.ws.WSResult.getDoubleValue(**
String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から java.lang.Double 値を取得します。

戻り値

値

getErrorMessage メソッド

構文

String **ianywhere.qanywhere.ws.WSResult.getErrorMessage()**

備考

エラー・メッセージを取得します。

戻り値

エラー・メッセージ

getFloatArrayValue メソッド

構文

Float[] **ianywhere.qanywhere.ws.WSResult.getFloatArrayValue(**
  String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から java.lang.Float 配列値を取得します。

戻り値

値

getFloatValue メソッド

構文

Float **ianywhere.qanywhere.ws.WSResult.getFloatValue(**
  String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から java.lang.Float 値を取得します。

戻り値

値

getIntegerArrayValue メソッド

構文

```
Integer[] ianywhere.qanywhere.ws.WSResult.getIntegerArrayValue(
  String elementName
)
throws WSException
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から java.lang.Integer 配列値を取得します。

戻り値

値

getIntegerValue メソッド

構文

```
Integer ianywhere.qanywhere.ws.WSResult.getIntegerValue(
  String elementName
)
throws WSException
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から java.lang.Integer 値を取得します。

戻り値

値

getLongArrayValue メソッド

構文

Long[] **ianywhere.qanywhere.ws.WSResult.getLongArrayValue(**
String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から java.lang.Long 配列値を取得します。

戻り値

値

getLongValue メソッド

構文

Long **ianywhere.qanywhere.ws.WSResult.getLongValue(**
String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から java.lang.Long 値を取得します。

戻り値

値

getObjectArrayValue メソッド

構文

```
Object[] ianywhere.qanywhere.ws.WSResult.getObjectArrayValue(
  String elementName
)
throws WSException
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から複合型の値の配列を取得します。

戻り値

値

getObjectValue メソッド

構文

```
Object ianywhere.qanywhere.ws.WSResult.getObjectValue(
  String elementName
)
throws WSException
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から複合型の値を取得します。

戻り値

値

getPrimitiveBooleanArrayValue メソッド

構文

```
boolean[] ianywhere.qanywhere.ws.WSResult.getPrimitiveBooleanArrayValue(
  String elementName
)
throws WSException
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から boolean 配列値を取得します。

戻り値

値

getPrimitiveBooleanValue メソッド

構文

```
boolean ianywhere.qanywhere.ws.WSResult.getPrimitiveBooleanValue(
  String elementName
)
throws WSException
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から boolean 値を取得します。

戻り値

値

getPrimitiveByteArrayValue メソッド

構文

byte[] **ianywhere.qanywhere.ws.WSResult.getPrimitiveByteArrayValue(**
String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から byte 配列値を取得します。

戻り値

値

getPrimitiveByteValue メソッド

構文

byte **ianywhere.qanywhere.ws.WSResult.getPrimitiveByteValue(**
String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から byte 値を取得します。

戻り値

値

getPrimitiveCharArrayValue メソッド

構文

```
char[] ianywhere.qanywhere.ws.WSResult.getPrimitiveCharArrayValue(
  String elementName
)
throws WSException
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から char 配列値を取得します。

戻り値

値

getPrimitiveCharValue メソッド

構文

```
char ianywhere.qanywhere.ws.WSResult.getPrimitiveCharValue(
  String elementName
)
throws WSException
```

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から char 値を取得します。

戻り値

値

getPrimitiveDoubleArrayValue メソッド

構文

double[] **ianywhere.qanywhere.ws.WSResult.getPrimitiveDoubleArrayValue(**
String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から double 配列値を取得します。

戻り値

値

getPrimitiveDoubleValue メソッド

構文

double **ianywhere.qanywhere.ws.WSResult.getPrimitiveDoubleValue(**
String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から double 値を取得します。

戻り値

値

getPrimitiveFloatArrayValue メソッド

構文

float[] **ianywhere.qanywhere.ws.WSResult.getPrimitiveFloatArrayValue(**
  String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から float 配列値を取得します。

戻り値

値

getPrimitiveFloatValue メソッド

構文

float **ianywhere.qanywhere.ws.WSResult.getPrimitiveFloatValue(**
  String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から float 値を取得します。

戻り値

値

getPrimitiveIntArrayValue メソッド

構文

int[] **ianywhere.qanywhere.ws.WSResult.getPrimitiveIntArrayValue(**
String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から i nt 配列値を取得します。

戻り値

値

getPrimitiveIntValue メソッド

構文

int **ianywhere.qanywhere.ws.WSResult.getPrimitiveIntValue(**
String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から i nt 値を取得します。

戻り値

値

getPrimitiveLongArrayValue メソッド

構文

long[] **ianywhere.qanywhere.ws.WSResult.getPrimitiveLongArrayValue(**
  String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から long 配列値を取得します。

戻り値

値

getPrimitiveLongValue メソッド

構文

long **ianywhere.qanywhere.ws.WSResult.getPrimitiveLongValue(**
  String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から long 値を取得します。

戻り値

値

getPrimitiveShortArrayValue メソッド

構文

short[] **ianywhere.qanywhere.ws.WSResult.getPrimitiveShortArrayValue(**
String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から short 配列値を取得します。

戻り値

値

getPrimitiveShortValue メソッド

構文

short **ianywhere.qanywhere.ws.WSResult.getPrimitiveShortValue(**
String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から short 値を取得します。

戻り値

値

getRequestID メソッド

構文

String **ianywhere.qanywhere.ws.WSResult.getRequestID()**

備考

この WSResult が表す要求 ID を取得します。

要求の作成時とは異なるアプリケーションで実行される Web サービス要求に対応する WSResult を取得するために、この要求 ID が必要な場合は、アプリケーションの実行と実行の間でこの要求 ID を保持する必要があります。

戻り値

要求 ID

getShortArrayValue メソッド

構文

Short[] **ianywhere.qanywhere.ws.WSResult.getShortArrayValue(**
  String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から java.lang.Short 配列値を取得します。

戻り値

値

getShortValue メソッド

構文

Short **ianywhere.qanywhere.ws.WSResult.getShortValue(**
  String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から java.lang.Short 値を取得します。

戻り値

値

getStatus メソッド

構文

int **ianywhere.qanywhere.ws.WSResult.getStatus()**

備考

この WSResult のステータスを取得します。

戻り値

ステータス・コード

参照

WSStatus クラス

getStringArrayValue メソッド

構文

String[] **ianywhere.qanywhere.ws.WSResult.getStringArrayValue(**
  String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から String 配列値を取得します。

戻り値

値

getStringValue メソッド

構文

String **ianywhere.qanywhere.ws.WSResult.getStringValue(**
  String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から String 値を取得します。

戻り値

値

getValue メソッド

構文

Object **ianywhere.qanywhere.ws.WSResult.getValue(**
  String *elementName*
)
throws **WSException**

パラメータ

- **elementName**　この値の WSDL ドキュメント内の要素名

スロー

- 値の取得で問題が発生した場合にスローされます。

備考

この WSResult から複合型の値を取得します。

戻り値

値

WSStatus クラス

構文

public **ianywhere.qanywhere.ws.WSStatus**

備考

このクラスは、Web サービス要求のステータス・コードを定義します。

メンバ

ianywhere.qanywhere.ws.WSStatus のすべてのメンバ。継承されるすべてのメンバも含まれます。

- 「STATUS_ERROR 変数」 673 ページ
- 「STATUS_QUEUED 変数」 673 ページ
- 「STATUS_RESULT_AVAILABLE 変数」 673 ページ
- 「STATUS_SUCCESS 変数」 673 ページ

STATUS_ERROR 変数

構文

final int **ianywhere.qanywhere.ws.WSStatus.STATUS_ERROR**

備考

要求の処理でエラーがありました。

STATUS_QUEUED 変数

構文

final int **ianywhere.qanywhere.ws.WSStatus.STATUS_QUEUED**

備考

要求はサーバへの配信用キューに登録されました。

STATUS_RESULT_AVAILABLE 変数

構文

final int **ianywhere.qanywhere.ws.WSStatus.STATUS_RESULT_AVAILABLE**

備考

要求の結果を取得できます。

STATUS_SUCCESS 変数

構文

final int **ianywhere.qanywhere.ws.WSStatus.STATUS_SUCCESS**

備考

要求は正常に処理されました。

第 15 章

QAnywhere SQL API リファレンス

目次

メッセージのプロパティ、ヘッダ、内容

この項では、メッセージ・ヘッダ、メッセージの内容、メッセージ・プロパティを容易に設定できる QAnywhere SQL ストアド・プロシージャについて説明します。

メッセージ・ヘッダ

メッセージ・ヘッダ情報の取得と設定は、次のストアド・プロシージャを使用して行います。

「メッセージ・ヘッダ」 220 ページを参照してください。

ml_qa_getaddress

メッセージの QAnywhere アドレスを返します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

戻り値

VARCHAR(128) 型の QAnywhere のメッセージ・アドレス。QAnywhere のメッセージ・アドレスは、*id¥queue-name* の形式を取ります。

このヘッダは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「QAnywhere メッセージ・アドレス」 56 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ

例

次の例では、受信されたメッセージのアドレスをデータベース・コンソールに出力します。

