SYBASE iAnywhere

QAnywhere

2009 年 2 月

バージョン 11.0.1

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、モバイル・クライアント、ワイヤレス・クライアント、デスクトップ・クライアント、およびラップトップ・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、QAnywhere について説明します。

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 のマニュアルについて

SQL Anywhere の完全なマニュアルは 4 つの形式で提供されており、いずれも同じ情報が含まれています。

- **HTML ヘルプ**　オンライン・ヘルプには、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。

  Microsoft Windows オペレーティング・システムを使用している場合は、オンライン・ヘルプは HTML ヘルプ (CHM) 形式で提供されます。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 11] - [マニュアル] - [オンライン・マニュアル]** を選択します。

  管理ツールのヘルプ機能でも、同じオンライン・マニュアルが使用されます。

- **Eclipse**　UNIX プラットフォームでは、完全なオンライン・ヘルプは Eclipse 形式で提供されます。マニュアルにアクセスするには、SQL Anywhere 11 インストール環境の *bin32* または *bin64* ディレクトリから *sadoc* を実行します。

- **DocCommentXchange**　DocCommentXchange は、SQL Anywhere マニュアルにアクセスし、マニュアルについて議論するためのコミュニティです。

  DocCommentXchange は次の目的に使用できます (現在のところ、日本語はサポートされておりません)。

  ○ マニュアルを表示する

  ○ マニュアルの項目について明確化するために、ユーザによって追加された内容を確認する

  ○ すべてのユーザのために、今後のリリースでマニュアルを改善するための提案や修正を行う

  http://dcx.sybase.com を参照してください。

- **PDF**　SQL Anywhere の完全なマニュアル・セットは、Portable Document Format (PDF) 形式のファイルとして提供されます。内容を表示するには、PDF リーダが必要です。Adobe Reader をダウンロードするには、http://get.adobe.com/reader/ にアクセスしてください。

  Microsoft Windows オペレーティング・システムで PDF 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 11] - [マニュアル] - [オンライン・マニュアル - PDF]** を選択します。

  UNIX オペレーティング・システムで PDF マニュアルにアクセスするには、Web ブラウザを使用して *install-dir/documentation/ja/pdf/index.html* を開きます。

マニュアル・セットに含まれる各マニュアルについて

SQL Anywhere のマニュアルは次の構成になっています。

- **『SQL Anywhere 11 - 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 11 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 11 - 変更点とアップグレード』** このマニュアルでは、SQL Anywhere 11 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 データベースを実行、管理、構成する方法について説明します。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、管理ユーティリティとオプションについて説明します。
- **『SQL Anywhere サーバ - プログラミング』** このマニュアルでは、C、C++、Java、PHP、Perl、Python、および Visual Basic や Visual C# などの .NET プログラミング言語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。ADO.NET や ODBC などのさまざまなプログラミング・インタフェースについても説明します。
- **『SQL Anywhere サーバ - SQL リファレンス』** このマニュアルでは、システム・プロシージャとカタログ (システム・テーブルとビュー) に関する情報について説明します。また、SQL Anywhere での SQL 言語の実装 (探索条件、構文、データ型、関数) についても説明します。
- **『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。
- **『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 - 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。また、dbmlsync API についても説明します。dbmlsync API を使用すると、同期を C++ または .NET のクライアント・アプリケーションにシームレスに統合できます。
- **『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 - サーバ起動同期』** このマニュアルでは、Mobile Link サーバ起動同期について説明します。この機能により、Mobile Link サーバは同期を開始したり、リモート・デバイス上でアクションを実行することができます。
- **『QAnywhere』** このマニュアルでは、モバイル・クライアント、ワイヤレス・クライアント、デスクトップ・クライアント、およびラップトップ・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、QAnywhere について説明します。
- **『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。

- **『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 - 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・デバイス、モバイル・デバイス、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 Mobile、または Windows を搭載しているハンドヘルド・デバイス、モバイル・デバイス、または埋め込みデバイスの Web ベースのデータベース・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 - .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・デバイス、モバイル・デバイス、または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 J』** このマニュアルでは、Ultra Light J について説明します。Ultra Light J を使用すると、Java をサポートしている環境用のデータベース・アプリケーションを開発し、配備することができます。Ultra Light J は、BlackBerry スマートフォンと Java SE 環境をサポートしており、iAnywhere Ultra Light データベース製品がベースになっています。
- **『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示し、その診断情報を説明します。

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表記規則について説明します。

オペレーティング・システム

SQL Anywhere はさまざまなプラットフォームで稼働します。ほとんどの場合、すべてのプラットフォームで同じように動作しますが、いくつかの相違点や制限事項があります。このような相違点や制限事項は、一般に、基盤となっているオペレーティング・システム (Windows、UNIX など) に由来しており、使用しているプラットフォームの種類 (AIX、Windows Mobile など) またはバージョンに依存していることはほとんどありません。

オペレーティング・システムへの言及を簡素化するために、このマニュアルではサポートされているオペレーティング・システムを次のようにグループ分けして表記します。

- **Windows** Microsoft Windows ファミリを指しています。これには、主にサーバ、デスクトップ・コンピュータ、ラップトップ・コンピュータで使用される Windows Vista や Windows XP、およびモバイル・デバイスで使用される Windows Mobile が含まれます。

  特に記述がないかぎり、マニュアル中に Windows という記述がある場合は、Windows Mobile を含むすべての Windows ベース・プラットフォームを指しています。

- **UNIX**　特に記述がないかぎり、マニュアル中に UNIX という記述がある場合は、Linux および Mac OS X を含むすべての UNIX ベース・プラットフォームを指しています。

ディレクトリとファイル名

ほとんどの場合、ディレクトリ名およびファイル名の参照形式はサポートされているすべてのプラットフォームで似通っており、それぞれの違いはごくわずかです。このような場合は、Windows の表記規則が使用されています。詳細がより複雑な場合は、マニュアルにすべての関連形式が記載されています。

ディレクトリ名とファイル名の表記を簡素化するために使用されている表記規則は次のとおりです。

- **大文字と小文字のディレクトリ名**　Windows と UNIX では、ディレクトリ名およびファイル名には大文字と小文字が含まれている場合があります。ディレクトリやファイルが作成されると、ファイル・システムでは大文字と小文字の区別が維持されます。

  Windows では、ディレクトリおよびファイルを参照するとき、大文字と小文字は**区別されません**。大文字と小文字を混ぜたディレクトリ名およびファイル名は一般的に使用されますが、参照するときはすべて小文字を使用するのが通常です。SQL Anywhere では、*Bin32* や *Documentation* などのディレクトリがインストールされます。

  UNIX では、ディレクトリおよびファイルを参照するとき、大文字と小文字は**区別されます**。大文字と小文字を混ぜたディレクトリ名およびファイル名は一般的に使用されません。ほとんどの場合は、すべて小文字の名前が使用されます。SQL Anywhere では、*bin32* や *documentation* などのディレクトリがインストールされます。

  このマニュアルでは、ディレクトリ名に Windows の形式を使用しています。ほとんどの場合、大文字と小文字が混ざったディレクトリ名をすべて小文字に変換すると、対応する UNIX 用のディレクトリ名になります。

- **各ディレクトリおよびファイル名を区切るスラッシュ**　マニュアルでは、ディレクトリの区切り文字に円記号を使用しています。たとえば、PDF 形式のマニュアルは *install-dir¥Documentation¥ja¥pdf* にあります。これは Windows の形式です。

  UNIX では、円記号をスラッシュに置き換えます。PDF マニュアルは *install-dir/documentation/ja/pdf* にあります。

- **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、*.exe* や *.bat* などの拡張子が付きます。UNIX では、実行ファイルの名前に拡張子は付きません。

  たとえば、Windows でのネットワーク・データベース・サーバは *dbsrv11.exe* です。UNIX では *dbsrv11* です。

- ***install-dir***　インストール・プロセス中に、SQL Anywhere をインストールするロケーションを選択します。このロケーションを参照する環境変数 SQLANY11 が作成されます。このマニュアルでは、そのロケーションを *install-dir* と表します。

  たとえば、マニュアルではファイルを install-dir¥readme.txt のように参照します。これは、Windows では、%SQLANY11%¥readme.txt に対応します。UNIX では、$SQLANY11/readme.txt または ${SQLANY11}/readme.txt に対応します。

install-dir のデフォルト・ロケーションの詳細については、「SQLANY11 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・プロセス中に、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します。このロケーションを参照する環境変数 SQLANYSAMP11 が作成されます。このマニュアルではそのロケーションを *samples-dir* と表します。

  Windows エクスプローラ・ウィンドウで *samples-dir* を開くには、**[スタート] - [プログラム] - [SQL Anywhere 11] - [サンプル・アプリケーションとプロジェクト]** を選択します。

  samples-dir のデフォルト・ロケーションの詳細については、「SQLANYSAMP11 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。

コマンド・プロンプトとコマンド・シェル構文

ほとんどのオペレーティング・システムには、コマンド・シェルまたはコマンド・プロンプトを使用してコマンドおよびパラメータを入力する方法が、1 つ以上あります。Windows のコマンド・プロンプトには、コマンド・プロンプト (DOS プロンプト) および 4NT があります。UNIX のコマンド・シェルには、Korn シェルおよび bash があります。各シェルには、単純コマンドからの拡張機能が含まれています。拡張機能は、特殊文字を指定することで起動されます。特殊文字および機能は、シェルによって異なります。これらの特殊文字を誤って使用すると、多くの場合、構文エラーや予期しない動作が発生します。

このマニュアルでは、一般的な形式のコマンド・ラインの例を示します。これらの例に、シェルにとって特別な意味を持つ文字が含まれている場合、その特定のシェル用にコマンドを変更することが必要な場合があります。このマニュアルではコマンドの変更について説明しませんが、通常、その文字を含むパラメータを引用符で囲むか、特殊文字の前にエスケープ文字を記述します。

次に、プラットフォームによって異なるコマンド・ライン構文の例を示します。

- **カッコと中カッコ** 一部のコマンド・ライン・オプションは、詳細な値を含むリストを指定できるパラメータを要求します。リストは通常、カッコまたは中カッコで囲まれています。このマニュアルでは、カッコを使用します。次に例を示します。

  ```
  -x tcpip(host=127.0.0.1)
  ```

  カッコによって構文エラーになる場合は、代わりに中カッコを使用します。

  ```
  -x tcpip{host=127.0.0.1}
  ```

  どちらの形式でも構文エラーになる場合は、シェルの要求に従ってパラメータ全体を引用符で囲む必要があります。

  ```
  -x "tcpip(host=127.0.0.1)"
  ```

- **引用符** パラメータの値として引用符を指定する必要がある場合、その引用符はパラメータを囲むために使用される通常の引用符と競合する可能性があります。たとえば、値に二重引用符を含む暗号化キーを指定するには、キーを引用符で囲み、パラメータ内の引用符をエスケープします。

  ```
  -ek "my ¥"secret¥" key"
  ```


多くのシェルでは、キーの値は my "secret" key のようになります。

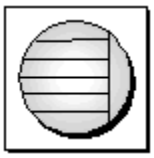
- **環境変数**　マニュアルでは、環境変数設定が引用されます。Windows のシェルでは、環境変数は構文 *%ENVVAR%* を使用して指定されます。UNIX のシェルでは、環境変数は構文 *$ENVVAR* または *${ENVVAR}* を使用して指定されます。

グラフィック・アイコン

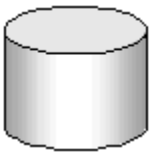
このマニュアルでは、次のアイコンを使用します。

- 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。



- データベース。ハイレベルの図では、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。

- プログラミング・インタフェース。

ドキュメンテーション・チームへのお問い合わせ

このヘルプに関するご意見、ご提案、フィードバックをお寄せください。

SQL Anywhere ドキュメンテーション・チームへのご意見やご提案は、弊社までご連絡ください。頂戴したご意見はマニュアルの向上に役立たせていただきます。ぜひとも、ご意見をお寄せください。

DocCommentXchange

DocCommentXchange を使用して、ヘルプ・トピックに関するご意見を直接お寄せいただくこともできます。DocCommentXchange (DCX) は、SQL Anywhere マニュアルにアクセスしたり、マニュアルについて議論するためのコミュニティです。DocCommentXchange は次の目的に使用できます (現在のところ、日本語はサポートされておりません)。

- マニュアルを表示する
- マニュアルの項目について明確化するために、ユーザによって追加された内容を確認する
- すべてのユーザのために、今後のリリースでマニュアルを改善するための提案や修正を行う

http://dcx.sybase.com を参照してください。

詳細情報の検索／テクニカル・サポートの依頼

詳しい情報やリソースについては、iAnywhere デベロッパー・コミュニティ (http://www.iAnywhere.jp/developers/index.html) 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。バージョンおよびビルド番号を調べるには、コマンド **dbeng11 -v** を実行します。

ニュースグループは、ニュース・サーバ *forums.sybase.com* にあります。

以下のニュースグループがあります。

- ianywhere.public.japanese.general

Web 開発に関する問題については、http://groups.google.com/group/sql-anywhere-web-development を参照してください。

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。

QAnywhere テクノロジの概要

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・デバイス用のアプリケーション間メッセージングの開発を支援します。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。

蓄積転送テクノロジを使用して、QAnywhere はリモート・アプリケーションとモバイル・アプリケーションに、安全で保証されたメッセージ配信機能を提供します。QAnywhere によってネットワークの速度と信頼性の低下という課題が自動的に対処されるので、接続性、通信、セキュリティに関わる問題の代わりに、アプリケーションの機能に集中的に取り組むことができます。QAnywhere の蓄積転送テクノロジによって、ネットワークに接続されていない場合でも、アプリケーションは常に使用可能になります。

QAnywhere によって、随時接続環境での通信が実現します。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。QAnywhere 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。

QAnywhere には、次のような追加機能があります。

- 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、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、優先度の低い大きなメッセージがピーク時以外の時間に配信されるようにすることなども可能です。
- 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 Mobile 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、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ます。
- プログラミング・インタフェースとして C++、Java、.NET および SQL の 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 にはモバイル Web サービス・インタフェースが含まれており、エンタープライズ Web サービスを基礎とする信頼性の高いモバイル・アプリケーションの作成を支援します。

QAnywhere は、Mobile Link 同期テクノロジを基礎として構築されています。

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 Mobile 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 認定の 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 経由の 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」 5 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」 7 ページを参照してください。

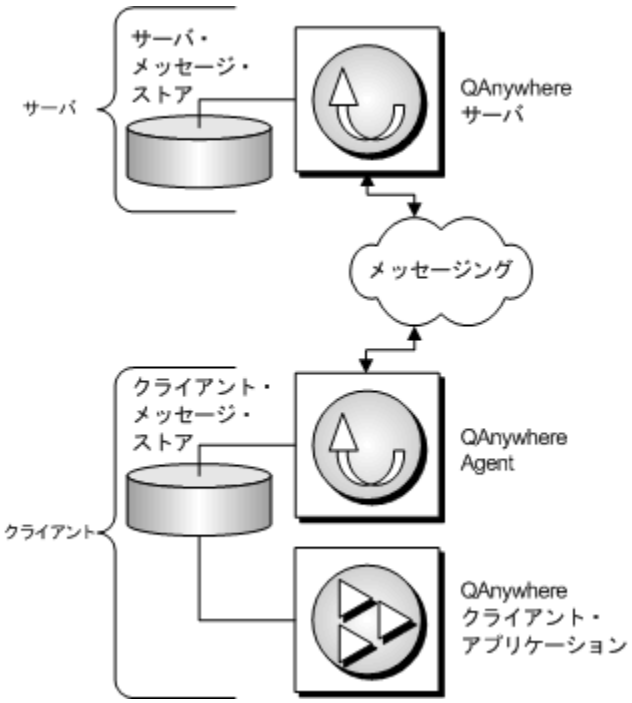
- **外部メッセージング・システムとのメッセージング**　JMS プロバイダを提供する外部システム (BEA WebLogic や Sybase EAServer など) を介して QAnywhere クライアント間でメッセージ交換を行う場合です。

  「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」 8 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。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であればいずれのデータベースでもかまいません。
- **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はリレーショナル・データベースに格納されます。QAnywhere では、SQL Anywhere データベースと Ultra Light データベースをサポートしています。データ同期アプリケーションには、SQL Anywhere データベースを推奨します。メッセージの格納と転送専用のアプリケーションには、Ultra Light データベースを推奨します。
- **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。「有効な Mobile Link での QAnywhere の起動」 32 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 の起動」 51 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」 57 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 クライアントは、「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、スケジュール設定、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。カスタム・ポリシーでは、転送ルールを使用してメッセージ転送に詳細な制御を追加します。

「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」 54 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

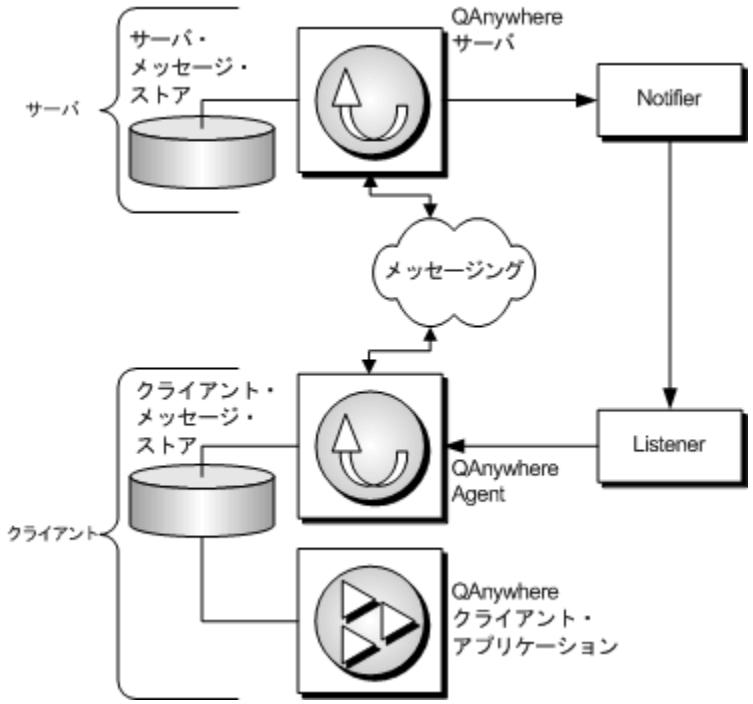
Push 通知は、サーバから QAnywhere クライアントに配信される特殊なメッセージです。Push 通知は、メッセージがサーバ・メッセージ・ストアに着信したときに送信されます。メッセージング・サーバは、受信側クライアントの Listener に Push 要求を自動的に通知します。クライアントは通知を受信するとメッセージ転送を開始し、サーバで待機しているメッセージを受信したり、カスタム・アクションを実行したりします。

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」 54 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」 5 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、簡単なメッセージング・シナリオに次のコンポーネントが追加されています。

- **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」 54 ページを参照してください。

参照

- 「Push 通知の使用方法」 37 ページ
- 「非同期的なメッセージ受信」 83 ページ
- 「Mobile Link サーバ起動同期の概要」 『Mobile Link - サーバ起動同期』