```
begin
   declare @msgid varchar(128);
   declare @addr varchar(128);
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @addr = ml_qa_getaddress( @msgid );
   message 'message to address ' || @addr || ' received';
   commit;
end
```

ml_qa_getexpiration

メッセージの有効期限を返します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

戻り値

TIMESTAMP 型の有効期限。有効期限がない場合は NULL が返されます。

備考

ml_qa_putmessage の処理が完了した後で、意図した受信者によって指定の時間内にメッセージが受信されない場合、メッセージは期限切れになります。期限切れになったメッセージは、QAnywhere のデフォルトの削除ルールを使って削除できます。

このヘッダは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「メッセージの削除ルール」 256 ページ
- 「QAnywhere メッセージの送信」 73 ページ
- 「ml_qa_setexpiration」 682 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ

例

次の例では、受信されたメッセージの有効期限をデータベース・コンソールに出力します。

```
begin
   declare @msgid varchar(128);
   declare @expires timestamp;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @expires = ml_qa_getexpiration( @msgid );
   message 'message would have expired at ' || @expires || ' if it had not been received';
   commit;
end
```

ml_qa_getinreplytoid

メッセージの in-reply-to ID を返します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

戻り値

VARCHAR(128) 型の in-reply-to ID。

備考

クライアントは、In-Reply-To ID ヘッダ・フィールドを使用してメッセージ間リンクを設定できます。これは、応答メッセージをそれに対応する要求メッセージとリンクさせる場合によく使用されます。

in-reply-to ID は、返信対象メッセージの ID です。

このヘッダは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_setinreplytoid」 683 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ

例

次の例では、受信されたメッセージの in-reply-to ID をデータベース・コンソールに出力します。

```
begin
   declare @msgid varchar(128);
   declare @inreplytoid varchar(128);
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @inreplytoid = ml_qa_getinreplytoid( @msgid );
   message 'message is likely a reply to the message with id ' || @inreplytoid;
   commit;
end
```

ml_qa_getpriority

メッセージの優先度レベルを返します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

戻り値

INTEGER 型の優先度レベル。

備考

QAnywhere API では、10 レベルの優先度が定義されています。0 が最低の優先度、9 が最高の優先度を表します。クライアントは、優先度 0 ー 4 を通常のメッセージ、優先度 5 ー 9 を緊急度の高いメッセージとみなす必要があります。

このヘッダは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_setpriority」 684 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ

例

次の例では、受信されたメッセージの優先度をデータベース・コンソールに出力します。

```
begin
   declare @msgid varchar(128);
   declare @priority integer;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @priority = ml_qa_getpriority( @msgid );
   message 'a message with priority ' || @priority || ' has been received';
   commit;
end
```

ml_qa_getredelivered

受信されたが受信確認されていないメッセージであるかどうかを示す値を返します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

戻り値

再配信のステータスを示す BIT 型の値。**1** はメッセージが再配信中であることを示し、**0** は再配信中ではないことを示します。

備考

受信されたが受信確認されていないメッセージは、再配信される場合があります。たとえば、メッセージは受信されたものの、メッセージを受信したアプリケーションがメッセージの内容の処理を完了する前にクラッシュした場合などです。このような場合には、メッセージは再配信と

マーク付けされて、メッセージの処理が完了していない可能性があることを示す警告が受信側に送信されます。

たとえば、次のような手順を経てメッセージが受信されるとします。

1. 非トランザクション志向の QAnywhere Manager を使用するアプリケーションがメッセージを受信します。
2. このアプリケーションは、T1 というデータベース・テーブルにメッセージの内容とメッセージ ID を書き込み、変更をコミットします。
3. アプリケーションがメッセージの受信を確認します。

手順 1 と 2、または手順 2 と 3 の間でアプリケーションに障害が発生した場合は、アプリケーションの再起動時にメッセージが再配信されます。

手順 1 と 2 の間で障害が発生した場合は、手順 2 と 3 を実行してメッセージを再配信してください。手順 2 と 3 の間で障害が発生した場合は、メッセージはすでに処理されているので、メッセージの確認のみ必要です。

アプリケーションの障害時に何が起きたかを特定するには、アプリケーションで ml_qa_getredelivered を呼び出して、メッセージがすでに再配信されていたかどうかをチェックします。テーブル T1 で検索する必要があるのは、再配信されるメッセージだけです。アプリケーションで障害が発生することはほとんどないので、この方法は、受信したメッセージのメッセージ ID にアプリケーションからアクセスして、メッセージがテーブル T1 に格納されているかどうかをチェックするよりも効率的です。

このヘッダは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ

例

次の例では、受信されたメッセージについて、以前に配信されたが受信が確認されていない場合に、メッセージ ID をデータベース・コンソールに出力します。

```
begin
   declare @msgid varchar(128);
   declare @redelivered bit;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @redelivered = ml_qa_getredelivered( @msgid );
   if @redelivered = 1 then
      message 'message with message ID ' || @msgid || ' has been redelivered';
   end if;
   commit;
end
```


ml_qa_getreplytoaddress

メッセージの返信先アドレスを返します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

戻り値

VARCHAR(128) 型の返信アドレス。

備考

このヘッダは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_setreplytoaddress」 685 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ

例

次の例では、受信されたメッセージに返信アドレスがある場合に、「message received」という内容のメッセージを返信アドレスに送信します。

```
begin
   declare @msgid varchar(128);
   declare @rmsgid varchar(128);
   declare @replytoaddr varchar(128);
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @replytoaddr = ml_qa_getreplytoaddress( @msgid );
   if @replytoaddr is not null then
      set @rmsgid = ml_qa_createmessage();
      call ml_qa_settextcontent( @rmsgid, 'message received' );
      call ml_qa_putmessage( @rmsgid, @replytoaddr );
   end if;
   commit;
end
```

ml_qa_gettimestamp

メッセージの作成時刻を返します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

戻り値

TIMESTAMP 型のメッセージ作成時刻。

備考

メッセージの Timestamp ヘッダ・フィールドには、メッセージが作成された時刻が格納されます。これは、協定世界時 (UTC: Coordinated Universal Time) です。この時刻は、メッセージが実際に送信された時刻ではないので注意してください。メッセージの実際の送信時刻は、トランザクションの進行状況やクライアント側のキューに登録されているその他のメッセージの影響で、作成時刻よりも遅れる可能性があります。

このヘッダは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ

例

次の例では、受信されたメッセージの作成時刻をデータベース・コンソールに出力します。

```
begin
   declare @msgid varchar(128);
   declare @ts timestamp;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @ts = ml_qa_gettimestamp( @msgid );
   message 'message received with create time: ' || @ts ;
   commit;
end
```

ml_qa_setexpiration

メッセージの有効期限を設定します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

項目	説明	備考
2	有効期限	TIMESTAMP

備考

このヘッダは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_getexpiration」 677 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ

例

次の例では、メッセージが作成されて 3 日以内に配信されない場合に、メッセージが期限切れになります。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setexpiration( @msgid, dateadd( day, 3, current timestamp ) );
   call ml_qa_settextcontent( @msgid, 'time-limited offer' );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

ml_qa_setinreplytoid

メッセージの in-reply-to ID を設定します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	in-reply-to ID	VARCHAR(128)

備考

in-reply-to ID は、返信を追跡するために電子メール・システムで使用する in-reply-to ID のようなものです。

通常は、返信対象メッセージがある場合に、そのメッセージ ID として in-reply-to ID を設定します。

クライアントは、In-Reply-To ID ヘッダ・フィールドを使用してメッセージ間リンクを設定できます。これは、応答メッセージをそれに対応する要求メッセージとリンクさせる場合によく使用されます。

メッセージの送信後にこのヘッダを変更することはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_getinreplytoid」 677 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ

例

次の例では、返信アドレスが指定されたメッセージが受信されたら、in-reply-to-id にメッセージ ID を指定して返信メッセージを作成し、送信します。

```
begin
   declare @msgid varchar(128);
   declare @rmsgid varchar(128);
   declare @replyaddr varchar(128);
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @replyaddr = ml_qa_getreplyaddress( @msgid );
   if @replyaddr is not null then
      set @rmsgid = ml_qa_createmessage();
      call ml_qa_settextcontent( @rmsgid, 'message received' );
      call ml_qa_setinreplytoid( @rmsgid, @msgid );
      call ml_qa_putmessage( @rmsgid, @replyaddr );
   end if;
   commit;
end
```

ml_qa_setpriority

メッセージの優先度を設定します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	優先度	INTEGER

備考

QAnywhere API では、10 レベルの優先度が定義されています。0 が最低の優先度、9 が最高の優先度を表します。クライアントは、優先度 0 ー 4 を通常のメッセージ、優先度 5 ー 9 を緊急度の高いメッセージとみなす必要があります。

メッセージの送信後にこのヘッダを変更することはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_getpriority」 678 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ

例

次の例では、優先度の高いメッセージを送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setpriority( @msgid, 9 );
   call ml_qa_settextcontent( @msgid, 'priority content' );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

ml_qa_setreplytoaddress

メッセージの返信アドレスを設定します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	返信アドレス	VARCHAR(128)

備考

メッセージの送信後にこのヘッダを変更することはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_getreplytoaddress」 681 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ

例

次の例では、返信アドレスをメッセージに追加します。メッセージの受信者はこの返信アドレスを使用して、返信メッセージを作成できます。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setreplytoaddress( @msgid, 'myaddress' );
   call ml_qa_settextcontent( @msgid, 'some content' );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
```

```
    commit;
end
```