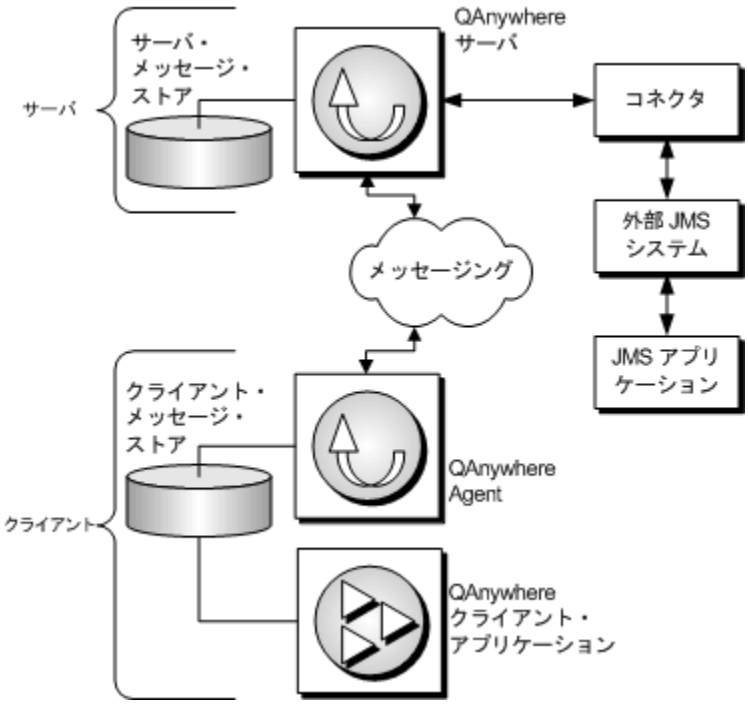
外部メッセージング・システムとのメッセージングのシナリオ

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」 5 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

- 「コネクタ」 161 ページ
- 「チュートリアル：JMS コネクタの使用」 176 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 Central または cancelMessage API 呼び出し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、もう一度配信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。

SQL Anywhere と Ultra Light のいずれを使用するかの決定

QAnywhere クライアント・アプリケーションでは、クライアント・メッセージ・ストアとして Ultra Light データベースを使用できるようになっています。これにより、純粋なメッセージング・アプリケーション用のより軽量なソリューションがモバイル・デバイスに提供されます。純粋なメッセージング・アプリケーションとは、メッセージの格納や転送はできるが、データの同期はできないアプリケーションを意味します。

Ultra Light の主な利点には、次のようなものがあります。

- アプリケーションの占有容量が小さく、SQL Anywhere のフル・インストールが不要。
- プロセスの占有容量が小さい。QAnywhere Agent では、3 つのプロセス (uleng11、dblsn、qauagent) だけが必要で、SQL Anywhere のように 4 つのプロセス (dbeng11、dbmlsync、dblsn、qaagent) は不要です。

Ultra Light の制限事項

SQL Anywhere と Ultra Light のどちらを使用するか決定する際は、Ultra Light に次のような制限があることに注意してください。

- 転送ルールの条件構文で、"ESCAPE" キーワードはサポートされていません。
- プロパティ属性のサポートが制限されています。プロパティ属性の機能は、事前に定義された ias_Network プロパティでのみ使用可能です。「カスタムのクライアント・メッセージ・ストア・プロパティ」 793 ページを参照してください。

推奨事項

一般的に、SQL Anywhere が既存していない場合は、SQL Anywhere ではなく必ず Ultra Light を使用する必要があります。SQL Anywhere は、すでに実装されている SQL Anywhere データ同期ソリューションと並行してメッセージング機能を追加したいという状況のときに使用可能です。しかし、純粋なメッセージング環境では、すべて Ultra Light を推奨します。

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 11] - [Sybase Central] を選択します。
2. **[接続] - [QAnywhere 11 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」 23 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。

   「有効な Mobile Link での QAnywhere の起動」 32 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 または Ultra Light 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」 25 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」 57 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。

   「コネクタ」 161 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 の起動」 51 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」 109 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」 209 ページ
- 「チュートリアル：JMS コネクタの使用」 176 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。
- 次の QAnywhere ニュースグループに質問を投稿できます。
  ianywhere.public.sqlanywhere.qanywhere

メッセージ

目次

QAnywhere メッセージの概要

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。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異なります。

QAnywhere では、次の定義済みメッセージ・ヘッダをサポートしています。

- メッセージ ID
- メッセージの作成時刻のタイムスタンプ
- 返信先アドレス
- メッセージ・アドレス
- メッセージの再配信のステータス
- メッセージの有効期限
- メッセージの優先度
- どのメッセージへの返信であるかを示すメッセージ ID

メッセージ・ヘッダの詳細については、「メッセージ・ヘッダ」 720 ページを参照してください。

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。「事前に定義されたメッセージ・プロパティ」 723 ページを参照してください。
- **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。「カスタムのメッセージ・プロパティ」 725 ページを参照してください。

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」 726 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
- ias_DeliveryCount
- ias_MessageType
- ias_RASNames
- ias_NetworkStatus
- ias_Originator
- ias_Status
- ias_StatusTime

メッセージ・プロパティの詳細については、「メッセージ・プロパティ」 723 ページを参照してください。

送信先の概要

QAnywhere では、メッセージは送信先宛てに送信されます。送信先は常に識別子とキュー名で構成され、その間はバックスラッシュ (¥) で分割されています。次に例を示します。

```
ianywhere.connector.tibco¥SomeQueue
DEV007¥app_queue1
SalesTeam¥queue1
```

バックスラッシュの前の識別子の意味は、メッセージの宛先が JMS アプリケーションか、送信先エイリアスか、モバイル・アプリケーションかによって異なります。

最初の例は、JMS アプリケーション宛てにメッセージを送信する場合を示しています。この場合、識別子は Mobile Link サーバで実行されている JMS コネクタの ID になります。「JMS コネクタ」 162 ページを参照してください。

2 つ目の例は、モバイル・アプリケーション宛てにメッセージを送信する場合を示しています。この場合、識別子は QAnywhere メッセージ・ストアのメッセージ・ストア ID になります。「クライアント・メッセージ・ストアの設定」 25 ページと「-id オプション」 750 ページを参照してください。

3 つ目の例は、送信先エイリアス宛てにメッセージを送信する場合を示しています。この場合、識別子は送信先エイリアス名になります。「送信先エイリアス」 158 ページを参照してください。

送信先のキュー名は、識別子が JMS コネクタ ID の場合は JMS システムで定義されたキューを指し、識別子がメッセージ・ストア ID か送信先エイリアスの場合は QAnywhere アプリケーション・キューを指します。

注意
QAnywhere の送信先は、必ず英字で指定する必要があります。

QAnywhere メッセージの送信先および送信の詳細については、次の項を参照してください。

- 「QAnywhere メッセージの送信」 73 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」 54 ページ

メッセージ・ストア

目次

メッセージ・ストアの概要

QAnywhere メッセージ・ストアは、メッセージが一時的に格納されるレポジトリです。メッセージは、サーバでは「サーバ・メッセージ・ストア」に、クライアントでは「クライアント・メッセージ・ストア」に、一時的に格納できます。指定した期間だけ保存したい場合は、サーバの「アーカイブ・メッセージ・ストア」に格納できます。

メッセージ・ストアの詳細については、次の項目を参照してください。

- 「サーバ・メッセージ・ストア」 23 ページ
- 「クライアント・メッセージ・ストア」 25 ページ

サーバ・メッセージ・ストア

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 RDBMS (MySQL と DB2 メインフレームを除く) 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。

サーバ・メッセージ・ストアの設定

サーバ・メッセージ・ストアを設定すると、アーカイブ・メッセージ・ストアが作成されます。アーカイブ・メッセージ・ストアはサーバ・メッセージ・ストアと共存するテーブルのセットで、削除待機中のすべてのメッセージがここに格納されます。定期的に実行されるシステム・プロセスでは、サーバ・メッセージ・ストアにあるメッセージのうち最終ステータスに到達したものすべてを削除し、アーカイブ・メッセージ・ストアに挿入することで、メッセージ・ストア間でのメッセージ転送を行います。メッセージは、サーバ削除ルールによって削除されるまで、アーカイブ・メッセージ・ストアに残ります。アーカイブ・メッセージ・ストアを使用することにより、同期中にフィルタする必要のあるメッセージの量が最小限に抑えられるため、サーバ・メッセージ・ストアのパフォーマンスが向上します。「アーカイブ・メッセージ・ストア要求」 185 ページを参照してください。

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。大文字と小文字が区別されない比較や文字列操作のため、統合データベースを設定する必要があります。**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 11 へのアップグレード」 『SQL Anywhere 11 - 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、次のコマンドを実行します。

```
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。

```
%SQLANY11%¥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。

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、アクティブなクライアントのクエリやメッセージ・ストア・プロパティのクエリ、メッセージのクエリなど、さまざまな機能を実行できます。

サーバ管理要求で実行できる機能や、その使用方法については、「サーバ管理要求の作成」 184 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

クライアント・メッセージ・ストアには、リモート・デバイス上の SQL Anywhere または Ultra Light データベースを使用できます。データ同期アプリケーションには、SQL Anywhere データベースを推奨します。メッセージの格納と転送専用のアプリケーションには、Ultra Light データベースを推奨します。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。Ultra Light データベースをクライアント・メッセージ・ストアとして使用する場合、QAnywhere API は、インプロセスの Ultra Light ランタイムではなく Ultra Light エンジンを使用してメッセージ・ストアにアクセスします。

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。QAnywhere API を使用する QAnywhere アプリケーション以外のアプリケーションから QAnywhere メッセージ・ストア・データベースにアクセスしないでください。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」 140 ページを参照してください。

クライアント・メッセージ・ストアの設定

♦ **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 または Ultra Light 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 - データベース管理』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 または QAnywhere Ultra Light Agent 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」 747 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。エージェントは、データベースを初期化した後、停止します。

     -si オプションを指定した qaagent を実行して QAnywhere を初期化すると、QAnywhere Agent は QAnywhere メッセージングで必要なクライアント・システム・テーブルを作成します。QAnywhere では、サーバ・システム・テーブルも使用します。サーバ・システム・テーブルは Mobile Link セットアップのインストール時に作成されます。すべての QAnywhere システム・テーブルは、ml_qa_ で始まる名前になります。これは変更できません。

     「-si オプション」 762 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」 26 ページと「-id オプション」 750 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 mlsrv11 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」 33 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」 140 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」 763 ページと「-sur オプション」 764 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 ユーティリティのリファレンス」 743 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 dbsrv11 -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 の -i および -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 ユーティリティのリファレンス」 743 ページを参照してください。

SQL Anywhere と Ultra Light クライアントの相違点

QAnywhere クライアント・アプリケーションでは、クライアント・メッセージ・ストアとして Ultra Light データベースを使用できるようになっています。これにより、純粋なメッセージング・アプリケーション用のより軽量なソリューションがモバイル・デバイスに提供されます。純粋なメッセージング・アプリケーションとは、メッセージの格納や転送はできるが、データの同期はできないアプリケーションを意味します。

Ultra Light の主な利点には、次のようなものがあります。

- アプリケーションの占有容量が小さく、SQL Anywhere のフル・インストールが不要。
- プロセスの占有容量が小さい。QAnywhere Agent では、3 つのプロセス (uleng11、dblsn、qauagent) だけが必要で、SQL Anywhere のように 4 つのプロセス (dbeng11、dbmlsync、dblsn、qaagent) は不要です。

SQL Anywhere と Ultra Light のどちらを使用するか決定する際は、Ultra Light に次のような制限があることに注意してください。

- 転送ルールの条件構文で、"ESCAPE" キーワードはサポートされていません。
- プロパティ属性のサポートが制限されています。プロパティ属性の機能は、事前に定義された ias_Network プロパティでのみ使用可能です。「カスタムのクライアント・メッセージ・ストア・プロパティ」 793 ページを参照してください。

一般的に、SQL Anywhere が既存していない場合は、SQL Anywhere ではなく必ず Ultra Light を使用する必要があります。SQL Anywhere は、すでに実装されている SQL Anywhere データ同期ソリューションと並行してメッセージング機能を追加したいという状況のときに使用可能です。しかし、純粋なメッセージング環境では、すべて Ultra Light を推奨します。

アプリケーション側から見ると、Ultra Light のクライアント API は SQL Anywhere の場合と同じです。ただし、Ultra Light メッセージ・ストアの場合は、QAManager 設定プロパティに DATABASE_TYPE=UltraLite という設定が含まれている必要があるという点が異なります。DATABASE_TYPE プロパティが設定されていない場合、デフォルトは SQLAnywhere になります。

Ultra Light でサポートされているクライアント API は、C# (Microsoft .NET の場合) と Java です。Ultra Light では、C++ と SQL の API はサポートされません。

アプリケーション側から見たもう一つの相違点は、Ultra Light 用 QAnywhere Agent が qauagent.exe であることです。Ultra Light 用 QAnywhere Agent では、QAnywhere Agent と同じオプションの多くがサポートされています。ただし、次のような例外があります。

- -sv は Ultra Light に不適用
- -pc[+|-] は qauagent でサポートされていない
- -sur は Ultra Light に不適用
- -c 接続パラメータは「Ultra Light 接続パラメータ」『Ultra Light データベース管理とリファレンス』に記載されているものに制限される

一般的に、Ultra Light 用 QAnywhere Agent では転送ルールを完全にサポートしています。制限されているのはプロパティ属性のサポートです。転送ルールでは、事前に定義された ias_Network プロパティの属性のうち、次のもののみを使用できます。

- ias_Network.Cost
- ias_Network.CommunicationAddress
- ias_Network.CommunicationType

参照

- 「カスタムのクライアント・メッセージ・ストア・プロパティ」 793 ページ
- 「qauagent ユーティリティ」 768 ページ

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。

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、事前定義またはカスタムのプロパティの名前を最初のパラメータとして渡すことができます。

「クライアント・メッセージ・ストア・プロパティの管理」 156 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」 809 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、いくつかのクライアント・メッセージ・ストア・プロパティは事前に定義されています。事前に定義されたメッセージ・ストア・プロパティは次のとおりです。

- ias_Adapters
- ias_MaxDeliveryAttempts
- ias_MaxDownloadSize
- ias_MaxUploadSize
- ias_Network
- ias_Network.Adapter
- ias_Network.RAS
- ias_Network.IP
- ias_Network.MAC
- ias_RASNames
- ias_StoreID
- ias_StoreInitialized
- ias_StoreVersion

クライアント・メッセージ・ストアの事前に定義された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。

- 「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」 792 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 (SQL Anywhere メッセージ・ストアのみ)
- **ias_** で始まるすべての名前

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。

Ultra Light をクライアント・メッセージ・ストアとして使用する際のプロパティ属性は、限定的にサポートされています。Ultra Light メッセージ・ストアでは、事前定義の ias_Network プロパティのみをサポートしています。

カスタムのクライアント・メッセージ・ストア・プロパティの使用については、次の項を参照してください。

- 「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」 793 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」 792 ページ

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」 23 ページを参照してください。

2. -m オプションを指定して mlsrv11 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。

   「有効な Mobile Link での QAnywhere の起動」 32 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」 33 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]** を選択します。

有効な Mobile Link での QAnywhere 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。「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。

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 (mlsrv11) 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 mlsrv11 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、「mlsrv11 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」 165 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 (*samples-dir¥QAnywhere¥server¥qanyserv.db*) を使用して QAnywhere のメッセージングを開始するには、次のコマンドを実行します。

```
mlsrv11 -m -c "dsn=QAnywhere 11 Demo"
```

QAnywhere のサンプルのサーバ・メッセージ・ストアでは、**QAnywhere 11 Demo** という ODBC データ・ソースを使用します。

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 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。
- mlsrv11 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」 750 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初めて同期を実行すると、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 11] - [Sybase Central]** を選択します。
   - **[接続] - [QAnywhere 11 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」 33 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 11] - [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。

リレー・サーバの使用

リレー・サーバを使用して Mobile Link サーバと通信するには、ネットワーク・プロトコルとして HTTP または HTTPS を使用するように QAnywhere Agent を設定します。

次に例を示します。

```
qaagent -c "dbf=mystore.db;eng=mystore;dbn=mystore;uid=ml_qa_user;pwd=qanywhere"
   -x http(host=webserver01;port=80;url_suffix=/ias_relay_server/client/rs_client.dll/farm1/)
```


参照

- 「リレー・サーバ」『Mobile Link - サーバ管理』
- 「Mobile Link サーバをサーバ・ファームで実行する」『Mobile Link - サーバ管理』

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」 25 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」 57 ページを参照してください。

3. QAnywhere Agent を起動します。

   「QAnywhere Agent の起動」 51 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]** を選択します。

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 のスケジュール設定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」 71 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」 54 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」 7 ページ
- 「Push 通知の通知」 71 ページ
- 「-push オプション」 759 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」 38 ページ
- 「Listener の設定」 40 ページ
- 「QAnywhere ゲートウェイの設定」 40 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」 759 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」 54 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、「サーバ起動同期の Mobile Link サーバ設定」『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 のウィンドウが表示されない。
- 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 mlsrv11 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 （resend interval)
- request_cursor プロパティの存続期間 (time to live)

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' )
```


「Notifier 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 要求の要件」 『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 イベント」 『Mobile Link - サーバ起動同期』を参照してください。

参照

- 「Notifier イベントおよびプロパティの設定」 『Mobile Link - サーバ起動同期』
- 「サーバ起動同期の Mobile Link サーバ設定」 『Mobile Link - サーバ起動同期』
- 「Notifier」 『Mobile Link - サーバ起動同期』
- 「ml_add_property システム・プロシージャ」 『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設定されています。ただし、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』
- 「Windows デバイス用の Listener ユーティリティ」 『Mobile Link - サーバ起動同期』
- 「QAnywhere ゲートウェイの設定」 40 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、「サーバ起動同期の Mobile Link サーバ設定」『Mobile Link - サーバ起動同期』と「ゲートウェイ・プロパティ」 『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」 40 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」 759 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

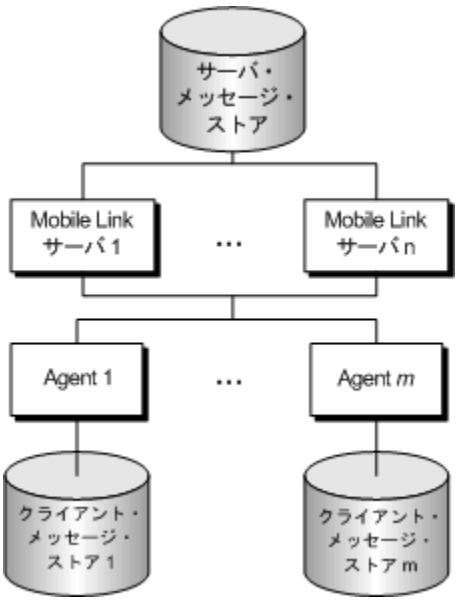
QAnywhere アプリケーションは、フェールオーバ・メカニズムを使用するように設定できます。フェールオーバ・メカニズムを使用すると、Mobile Link サーバで障害が発生しても代替サーバを使用できます。フェールオーバをサポートするには、各 QAnywhere Agent を起動するときに、Mobile Link サーバのリストを指定する必要があります。リストの先頭に指定された Mobile Link サーバがプライマリ・サーバになります。残りのサーバは代替サーバになります。

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・ストアが存在するコンピュータ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」 766 ページ
- 「-fd オプション」 748 ページ
- 「-fr オプション」 749 ページ
- 「QAnywhere Agent の起動」 51 ページ

QAnywhere Agent の概要

目次

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。

メッセージ転送ポリシー

QAnywhere クライアントは、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、次のようなものがあります。

- 「スケジュール設定ポリシー」 44 ページ
- 「自動ポリシー」 45 ページ
- 「オン・デマンド・ポリシー」 46 ページ
- 「カスタム・ポリシー」 46 ページ

スケジュール設定ポリシーは、指定された時間間隔でメッセージの転送を開始します。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。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。カスタム・ポリシーでは、転送ルールを使用してメッセージ転送に詳細な制御を追加します。

スケジュール設定ポリシー

スケジュール設定ポリシーは、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。

スケジュール設定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]** ウィンドウで **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。

スケジュール設定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を受信した。
- 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 列挙体」 226 ページ
  - C++ :「AcknowledgementMode クラス」 406 ページ
  - Java :「AcknowledgementMode インタフェース」 524 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・ステータス変更通知を受信した。
- 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」 317 ページ
- C++：「triggerSendReceive 関数」 477 ページ
- Java：「triggerSendReceive メソッド」 599 ページ
- SQL：「ml_qa_triggersendreceive」 718 ページ

自動ポリシー

自動ポリシーは、メッセージが送受信されるたびに同期を行い、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。このポリシーは、頻繁にメッセージを送受信するアプリケーションにはおすすめしません。

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」 301 ページ
  - C++：「putMessage 関数」 468 ページ
  - Java：「putMessage メソッド」 588 ページ
  - SQL：「ml_qa_putmessage」 717 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 列挙体」 226 ページ
  - C++：「AcknowledgementMode クラス」 406 ページ
  - Java：「AcknowledgementMode インタフェース」 524 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」 37 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」 71 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」 317 ページ
  - C++：「triggerSendReceive 関数」 477 ページ
  - Java：「triggerSendReceive メソッド」 599 ページ
  - SQL：「ml_qa_triggersendreceive」 718 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」 317 ページ
- C++：「triggerSendReceive 関数」 477 ページ
- Java：「triggerSendReceive メソッド」 599 ページ
- SQL：「ml_qa_triggersendreceive」 718 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」 70 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。

アプリケーション用にカスタムのポリシー・ルールを作成するときは、メッセージが他のルールによって誤って見過ごされないように、すべてを含むデフォルトのルールを含めることをおすすめします。たとえば、次のルールでは最低 1 日前のメッセージを同期します。