メッセージ・プロパティ

カスタム・メッセージ・プロパティの取得と設定、および事前定義済みメッセージ・プロパティの取得は、次のストアド・プロシージャを使用して行います。

「メッセージ・プロパティ」 223 ページを参照してください。

ml_qa_getbooleanproperty

指定されたメッセージ・プロパティを SQL BIT データ型で返します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)

戻り値

BIT 型のプロパティ値。

備考

メッセージ・プロパティ値が範囲外の場合は、SQL エラー SQLSTATE 22003 が発生します。

このプロパティは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_setbooleanproperty」 694 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ
- 「カスタムのメッセージ・プロパティ」 225 ページ

例

次の例では、受信されたメッセージの boolean 型プロパティ mybooleanproperty の値をデータベース・コンソールに出力します。

```
begin
   declare @msgid varchar(128);
   declare @prop bit;
   set @msgid = ml_qa_getmessage( 'myaddress' );
```



```
    set @prop = ml_qa_getbooleanproperty( @msgid, 'mybooleanproperty' );
    message 'message property mybooleanproperty is set to ' || @prop;
    commit;
end
```

ml_qa_getbyteproperty

指定されたメッセージ・プロパティを SQL TINYINT データ型で返します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)

戻り値

TINYINT 型のプロパティ値。

備考

メッセージ・プロパティ値が範囲外の場合は、SQL エラー SQLSTATE 22003 が発生します。

このプロパティは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_setbyteproperty」 695 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ
- 「カスタムのメッセージ・プロパティ」 225 ページ

例

次の例では、受信されたメッセージの byte 型プロパティ mybyteproperty の値をデータベース・コンソールに出力します。

```
begin
    declare @msgid varchar(128);
    declare @prop tinyint;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @prop = ml_qa_getbyteproperty( @msgid, 'mybyteproperty' );
    message 'message property mybyteproperty is set to ' || @prop;
    commit;
end
```

ml_qa_getdoubleproperty

指定されたメッセージ・プロパティを SQL DOUBLE データ型で返します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)

戻り値

DOUBLE 型のプロパティ値。

備考

メッセージ・プロパティ値が範囲外の場合は、SQL エラー SQLSTATE 22003 が発生します。

このプロパティは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_setdoubleproperty」 695 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ
- 「カスタムのメッセージ・プロパティ」 225 ページ

例

次の例では、受信されたメッセージの double 型プロパティ mydoubleproperty の値をデータベース・コンソールに出力します。

```
begin
   declare @msgid varchar(128);
   declare @prop double;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @prop = ml_qa_getdoubleproperty( @msgid, 'mydoubleproperty' );
   message 'message property mydoubleproperty is set to ' || @prop;
   commit;
end
```

ml_qa_getfloatproperty

指定されたメッセージ・プロパティを SQL FLOAT データ型で返します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)

戻り値

FLOAT 型のプロパティ値。

備考

メッセージ・プロパティ値が範囲外の場合は、SQL エラー SQLSTATE 22003 が発生します。

このプロパティは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_setfloatproperty」 696 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ
- 「カスタムのメッセージ・プロパティ」 225 ページ

例

次の例では、受信されたメッセージの float 型プロパティ myfloatproperty の値をデータベース・コンソールに出力します。

```
begin
   declare @msgid varchar(128);
   declare @prop float;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @prop = ml_qa_getfloatproperty( @msgid, 'myfloatproperty' );
   message 'message property myfloatproperty is set to ' || @prop;
   commit;
end
```

ml_qa_getintproperty

指定されたメッセージ・プロパティを SQL INTEGER データ型で返します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)

戻り値

INTEGER 型のプロパティ値。

備考

メッセージ・プロパティ値が範囲外の場合は、SQL エラー SQLSTATE 22003 が発生します。

このプロパティは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_setintproperty」 697 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ
- 「カスタムのメッセージ・プロパティ」 225 ページ

例

次の例では、受信されたメッセージの integer 型プロパティ myintproperty の値をデータベース・コンソールに出力します。

```
begin
   declare @msgid varchar(128);
   declare @prop integer;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @prop = ml_qa_getintproperty( @msgid, 'myintproperty' );
   message 'message property myintproperty is set to ' || @prop;
   commit;
end
```

ml_qa_getlongproperty

指定されたメッセージ・プロパティを SQL BIGINT データ型で返します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)

戻り値

BIGINT 型のプロパティ値。

備考

メッセージ・プロパティ値が範囲外の場合は、SQL エラー SQLSTATE 22003 が発生します。

このプロパティは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_setlongproperty」 698 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ
- 「カスタムのメッセージ・プロパティ」 225 ページ

ml_qa_getpropertynames

指定されたメッセージのプロパティ名を取得します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

備考

このストアド・プロシージャは、指定されたメッセージのプロパティ名の結果セットを開きます。メッセージ ID パラメータが、受信されたメッセージの ID であることが必要です。

結果セットは単一の VARCHAR(128) カラムです。このカラムの各行にメッセージ・プロパティ名が含まれています。QAnywhere の予約済みプロパティ名 ("ias_" または "QA" のプレフィクスが付いているプロパティ名) は返されません。

このプロパティは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ
- 「カスタムのメッセージ・プロパティ」 225 ページ

例

次の例では、メッセージ ID が msgid であるメッセージに対するプロパティ名の結果セット上でカーソルを宣言します。次に、アドレスが clientid¥queuename であるメッセージを取得し、カーソルを開いてそのメッセージのプロパティ名にアクセスしてから、次のプロパティ名をフェッチします。

```
begin
   declare prop_name_cursor cursor for
      call ml_qa_getpropertynames( @msgid );
   declare @msgid varchar(128);
   declare @name varchar(128);

   set @msgid = ml_qa_getmessage( 'clientid¥queuename' );
   open prop_name_cursor;
   lp: loop
      fetch next prop_name_cursor into name;
      if sqlcode <> 0 then leave lp end if;
      ...
   end loop;
   close prop_name_cursor;
end
```

ml_qa_getshortproperty

指定されたメッセージ・プロパティを SQL SMALLINT データ型で返します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)

戻り値

SMALLINT 型のプロパティ値。

備考

メッセージ・プロパティ値が範囲外の場合は、SQL エラー SQLSTATE 22003 が発生します。

このプロパティは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_setshortproperty」 699 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ
- 「カスタムのメッセージ・プロパティ」 225 ページ

例

次の例では、受信されたメッセージの short 型プロパティ myshortproperty の値をデータベース・コンソールに出力します。

```
begin
   declare @msgid varchar(128);
   declare @prop smallint;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @prop = ml_qa_getshortproperty( @msgid, 'myshortproperty' );
   message 'message property myshortproperty is set to ' || @prop;
   commit;
end
```

ml_qa_getstringproperty

指定されたメッセージ・プロパティを SQL LONG VARCHAR データ型で返します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)

戻り値

LONG VARCHAR 型のプロパティ値。

備考

メッセージ・プロパティ値が範囲外の場合は、SQL エラー SQLSTATE 22003 が発生します。

このプロパティは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ

- 「ml_qa_setstringproperty」 699 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ
- 「カスタムのメッセージ・プロパティ」 225 ページ

例

次の例では、受信されたメッセージの string 型プロパティ mystringproperty の値をデータベース・コンソールに出力します。

```
begin
   declare @msgid varchar(128);
   declare @prop long varchar;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @prop = ml_qa_getstringproperty( @msgid, 'mystringproperty' );
   message 'message property mystringproperty is set to ' || @prop;
   commit;
end
```

ml_qa_setbooleanproperty

指定されたメッセージ・プロパティを SQL BIT データ型で設定します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)
3	プロパティ値	BIT

備考

メッセージの送信後にこのプロパティを変更することはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_getbooleanproperty」 686 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ
- 「カスタムのメッセージ・プロパティ」 225 ページ

例

次の例では、メッセージを作成し、boolean 型プロパティ mybooleanproperty1 と mybooleanproperty2 を設定して、アドレス clientid¥queuename に送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
```

```
    call ml_qa_setbooleanproperty( @msgid, 'mybooleanproperty1', 0 );
    call ml_qa_setbooleanproperty( @msgid, 'mybooleanproperty2', 1 );
    call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
    commit;
end
```

ml_qa_setbyteproperty

指定されたメッセージ・プロパティを SQL TINYINT データ型で設定します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)
3	プロパティ値	TINYINT

備考

メッセージの送信後にこのプロパティを変更することはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_getbyteproperty」 687 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ
- 「カスタムのメッセージ・プロパティ」 225 ページ

例

次の例では、メッセージを作成し、byte 型プロパティ mybyteproperty1 と mybyteproperty2 を設定して、アドレス clientid¥queuename に送信します。

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setbyteproperty( @msgid, 'mybyteproperty1', 0 );
    call ml_qa_setbyteproperty( @msgid, 'mybyteproperty2', 255 );
    call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
    commit;
end
```

ml_qa_setdoubleproperty

指定されたメッセージ・プロパティを SQL DOUBLE データ型で設定します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)
3	プロパティ値	DOUBLE

備考

メッセージの送信後にこのプロパティを変更することはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_getdoubleproperty」 688 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ
- 「カスタムのメッセージ・プロパティ」 225 ページ

例

次の例では、メッセージを作成し、double 型プロパティ mydoubleproperty1 と mydoubleproperty2 を設定して、アドレス clientid¥queuename に送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setdoubleproperty( @msgid, 'mydoubleproperty1', -12.34e-56 );
   call ml_qa_setdoubleproperty( @msgid, 'mydoubleproperty2', 12.34e56 );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

ml_qa_setfloatproperty

指定されたメッセージ・プロパティを SQL FLOAT データ型で設定します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)
3	プロパティ値	FLOAT

備考

メッセージの送信後にこのプロパティを変更することはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_getfloatproperty」 688 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ
- 「カスタムのメッセージ・プロパティ」 225 ページ

例

次の例では、メッセージを作成し、float 型プロパティ myfloatproperty1 と myfloatproperty2 を設定して、アドレス clientid¥queuename に送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setfloatproperty( @msgid, 'myfloatproperty1', -1.3e-5 );
   call ml_qa_setfloatproperty( @msgid, 'myfloatproperty2', 1.3e5 );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

ml_qa_setintproperty

指定されたメッセージ・プロパティを SQL INTEGER データ型で設定します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)
3	プロパティ値	INTEGER

備考

メッセージの送信後にこのプロパティを変更することはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_getintproperty」 689 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ
- 「カスタムのメッセージ・プロパティ」 225 ページ

例

次の例では、メッセージを作成し、integer 型プロパティ myintproperty1 と myintproperty2 を設定して、アドレス clientid¥queuename に送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setintproperty( @msgid, 'myintproperty1', -1234567890 );
   call ml_qa_setintproperty( @msgid, 'myintproperty2',  1234567890 );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

ml_qa_setlongproperty

指定されたメッセージ・プロパティを SQL BIGINT データ型で設定します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)
3	プロパティ値	BIGINT

備考

メッセージの送信後にこのプロパティを変更することはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_getlongproperty」 690 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ
- 「カスタムのメッセージ・プロパティ」 225 ページ

例

次の例では、メッセージを作成し、long 型プロパティ mylongproperty1 と mylongproperty2 を設定して、アドレス clientid¥queuename に送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setlongproperty( @msgid, 'mylongproperty1', -12345678900987654321 );
   call ml_qa_setlongproperty( @msgid, 'mylongproperty2',  12345678900987654321 );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

ml_qa_setshortproperty

指定されたメッセージ・プロパティを SQL SMALLINT データ型で設定します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)
3	プロパティ値	SMALLINT

備考

メッセージの送信後にこのプロパティを変更することはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_getshortproperty」 692 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ
- 「カスタムのメッセージ・プロパティ」 225 ページ

例

次の例では、メッセージを作成し、short 型プロパティ myshortproperty1 と myshortproperty2 を設定して、アドレス clientid¥queuename に送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setshortproperty( @msgid, 'myshortproperty1', -12345 );
   call ml_qa_setshortproperty( @msgid, 'myshortproperty2',  12345 );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

ml_qa_setstringproperty

指定されたメッセージ・プロパティを SQL LONG VARCHAR データ型で設定します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

項目	説明	備考
2	プロパティ名	VARCHAR(128)
3	プロパティ値	LONG VARCHAR

備考

メッセージの送信後にこのプロパティを変更することはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_getstringproperty」 693 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ
- 「カスタムのメッセージ・プロパティ」 225 ページ