```
auto=DATEADD( day, 1, ias_statustime ) < ias_currenttimestamp
```

次に、同期の有効性に影響する要素のリストを示します。自分でカスタムのポリシー・ルールを作成する際は、これらの要素を考慮する必要があります。

- メッセージ・サイズ
- 同期の頻度
- 帯域幅とネットワークの信頼性
- 優先度が高いメッセージ
- データ転送コスト

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」 810 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」 301 ページ
  - C++：「putMessage 関数」 468 ページ
  - Java：「putMessage メソッド」 588 ページ
  - SQL：「ml_qa_putmessage」 717 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 列挙体」 226 ページ
  - C++：「AcknowledgementMode クラス」 406 ページ
  - Java：「AcknowledgementMode インタフェース」 524 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」 37 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」 317 ページ
  - C++：「triggerSendReceive 関数」 477 ページ
  - Java：「triggerSendReceive メソッド」 599 ページ
  - SQL：「ml_qa_triggersendreceive」 718 ページ

転送ステータスの概要

メッセージの転送ステータスを判断するには、Sybase Central を使用する方法が最も簡単です。**[メッセージ・プロパティ]** ウィンドウの **[一般]** タブを開き、メッセージの転送ステータスを表示します。値は次のいずれかです。

- **transmitted**　メッセージは送信されました。
- **transmitting**　メッセージは送信中です。
- **untransmitted**　メッセージは送信されていません。
- **do_not_transmit**　メッセージは送信されません。

メッセージのステータスの概要

メッセージのステータスを判断するには、Sybase Central を使用する方法が最も簡単です。**[メッセージ・プロパティ]** ウィンドウの **[一般]** タブを開き、メッセージのステータスを表示します。値は次のいずれかです。

- **保留**　メッセージは送信されましたが、まだ受信されていません。
- **受信中**　メッセージは受信中であるか、または受信されましたがまだ確認されていません。
- **最終**　メッセージは最終ステータスになりました。
- **失効**　有効期限が切れる前にメッセージが受信されませんでした。
- **キャンセル済み**　メッセージはキャンセルされました。
- **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ませんでした。
- **受信済み**　メッセージは受信され、確認されました。

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。

QAnywhere では、ルールとはメッセージの転送を発生させる時期、転送するメッセージ、メッセージを削除する時期を決定する論理のことです。ルールは、サーバ側とクライアント側で指定できます。

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。

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。転送ルールは、サーバとクライアントの両方に対して指定できます。

転送ルール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。

- 「クライアント側の転送ルール」 819 ページ
- 「サーバ側の転送ルール」 820 ページ
- 「ルールの構文」 810 ページ
- 「ルール変数」 816 ページ

削除ルール

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。削除ルールは、サーバとクライアントの両方に対して指定できます。

削除ルールの使用については、「メッセージの削除ルール」 823 ページを参照してください。

QAnywhere Agent の起動

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]** ウィンドウで **[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」 747 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]** ウィンドウの **[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」 750 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]** ウィンドウで **[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」 766 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 ユーティリティ」 744 ページを参照してください。

Windows Mobile 上での qaagent の起動

Windows Mobile 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」 761 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」 750 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、QAnywhere Agent のメッセージ・ウィンドウで **[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。

参照

- 「qastop ユーティリティ」 789 ページ
- 「-qi オプション」 761 ページ

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」 759 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 または Ultra Light 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、データベース・サーバが必要です。たとえば、Windows Mobile でのデータベース・サーバは *dbsrv11.exe* です。Windows でのデータベース・サーバは、パーソナル・データベース・サーバ *dbeng11.exe* 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」 747 ページを参照してください。

QAnywhere Agent の配備

配備の詳細については、「QAnywhere アプリケーションの配備」 134 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、カスタム・ポリシーと、スケジュール設定、自動、オン・デマンドの 3 つの事前定義済みポリシーがあります。

ポリシーは、次の方法で指定できます。

- Sybase Central の QAnywhere プラグインを使用し、**[ツール] - [QAnywhere 11] - [SQL Anywhere のエージェント設定ファイルの新規作成]** を選択する。ポリシーは、**エージェント設定ファイルの [プロパティ]** ウィンドウの **[一般]** タブで指定します。このタスクにより、Sybase Central の規則に従って拡張子 *.qaa* の付いたファイルが作成されます。

  Sybase Central の QAnywhere プラグインを使用してカスタムのプロパティを指定するには、**[ツール] - [QAnywhere 11] - [新しいエージェント・ルール・ファイル]** を選択します。このタスクにより、Sybase Central の規則に従って拡張子 *.qar* の付いたファイルが作成されます。
- -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。

参照

- 「メッセージ転送ポリシー」 44 ページ
- 「転送ルール」 49 ページ
- 「-policy オプション」 758 ページ

信頼性の低いネットワークでの対応

インクリメンタル・アップロードとインクリメンタル・ダウンロードを使用しないと、メッセージは単体で送信されるため、メッセージのアップロード中またはダウンロード中にネットワーク接続が失われた場合にメッセージの転送が失敗します。インクリメンタル・アップロードとインクリメンタル・ダウンロードを使用すれば、大きなメッセージを小さなメッセージ・パーツに分割できます。メッセージを小さなパーツに分けて、各メッセージ・パーツを個別に送信できるため、何度か同期を行いながらメッセージを段階的にアップロードまたはダウンロードできます。送信先ですべてのメッセージ・パーツが受信されると、完全なメッセージが受信されたことになります。

インクリメンタル・アップロードを実装する方法については、「-iu オプション」 751 ページを参照してください。

インクリメンタル・ダウンロードを実装する方法については、「-idl オプション」 751 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、Java 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 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」 217 ページを参照してください。

「クライアント用 QAnywhere .NET API (.NET 2.0)」 226 ページを参照してください。

QAnywhere C++ API

QAnywhere C++ API は、Microsoft Visual Studio 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。

- *install-dir¥sdk¥include* 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*)

- *install-dir¥sdk¥lib¥x86* と *install-dir¥sdk¥lib¥ce¥arm.50* にあるインポート・ライブラリ (*qany11.lib*)
- *install-dir¥bin32* と *install-dir¥ce¥arm.50* にあるランタイム DLL (*qany11.dll*)

API にアクセスするには、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。

「QAnywhere C++ API リファレンス」 405 ページを参照してください。

QAnywhere Java API

QAnywhere Java API は JRE 1.4.2 以上をサポートしています。モバイル Web サービスの wsdl コンパイラでは、JDK 1.5.0 以上と互換性を持つ Java クラスが生成されます。

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 11 インストール環境のサブディレクトリ *documentation¥ja¥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 11 インストール環境のサブディレクトリ *bin32* にある Ultra Light メッセージ・ストア用のランタイム DLL (*qadbiuljni.dll*)
- SQL Anywhere 11 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」 523 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」 677 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」 8 ページ
- 「JMS コネクタ」 162 ページ
- 「チュートリアル：JMS コネクタの使用」 176 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」 109 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」 62 ページ
   - 「C++ アプリケーションの設定」 64 ページ
   - 「Java アプリケーションの設定」 66 ページ
   - 「SQL アプリケーションの設定」 67 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」 98 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。
   - 「QAnywhere メッセージの概要」 16 ページ
   - 「クライアント・メッセージ・ストア・プロパティ」 28 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」 134 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」 209 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 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 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 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 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、**[参照の追加]** を選択します。
3. **[.NET]** タブで、**[参照]** をクリックして *iAnywhere.QAnywhere.Client.dll* を検索します。デフォルト・ロケーションは次のとおりです。
   - .NET Framework 2.0：*install-dir¥Assembly¥v2*
   - .NET Compact Framework 2.0：*install-dir¥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]** ウィンドウを開き、**[.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、ウィンドウを閉じます。

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 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 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。次に例を示します。

```
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。「マルチスレッドでの考慮事項」 97 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」 318 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 列挙体」 226 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

- 「クライアント用 QAnywhere .NET API (.NET 2.0)」 226 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」 478 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」 442 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。「マルチスレッドでの考慮事項」 97 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」 101 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」 99 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」 406 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」 405 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。デフォルトでは、このファイルは *install-dir¥Java* 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。「マルチスレッドでの考慮事項」 97 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」 524 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」 523 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 のほど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」 57 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」 715 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」 677 ページ

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」 25 ページを参照してください。

各アドレスに関連付けられるアプリケーションは、一度に 1 つまでです。同じアドレスで複数のアプリケーションを実行すると、メッセージ取得中に不確定な動作が発生する場合があります。

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」 70 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・プロパティの設定」 165 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。

- 「JMS コネクタへの QAnywhere メッセージの送信」 167 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」 169 ページ
- 「コネクタ」 161 ページ

システム・キュー

system というキューは、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」 70 ページ
- 「Push 通知の通知」 71 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。各文字列はパイプ記号で区切られます。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」 231 ページ
  - C++：「ADAPTER 変数」 409 ページ
  - Java：「ADAPTERS 変数」 527 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。各文字列はパイプ記号で区切られます。次の項を参照してください。

  - .NET：「RASNAMES フィールド」 236 ページ
  - C++：「RASNAMES 変数」 413 ページ
  - Java：「RASNAMES 変数」 531 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」 234 ページ
  - C++：「NETWORK_STATUS 変数」 412 ページ
  - Java：「NETWORK_STATUS 変数」 530 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」 723 ページの ias_MessageType
- 「システム・キュー」 69 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」 7 ページ
- 「Push 通知の使用方法」 37 ページ
- 「システム・キュー」 69 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」 723 ページの ias_MessageType
- .NET：「MessageProperties クラス」 229 ページ
- C++：「MessageProperties クラス」 408 ページ
- Java：「MessageProperties インタフェース」 525 ページ

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」 54 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。

   「QAnywhere メッセージの概要」 16 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」 301 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」 54 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。

「QAnywhere メッセージの概要」 16 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」 468 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」 54 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。

   「QAnywhere メッセージの概要」 16 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」 588 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」 54 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」 688 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」 717 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」 54 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」 349 ページ
- C++ クライアント：「QATransactionalManager クラス」 512 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」 318 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」 349 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」 512 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」 318 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」 282 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」 453 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」 572 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。

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

「GetMessageNoWait メソッド」 293 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。

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

「getMessageNoWait 関数」 464 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」 581 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」 713 ページ
- 「ml_qa_getmessagetimeout」 714 ページ
- 「ml_qa_getmessage」 711 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、MessageListener 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 デリゲート」 228 ページと「SetMessageListener メソッド」 308 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」 506 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」 473 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」 623 ページと「setMessageListener メソッド」 593 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*** という名前のストアド・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」 716 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」 812 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」 290 ページと「GetMessageBySelectorNoWait メソッド」 291 ページ
- C++：「getMessageBySelector 関数」 462 ページと「getMessageBySelectorNoWait 関数」 463 ページ
- Java：「getMessageBySelector メソッド」 579 ページと「getMessageBySelectorNoWait メソッド」 580 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 1 MB、Windows Mobile の場合は 64 KB) 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」 244 ページと「ReadText メソッド」 348 ページ
- C++：「readBinary 関数」 422 ページと「readText 関数」 510 ページ
- Java：「readBinary メソッド」 540 ページと「readText メソッド」 629 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」 723 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」 278 ページ
- C++：「browseMessages 関数」 450 ページ
- Java：「browseMessages メソッド」 570 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」 281 ページ
- C++：「browseMessagesByQueue 関数」 451 ページ
- Java：「browseMessagesByQueue メソッド」 571 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」 280 ページ
- C++：「browseMessagesByID 関数」 450 ページ
- Java：「browseMessagesByID メソッド」 570 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」 812 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」 282 ページ
- C++：「browseMessagesBySelector 関数」 452 ページ
- Java：「browseMessagesBySelector 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。

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。

QAException がメッセージ・リスナ委任でスローされ、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 デリゲート」 228 ページ
- 「MessageListener2 デリゲート」 229 ページ
- 「ExceptionListener デリゲート」 227 ページ
- 「ExceptionListener2 デリゲート」 227 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」 98 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、メッセージの受信を無制限に試みます。

詳細については、次の項を参照してください。

- 「QAException クラス」 258 ページ
- 「ErrorCode プロパティ」 268 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」 792 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」 433 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」 459 ページと「getLastErrorMsg 関数」 460 ページ
- QAManagerFactory：「getLastError 関数」 480 ページと「getLastErrorMsg 関数」 481 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。

QAException がメッセージ・リスナでスローされ、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」 623 ページ
- 「QAMessageListener2 インタフェース」 625 ページ
- 「QAException クラス」 553 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」 98 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、メッセージの受信を無制限に試みます。

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」 268 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」 792 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。

エラー値	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	クライアントのプロパティ・ファイルにアクセスできません。
1003	宛先が指定されていません。
1004	この関数は実装されていません。
1005	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。
1006	メッセージをクライアント・メッセージ・ストアに格納中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中にエラーが発生しました。
1008	バックグラウンド・スレッドを初期化中にエラーが発生しました。
1009	メッセージ・ストアへの接続を開くときにエラーが発生しました。
1010	クライアントのプロパティ・ファイルに無効なプロパティが存在します。
1011	ログ・ファイルを開くときにエラーが発生しました。
1012	予期しないメッセージの終わりに到達しました。
1013	メッセージ・ストアがデバイスの空きディスク領域と比べて大きすぎます。
1014	メッセージ・ストアがメッセージング向けに初期化されていません。
1015	キューの深さの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、指定された宛先には 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。すでに送信されたメッセージをキャンセルすることはできません。
1020	メッセージの確認中にエラーが発生しました。

エラー値	説明
1021	QAManager が開いていません。
1022	QAManager はすでに開いています。
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。
1025	サーバの最大同時要求数が少なすぎるため QAManager を開くことができません (データベース 「-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)。
1026	メッセージ・ストアからプロパティを取得中にエラーが発生しました。
1027	プロパティをメッセージ・ストアに格納中にエラーが発生しました。

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」 316 ページ
- C++：「stop 関数」 476 ページ
- Java：「stop メソッド」 599 ページ
- SQL：SQL API では、QAnywhere を停止できません。

マルチスレッドでの考慮事項

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されます。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、ブロックされます。

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」 99 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」 101 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 ～ 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。

  このプロパティはスタンドアロン・クライアントではサポートされていません。

  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」 139 ページを参照してください。

- **DATABASE_TYPE=string**　QAnywhere Manager が接続されているデータベースの種類を指定します。SQL データベースの場合は **sqlanywhere**、Ultra Light データベースの場合は **ultralite** を使用します。デフォルトでは、QAnywhere Manager は **sqlanywhere** を使用します。

- **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 1 MB、Windows Mobile では 64 KB です。

次に、スタンドアロン・クライアント専用のプロパティを示します。

- **ML_PROTOCOL_TYPE**　プロトコル・タイプを指定します。有効なオプションは、**tcpip**、**tls**、**http**、**https** です。

- **ML_PROTOCOL_PARAMS** Mobile Link 接続パラメータを指定します。
- **ML_PROTOCOL_USENAME** QAnywhere Agent の -mu オプションと同じ効果があります。
- **ML_PROTOCOL_PASSWORD** QAnywhere Agent の -mn オプションと同じ効果があります。
- **INC_UPLOAD** QAnywhere Agent の -iu オプションと同じ効果があります。
- **INC_DOWNLOAD** QAnywhere Agent の -idl オプションと同じ効果があります。
- **STORE_ID** QAnywhere Agent の -id オプションと同じ効果があります。
- **STORE_ENCRYPTION_KEY** メッセージ・ストアを暗号化するための暗号化キーを指定します。
- **POLICY** QAnywhere Agent の -policy オプションと同じ効果があります。
- **DELETE_PERIOD** 削除ルールの実行間隔を秒数で指定します。負数の値が指定された場合、削除ルールの実行は無効になります。
- **PUSH** QAnywhere Agent の -push オプションと同じ効果があります。

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すると、設定ファイルの **[プロパティ]** ウィンドウが開きます。このウィンドウ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」 320 ページ
- C++ API：「createQAManager 関数」 478 ページ
- Java API：「createQAManager メソッド」 601 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」 101 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」 98 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
DATABASE_TYPE=sqlanywhere
MAX_IN_MEMORY_MESSAGE_SIZE=2048
COMPRESSION_LEVEL=0
```

設定ファイルの参照

次のような内容で *mymanager.props* という名前の QAnywhere Manager の設定プロパティ・ファイルがあるとします。

```
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」 268 ページと「QAManagerFactory クラス」 318 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」 442 ページと「QAManagerFactory クラス」 478 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」 600 ページと「QAManager インタフェース」 562 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」 98 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.SetProperty( "DATABASE_TYPE", "sqlanywhere" );
mgr.Open( AcknowledgeMode.EXPLICIT_ACKNOWLEDGEMENT );
```

.NET API については、「QAManager インタフェース」 268 ページと「QAManagerFactory クラス」 318 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->setProperty( "DATABASE_TYPE", "sqlanywhere" );
mgr->open( AcknowledgementMode::EXPLICIT_ACKNOWLEDGEMENT );
```

C++ API については、「QAManager クラス」 442 ページと「QAManagerFactory クラス」 478 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.setStringProperty("DATABASE_TYPE", "sqlanywhere");
mgr.open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
```

Java API については、「QAManagerFactory クラス」 600 ページと「QAManager インタフェース」 562 ページを参照してください。

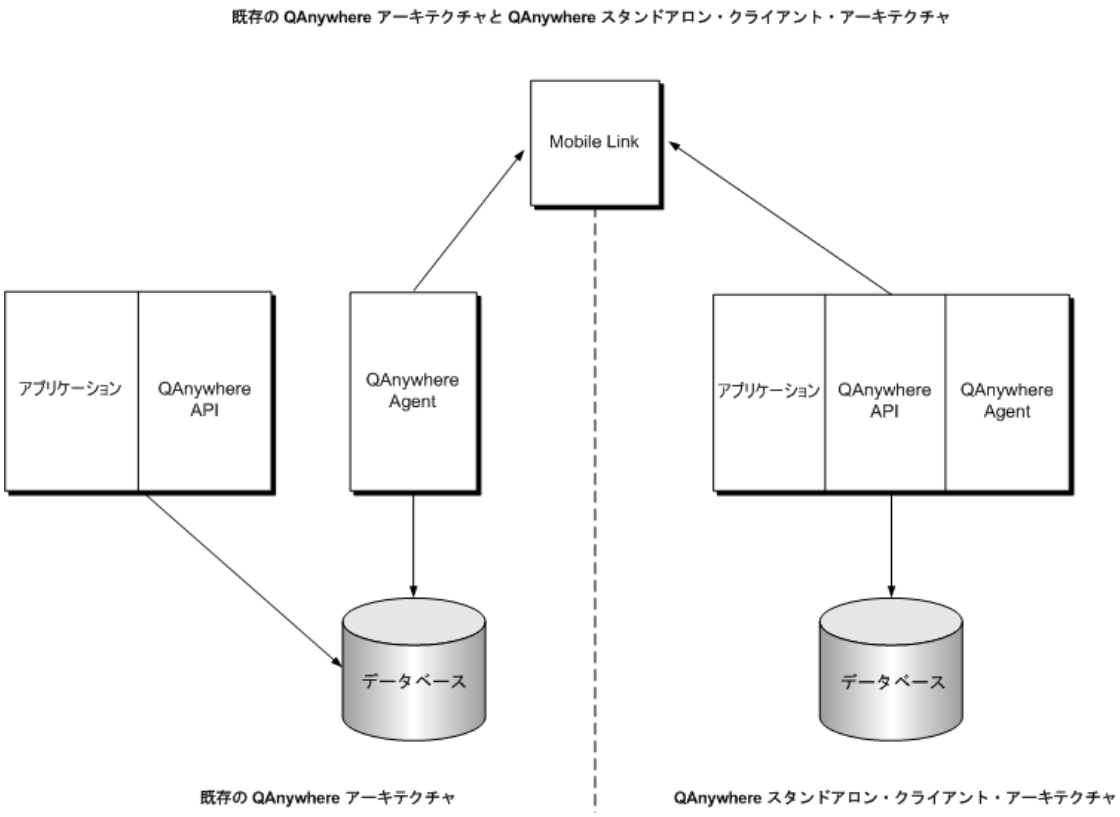
QAnywhere スタンドアロン・クライアント

目次

QAnywhere スタンドアロン・クライアントの概要

QAnywhere スタンドアロン・クライアントは、QAnywhere Agent を実行したり、データベースを管理したりせずに、メッセージング・システムを設定できる小型のクライアントです。スタンドアロン・クライアントでは、クライアント・ストア管理と QAnywhere Agent 機能の両方が、クライアント API にアクセスする同じプロセスに組み込まれています。そのため、クライアント・メッセージ・ストアの作成や保持が不要になり、QAnywhere Agent を個別のプロセスとして実行する必要はありません。

次の図に、スタンドアロン・クライアント・アーキテクチャと既存の QAnywhere アーキテクチャを示します。



スタンドアロン・クライアント・メッセージ・ストアの概要

メッセージ・ストアは、ストア ID にバインドされている Ultra Light データベースであり、スタンドアロン・クライアントによって自動的に作成および保持されます。QAnywhere API は、Ultra Light エンジンではなくインプロセスの Ultra Light ランタイムを使用してメッセージ・ストアにアクセスします。

スタンドアロン・クライアント・メッセージ・ストアが既存の QAnywhere メッセージ・ストアと異なるのは、一度に 1 つのユーザ・アプリケーションのみがメッセージ・ストアにアクセスできるという点です。つまり、同じデバイス上にある複数のスタンドアロン・クライアント・アプリケーションには、別々のメッセージ・ストアと個別のストア ID が必要です。この結果、スタンドアロン・クライアントには「ローカル・メッセージング」という概念がなく、メッセージが同じメッセージ・ストア内の別々のキューに送信される可能性があります。スタンドアロン・クライアントを介して送信されるすべてのメッセージは、別のメッセージ・ストアに対するものと見なされます。

クライアント・アプリケーションが実行していない場合でもメッセージはメッセージ・ストア内に存在する可能性があるため、最初の使用時に STORE_ENCRYPTION_KEY 値を設定することで、メッセージ・ストアを保護することができます。それ以降、ユーザがメッセージ・ストアを使用しようとするたびに、同じキーを指定する必要があります。この暗号化キーは、ディスク上のデータも暗号化します。「QAnywhere Manager の設定プロパティ」 98 ページを参照してください。

スタンドアロン・クライアントの配備

QAnywhere スタンドアロン・クライアントは、.NET 用の個別の DLL ファイルと Java 用の個別の JAR ファイルで配布されます。

- .NET 実装は、*iAnywhere.QAnywhere.StandAloneClient.dll* という DLL で配布され、ネームスペース iAnywhere.QAnywhere.StandAloneClient を使用します。「クライアント用 QAnywhere .NET API (.NET 2.0)」 226 ページを参照してください。
- Java 実装は、*qastandaloneclient.jar* という JAR ファイルで配布され、パッケージ名 ianywhere.qanywhere.standaloneclient を使用します。「クライアント用 QAnywhere Java API」 524 ページを参照してください。

スタンドアロン・クライアント API

スタンドアロン・クライアントの Java 実装と .NET 実装では、既存のクライアントと同じ API が保持されていますが、次の例外があります。

次の QAManager 設定プロパティは、スタンドアロン・クライアント専用です。

- **ML_PROTOCOL_TYPE** プロトコル・タイプを指定します。有効なオプションは、tcpip、tls、http、https です。
- **ML_PROTOCOL_PARAMS** Mobile Link 接続パラメータを指定します。
- **ML_PROTOCOL_USENAME** Mobile Link ユーザ名を指定します。QAnywhere Agent の -mu オプションと同じです。「-mu オプション」 754 ページを参照してください。
- **ML_PROTOCOL_PASSWORD** Mobile Link ユーザの新しいパスワードを指定します。QAnywhere Agent の -mn オプションと同じです。「-mn オプション」 753 ページを参照してください。
- **INC_UPLOAD** インクリメンタル・アップロードのサイズを指定します。QAnywhere Agent の -iu オプションと同じです。「-iu オプション」 751 ページを参照してください。
- **INC_DOWNLOAD** インクリメンタル・ダウンロードのサイズを指定します。QAnywhere Agent の -idl オプションと同じです。「-idl オプション」 751 ページを参照してください。
- **STORE_ID** スタンドアロン・クライアント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。QAnywhere Agent の -id オプションと同じです。「-id オプション」 750 ページを参照してください。
- **STORE_ENCRYPTION_KEY** メッセージ・ストアの暗号化に使用する暗号化キーを指定します。
- **POLICY** 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。QAnywhere Agent の -policy オプションと同じです。「-policy オプション」 758 ページを参照してください。
- **DELETE_PERIOD** 最終ステータスに達したメッセージを削除する間隔を秒数で指定します。
- **PUSH** Push 通知の配信方法を指定します。QAnywhere Agent の -push オプションと同じです。「-push オプション」 759 ページを参照してください。

次の QAManager 設定プロパティは、QAnywhere スタンドアロン・クライアントでサポートされていません。

- CONNECT_PARAMS
- DATABASE_TYPE

参照

- 「QAnywhere Manager の設定プロパティ」 98 ページ
- 「qaagent ユーティリティ」 744 ページ

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、接続された環境での場合と同じように 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、直列化などの処理を行います。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 11 - 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」 23 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。

   「有効な Mobile Link での QAnywhere の起動」 32 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」 25 ページを参照してください。

4. iAnywhere 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。

   「iAnywhere WSDL コンパイラの実行」 112 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」 114 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」 172 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 の起動」 51 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

- Java で天気予報 Web サービスを設定する方法を示す例は、「モバイル Web サービスの例」 124 ページで述べられています。
- 通貨換算 Web サービスを使用したモバイル Web サービスのサンプル・アプリケーションは、*samples-dir¥QAnywhere¥MobileWebServices* 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。このサンプルは、Java と C# の両方で提供されています。
- 次の QAnywhere ニュースグループに質問を投稿できます。
  ianywhere.public.sqlanywhere.qanywhere

モバイル Web サービス開発のヒント

- モバイル Web サービスの .NET アプリケーションについては、アプリケーションの実行プログラムに、WSDL コンパイラで生成されたプロキシ・クラスをコンパイルする必要があります。これらのプロキシ・クラスをそれ自体のアセンブリにコンパイルすることはできません。
- モバイル Web サービスの Java アプリケーションについては、JDK 1.5.x を使用する必要があります。
- iAnywhere WSDL コンパイラは、Char データ型をサポートしていません。Char の代わりに String データ型を使用することをおすすめします。

iAnywhere WSDL コンパイラの実行

iAnywhere WSDL コンパイラは、Web サービスが記述された WSDL ソースから、Java プロキシ・クラス、C# プロキシ・クラス、または SQL Anywhere 用の SQL SOAP クライアント・プロシージャのセットを生成します。ユーザはこれらをアプリケーションに含めることができます。

WSDL コンパイラで生成される Java クラスまたは C# クラスは、QAnywhere で使用します。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 ianywhere.qanywhere.ws.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：「Web サービス用 QAnywhere .NET API (.NET 2.0)」 354 ページ
- Java：「Web サービス用 QAnywhere Java API」 642 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 [*options*] *wsdl-uri*

wsdl-uri：

これは WSDL (Web Services Description Language) ソース (URL またはファイル) の指定です。

options：

- **-h**　ヘルプ・テキストを表示します。
- **-v**　冗長情報を表示します。
- **-o *output-directory***　生成ファイルの出力ディレクトリを指定します。
- **-l *language***　生成されたファイルの言語を指定します。これは、**java**、**cs**、または **sql** のいずれかです。このオプションは、必ず小文字で指定してください。
- **-d**　iAnywhere カスタマ・サポートに問い合わせるときに役立つデバッグ情報を表示します。

Java 固有の options：

- **-p *package***　パッケージ名を指定します。これを指定すると、デフォルトのパッケージ名を上書きできます。

C# 固有の options：

- **-n *namespace***　ネームスペースを指定します。これを指定すると、生成されたクラスを特定のネームスペースにラップできます。

SQL 固有の options :

- **-f *filename*** (必須) SQL 文が書き込まれる出力 SQL ファイルの名前を指定します。同じ名前のファイルが既存している場合は、この操作で上書きされます。
- **-p=*prefix*** 生成される関数またはプロシージャ名のプレフィクスを指定します。デフォルトのプレフィクスは、サービス名とピリオドです (例 : "WSDish.")。
- **-x** 関数の定義ではなくプロシージャの定義を生成します。

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 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 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 を指定します。
- 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 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 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 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、**[参照の追加]** を選択します。
3. **[参照]** タブで、次のディレクトリにある *iAnywhere.QAnywhere.Client.dll* と *iAnywhere.QAnywhere.WS.dll* を探します。
   - .NET Framework 2.0：*install-dir¥Assembly¥v2*
   - .NET Compact Framework 2.0：*install-dir¥ce¥Assembly¥v2*

   使用環境に応じた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、ソリューション・エクスプローラで **[参照]** ツリーを展開します。リストに *iAnywhere.QAnywhere.Client.dll* と *iAnywhere.QAnywhere.WS.dll* が表示されています。

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 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 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* および *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。「マルチスレッドでの考慮事項」 97 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」 318 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」 359 ページ
- WSResult：「Acknowledge メソッド」 373 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

- 「Web サービス用 QAnywhere .NET API (.NET 2.0)」 354 ページ
- 「クライアント用 QAnywhere .NET API (.NET 2.0)」 226 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¥¥" );
```

CurrencyConvertor サンプルの応答時間は、Web サービスの可用性によって異なります。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利用不可能になるときがある場合に便利です。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。「非同期の Web サービス要求」 121 ページを参照してください。

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* にあります。
   - *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。「マルチスレッドでの考慮事項」 97 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」 646 ページ
- WSResult :「acknowledge メソッド」 653 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

- 「Web サービス用 QAnywhere Java API」 642 ページ
- 「クライアント用 QAnywhere Java API」 524 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、*install-dir¥Java* にある *iawsrt.jar* です。

C# のランタイム・ライブラリは、*iAnywhere.QAnywhere.WS.dll* で、次のディレクトリにあります。

- .NET Framework 2.0：*install-dir¥Assembly¥v2*
- .NET Compact Framework 2.0：*install-dir¥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)

クラスパスに次のファイルを含めます。これらのファイルは *install-dir¥Java* にあります。

- *jaxrpc.jar*
- *qaclient.jar*
- *iawsrt.jar*

必要なランタイム・ライブラリ (.NET)

SQL Anywhere 11 をインストールすると、次のファイルはグローバル・アセンブリ・キャッシュ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」 121 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」 121 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();
   }
}
```

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。数分で作成可能なこのアプリケーションでは、QAnywhere の蓄積転送機能を使用します。これにより、オフラインのときでも天気予報に対する要求を発行し、利用可能になったら天気予報を見ることができます。

Global Weather Web サービス

次のコードは、Global Weather という Web サービスを記述したものです。公開されている天気予報 Web サービスからコピーした wsdl ファイルです。コードをファイルにコピーし、そのファイルに *globalweather.wsdl* という名前を付けます。

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" 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.webserviceX.NET"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:mime="http://schemas.xmlsoap.org/
wsdl/mime/" targetNamespace="http://www.webserviceX.NET" xmlns:wsdl="http://schemas.xmlsoap.org/
wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://www.webserviceX.NET">
      <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:element name="GetCitiesByCountry">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="CountryName" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetCitiesByCountryResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetCitiesByCountryResult" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="string" nillable="true" type="s:string" />
    </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:message name="GetCitiesByCountrySoapIn">
    <wsdl:part name="parameters" element="tns:GetCitiesByCountry" />
  </wsdl:message>
  <wsdl:message name="GetCitiesByCountrySoapOut">
    <wsdl:part name="parameters" element="tns:GetCitiesByCountryResponse" />
  </wsdl:message>
  <wsdl:message name="GetWeatherHttpGetIn">
    <wsdl:part name="CityName" type="s:string" />
    <wsdl:part name="CountryName" type="s:string" />
  </wsdl:message>
  <wsdl:message name="GetWeatherHttpGetOut">
    <wsdl:part name="Body" element="tns:string" />
  </wsdl:message>
  <wsdl:message name="GetCitiesByCountryHttpGetIn">
    <wsdl:part name="CountryName" type="s:string" />
  </wsdl:message>
  <wsdl:message name="GetCitiesByCountryHttpGetOut">
    <wsdl:part name="Body" element="tns:string" />
  </wsdl:message>
  <wsdl:message name="GetWeatherHttpPostIn">
    <wsdl:part name="CityName" type="s:string" />
    <wsdl:part name="CountryName" type="s:string" />
  </wsdl:message>
  <wsdl:message name="GetWeatherHttpPostOut">
    <wsdl:part name="Body" element="tns:string" />
  </wsdl:message>
  <wsdl:message name="GetCitiesByCountryHttpPostIn">
    <wsdl:part name="CountryName" type="s:string" />
  </wsdl:message>
  <wsdl:message name="GetCitiesByCountryHttpPostOut">
    <wsdl:part name="Body" element="tns:string" />
  </wsdl:message>
  <wsdl:portType name="GlobalWeatherSoap">
    <wsdl:operation name="GetWeather">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get weather report for all major cities
around the world.</documentation>
      <wsdl:input message="tns:GetWeatherSoapIn" />
      <wsdl:output message="tns:GetWeatherSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="GetCitiesByCountry">
     <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get all major cities by country name(full /
part).</documentation>
      <wsdl:input message="tns:GetCitiesByCountrySoapIn" />
      <wsdl:output message="tns:GetCitiesByCountrySoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="GlobalWeatherHttpGet">
    <wsdl:operation name="GetWeather">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get weather report for all major cities
around the world.</documentation>
      <wsdl:input message="tns:GetWeatherHttpGetIn" />
      <wsdl:output message="tns:GetWeatherHttpGetOut" />
    </wsdl:operation>
    <wsdl:operation name="GetCitiesByCountry">
     <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get all major cities by country name(full /
part).</documentation>
      <wsdl:input message="tns:GetCitiesByCountryHttpGetIn" />
      <wsdl:output message="tns:GetCitiesByCountryHttpGetOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="GlobalWeatherHttpPost">
    <wsdl:operation name="GetWeather">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get weather report for all major cities
around the world.</documentation>
```

```
      <wsdl:input message="tns:GetWeatherHttpPostIn" />
      <wsdl:output message="tns:GetWeatherHttpPostOut" />
    </wsdl:operation>
    <wsdl:operation name="GetCitiesByCountry">
     <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get all major cities by country name(full /
part).</documentation>
      <wsdl:input message="tns:GetCitiesByCountryHttpPostIn" />
      <wsdl:output message="tns:GetCitiesByCountryHttpPostOut" />
    </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.webserviceX.NET/GetWeather" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetCitiesByCountry">
      <soap:operation soapAction="http://www.webserviceX.NET/GetCitiesByCountry" style="document" /
>
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="GlobalWeatherHttpGet" type="tns:GlobalWeatherHttpGet">
    <http:binding verb="GET" />
    <wsdl:operation name="GetWeather">
      <http:operation location="/GetWeather" />
      <wsdl:input>
        <http:urlEncoded />
      </wsdl:input>
      <wsdl:output>
        <mime:mimeXml part="Body" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetCitiesByCountry">
      <http:operation location="/GetCitiesByCountry" />
      <wsdl:input>
        <http:urlEncoded />
      </wsdl:input>
      <wsdl:output>
        <mime:mimeXml part="Body" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="GlobalWeatherHttpPost" type="tns:GlobalWeatherHttpPost">
    <http:binding verb="POST" />
    <wsdl:operation name="GetWeather">
      <http:operation location="/GetWeather" />
      <wsdl:input>
        <mime:content type="application/x-www-form-urlencoded" />
      </wsdl:input>
      <wsdl:output>
        <mime:mimeXml part="Body" />
      </wsdl:output>
    </wsdl:operation>
```



```
    <wsdl:operation name="GetCitiesByCountry">
      <http:operation location="/GetCitiesByCountry" />
      <wsdl:input>
        <mime:content type="application/x-www-form-urlencoded" />
      </wsdl:input>
      <wsdl:output>
        <mime:mimeXml part="Body" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="GlobalWeather">
    <wsdl:port name="GlobalWeatherSoap" binding="tns:GlobalWeatherSoap">
      <soap:address location="http://www.webservicex.net/globalweather.asmx" />
    </wsdl:port>
    <wsdl:port name="GlobalWeatherHttpGet" binding="tns:GlobalWeatherHttpGet">
      <http:address location="http://www.webservicex.net/globalweather.asmx" />
    </wsdl:port>
    <wsdl:port name="GlobalWeatherHttpPost" binding="tns:GlobalWeatherHttpPost">
      <http:address location="http://www.webservicex.net/globalweather.asmx" />
    </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、現在のディレクトリの *NET¥webserviceX* サブ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 Version 10.0.1.3415
 * Do not edit this file.
 */

package NET.webserviceX;

import ianywhere.qanywhere.ws.*;
import ianywhere.qanywhere.client.QABinaryMessage;
import ianywhere.qanywhere.client.QAException;

import java.io.*;

import javax.xml.transform.*;
import javax.xml.transform.sax.*;
import javax.xml.transform.stream.*;

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
    {
        try {
            StringWriter sw = new StringWriter();
            SAXTransformerFactory stf = (SAXTransformerFactory) SAXTransformerFactory.newInstance();
            TransformerHandler hd = stf.newTransformerHandler();
            QABinaryMessage qaRequestMsg = null;

            hd.setResult( new StreamResult( sw ) );
            String        responsePartName = "GetWeatherResult";
            java.lang.String    returnValue;

            writeSOAPHeader( hd, "GetWeather", "http://www.webserviceX.NET" );
            WSBaseTypeSerializer.serialize(hd,"CityName",cityName,"string","http://www.w3.org/2001/
XMLSchema",true,true);
           WSBaseTypeSerializer.serialize(hd,"CountryName",countryName,"string","http://www.w3.org/2001/
XMLSchema",true,true);
            writeSOAPFooter( hd, "GetWeather" );

            qaRequestMsg = createQAMessage( sw.toString(), "http://www.webserviceX.NET/GetWeather",
"GetWeatherResponse" );

            WSResult wsResult = invokeWait( qaRequestMsg );

            returnValue = wsResult.getStringValue(responsePartName);

            return returnValue;
        } catch( TransformerConfigurationException e ) {
            throw new WSException( e );
        }
    }

    public WSResult asyncGetWeather(java.lang.String cityName,
                    java.lang.String countryName) throws QAException, WSException
    {
        try {
            StringWriter sw = new StringWriter();
            SAXTransformerFactory stf = (SAXTransformerFactory) SAXTransformerFactory.newInstance();
            TransformerHandler hd = stf.newTransformerHandler();
            QABinaryMessage qaRequestMsg = null;

            hd.setResult( new StreamResult( sw ) );

            writeSOAPHeader( hd, "GetWeather", "http://www.webserviceX.NET" );
            WSBaseTypeSerializer.serialize(hd,"CityName",cityName,"string","http://www.w3.org/2001/
XMLSchema",true,true);
           WSBaseTypeSerializer.serialize(hd,"CountryName",countryName,"string","http://www.w3.org/2001/
```



```
XMLSchema",true,true);
            writeSOAPFooter( hd, "GetWeather" );

            qaRequestMsg = createQAMessage( sw.toString(), "http://www.webserviceX.NET/GetWeather",
"GetWeatherResponse" );

            WSResult wsResult = invoke( qaRequestMsg );

            return wsResult;
        } catch( TransformerConfigurationException e ) {
            throw new WSException( e );
        }
    }


    public java.lang.String getCitiesByCountry(java.lang.String countryName) throws QAException,
WSException, WSFaultException
    {
        try {
            StringWriter sw = new StringWriter();
            SAXTransformerFactory stf = (SAXTransformerFactory) SAXTransformerFactory.newInstance();
            TransformerHandler hd = stf.newTransformerHandler();
            QABinaryMessage qaRequestMsg = null;

            hd.setResult( new StreamResult( sw ) );
            String        responsePartName = "GetCitiesByCountryResult";
            java.lang.String    returnValue;

            writeSOAPHeader( hd, "GetCitiesByCountry", "http://www.webserviceX.NET" );
         WSBaseTypeSerializer.serialize(hd,"CountryName",countryName,"string","http://www.w3.org/2001/
XMLSchema",true,true);
            writeSOAPFooter( hd, "GetCitiesByCountry" );

            qaRequestMsg = createQAMessage( sw.toString(), "http://www.webserviceX.NET/
GetCitiesByCountry", "GetCitiesByCountryResponse" );

            WSResult wsResult = invokeWait( qaRequestMsg );

            returnValue = wsResult.getStringValue(responsePartName);

            return returnValue;
        } catch( TransformerConfigurationException e ) {
            throw new WSException( e );
        }
    }

    public WSResult asyncGetCitiesByCountry(java.lang.String countryName) throws QAException,
WSException
    {
        try {
            StringWriter sw = new StringWriter();
            SAXTransformerFactory stf = (SAXTransformerFactory) SAXTransformerFactory.newInstance();
            TransformerHandler hd = stf.newTransformerHandler();
            QABinaryMessage qaRequestMsg = null;

            hd.setResult( new StreamResult( sw ) );

            writeSOAPHeader( hd, "GetCitiesByCountry", "http://www.webserviceX.NET" );
         WSBaseTypeSerializer.serialize(hd,"CountryName",countryName,"string","http://www.w3.org/2001/
XMLSchema",true,true);
            writeSOAPFooter( hd, "GetCitiesByCountry" );

            qaRequestMsg = createQAMessage( sw.toString(), "http://www.webserviceX.NET/
GetCitiesByCountry", "GetCitiesByCountryResponse" );
```

```
            WSResult wsResult = invoke( qaRequestMsg );

            return wsResult;
        } catch( TransformerConfigurationException e ) {
            throw new WSException( e );
        }
    }
}
```