例

次の例では、メッセージを作成し、string 型プロパティ mystringproperty1 と mystringproperty2 を設定して、アドレス clientid¥queuename に送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setstringproperty( @msgid, 'mystringproperty1', 'c:¥¥temp' );
   call ml_qa_setstringproperty( @msgid, 'mystringproperty2',  'first line¥nsecond line' );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

メッセージの内容

メッセージの内容の取得と設定は、次のストアド・プロシージャを使用して行います。

ml_qa_getbinarycontent

バイナリ・メッセージの内容を返します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

戻り値

LONG BINARY 型のメッセージの内容。

メッセージの内容がバイナリではなくテキストの場合は、NULL を返します。

この内容は、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_setbinarycontent」 703 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ
- 「ml_qa_getcontentclass」 701 ページ

例

次の例では、メッセージの暗号化された内容を復号化してデータベース・コンソールに出力します。

```
begin
   declare @msgid varchar(128);
   declare @content long binary;
   declare @plaintext long varchar;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @content = ml_qa_getbinarycontent( @msgid );
   set @plaintext = decrypt( @content, 'mykey' );
   message 'message content decrypted: ' || @plaintext;
   commit;
end
```

ml_qa_getcontentclass

メッセージ・タイプ (テキストまたはバイナリ) を返します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

戻り値

INTEGER 型の内容クラス。

次のいずれかの値を返します。

- **1**　メッセージの内容がバイナリであり、ストアド・プロシージャ ml_qa_getbinarycontent を使用して読み込む必要があることを示します。
- **2**　メッセージの内容がテキストであり、ストアド・プロシージャ ml_qa_gettextcontent を使用して読み込む必要があることを示します。

備考

この内容は、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ
- 「ml_qa_getbinarycontent」 700 ページ
- 「ml_qa_gettextcontent」 702 ページ

例

次の例では、受信されたメッセージの内容をデータベース・コンソールに出力します。

```
begin
   declare @msgid varchar(128);
   declare @contentclass integer;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @contentclass = ml_qa_getcontentclass( @msgid );
   if @contentclass = 1 then
      message 'message  binary is ' || ml_qa_getbinarycontent( @msgid );
   elseif @contentclass = 2 then
      message 'message  text is ' || ml_qa_gettextcontent( @msgid );
   end if;
   commit;
end
```

ml_qa_gettextcontent

テキスト・メッセージの内容を返します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

戻り値

LONG VARCHAR 型のメッセージの内容。

メッセージの内容がテキストではなくバイナリの場合は、NULL を返します。

備考

この内容は、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_settextcontent」 703 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ
- 「ml_qa_getcontentclass」 701 ページ

例

次の例では、メッセージの内容をデータベース・コンソールに出力します。

```
begin
   declare @msgid varchar(128);
   declare @content long binary;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @content = ml_qa_gettextcontent( @msgid );
   message 'message content: ' || @content ;
   commit;
end
```

ml_qa_setbinarycontent

メッセージにバイナリの内容を設定します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	内容	LONG BINARY

メッセージの送信後にこの内容を変更することはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_getbinarycontent」 700 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ

例

次の例では、暗号化した内容を含むメッセージを作成してから送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setbinarycontent( @msgid, encrypt( 'my secret message', 'mykey' ) );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

ml_qa_settextcontent

メッセージにテキストの内容を設定します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	内容	LONG VARCHAR

備考

メッセージの送信後にこの内容を変更することはできません。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_gettextcontent」 702 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ

例

次の例では、メッセージを作成してから、指定された内容を設定します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_settextcontent( @msgid, 'my simple message' );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

メッセージ・ストア・プロパティ

クライアント・メッセージ・ストアのプロパティの取得と設定は、次のストアド・プロシージャを使用して行います。

メッセージ・ストア・プロパティの詳細については、「クライアント・メッセージ・ストア・プロパティ」 230 ページを参照してください。

ml_qa_getstoreproperty

クライアント・メッセージ・ストアのプロパティを返します。

パラメータ

項目	説明	備考
1	プロパティ名	VARCHAR(128)

戻り値

LONG VARCHAR 型のプロパティ値。

備考

クライアント・メッセージ・ストアのプロパティは、このクライアント・メッセージ・ストアへのすべての接続から読み込むことができます。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_setstoreproperty」 705 ページ

例

次の例では、メッセージ・ストアの現在の同期ポリシーを取得して、データベース・コンソールに出力します。

```
begin
   declare @policy varchar(128);
   set @policy = ml_qa_getstoreproperty( 'policy' );
   message 'the current policy for synchronizing this message store is ' || @policy;
end
```

ml_qa_setstoreproperty

クライアント・メッセージ・ストアのプロパティを設定します。

パラメータ

項目	説明	備考
1	プロパティ名	VARCHAR(128)
2	プロパティ値	SMALLINT

備考

クライアント・メッセージ・ストアのプロパティは、このクライアント・メッセージ・ストアへのすべての接続から読み込むことができます。プロパティ値は、サーバに対しても同期されます。サーバでは転送ルールでプロパティ値を使用できます。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_getstoreproperty」 705 ページ

例

次の例では、メッセージ・ストアの同期ポリシーを自動に設定します。

```
begin
   call ml_qa_setstoreproperty( 'policy', 'automatic' );
   commit;
end
```

メッセージの管理

QAnywhere クライアント・トランザクションを管理するには、次のストアド・プロシージャを使用します。

ml_qa_createmessage

新規メッセージのメッセージ ID を返します。

戻り値

新規メッセージのメッセージ ID。

備考

このストアド・プロシージャは、メッセージを作成するために使用します。作成したメッセージは、内容、プロパティ、ヘッダを関連付けて送信できます。

内容、プロパティ、ヘッダを関連付けるには、ml_qa_set で始まる QAnywhere ストアド・プロシージャを使用します。たとえば、ml_qa_setbinarycontent を使用してバイナリ・メッセージを作成したり、ml_qa_settextcontent を使用してテキスト・メッセージを作成します。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「メッセージ・ヘッダ」 676 ページ
- 「メッセージ・プロパティ」 686 ページ
- 「メッセージの内容」 700 ページ