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、接続が有効なときに結果を処理します。

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。*RequestWeather.java* ファイルに次のコードをコピーします。

```
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。*ShowWeather.java* ファイルに次のコードをコピーします。

```
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 ".;%sqlany11%¥java¥iawsrt.jar;%sqlany11%¥java¥qaclient.jar" com¥myweather
¥GlobalWeatherSoap.java RequestWeather.java
javac -classpath ".;%sqlany11%¥java¥iawsrt.jar;%sqlany11%¥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。

```
qaagent -c "dbf=qanywhere.db;eng=qanywhere;uid=ml_qa_user;pwd=qanywhere"
```

QAnywhere サーバを起動します。

```
mlsrv11 -m -zu+ -c "dsn=QAnywhere 11 Demo;uid=ml_server;pwd=sql;start=dbsrv11" -v+ -ot
qanyserv.mls
```

♦ **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。**

GetWeather Web サービスに送信される 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行い、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・コネクタを作成します。

1. Sybase Central で、**[接続] - [QAnywhere 11 に接続]** を選択します。
2. **[ユーザ ID]** フィールドに **ml_server** と入力します。
3. **[パスワード]** フィールドに **sql** と入力します。
4. **[ODBC データ・ソース名]** をクリックし、QAnywhere 11 Demo の場所を参照します。
5. **[OK]** をクリックします。
6. **[ファイル] - [新規] - [コネクタ]** を選択します。
7. **[Web サービス]** をクリックします。**[次へ]** をクリックします。
8. **[コネクタ名]** フィールドに **ianywhere.connector.globalweather** と入力します。**[次へ]** をクリックします。
9. **[URL]** フィールドに **http://www.webservicex.net/globalweather.asmx** と入力します。**[完了]** をクリックします。

Web サービスの使用

Web サービスから天気予報を取得するための要求をキューに登録するには、次のように入力します。

```
java -classpath ".;%sqlany11%¥java¥iawsrt.jar;%sqlany11%¥java¥qaclient.jar" RequestWeather
```

要求 ID が返されます。

天気予報を見るには、次のように入力します。コマンドの末尾には要求 ID を入力します。この例では REQ123123123 となります。

```
java -classpath ".;%sqlany11%¥java¥iawsrt.jar;%sqlany11%¥java¥qaclient.jar" ShowWeather
REQ123123123
```

詳細な天気予報が返されます。

QAnywhere の配備

目次

QAnywhere アプリケーションの配備

QAnywhere では、SQL Anywhere メッセージ・ストアに対するサポートのために、C++、Java、および .NET API を提供しています。Java および .NET API では、Ultra Light メッセージ・ストアもサポートしています。QAnywhere アプリケーションの配備に必要なファイルは、お使いの Windows 環境、メッセージ・ストアの種類、および API の選択に基づいて異なります。モバイル Web サービス・アプリケーションを開発する場合は、それ以外のファイルも必要です。

以下に示すファイルの他に、QAnywhere アプリケーションでは次のファイルも必要です。

- 「SQL Anywhere Mobile Link クライアントの配備」『Mobile Link - サーバ管理』の Mobile Link 同期クライアント、リスナ、セキュリティ (任意) に関する各項に記載されたすべてのファイル。リスナ・ファイルは、Push 通知を使用している場合 (デフォルト設定) のみ必要です。
- 「データベース・サーバの配備」『SQL Anywhere サーバ - プログラミング』に記載された dbeng11 ファイルまたは dbsrv11 ファイル。

Sybase Central を配備する方法については、「管理ツールの配備」『SQL Anywhere サーバ - プログラミング』を参照してください。

Windows アプリケーション

すべてのディレクトリは、*install-dir* を基準とした相対ディレクトリです。

Windows Mobile 環境のファイル構造の詳細については、「Windows Mobile アプリケーション」『Mobile Link - サーバ管理』を参照してください。

次に、SQL Anywhere メッセージ・ストアをセットアップする際に必要なファイルのリストを示します。

クライアント API	Windows ファイル
C++	● *bin32¥qany11.dll* ● *bin32¥qaagent.exe* ● *bin32¥qastop.exe*
Java	● *bin32¥qaagent.exe* ● *bin32¥qastop.exe* ● *java¥qaclient.jar* ● *java¥jodbc.jar* モバイル Web サービス・アプリケーションについては、次のファイルも必要です。 ● *java¥iawsrt.jar* ● *java¥jaxrpc.jar*

クライアント API	Windows ファイル
.NET	● *bin32¥qazlib.dll* ● *bin32¥qaagent.exe* ● *bin32¥qastop.exe* ● *assembly¥v2¥iAnywhere.QAnywhere.Client.dll* ● *assembly¥v2¥iAnywhere.QAnywhere.Resources.dll* ● *assembly¥v2¥iAnywhere.Data.SQLAnywhere.dll* モバイル Web サービス・アプリケーションについては、次のファイルも必要です。 ● *Assembly¥v2¥iAnywhere.QAnywhere.WS.dll*

次に、Ultra Light メッセージ・ストアをセットアップする際に必要なファイルのリストを示します。

クライアント API	Windows ファイル
Java	● *bin32¥qauagent.exe* ● *bin32¥qastop.exe* ● *bin32¥qadbiuljni.dll* ● *java¥qaclient.jar* モバイル Web サービス・アプリケーションについては、次のファイルも必要です。 ● *java¥iawsrt.jar* ● *java¥jaxrpc.jar*
.NET	● *bin32¥qazlib.dll* ● *bin32¥qauagent.exe* ● *bin32¥qastop.exe* ● *assembly¥v2¥iAnywhere.QAnywhere.Client.dll* ● *assembly¥v2¥iAnywhere.QAnywhere.Resources.dll* ● *ultralite¥ultralite.NET¥assembly¥v2¥iAnywhere.Data.UltraLite.dll* モバイル Web サービス・アプリケーションについては、次のファイルも必要です。 ● *Assembly¥v2¥iAnywhere.QAnywhere.WS.dll*

Ultra Light メッセージ・ストアを作成するときは、Ultra Light データベース作成ユーティリティを使用して udb データベース・ファイルを作成し、それから QAnywhere Ultra Light Agent の -si オプションを使用してデータベースを初期化する必要があります。「Ultra Light データベース作成ユーティリティ (ulcreate)」『Ultra Light データベース管理とリファレンス』と「qauagent ユーティリティ」 768 ページを参照してください。

Windows Mobile アプリケーション

すべてのディレクトリは、*install-dir* を基準とした相対ディレクトリです。

Windows 環境のファイル構造の詳細については、「Windows アプリケーション」 『Mobile Link - サーバ管理』を参照してください。

次に、SQL Anywhere メッセージ・ストアをセットアップする際に必要なファイルのリストを示します。

クライアント API	Windows Mobile ファイル
C++	● *ce¥arm.50¥qany11.dll* ● *ce¥arm.50¥qaagent.exe* ● *ce¥arm.50¥qastop.exe*
Java	● *ce¥arm.50¥qaagent.exe* ● *ce¥arm.50¥qastop.exe* ● *java¥qaclient.jar* ● *java¥jodbc.jar* モバイル Web サービス・アプリケーションの場合、次のファイルも必要です。 ● *java¥iawsrt.jar* ● *java¥jaxrpc.jar*
.NET	● *ce¥arm.50¥qazlib.dll* ● *ce¥arm.50¥qaagent.exe* ● *ce¥arm.50¥qastop.exe* ● *ce¥assembly¥v2¥iAnywhere.QAnywhere.Client.dll* ● *ce¥assembly¥v2¥iAnywhere.QAnywhere.Resources.dll* ● *ce¥assembly¥v2¥iAnywhere.Data.SQLAnywhere.dll* モバイル Web サービス・アプリケーションについては、次のファイルも必要です。 ● *ce¥Assembly¥v2¥iAnywhere.QAnywhere.WS.dll*

次に、Ultra Light メッセージ・ストアをセットアップする際に必要なファイルのリストを示します。

クライアント API	Windows Mobile ファイル
Java	● *ce¥arm.50¥qauagent.exe* ● *ce¥arm.50¥qastop.exe* ● *ce¥arm.50¥qadbiuljni.dll* ● *java¥qaclient.jar* モバイル Web サービス・アプリケーションについては、次のファイルも必要です。 ● *java¥iawsrt.jar* ● *java¥jaxrpc.jar*
.NET	● *ce¥arm.50¥qazlib.dll* ● *ce¥arm.50¥qauagent.exe* ● *ce¥arm.50¥qastop.exe* ● *ce¥assembly¥v2¥iAnywhere.QAnywhere.Client.dll* ● *ce¥assembly¥v2¥iAnywhere.QAnywhere.Resources.dll* ● *ultralite¥ultralite.NET¥ce¥assembly¥v2¥iAnywhere.Data.UltraLite.dll* モバイル Web サービス・アプリケーションについては、次のファイルも必要です。 ● *ce¥Assembly¥v2¥iAnywhere.QAnywhere.WS.dll*

Ultra Light メッセージ・ストアを作成するときは、Ultra Light データベース作成ユーティリティを使用してデータベース・ファイルを作成し、それから QAnywhere Ultra Light Agent を使用してデータベースを初期化する必要があります。「Ultra Light データベース作成ユーティリティ (ulcreate)」『Ultra Light データベース管理とリファレンス』と「qauagent ユーティリティ」 768 ページを参照してください。

QAnywhere .NET API DLL の登録

QAnywhere .NET API DLL (*Assembly¥v2¥iAnywhere.QAnywhere.Client.dll*) は、Windows (Windows Mobile を除く) のグローバル・アセンブリ・キャッシュに登録する必要があります。グローバル・アセンブリ・キャッシュには、コンピュータに登録されているすべてのプログラムがリストされています。SQL Anywhere をインストールすると、インストール・プログラムによって登録されます。Windows Mobile の場合、この DLL を登録する必要はありません。

QAnywhere を配備する場合は、.NET Framework に含まれている gacutil ユーティリティを使用して、QAnywhere .NET API DLL (*Assembly¥v2¥iAnywhere.QAnywhere.Client.dll*) を登録してください。

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」 140 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」 141 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。

```
CREATE USER 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
ALTER USER 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と復号化」 『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」 747 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」 766 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 認定の暗号化には、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 11 - 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 ID ファイルを使用しています。

RSA を使用してセキュリティ保護された TCP/IP：

```
mlsrv11 -x tls(tls_type=rsa;identity=rsaserver.id;identity_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：

```
mlsrv11 -x tls(tls_type=ecc;identity=eccserver.id;identity_password=test)
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)：

```
mlsrv11 -x https(identity=rsaserver.id;identity_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」 754 ページ
- 「-mp オプション」 753 ページ
- 「Mobile Link ユーザ」 『Mobile Link - クライアント管理』

サーバ管理要求のセキュリティ保護

サーバ管理要求は、パスワードを使用して保護することができます。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」 799 ページを参照してください。

Mobile Link ユーザ認証ユーティリティを使用した新しいユーザの追加

セキュリティ保護を確実に行うには、Mobile Link ユーザ認証ユーティリティ (mluser) を使用してユーザを追加します。mluser ユーティリティを使用すると、統合データベース上のサーバ・メッセージ・ストアに Mobile Link ユーザ名を登録することができます。その後、QAnywhere Agent がユーザ名とパスワードのパラメータを使用し、メッセージ転送中にユーザの認証を行います。「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。

Push 通知を使用している場合は、リスナ用の Mobile Link ユーザ (dblsn) を追加する必要もあります。また追加された各ユーザに対して、ユーザ名 ias_[user]_lsn でリスナを追加する必要もあります。

-zu+ オプション付きで mlsrv11 を実行して新しいユーザを追加することもできますが、セキュリティ上問題がある場合はこのオプションを使用しないでください。-zu+ オプション付きで mlsrv11 を使用すると、すべての新しいユーザが、最初の同期時に統合データベースに追加されます。これは、認識されていないユーザが認証を行わずに追加されることを意味します。「-zu オプション」 『Mobile Link - サーバ管理』を参照してください。

リレー・サーバを使用する場合のセキュリティ

リレー・サーバを使用する場合は、リレー・サーバの設定ファイルのオプションを設定して、必要なセキュリティ・レベルを制御します。「リレー・サーバ設定ファイル」 『Mobile Link - サーバ管理』を参照してください。

Web サーバとのセキュリティ保護された通信ストリーム (信頼できる証明書付きの HTTPS など) を設定し、QAnywhere Agent と Web サーバの間の通信のセキュリティを確保します。qaagent 「-x オプション」 766 ページ の項を参照してください。

リレー・サーバと Mobile Link の間でセキュリティ保護された通信ストリームを設定する方法については、リレー・サーバのマニュアルを参照してください。「リレー・サーバ」 『Mobile Link - サーバ管理』を参照してください。

サーバ・メッセージ・ストアの管理

目次

転送ルール

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。転送ルールは、サーバとクライアントの両方に対して指定できます。

サーバ側の転送ルールの管理は、サーバ・メッセージ・ストアの管理の重要な部分です。

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」 200 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」 148 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。この方法は推奨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)」 820 ページを参照してください。

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」 200 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」 148 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。この方法は推奨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)」 820 ページを参照してください。

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 11] - [Sybase Central]** を選択します。
   - **[接続] - [QAnywhere 11 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・メッセージ・ストア]** の下で、データ・ソース名を選択します。
3. **[ファイル] - [プロパティ]** を選択します。
4. **[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。
5. **[新規]** をクリックして、ルールを追加します。
6. 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 **[定数]** を選択して、条件を追加します。
7. **[OK]** をクリックして終了します。

サーバ管理要求でのサーバ側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、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。

サーバ管理要求を使用して転送ルールを指定する方法については、「サーバ管理要求での転送ルールの指定」 200 ページを参照してください。

メッセージ・アーカイブの管理

アーカイブ・メッセージ・ストアはサーバ・メッセージ・ストアと共存するテーブルのセットで、削除待機中のすべてのメッセージがここに格納されます。定期的に実行されるシステム・プロセスでは、サーバ・メッセージ・ストアにあるメッセージのうち最終ステータスに到達したものすべてを削除し、アーカイブ・メッセージ・ストアに挿入することで、メッセージ・ストア間でのメッセージ転送を行います。

メッセージは、サーバ削除ルールによって削除されるまで、アーカイブ・メッセージ・ストアに残ります。アーカイブ・メッセージ・ストアを使用することにより、同期中にフィルタする必要のあるメッセージの量が最小限に抑えられるため、サーバ・メッセージ・ストアのパフォーマンスが向上します。「削除ルール」 50 ページを参照してください。

サーバ管理要求の使用

サーバ管理要求は、クライアントからサーバへ送信される特殊なメッセージです。サーバ管理要求には、さまざまな機能の実行をサーバに命令する、フォーマット指定された XML コンテンツが含まれています。

サーバ管理要求を使用してサーバ・メッセージ・ストアを管理するには、次の機能を使用します。

- 「クライアント転送ルールの更新」 186 ページ
- 「メッセージのキャンセル」 186 ページ
- 「メッセージの削除」 188 ページ

サーバ管理要求の使用の詳細については、次の項を参照してください。

- 「サーバ管理要求でのサーバ・メッセージ・ストアの管理」 186 ページ
- 「サーバ管理要求の作成」 184 ページ
- 「サーバ管理要求のリファレンス」 731 ページ

クライアント・メッセージ・ストアの管理

目次

QAnywhere クライアントのモニタ

QAnywhere クライアントは、サーバ管理要求または Sybase Central を使用してモニタすることができます。

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。「QAnywhere クライアントのモニタ」 206 ページを参照してください。

Sybase Central で、サーバ・メッセージ・ストアの **[クライアント]** ウィンドウ枠を使用して、現在サーバ上にあるクライアントのリストを表示します。

クライアント・プロパティのモニタ

QAnywhere クライアント・プロパティは、サーバ管理要求または Sybase Central を使用してモニタすることができます。

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。要求の応答によってリスト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。「プロパティのモニタ」 206 ページを参照してください。

Sybase Central で、QAnywhere の **[クライアント・プロパティ]** ウィンドウを使用してクライアント・プロパティを表示および変更できます。

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」 796 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は、クライアントに送信されるメッセージをフィルタするために転送ルールで使用したり、追加するメッセージを決定するために削除ルールで使用することができます。

「QAnywhere 転送ルールと削除ルール」 809 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」 182 ページを参照してください。

送信先エイリアス

目次

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」 201 ページを参照してください。

- Sybase Central

  「Sybase Central の使用」 158 ページを参照してください。

Sybase Central の使用

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 11] - [Sybase Central]** を選択します。
   - **[接続] - [QAnywhere 11 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。

コネクタ

目次

JMS コネクタ

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 配備では、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」 8 ページを参照してください。

サーバ・ファーム環境で QAnywhere メッセージング対応 Mobile Link を実行すると、プライマリ・サーバの QAnywhere コネクタのみが起動します。プライマリ・サーバが失敗した場合、新しいプライマリ・サーバで QAnywhere コネクタが自動的に起動するため、JMS などの外部メッセージング・システムでデータを交換する間もメッセージの順序は保持されます。詳細については、「Mobile Link サーバをサーバ・ファームで実行する」 『Mobile Link - サーバ管理』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]** - **[新規]** - **[コネクタ]** を選択します。
4. **[JMS]** をクリックし、**[使用している JMS システムを指定してください。]** リストで使用している Web サーバの種類を選択します。**[次へ]** をクリックします。
5. **[コネクタ名]** ページで、次の操作を行います。
   - **[コネクタ名]** フィールドに、QAnywhere クライアントがコネクタを指定するために使用するコネクタのアドレスを入力します。「JMS コネクタへの QAnywhere メッセージの送信」 167 ページを参照してください。
   - **[受信側]** フィールドに、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で使用されるキュー名を入力します。
   - **[次へ]** をクリックします。
6. **[JNDI の設定]** ページで、次の操作を行います。
   - **[JNDI ファクトリ]** フィールドに、外部 JMS JNDI ネーム・サービスへのアクセスに使用されるファクトリ名を入力します。
   - **[ネーム・サービス URL]** フィールドに、JMS JNDI ネーム・サービスにアクセスするための URL を入力します。
   - **[ユーザ名]** フィールドに、外部 JMS JNDI ネーム・サービスに接続するための認証名を入力します。
   - **[パスワード]** フィールドに、外部 JMS JNDI ネーム・サービスに接続するための認証パスワードを入力します。
   - **[次へ]** をクリックします。
7. **[JMS キューの設定]** ページで、次の操作を行います。
   - **[キュー・ファクトリ]** フィールドに、外部 JMS プロバイダのキュー・ファクトリ名を入力します。
   - **[ユーザ名]** フィールドに、外部 JMS キュー接続に接続するためのユーザ ID を入力します。
   - **[パスワード]** フィールドに、外部 JMS キュー接続に接続するためのパスワードを入力します。

- **[次へ]** をクリックします。

8. **[JMS トピックの設定]** ページで、次の操作を行います。
   - **[トピック・ファクトリ]** フィールドに、外部 JMS プロバイダのトピック・ファクトリ名を入力します。
   - **[ユーザ名]** フィールドに、外部 JMS トピック接続に接続するためのユーザ ID を入力します。
   - **[パスワード]** フィールドに、外部 JMS トピック接続に接続するためのパスワードを入力します。
   - **[完了]** をクリックします。
9. **[OK]** をクリックします。
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」 165 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・プロパティの設定」 165 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、「サーバ管理要求によるコネクタの管理」 189 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。

   「JMS コネクタへの QAnywhere メッセージの送信」 167 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」 169 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¥jms* 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。
- 次の QAnywhere ニュースグループに質問を投稿できます。
  ianywhere.public.sqlanywhere.qanywhere
- サーバ・ファーム環境でメッセージングを設定する方法については、「Mobile Link サーバをサーバ・ファームで実行する」 『Mobile Link - サーバ管理』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 (mlsrv11) 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。

```
mlsrv11 -sl java(-cp
"jmsclient.jar") -m -c "dsn=QAnywhere 11.0 Demo" ...
```

JMS コネクタ・プロパティの設定

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」 163 ページを参照してください。
- Sybase Central の **[コネクタ・プロパティ]** ウィンドウ
- サーバ管理要求

  「コネクタの作成と設定」 189 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・プロパティのリストについては、「JMS コネクタ・プロパティ」 802 ページを参照してください。

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

- 「JMS コネクタへの QAnywhere メッセージの送信」 167 ページ
- 「JMS コネクタ・プロパティの設定」 165 ページ
- 「コネクタの作成と設定」 189 ページ

JMS コネクタへの QAnywhere メッセージの送信

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」 69 ページ
- 「JMS コネクタ・プロパティの設定」 165 ページ

JMS コネクタから QAnywhere クライアントへのメッセージの送信

QAnywhere メッセージは、JMS メッセージに自然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 Unicode 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	

QAnywhere	JMS	説明
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
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 のプロパティに自然にマッピングされ、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」 69 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 メッセージに自然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 Unicode 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 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 のプロパティに自然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。

Web サービス・コネクタ

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行います。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。

サーバ・ファーム環境で QAnywhere メッセージング対応 Mobile Link を実行すると、プライマリ・サーバの QAnywhere コネクタのみが起動します。プライマリ・サーバが失敗した場合、新しいプライマリ・サーバで QAnywhere コネクタが自動的に起動するため、JMS などの外部メッセージング・システムでデータを交換する間もメッセージの順序は保持されます。詳細については、「Mobile Link サーバをサーバ・ファームで実行する」 『Mobile Link - サーバ管理』を参照してください。

Web サービス・コネクタの設定

♦ **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]** - **[新規]** - **[コネクタ]** を選択します。
3. **[Web サービス]** をクリックします。**[次へ]** をクリックします。
4. **[コネクタ名]** フィールドに、QAnywhere クライアントがコネクタを指定するために使用するコネクタのアドレスを入力します。**[次へ]** をクリックします。
5. **[URL]** フィールドに、Web サービスの URL (*http://localhost:8080/qanyserv/F2C* など) を入力します。**[次へ]** をクリックします。

   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 **[プロキシ経由で Web サービスにアクセスする]** をクリックし、次のフィールドに入力します。
   - **[HTTP ユーザ名]** フィールドにユーザ名を入力します。値は、プロパティ webservice.http.authName に設定されます。
   - **[HTTP パスワード]** フィールドにユーザ・パスワードを入力します。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]** フィールドにホスト名を入力します。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。
   - **[プロキシ・ポート]** フィールドに、プロキシ・サーバ上の接続先ポートを入力します。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。

- **[プロキシ・ユーザ名]** フィールドに、プロキシで認証が必要な場合に使用されるプロキシ・ユーザ名を入力します。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。
- **[完了]** をクリックします。

7. 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」 173 ページを参照してください。

   サーバ管理要求の使用の詳細については、「サーバ管理要求によるコネクタの管理」 189 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」 172 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]** ウィンドウまたは 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] ウィンドウを開くには、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。

♦ **Web サービスのプロパティを表示するには、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. 左ウィンドウ枠の **[サーバ・メッセージ・ストア]** の下で、データ・ソースの名前を選択します。
3. 右ウィンドウ枠で **[コネクタ]** タブを選択し、Web サービス・コネクタの名前を選択します。
4. **[ファイル]** - **[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 Mobile Link メッセージ・ウィンドウ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**の **[コネクタ名]** ページにある **[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 ～ 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]** ウィンドウの **[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 メッセージ・ウィンドウと Mobile Link サーバ・メッセージ・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]** ウィンドウの **[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]** ウィンドウの **[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。

Web サービス・コネクタへのメッセージの送信

メッセージは、モバイル Web サービス API を介して Web サービス・コネクタに送信されます。ianywhere.qanywhere.ws.WSBase クラスの setProperty メソッドを使用して、WS_CONNECTOR_ADDRESS プロパティに Web サービス・コネクタの ID を設定します。「WSBase クラス」 642 ページを参照してください。

たとえば、CurrencyConvertor サンプルの次のコード行が指定されると、Web サービス要求の作成に使用された Web サービス API が、Web サービス・コネクタを介してこれらの要求をメッセージとして送信します。

```
service.setProperty(
   "WS_CONNECTOR_ADDRESS",
   "ianywhere.connector.currencyconvertor¥¥" );
```

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 クライアント・アプリケーションと QAnywhere クライアント・アプリケーションの間でメッセージを送信します。

必要なソフトウェア

- SQL Anywhere 11
- Java ソフトウェア開発キット
- JMS コネクタ

前提知識と経験

次の知識と経験が必要です。

- Java の知識
- JMS コネクタの設定に関する基礎知識

目的

次の項目について、知識と経験を得ることができます。

- サンプルの QAnywhere アプリケーションと通信するための JMS コネクタの設定
- JMS メッセージ・システムとサンプルの QAnywhere アプリケーションの間でのメッセージの送信

主要な概念

ここでは、次の手順に従って、SQL Anywhere サンプル・データベースを使用した JMS メッセージ・システムと QAnywhere との接続を行います。

- メッセージの送受信のための JMS コネクタの準備
- QAnywhere サーバとクライアントのコンポーネント、および JMS クライアントの実行
- QAnywhere クライアントと JMS クライアントの間でのメッセージの送信

関連項目

JMS コネクタの使用の詳細については、「コネクタ」 161 ページを参照してください。

レッスン 1：クライアントおよびサーバのコンポーネントの設定

♦ **JMS プロバイダを準備するには、次の手順に従います。**

1. サーバを起動する方法については、JMS サーバのマニュアルを参照してください。

2. JMS サーバで次のキューを作成します。
   - **testmessage** TestMessage サンプルでは、このキュー名を使用してメッセージを受信します。
   - **qanywhere_receive** QAnywhere の JMS コネクタでは、このキュー名を使用してメッセージを受信します。

   キューを作成した後、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のコンポーネントを起動するには、次の手順に従います。

1. Sybase Central を使用して、JMS システムの QAnywhere JMS コネクタを作成します。「JMS コネクタの設定」 163 ページを参照してください。
2. コマンド・プロンプトで次のコマンドを実行します。

   ```
   mlsrv11 -m -c "dsn=QAnywhere 11 Demo" -sl
   java(-cp JMS-client-jar-files) -vcrs
   -zu+
   ```

   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。

   Mobile Link サーバがメッセージングに対応した状態で起動します。
3. **[スタート] - [プログラム] - [SQL Anywhere 11] - [QAnywhere] - [SQL Anywhere の使用に関するチュートリアル] - [SQL Anywhere 用 QAnywhere Agent -- saclient1]** を選択します。

   QAnywhere Agent がロードされます。
4. **[スタート] - [プログラム] - [SQL Anywhere 11] - [QAnywhere] - [SQL Anywhere の使用に関するチュートリアル] - [TestMessage -- saclient1]** を選択します。

   QAnywhere サンプル・アプリケーションがロードされます。

♦ JMS 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・プロンプトで次のコマンドを実行します。

   ```
   edit samples-dir/QAnywhere/JMS/TestMessage/build.bat
   ```
2. *build.bat* ファイルのコードを調べて、JMS サーバのファイル・パスが正しいことを確認します。

   たとえば、EAServer を使用している場合、デフォルト設定は **easerver** 見出しの下で定義されます。

   ```
   :easerver
   REM For EAServer, compile with the following JAR files
   SET easerver_install=c:¥program files¥sybase¥easerver6
   SET jmsjars=%easerver_install%¥lib¥eas-client-15.jar
   goto build_app
   ```


c:¥program files¥sybase¥easerver6 ディレクトリに EAServer がない場合は、正しいインストール・ディレクトリを示すように **easerver_install** 変数を更新します。**jmsjars** 変数が、JMS サーバの jar ファイルの正しいロケーションを示していることを確認します。

お使いの JMS サーバがリストにない場合は、バッチ・ファイルの最初の方で定義した **custom** ヘッダ設定を使用して、自分の JMS ファイル・パスのロケーションを定義します。

完了したら、変更内容を保存してエディタを終了します。

3. コマンド・プロンプトで、次のコマンドを実行し、JMS TestMessage クライアントをコンパイルします。

```
build.bat JMS-server-name
```

JMS-server-name は *build.bat* でヘッダ名として示されている JMS サーバの名前です。設定可能な値は、**easerver**、**fioranomq**、**jboss**、**tibco**、**weblogic**、および **custom** です。デフォルトでは、*build.bat* は **easerver** を使用します。

4. コマンド・プロンプトで次のコマンドを実行します。

```
edit samples-dir/QAnywhere/JMS/TestMessage/run.bat
```

5. *run.bat* ファイルのコードを調べて、JMS サーバのファイル・パスが正しいことを確認します。

たとえば、EAServer を使用している場合、デフォルト設定は **easerver** 見出しの下で定義されます。

```
:easerver
REM For EAServer, compile with the following JAR files
SET easerver_install=c:¥program files¥sybase¥easerver6
SET jmsjars=%easerver_install%¥lib¥eas-client-15.jar
goto build_app
```

c:¥program files¥sybase¥easerver6 ディレクトリに EAServer がない場合は、正しいインストール・ディレクトリを示すように **easerver_install** 変数を更新します。**jmsjars** 変数が、JMS サーバの jar ファイルの正しいロケーションを示していることを確認します。

お使いの JMS サーバがリストにない場合は、バッチ・ファイルの最初の方で定義した **custom** ヘッダ設定を使用して、自分の JMS ファイル・パスのロケーションを定義します。

完了したら、変更内容を保存してエディタを終了します。

6. コマンド・プロンプトで、次のコマンドを実行し、JMS TestMessage クライアントを実行します。

```
run.bat JMS-server-name
```

JMS-server-name は *build.bat* でヘッダ名として示されている JMS サーバの名前です。設定可能な値は、**easerver**、**fioranomq**、**jboss**、**tibco**、**weblogic**、および **custom** です。デフォルトでは、*build.bat* は **easerver** を使用します。

7. 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 **[TestMessage -- saclient1]** 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。
2. **[Destination ID]** フィールドに **saclient1** と入力します。
3. **[Subject]** フィールドと **[Message]** フィールドにサンプル・テキストを入力します。
4. **[Send]** をクリックします。

   メッセージが受信されたことを示すウィンドウが表示されます。

レッスン 3：QAnywhere クライアントから JMS クライアントへのメッセージの送信

♦ **QAnywhere の JMS コネクタの名前を見つけるには、次の手順に従います。**

1. **[スタート] - [プログラム] - [SQL Anywhere] - [Sybase Central]** を選択します。
2. **[接続] - [QAnywhere 11 に接続]** を選択します。
3. **[ODBC データ・ソース名]** をクリックします。
4. **[参照]** をクリックし、**[QAnywhere 11 Demo]** を選択します。
5. **[OK]** をクリックします。
6. **[OK]** をクリックします。
7. **[コネクタ]** タブを選択します。

   右ウィンドウ枠には、すべてのアクティブな JMS コネクタのリストが表示されます。
8. 名前フィールドを確認します。

   リストには、アクティブな QAnywhere JMS コネクタが 1 つだけ含まれているはずです。コネクタの名前が、名前フィールドの下に表示されます。

♦ **QAnywhere クライアントから JMS 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. **[saclient1 - TestMessage]** ウィンドウで、**[Message] - [New]** をクリックします。
2. **[Destination ID]** フィールドに JMS システムの名前を入力します。
3. **[Subject]** フィールドと **[Message]** フィールドにサンプル・テキストを入力します。
4. **[Send]** をクリックします。

   メッセージが受信されたことを示すウィンドウが表示されます。

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。

♦ **すべてのアプリケーションを停止するには、次の手順に従います。**

1. JMS TestMessage クライアント・アプリケーションを閉じるには、**[File] - [Exit]** を選択します。
2. **[saclient1 - TestMessage Client]** ウィンドウを閉じるには、**[File] - [Exit]** を選択します。
3. **[saclient1 - QAnywhere Agent]** ウィンドウと Mobile Link サーバを閉じるには、それぞれのウィンドウで **[シャットダウン]** を選択します。
4. JMS コネクタから切断する方法については、JMS サーバのマニュアルを参照してください。

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」 191 ページと「コネクタを閉じる」 191 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」 192 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」 200 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」 204 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」 198 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」 186 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」 799 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」 70 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 );
```

サーバ管理要求の認証

「ianywhere.qa.server.password.e」 サーバ・プロパティは、サーバ管理要求を認証する際のパスワードを指定するために使用されます。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・プロパティ」 799 ページを参照してください。

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。

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。

各サーバ管理要求は <actions> タグで開始します。

actionsResponseId タグ

actionsResponseId タグを <actions> タグのサブタグとして使用し、<action> タグ内の処理の進捗を追跡およびレポートします。レポートは、<action> タグが処理されるとサーバによって作成されます。

レポートには、<actionsResponseId> タグの ID と、要求によって生成されたエラー・メッセージが含まれます。レポートが作成されると、そのレポートはサーバ管理要求の返信アドレスに送信されます。

次に、actionsResponseId タグを使用したサーバ管理要求の例を示します。

```
<?xml version="1.0" encountered="UTF-8"?>
<actions>
 <actionsResponseId>myActionID</actionsResponseId>
 <MessageDetailsRequest>
  <request>
   <requestId>testRequest</requestId>
   <condition>
    <priority>9</priority>
   </condition>
   <status/>
   <address/>
  </request>
 </MessageDetailsRequest>
</actions>
```

次に、myActionId 要求がエラーを生成しなかった場合の actionsResponseId レポートの例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<ActionsResponse>
<actionsResponseId>myActionId</actionsResponseId>
<error/>
</ActionsResponse>
```

アーカイブ・メッセージ・ストア要求

アーカイブ・メッセージ・ストア内のメッセージの詳細を表示するには、<condition> タグのサブタグとして <archived> タグを使用します。<archived> タグを省略すると、レポートにはサーバ・メッセージ・ストアのメッセージのみが含まれます。