例

次の例では、メッセージを作成して内容を設定し、アドレス clientid¥queuename に送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_settextcontent( @msgid, 'some content' );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

ml_qa_getmessage

キューに登録されている、指定されたアドレスを持つメッセージのうち、次に取り出せるもののメッセージ ID を返します。メッセージがキューに登録されるまで処理をブロックします。

パラメータ

項目	説明	備考
1	アドレス	VARCHAR(128)

戻り値

VARCHAR(128) 型の メッセージ ID。

このアドレスを持つ、キューに登録されたメッセージがない場合は、NULL を返します。

備考

指定された QAnywhere メッセージ・アドレス宛ての待機中のメッセージがあるかどうかを同期的にチェックするには、このストアド・プロシージャを使用します。指定された QAnywhere アドレス宛てのメッセージが発生した場合に、非同期ですぐに SQL プロシージャを呼び出すには、Listener を使用します。

このストアド・プロシージャは、メッセージがキューに登録されるまで処理をブロックします。

ブロックを回避する方法の詳細については、「ml_qa_getmessagenowait」 709 ページまたは「ml_qa_getmessagetimeout」 710 ページを参照してください。

返されるメッセージ ID に対応するメッセージが受信されたとみなされるのは、現在のトランザクションがコミットされた後です。受信がコミットされたメッセージは、このストアド・プロシージャやいかなる QAnywhere API でも受信することができません。同様に、現在のトランザクションがロールバックされると、メッセージが受信されていないことを意味します。このため、以降の ml_qa_getmessage の呼び出しで同じメッセージ ID が返される可能性があります。

受信されたメッセージのプロパティと内容は、現在のトランザクションに対してコミットまたはロールバックが実行されるまでの間、各種の ml_qa_get ストアド・プロシージャを使用して読み込むことができます。現在のトランザクションに対してコミットまたはロールバックが実行された時点で、そのメッセージ・データは読み込めなくなります。コミットを実行する前に、メッセージ内の必要なデータを表形式のデータとして、または SQL 変数に保存してください。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_getmessagenowait」 709 ページ
- 「ml_qa_getmessagetimeout」 710 ページ
- 「メッセージ・ヘッダ」 676 ページ
- 「メッセージ・プロパティ」 686 ページ
- 「メッセージの内容」 700 ページ

例

次の例では、アドレス myaddress に送信されたすべてのメッセージの内容を表示します。

```
begin
   declare @msgid varchar(128);
   loop
      set @msgid = ml_qa_getmessage( 'myaddress' );
      message 'a message with content ' || ml_qa_gettextcontent( @msgid ) || ' has been received';
      commit;
   end loop;
end
```

ml_qa_getmessagenowait

現在キューに登録されている、指定されたアドレスを持つメッセージのうち、次に取り出せるもののメッセージ ID を返します。

パラメータ

項目	説明	備考
1	アドレス	VARCHAR(128)

戻り値

VARCHAR(128) 型の メッセージ ID。

キューに登録されている、指定されたアドレスを持つメッセージのうち、次に取り出せるもののメッセージ ID を返します。このアドレスを持つ、キューに登録されたメッセージがない場合は、NULL を返します。

備考

指定された QAnywhere メッセージ・アドレス宛ての待機中のメッセージがあるかどうかを同期的にチェックするには、このストアド・プロシージャを使用します。指定された QAnywhere アドレス宛てのメッセージが発生した場合に、非同期ですぐに SQL プロシージャを呼び出すには、Listener を使用します。

メッセージが準備できるまで処理をブロックする方法の詳細については、「ml_qa_getmessage」 707 ページと「ml_qa_getmessagetimeout」 710 ページを参照してください。

返されるメッセージ ID に対応するメッセージが受信されたとみなされるのは、現在のトランザクションがコミットされた後です。受信がコミットされたメッセージは、このストアド・プロシージャやいかなる QAnywhere API でも受信することができません。同様に、現在のトランザクションがロールバックされると、メッセージが受信されていないことを意味します。このため、以降の ml_qa_getmessage の呼び出しで同じメッセージ ID が返される可能性があります。

受信されたメッセージのプロパティと内容は、現在のトランザクションに対してコミットまたはロールバックが実行されるまでの間、各種の ml_qa_get ストアド・プロシージャを使用して読み込むことができます。現在のトランザクションに対してコミットまたはロールバックが実行された時点で、そのメッセージ・データは読み込めなくなります。コミットを実行する前に、メッセージ内の必要なデータを表形式のデータとして、または SQL 変数に保存してください。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「QAnywhere メッセージ・アドレス」 56 ページ
- 「Listener」 『Mobile Link - サーバ起動同期』
- 「ml_qa_getmessagetimeout」 710 ページ
- 「メッセージ・ヘッダ」 676 ページ
- 「メッセージ・プロパティ」 686 ページ
- 「メッセージの内容」 700 ページ

例

次の例では、キューに登録されている、アドレス myaddress を持つメッセージがすべて読み込まれるまで、これらすべてのメッセージの内容を表示します (一般には、各メッセージが読み込まれた後ではなく、最後のメッセージが読み込まれた後でコミットする方が効率的です)。

```
begin
   declare @msgid varchar(128);
   loop
      set @msgid = ml_qa_getmessagenowait( 'myaddress' );
      if @msgid is null then leave end if;
      message 'a message with content ' || ml_qa_gettextcontent( @msgid ) || ' has been received';
   end loop;
   commit;
end
```

ml_qa_getmessagetimeout

指定されたタイムアウト期間だけ待機してから、キューに登録されている、指定されたアドレスを持つメッセージのうち、次に取り出せるもののメッセージ ID を返します。

パラメータ

項目	説明	備考
1	アドレス	VARCHAR(128)
2	タイムアウト (ミリ秒単位)	INTEGER

戻り値

VARCHAR(128) 型の メッセージ ID。

タイムアウト時間内に、このアドレスを持つキューに登録されたメッセージがない場合は、NULL を返します。

備考

指定された QAnywhere メッセージ・アドレス宛ての待機中のメッセージがあるかどうかを同期的にチェックするには、このストアド・プロシージャを使用します。指定された QAnywhere アドレス宛てのメッセージが発生した場合に、非同期ですぐに SQL プロシージャを呼び出すには、Listener を使用します。