アーカイブ・メッセージ・ストア内にメッセージが存在するかどうかを判断するには、<request> タグのサブタグとして <archived> タグを使用します。

例

次の要求では、testRequest がアーカイブ・メッセージ・ストア内に存在する場合は true 、サーバ・メッセージ・ストア内に存在する場合は false が返されます。

```
<request>
  <requestID>testRequest</requestID>
  <status/>
  <archived/>
</request>
```

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。「サーバ管理要求を使用した送信先エイリアスの作成」 201 ページを参照してください。

送信先エイリアスの詳細については、「送信先エイリアス」 158 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」 182 ページを参照してください。

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」 182 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。クライアント転送ルールを変更するには、サーバ管理要求で 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最終ステータスに至っておらず、受信者に送信され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	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、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」 732 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します。これにより、サーバが再起動された場合で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。「サーバ管理要求の親タグ」 732 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</requestId>
   <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>
```

サーバ管理要求によるコネクタの管理

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」 182 ページを参照してください。

参照

- 「コネクタ」 161 ページ
- 「Web サービス・コネクタ」 172 ページ

コネクタの作成と設定

コネクタを作成するには、<SetProperty> を使用してプロパティを追加し、<OpenConnector> を使用します。

例

次の例では、サーバ管理要求は最初に関係する複数のプロパティを設定して、クライアントの ianywhere.connector.jboss にこれらのプロパティを関連付けます。ianywhere.connector.jboss は新しいコネクタのクライアント ID です。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、<SetProperty> を使用してクライアントのすべてのプロパティを削除します。

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>
   <SetProperty>
      <prop>
     <client>ianywhere.connector.jboss</client>
     <name>ianywhere.connector.nativeConnection</name>
    </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

- 「コネクタ」 161 ページ
- 「Web サービス・コネクタ」 172 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

- 「コネクタ」 161 ページ
- 「Web サービス・コネクタ」 172 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 Listener) 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	説明
<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、リスト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>	「サーバ管理要求の親タグ」 732 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> タグと <schedule> 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 メッセージでは <onEvent> タグと <schedule> タグが除外されているので、これは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化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、格納されているイベント・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。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が「実行中」に変わると、コネクタは初期化されます。

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」 190 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」 182 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、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」 182 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、myClient 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>myClient</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>myClient</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」 158 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」 182 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」 812 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」 182 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」 809 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」 182 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」 732 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。「サーバ管理要求の親タグ」 732 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要求します。
<archived>	メッセージがアーカイブ・ストア内に存在するかどうかを要求します。
<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」 732 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。
<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。「サーバ管理要求の親タグ」 732 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。

<Request> タグのサブタグ	説明
<transmissionStatus>	各メッセージの転送ステータスを要求します。

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」 182 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。要求の応答によってリスト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」 182 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>
```


チュートリアル：TestMessage の解説

目次

チュートリアルの概要

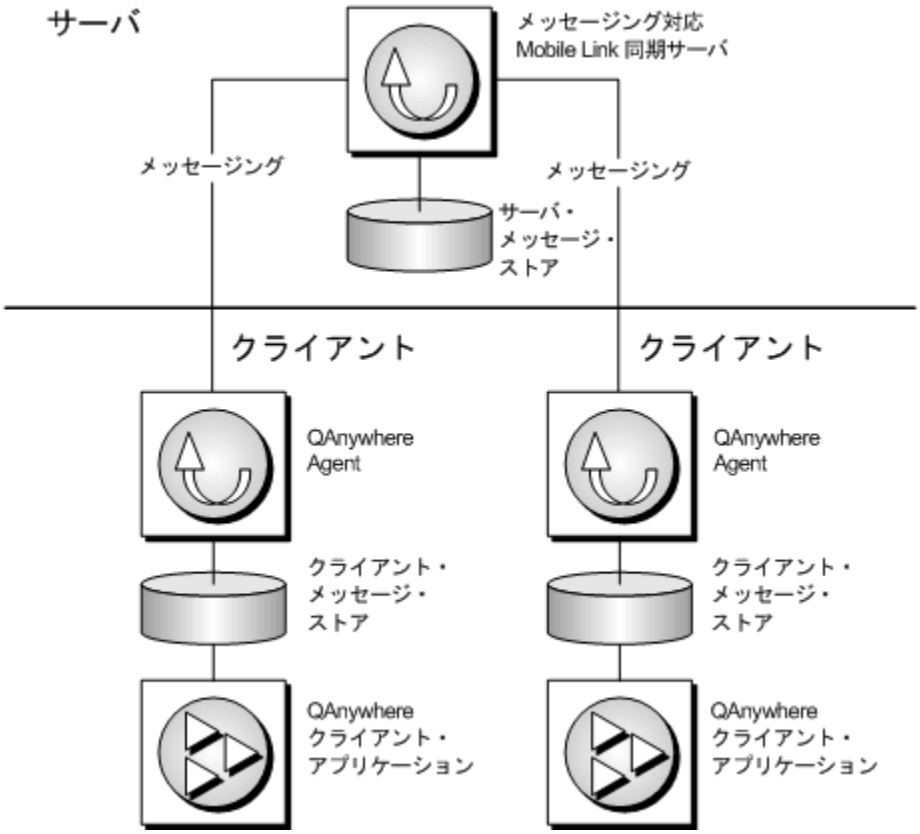
TestMessage は QAnywhere クライアント・アプリケーションのサンプルです。このアプリケーションは、QAnywhere を使用してユーザ独自のメッセージ・クライアント・アプリケーションを作成する方法を示しています。TestMessage では、単一のクライアント間インタフェースを使用して、メッセージを送信、受信、表示します。このサンプルでは、QAnywhere メッセージングの機能をわかりやすく示すため、人間が読めるテキスト・メッセージを使用しています。ただし、QAnywhere はテキスト以外のメッセージも扱うことができます。QAnywhere は、多数のクライアントの間でメッセージ・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 Mobile 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 Mobile 用には、C++、Visual Basic、C#、Java に対応したソース・コードが提供されています。.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. **[スタート] - [プログラム] - [SQL Anywhere 11] - [QAnywhere] - [Mobile Link QAnywhere のサンプル]** を選択します。

   または、コマンド・プロンプトで次のコマンドを実行します。

   ```
   mlsrv11 -m -c "dsn=QAnywhere 11 Demo" -vcrs -zu+
   ```

   このコマンド・ラインでは、以下に示す mlsrv11 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 11.0 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。

参照

- 「有効な Mobile Link での QAnywhere の起動」 32 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」 13 ページ
- 「簡単なメッセージング・シナリオ」 5 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. **[スタート] - [プログラム] - [SQL Anywhere 11] - [QAnywhere] - [SQL Anywhere の使用に関するチュートリアル] - [SQL Anywhere 用 QAnywhere Agent -- saclient1]** を選択します。

   QAnywhere 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。

   次の手順に進む前に、QAnywhere Agent の最初のインスタンスが起動するまで数秒待機する必要があります。

3. **[スタート] - [プログラム] - [SQL Anywhere 11] - [QAnywhere] - [SQL Anywhere の使用に関するチュートリアル] - [SQL Anywhere 用 QAnywhere Agent -- saclient2]** を選択します。

   2 番目の **QAnywhere Agent** が起動し、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。

♦ TestMessage を開始する

1. **[スタート] - [プログラム] - [SQL Anywhere 11] - [QAnywhere] - [SQL Anywhere の使用に関するチュートリアル] - [TestMessage -- saclient1]** を選択します。
2. **[saclient1 - TestMessage]** ウィンドウを画面右側に移動します。
3. **[saclient1 - TestMessage]** ウィンドウで、**[Tools] - [Options]** をクリックします。
4. **testmessage** が **[Queue name used to listen for incoming messages]** フィールドに表示されていることを確認します。**[Cancel]** をクリックします。

5. **[スタート] - [プログラム] - [SQL Anywhere 11] - [QAnywhere] - [SQL Anywhere の使用に関するチュートリアル] - [TestMessage -- saclient2]** を選択します。
6. **[saclient2 - TestMessage]** ウィンドウを画面左側に移動します。
7. **[saclient2 - TestMessage]** ウィンドウで、**[Tools] - [Options]** をクリックします。
8. **testmessage** が **[Queue name used to listen for incoming messages]** フィールドに表示されていることを確認します。**[Cancel]** をクリックします。

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設定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。

- **スケジュール設定**　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。
- **自動**　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。
- **オン・デマンド**　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。
- **カスタム**　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。

参照

- 「QAnywhere Agent の起動」 51 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」 54 ページ
- 「qaagent ユーティリティ」 744 ページ
- 「QAnywhere 転送ルールと削除ルール」 809 ページ
- 「QAnywhere クライアント・アプリケーションの作成」 57 ページ
- *samples-dir¥QAnywhere* にある QAnywhere のサンプル。*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

このレッスンでは、TestMessage の saclient1 アプリケーションから saclient2 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. **[saclient1 - TestMessage]** ウィンドウで、**[Message] - [New]** をクリックします。
2. **[Destination ID]** フィールドに **saclient2** と入力します。
3. **[Subject]** フィールドに、現在時刻を入力します。現在時刻を使用すると、個々のメッセージを追跡しやすくなります。
4. **[Message]** フィールドに **sample** と入力します。
5. **[Send]** をクリックします。
6. **[OK]** をクリックします。
7. **[saclient2 - TestMessage]** ウィンドウで、メッセージを選択します。メッセージの内容が下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。

参照

- 「QAnywhere メッセージ・アドレス」 69 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」 823 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。

コードの多くは、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¥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++ バージョンは、*Samples¥QAnywhere¥Windows¥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 C# バージョンは、*Samples¥QAnywhere¥Windows¥.NET¥CS¥TestMessage¥TestMessage.sln* として提供されています。
- Java バージョンは、*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¥Windows Mobile Classic¥.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 2005 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 内で目的のプロジェクトを開きます。たとえば、*Samples¥QAnywhere¥Windows¥.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*) 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するアセンブリは、*iAnywhere.QAnywhere.Client.dll* という DLL として提供されています。ファイルは、次の場所にあります。

   - .NET Framework 2.0：*install-dir¥Assembly¥v2*
   - .NET Compact Framework 2.0：*install-dir¥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();
     ```

     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、MessageList_Load メソッド内で定義されたとおりに宣言する。

  ```
  _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」 7 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的に受信されます。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] ウィンドウで設定された 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]** ウィンドウ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 は送信先アドレスです。送信先アドレスは、この関数の引数として渡されます。

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

参照

- 「QAnywhere C++ API リファレンス」 405 ページ
- 「クライアント用 QAnywhere .NET API (.NET 2.0)」 226 ページ
- 「QAnywhere クライアント・アプリケーションの作成」 57 ページ
- *samples-dir¥QAnywhere* にある TestMessage サンプル。*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。

QAnywhere リファレンス

この項では、QAnywhere API の参照マニュアルを示します。

QAnywhere .NET API リファレンス

目次

クライアント用 QAnywhere .NET API (.NET 2.0)

ネームスペース

- iAnywhere.QAnywhere.Client (通常のクライアントの場合)
- iAnywhere.QAnywhere.StandAloneClient (スタンドアロン・クライアントの場合)

AcknowledgementMode 列挙体

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」 62 ページを参照してください。

暗黙的な受信確認の場合、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受信確認されます。メッセージを同期的に受信する場合、メッセージは QAManagerBase.GetMessage(string) メソッドが返されると受信確認されます。メッセージを非同期的に受信する場合、メッセージはイベント処理関数が返されると受信確認されます。

メンバ名	説明
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」 268 ページ
- 「QATransactionalManager インタフェース」 349 ページ
- 「QAManagerBase インタフェース」 273 ページ

ExceptionListener デリゲート

ExceptionListener デリゲートの定義です。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。

参照

- 「SetExceptionListener メソッド」 304 ページ

ExceptionListener2 デリゲート

ExceptionListener2 デリゲートの定義です。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。

参照

- 「SetExceptionListener2 メソッド」 305 ページ

MessageListener デリゲート

MessageListener デリゲートの定義です。MessageListener は 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

- 「SetMessageListener メソッド」 308 ページ

MessageListener2 デリゲート

MessageListener2 デリゲートの定義です。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。

参照

- 「SetMessageListener2 メソッド」 309 ページ

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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

- 「MessageProperties のメンバ」 230 ページ
- 「QAMessage インタフェース」 324 ページ
- 「GetIntProperty メソッド」 334 ページ
- 「GetStringProperty メソッド」 338 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」 231 ページ	システム・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」 231 ページ	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」 232 ページ	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」 232 ページ	システム・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」 233 ページ	システム・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」 234 ページ	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」 234 ページ	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。
「ORIGINATOR フィールド」 235 ページ	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」 235 ページ	システム・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」 236 ページ	システム・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」 236 ページ	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。

メンバ名	説明
「STATUS_TIME フィールド」 237 ページ	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」 237 ページ	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

システム・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」 792 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」 229 ページ
- 「MessageProperties のメンバ」 230 ページ
- 「MessageProperties クラス」 229 ページ
- 「GetStringProperty メソッド」 338 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」 792 ページを参照してください。

参照

- 「MessageProperties クラス」 229 ページ
- 「MessageProperties のメンバ」 230 ページ
- 「MessageProperties クラス」 229 ページ
- 「GetStringProperty メソッド」 338 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」 229 ページ
- 「MessageProperties のメンバ」 230 ページ
- 「MessageProperties クラス」 229 ページ
- 「GetIntProperty メソッド」 334 ページ

IP フィールド

システム・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」 792 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」 229 ページ
- 「MessageProperties のメンバ」 230 ページ
- 「MessageProperties クラス」 229 ページ
- 「GetStringProperty メソッド」 338 ページ

MAC フィールド

システム・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」 792 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」 229 ページ
- 「MessageProperties のメンバ」 230 ページ
- 「MessageProperties クラス」 229 ページ
- 「GetStringProperty メソッド」 338 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」 229 ページ
- 「MessageProperties のメンバ」 230 ページ
- 「MessageProperties クラス」 229 ページ
- 「MessageType 列挙体」 240 ページ
- 「GetIntProperty メソッド」 334 ページ
- 「GetStringProperty メソッド」 338 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」 792 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」 229 ページ
- 「MessageProperties のメンバ」 230 ページ
- 「MessageProperties クラス」 229 ページ
- 「GetIntProperty メソッド」 334 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」 229 ページ
- 「MessageProperties のメンバ」 230 ページ
- 「MessageProperties クラス」 229 ページ
- 「GetStringProperty メソッド」 338 ページ

RAS フィールド

システム・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」 792 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」 229 ページ
- 「MessageProperties のメンバ」 230 ページ
- 「MessageProperties クラス」 229 ページ
- 「GetStringProperty メソッド」 338 ページ

RASNAMES フィールド

システム・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」 792 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」 229 ページ
- 「MessageProperties のメンバ」 230 ページ
- 「MessageProperties クラス」 229 ページ
- 「GetStringProperty メソッド」 338 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 列挙体」 353 ページ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」 229 ページ
- 「MessageProperties のメンバ」 230 ページ
- 「StatusCodes 列挙体」 353 ページ
- 「MessageProperties クラス」 229 ページ
- 「GetIntProperty メソッド」 334 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」 229 ページ
- 「MessageProperties のメンバ」 230 ページ
- 「GetProperty メソッド」 336 ページ
- 「MessageProperties クラス」 229 ページ
- 「GetProperty メソッド」 336 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 列挙体」 353 ページ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」 229 ページ
- 「MessageProperties のメンバ」 230 ページ
- 「StatusCodes 列挙体」 353 ページ
- 「MessageProperties クラス」 229 ページ
- 「GetIntProperty メソッド」 334 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」 28 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」 239 ページ	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。

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」 239 ページ	MessageStoreProperties クラスの新しいインスタンスを初期化します。「MessageStoreProperties クラス」 238 ページを参照してください。

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();**

参照

- 「MessageStoreProperties クラス」 238 ページ

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。

構文

Visual Basic
Public Shared **MAX_DELIVERY_ATTEMPTS** As String

C#
public const string **MAX_DELIVERY_ATTEMPTS**;

備考

このフィールドの値は ias_MaxDeliveryAttempts です。

MessageType 列挙体

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 列挙体

QAMessage プロパティ型列挙体。C# の型に自然に対応します。

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。

メンバ名	説明
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 string であることを示します。
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た時点でプロバイダは 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」 243 ページ	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」 244 ページ	QABinaryMessage インスタンスの本文未読部分の先頭から、指定されたバイト数を読み込みます。
「ReadBoolean メソッド」 246 ページ	QABinaryMessage インスタンスのメッセージ本文未読部分の先頭から boolean 値を読み込みます。
「ReadChar メソッド」 246 ページ	QABinaryMessage メッセージ本文未読部分の先頭から char 値を読み込みます。
「ReadDouble メソッド」 247 ページ	QABinaryMessage メッセージ本文未読部分の先頭から double 値を読み込みます。
「ReadFloat メソッド」 247 ページ	QABinaryMessage メッセージ本文未読部分の先頭から float 値を読み込みます。
「ReadInt メソッド」 248 ページ	QABinaryMessage メッセージ本文未読部分の先頭から int 値を読み込みます。
「ReadLong メソッド」 248 ページ	QABinaryMessage メッセージ本文未読部分の先頭から long 値を読み込みます。
「ReadSbyte メソッド」 249 ページ	QABinaryMessage メッセージ本文未読部分の先頭から signed byte 値を読み込みます。
「ReadShort メソッド」 250 ページ	QABinaryMessage メッセージ本文未読部分の先頭から short 値を読み込みます。
「ReadString メソッド」 250 ページ	QABinaryMessage メッセージ本文未読部分の先頭から string 値を読み込みます。
「Reset メソッド」 251 ページ	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」 252 ページ	QABinaryMessage インスタンスのメッセージ本文に byte 配列値を追加します。

メンバ名	説明
「WriteBoolean メソッド」 253 ページ	QABinaryMessage 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」 253 ページ	QABinaryMessage 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」 254 ページ	QABinaryMessage 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」 255 ページ	QABinaryMessage 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」 255 ページ	QABinaryMessage インスタンスのメッセージ本文に int 値を追加します。
「WriteLong メソッド」 256 ページ	QABinaryMessage 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」 256 ページ	QABinaryMessage 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」 257 ページ	QABinaryMessage 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」 258 ページ	QABinaryMessage インスタンスのメッセージ本文に string 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「QABinaryMessage インタフェース」 241 ページ

ReadBinary メソッド

QABinaryMessage インスタンスの本文未読部分の先頭から、指定されたバイト数を読み込みます。

構文

Visual Basic
```
Public Function ReadBinary( _
  ByVal bytes As Byte(), _
) As Integer
```

C#
```
public int ReadBinary(
  byte[] bytes
);
```

パラメータ

- **bytes**　読み込まれるバイトを含む byte 配列。

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」 258 ページ - 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「WriteBinary メソッド」 252 ページ

ReadBinary メソッド

QABinaryMessage インスタンスの本文未読部分の先頭から、指定されたバイト数を読み込みます。

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を含む byte 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」 258 ページ - 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「WriteBinary メソッド」 252 ページ

ReadBinary メソッド

QABinaryMessage インスタンスの本文未読部分の先頭から、指定されたバイト数を読み込みます。

構文

Visual Basic

```
Public Function ReadBinary( _
  ByVal bytes As Byte(), _
  ByVal offset As Integer, _
  ByVal len As Integer _
) As Integer
```

C#