返されるメッセージ ID に対応するメッセージが受信されたとみなされるのは、現在のトランザクションがコミットされた後です。受信がコミットされたメッセージは、このストアド・プロシージャやいかなる QAnywhere API でも受信できません。同様に、現在のトランザクションがロールバックされると、メッセージが受信されていないことを意味します。このため、以降の ml_qa_getmessage の呼び出しで同じメッセージ ID が返される可能性があります。

受信されたメッセージのプロパティと内容は、現在のトランザクションに対してコミットまたはロールバックが実行されるまでの間、各種の ml_qa_get ストアド・プロシージャを使用して読み込むことができます。現在のトランザクションに対してコミットまたはロールバックが実行され

た時点で、そのメッセージ・データは読み込めなくなります。コミットを実行する前に、メッセージ内の必要なデータを表形式のデータとして、または SQL 変数に保存してください。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_getmessage」 707 ページ
- 「ml_qa_getmessagenowait」 709 ページ

例

次の例では、アドレス myaddress に送信されたすべてのメッセージの内容をデータベース・コンソールに出力します。メッセージが受信されなかった場合は、データベース・コンソールを 10 秒ごとに更新します。

```
begin
   declare @msgid varchar(128);
   loop
      set @msgid = ml_qa_getmessagetimeout( 'myaddress', 10000 );
      if @msgid is null then
         message 'waiting for a message...';
      else
         message 'a message with content ' || ml_qa_gettextcontent( @msgid ) || ' has been received';
      commit;
      end if;
   end loop;
end
```

ml_qa_grant_messaging_permissions

QAnywhere ストアド・プロシージャを使用するパーミッションを他のユーザに与えます。

パラメータ

項目	説明	備考
1	データベース・ユーザ ID	VARCHAR(128)

備考

QAnywhere ストアド・プロシージャを実行するパーミッションを自動的に与えられるのは、DBA 権限を持つユーザだけです。それ以外のユーザに対しては、DBA 権限を持つユーザがこのストアド・プロシージャを実行して、パーミッションを付与する必要があります。

このプロシージャは ml_qa_message_group というグループにユーザを追加して、QAnywhere の全ストアド・プロシージャを実行するパーミッションを付与します。

参照

- 「SQL アプリケーションの設定」 66 ページ

例

たとえば、user1 というデータベース ID を持つユーザにメッセージング・パーミッションを付与するには、次の SQL コードを実行します。

```
call dbo.ml_qa_grant_messaging_permissions( 'user1' )
```

ml_qa_listener_queue

メッセージを非同期に受信するために、**ml_qa_listener_*queue*** というストアド・プロシージャ (*queue* はメッセージ・キューの名前) を作成します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は QAnywhere Listener から取得できます。

備考

注意
このストアド・プロシージャは他のすべての QAnywhere ストアド・プロシージャとは異なり、デフォルトでは提供されていません。ユーザがストアド・プロシージャ **ml_qa_listener_*queue*** (*queue* はメッセージ・キューの名前) を作成すると、そのストアド・プロシージャは QAnywhere によって使用されます。

メッセージは接続で同期的に受信できますが、多くの場合は非同期で受信する方が便利です。特定のアドレスを持つメッセージがキューに登録された場合に呼び出されるストアド・プロシージャを作成できます。このプロシージャの名前は、**ml_qa_listener_*queue*** (*queue* はメッセージ・キューの名前) にする必要があります。このようなプロシージャが存在する場合、そのプロシージャは、指定されたアドレスを持つメッセージがキューに登録されるたびに呼び出されます。

このプロシージャは別の接続から呼び出されます。このプロシージャの実行中に SQL エラーが発生しないかぎり、メッセージの確認とコミットは自動的に行われます。

このプロシージャ内では、コミットやロールバックを実行しないでください。

キューの名前は QAnywhere アドレスの一部です。詳細については、「QAnywhere メッセージ・アドレス」 56 ページを参照してください。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「非同期的なメッセージ受信」 83 ページ
- 「同期的なメッセージ受信」 82 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ

例

次の例では、アドレス executesql を持つメッセージがキューに登録されるたびに呼び出されるプロシージャを作成します。この例のプロシージャでは、メッセージの内容が、現在のデータベースに対して実行できる SQL 文であるものと想定しています。

```
CREATE PROCEDURE ml_qa_listener_executesql(IN @msgid VARCHAR(128))
begin
   DECLARE @execstr LONG VARCHAR;
   SET @execstr = ml_qa_gettextcontent( @msgid );
   EXECUTE IMMEDIATE @execstr;
end
```

ml_qa_putmessage

メッセージを送信します。

パラメータ

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	アドレス	VARCHAR(128)

備考

指定するメッセージ ID は、ml_qa_createmessage を使用して、すでに作成されていることが必要です。ml_qa_putmessage を呼び出す前にそのメッセージ ID に関連付けられていた内容、プロパティ、ヘッダだけが、メッセージとともに送信されます。ml_qa_putmessage の呼び出し後に追加されたものは無視されます。

送信のためにメッセージが実際にキューに登録される前に、コミットを実行する必要があります。

参照

- 「SQL アプリケーションの設定」 66 ページ
- 「ml_qa_createmessage」 707 ページ
- 「ml_qa_getmessage」 707 ページ

例

次の例では、「a simple message」という内容のメッセージを作成し、アドレス clientid¥queuename に送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_settextcontent( @msgid, 'a simple message' );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

ml_qa_triggersendreceive

Mobile Link サーバとのメッセージの同期をトリガします。

備考

通常、メッセージ同期は QAnywhere Agent が処理します。ただし、同期ポリシーがオンデマンドの場合、メッセージ同期をトリガするのはアプリケーション側の役割です。これを行うには、このストアド・プロシージャを使用します。メッセージ同期のトリガは、現在のトランザクションがコミットされるまで適用されません。

参照

- 「SQL アプリケーションの設定」 66 ページ

例

次の例では、メッセージの送信後すぐに、メッセージ転送が開始されます。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_settextcontent( @msgid, 'my simple message' );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   call ml_qa_triggersendreceive();
   commit;
end
```

索引

記号

A

B

C

D

E

F

G

I

J

L

M

N

O

P

Q

R

S

T

U

W

X

あ

い

え

お

か

き

く

け

こ

さ

し

す

と

に

ね

は

ひ

ふ

も

よ

ら

る

れ

ろ