```
public int ReadBinary(
  byte[] bytes,
  int offset,
  int len
);
```

パラメータ

- **bytes**　読み込まれるバイトを含む byte 配列。
- **offset**　コピー先の配列の開始オフセット。
- **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」 258 ページ - 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「WriteBinary メソッド」 252 ページ

ReadBoolean メソッド

QABinaryMessage インスタンスのメッセージ本文未読部分の先頭から 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」 258 ページ - 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「QABinaryMessage インタフェース」 241 ページ
- 「WriteBoolean メソッド」 253 ページ

ReadChar メソッド

QABinaryMessage メッセージ本文未読部分の先頭から 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」 258 ページ - 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「QABinaryMessage インタフェース」 241 ページ
- 「WriteChar メソッド」 253 ページ

ReadDouble メソッド

QABinaryMessage メッセージ本文未読部分の先頭から 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」 258 ページ - 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「QABinaryMessage インタフェース」 241 ページ
- 「WriteDouble メソッド」 254 ページ

ReadFloat メソッド

QABinaryMessage メッセージ本文未読部分の先頭から 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」 258 ページ - 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「QABinaryMessage インタフェース」 241 ページ
- 「WriteFloat メソッド」 255 ページ

ReadInt メソッド

QABinaryMessage メッセージ本文未読部分の先頭から 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」 258 ページ - 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「QABinaryMessage インタフェース」 241 ページ
- 「WriteInt メソッド」 255 ページ

ReadLong メソッド

QABinaryMessage メッセージ本文未読部分の先頭から 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」 258 ページ - 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「QABinaryMessage インタフェース」 241 ページ
- 「WriteLong メソッド」 256 ページ

ReadSbyte メソッド

QABinaryMessage メッセージ本文未読部分の先頭から 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」 258 ページ - 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「QABinaryMessage インタフェース」 241 ページ
- 「WriteSbyte バイト」 256 ページ

ReadShort メソッド

QABinaryMessage メッセージ本文未読部分の先頭から 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」 258 ページ - 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「QABinaryMessage インタフェース」 241 ページ
- 「WriteShort メソッド」 257 ページ

ReadString メソッド

QABinaryMessage メッセージ本文未読部分の先頭から string 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 string 値。

例外

- 「QAException クラス」 258 ページ - 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「QABinaryMessage インタフェース」 241 ページ
- 「WriteString メソッド」 258 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「QABinaryMessage インタフェース」 241 ページ

WriteBinary メソッド

QABinaryMessage インスタンスのメッセージ本文に byte 配列値を追加します。

構文

Visual Basic
```
Public Sub WriteBinary( _
  ByVal val As Byte() _
)
```

C#
```
public void WriteBinary(
  byte[] val
);
```

パラメータ

- **val**　メッセージ本文に書き込む byte 配列値。

参照

- 「QABinaryMessage インタフェース」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「ReadBinary メソッド」 244 ページ

WriteBinary メソッド

QABinaryMessage インスタンスのメッセージ本文に byte 配列値を追加します。

構文

Visual Basic

```
Public Sub WriteBinary( _
  ByVal val As Byte(), _
  ByVal len As Integer _
)
```

C#

```
public void WriteBinary(
  byte[] val,
  int len
);
```

パラメータ

- **val** メッセージ本文に書き込む byte 配列値。
- **len** 書き込まれるバイト数。

参照

- 「QABinaryMessage インタフェース」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「ReadBinary メソッド」 244 ページ

WriteBinary メソッド

QABinaryMessage インスタンスのメッセージ本文に byte 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 byte 配列値。
- **offset**　byte 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。

参照

- 「QABinaryMessage インタフェース」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「ReadBinary メソッド」 244 ページ

WriteBoolean メソッド

QABinaryMessage 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「QABinaryMessage インタフェース」 241 ページ
- 「ReadBoolean メソッド」 246 ページ

WriteChar メソッド

QABinaryMessage 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「QABinaryMessage インタフェース」 241 ページ
- 「ReadChar メソッド」 246 ページ

WriteDouble メソッド

QABinaryMessage 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「QABinaryMessage インタフェース」 241 ページ
- 「ReadDouble メソッド」 247 ページ

WriteFloat メソッド

QABinaryMessage 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 int 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「QABinaryMessage インタフェース」 241 ページ
- 「ReadFloat メソッド」 247 ページ

WriteInt メソッド

QABinaryMessage インスタンスのメッセージ本文に int 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

int 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「QABinaryMessage インタフェース」 241 ページ
- 「ReadInt メソッド」 248 ページ

WriteLong メソッド

QABinaryMessage 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「QABinaryMessage インタフェース」 241 ページ
- 「ReadLong メソッド」 248 ページ

WriteSbyte バイト

QABinaryMessage 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「QABinaryMessage インタフェース」 241 ページ
- 「ReadSbyte メソッド」 249 ページ

WriteShort メソッド

QABinaryMessage 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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「QABinaryMessage インタフェース」 241 ページ
- 「ReadShort メソッド」 250 ページ

WriteString メソッド

QABinaryMessage インスタンスのメッセージ本文に string 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 string 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 を呼び出す必要があります。「ReadString メソッド」 250 ページを参照してください。

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」 241 ページ
- 「QABinaryMessage のメンバ」 242 ページ
- 「QABinaryMessage インタフェース」 241 ページ
- 「ReadString メソッド」 250 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 の例外をキャッチします。

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」 260 ページ	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」 260 ページ	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
「DetailedMessage プロパティ」 268 ページ	例外の詳細な説明です。
「ErrorCode プロパティ」 268 ページ	例外のエラー・コードです。
「NativeErrorCode プロパティ」 268 ページ	例外のネイティブ・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たヘルプ・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。
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。

COMMON_ALREADY_OPEN_ERROR フィールド

QAManager はすでに開いています。

構文

Visual Basic
Public Shared **COMMON_ALREADY_OPEN_ERROR** As Integer

C#
public const int **COMMON_ALREADY_OPEN_ERROR**;

COMMON_GETQUEUEDEPTH_ERROR フィールド

キューの深さの取得中にエラーが発生しました。

構文

Visual Basic
Public Shared **COMMON_GETQUEUEDEPTH_ERROR** As Integer

C#
public const int **COMMON_GETQUEUEDEPTH_ERROR**;

COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG フィールド

フィルタが ALL の場合、指定された宛先には QAManagerBase.getQueueDepth を使用できません。

構文

Visual Basic
Public Shared **COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG** As Integer

C#
public const int **COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG**;

COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID フィールド

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。

構文

Visual Basic
Public Shared **COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID** As Integer

C#
public const int **COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID**;

COMMON_GET_INIT_FILE_ERROR フィールド

クライアントのプロパティ・ファイルにアクセスできません。

構文

Visual Basic
Public Shared **COMMON_GET_INIT_FILE_ERROR** As Integer

C#
public const int **COMMON_GET_INIT_FILE_ERROR**;

COMMON_GET_PROPERTY_ERROR フィールド

メッセージ・ストアからプロパティを取得中にエラーが発生しました。

構文

Visual Basic
Public Shared **COMMON_GET_PROPERTY_ERROR** As Integer

C#
public const int **COMMON_GET_PROPERTY_ERROR**;

COMMON_INIT_ERROR フィールド

初期化エラーです。

構文

Visual Basic
Public Shared **COMMON_INIT_ERROR** As Integer

C#
public const int **COMMON_INIT_ERROR**;

COMMON_INIT_THREAD_ERROR フィールド

バックグラウンド・スレッドを初期化中にエラーが発生しました。

構文

Visual Basic
Public Shared **COMMON_INIT_THREAD_ERROR** As Integer

C#
public const int **COMMON_INIT_THREAD_ERROR**;

COMMON_INVALID_PROPERTY フィールド

クライアントのプロパティ・ファイルに無効なプロパティが存在します。

構文

Visual Basic
Public Shared **COMMON_INVALID_PROPERTY** As Integer

C#
public const int **COMMON_INVALID_PROPERTY**;

COMMON_MSG_ACKNOWLEDGE_ERROR フィールド

メッセージの確認中にエラーが発生しました。

構文

Visual Basic
Public Shared **COMMON_MSG_ACKNOWLEDGE_ERROR** As Integer

C#
public const int **COMMON_MSG_ACKNOWLEDGE_ERROR**;

COMMON_MSG_CANCEL_ERROR フィールド

メッセージのキャンセル中にエラーが発生しました。

構文

Visual Basic
Public Shared **COMMON_MSG_CANCEL_ERROR** As Integer

C#
public const int **COMMON_MSG_CANCEL_ERROR**;

COMMON_MSG_CANCEL_ERROR_SENT フィールド

メッセージのキャンセル中にエラーが発生しました。すでに送信されたメッセージをキャンセルすることはできません。

構文

Visual Basic
Public Shared **COMMON_MSG_CANCEL_ERROR_SENT** As Integer

C#
public const int **COMMON_MSG_CANCEL_ERROR_SENT**;

COMMON_MSG_NOT_WRITEABLE_ERROR フィールド

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。

構文

Visual Basic
Public Shared **COMMON_MSG_NOT_WRITEABLE_ERROR** As Integer

C#
public const int **COMMON_MSG_NOT_WRITEABLE_ERROR**;

COMMON_MSG_RETRIEVE_ERROR フィールド

クライアント・メッセージ・ストアからメッセージを取得中にエラーが発生しました。

構文

Visual Basic
Public Shared **COMMON_MSG_RETRIEVE_ERROR** As Integer

C#
public const int **COMMON_MSG_RETRIEVE_ERROR**;

COMMON_MSG_STORE_NOT_INITIALIZED フィールド

メッセージ・ストアがメッセージング向けに初期化されていません。

構文

Visual Basic
Public Shared **COMMON_MSG_STORE_NOT_INITIALIZED** As Integer

C#
public const int **COMMON_MSG_STORE_NOT_INITIALIZED**;

COMMON_MSG_STORE_TOO_LARGE フィールド

メッセージ・ストアがデバイスの空きディスク領域と比べて大きすぎます。

構文

Visual Basic
Public Shared **COMMON_MSG_STORE_TOO_LARGE** As Integer

C#
public const int **COMMON_MSG_STORE_TOO_LARGE**;

COMMON_NOT_OPEN_ERROR フィールド

QAManager が開いていません。

構文

Visual Basic
Public Shared **COMMON_NOT_OPEN_ERROR** As Integer

C#
public const int **COMMON_NOT_OPEN_ERROR**;

COMMON_NO_DEST_ERROR フィールド

宛先が指定されていません。

構文

Visual Basic
Public Shared **COMMON_NO_DEST_ERROR** As Integer

C#
public const int **COMMON_NO_DEST_ERROR**;

COMMON_NO_IMPLEMENTATION フィールド

この関数は実装されていません。

構文

Visual Basic
Public Shared **COMMON_NO_IMPLEMENTATION** As Integer

C#
public const int **COMMON_NO_IMPLEMENTATION**;

COMMON_OPEN_ERROR フィールド

メッセージ・ストアへの接続を開くときにエラーが発生しました。

構文

Visual Basic
Public Shared **COMMON_OPEN_ERROR** As Integer

C#
public const int **COMMON_OPEN_ERROR**;

COMMON_OPEN_LOG_FILE_ERROR フィールド

ログ・ファイルを開くときにエラーが発生しました。

構文

Visual Basic
Public Shared **COMMON_OPEN_LOG_FILE_ERROR** As Integer

C#
public const int **COMMON_OPEN_LOG_FILE_ERROR**;

COMMON_OPEN_MAXTHREADS_ERROR フィールド

サーバの最大同時要求数が少なすぎるため QAManager を開くことができません。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。

構文

Visual Basic
Public Shared **COMMON_OPEN_MAXTHREADS_ERROR** As Integer

C#
public const int **COMMON_OPEN_MAXTHREADS_ERROR**;

COMMON_SELECTOR_SYNTAX_ERROR フィールド

指定されたセレクタに構文エラーがあります。

構文

Visual Basic
Public Shared **COMMON_SELECTOR_SYNTAX_ERROR** As Integer

C#
public const int **COMMON_SELECTOR_SYNTAX_ERROR**;

COMMON_SET_PROPERTY_ERROR フィールド

プロパティをメッセージ・ストアに格納中にエラーが発生しました。

構文

Visual Basic
Public Shared **COMMON_SET_PROPERTY_ERROR** As Integer

C#
public const int **COMMON_SET_PROPERTY_ERROR**;

COMMON_TERMINATE_ERROR フィールド

終了エラーです。

構文

Visual Basic
Public Shared **COMMON_TERMINATE_ERROR** As Integer

C#
public const int **COMMON_TERMINATE_ERROR**;

COMMON_UNEXPECTED_EOM_ERROR フィールド

予期しないメッセージの終わりに到達しました。

構文

Visual Basic
Public Shared **COMMON_UNEXPECTED_EOM_ERROR** As Integer

C#
public const int **COMMON_UNEXPECTED_EOM_ERROR**;

COMMON_UNREPRESENTABLE_TIMESTAMP フィールド

タイムスタンプが許容範囲外です。

構文

Visual Basic
Public Shared **COMMON_UNREPRESENTABLE_TIMESTAMP** As Integer

C#
public const int **COMMON_UNREPRESENTABLE_TIMESTAMP**;

QA_NO_ERROR フィールド

エラーはありません。

構文

Visual Basic
Public Shared **QA_NO_ERROR** As Integer

C#
public const int **QA_NO_ERROR**;

DetailedMessage プロパティ

例外の詳細な説明です。

構文

Visual Basic
Public Readonly Property **DetailedMessage** As String

C#
public string **DetailedMessage** {get;}

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;}

NativeErrorCode プロパティ

例外のネイティブ・エラー・コードです。

構文

Visual Basic
Public Readonly Property **NativeErrorCode** As Integer

C#
public int **NativeErrorCode** {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」 273 ページ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」 269 ページ
- 「AcknowledgementMode 列挙体」 226 ページ
- 「QATransactionalManager インタフェース」 349 ページ

QAManager のメンバ

パブリック・メソッド

メンバ名	説明
「Acknowledge メソッド」 270 ページ	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」 270 ページ	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」 271 ページ	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」 272 ページ	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」 273 ページ	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 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。

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

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 に変わります。QAMessage は、MessageProperties.STATUS 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」 823 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 268 ページ
- 「QAManager のメンバ」 269 ページ
- 「QAManager インタフェース」 268 ページ
- 「AcknowledgeUntil メソッド」 271 ページ
- 「StatusCodes 列挙体」 353 ページ
- 「MessageProperties クラス」 229 ページ
- 「AcknowledgeAll メソッド」 270 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

QAMessage が受信確認されると、その MessageProperties.STATUS 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」 823 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 268 ページ
- 「QAManager のメンバ」 269 ページ
- 「QAManager インタフェース」 268 ページ
- 「Acknowledge メソッド」 270 ページ
- 「AcknowledgeUntil メソッド」 271 ページ
- 「StatusCodes 列挙体」 353 ページ
- 「MessageProperties クラス」 229 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

QAMessage が受信確認されると、その MessageProperties.STATUS 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」 823 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 268 ページ
- 「QAManager のメンバ」 269 ページ
- 「QAManager インタフェース」 268 ページ
- 「Acknowledge メソッド」 270 ページ
- 「AcknowledgeAll メソッド」 270 ページ
- 「StatusCodes 列挙体」 353 ページ
- 「MessageProperties クラス」 229 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。

データベース接続エラーが検出された場合は、Close メソッドに続けて Open メソッドを呼び出し、QAManager に再接続することができます。QAManager に再接続するときに、QAManager を再作成したり、プロパティをリセットしたり、メッセージ・リスナをリセットしたりする必要はありません。最初の Open 呼び出し時に QAManager のプロパティを変更することはできません。また、2 度目以降に Open を呼び出すときも、同じ受信確認モードを指定する必要があります。

例外

- 「QAException クラス」 258 ページ - 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」 268 ページ
- 「QAManager のメンバ」 269 ページ
- 「Close メソッド」 283 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

これらのメッセージは、QAManagerBase.GetMessage を使用して再受信する必要があります。

例外

- 「QAException クラス」 258 ページ - 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」 268 ページ
- 「QAManager のメンバ」 269 ページ
- 「QAManager インタフェース」 268 ページ
- 「GetMessage メソッド」 289 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」 54 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」 28 ページと「MessageStoreProperties クラス」を参照してください。

参照

- 「QAManagerBase のメンバ」 274 ページ
- 「CreateBinaryMessage メソッド」 284 ページ
- 「TriggerSendReceive メソッド」 317 ページ
- 「Close メソッド」 283 ページ
- 「QAException クラス」 258 ページ

QAManagerBase のメンバ

パブリック・プロパティ

メンバ名	説明
「Mode プロパティ」 278 ページ	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」 278 ページ	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」 279 ページ	このメソッドは推奨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」 280 ページ	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」 281 ページ	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」 282 ページ	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」 282 ページ	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」 283 ページ	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」 284 ページ	QABinaryMessage オブジェクトを作成します。
「CreateTextMessage メソッド」 284 ページ	QATextMessage オブジェクトを作成します。
「GetBooleanStoreProperty メソッド」 285 ページ	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」 286 ページ	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。
「GetFloatStoreProperty メソッド」 287 ページ	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」 288 ページ	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」 288 ページ	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」 289 ページ	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。

メンバ名	説明
「GetMessageBySelector メソッド」 290 ページ	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」 291 ページ	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」 292 ページ	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」 293 ページ	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」 294 ページ	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」 295 ページ	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」 296 ページ	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」 297 ページ	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」 298 ページ	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」 298 ページ	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」 299 ページ	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」 300 ページ	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 string 値を取得します。
「PropertyExists メソッド」 300 ページ	指定されたプロパティに値が存在するかどうかをチェックします。
「PutMessage メソッド」 301 ページ	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」 302 ページ	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」 303 ページ	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」 304 ページ	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」 304 ページ	QAnywhere メッセージの非同期的な処理時に QAExceptions を受信するように、ExceptionListener デリゲートを設定します。「ExceptionListener デリゲート」 227 ページを参照してください。
「SetExceptionListener2 メソッド」 305 ページ	QAnywhere メッセージの非同期的な処理時に QAExceptions を受信するように、ExceptionListener2 デリゲートを設定します。「ExceptionListener2 デリゲート」 227 ページを参照してください。
「SetFloatStoreProperty メソッド」 306 ページ	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」 307 ページ	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」 307 ページ	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」 308 ページ	QAnywhere メッセージを非同期に受信するように、MessageListener デリゲートを設定します。「MessageListener デリゲート」 228 ページを参照してください。
「SetMessageListener2 メソッド」 309 ページ	QAnywhere メッセージを非同期的に受信するように、MessageListener2 デリゲートを設定します。「MessageListener2 デリゲート」 229 ページを参照してください。
「SetMessageListenerBySelector メソッド」 310 ページ	メッセージ・セレクタを使用して、QAnywhere メッセージを非同期的に受信するように、MessageListener デリゲートを設定します。「MessageListener デリゲート」 228 ページを参照してください。
「SetMessageListenerBySelector2 メソッド」 311 ページ	メッセージ・セレクタを使用して、QAnywhere メッセージを非同期的に受信するように、MessageListener2 デリゲートを設定します。「MessageListener2 デリゲート」 229 ページを参照してください。
「SetProperty メソッド」 312 ページ	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。

メンバ名	説明
「SetSbyteStoreProperty メソッド」 313 ページ	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」 313 ページ	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」 314 ページ	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」 315 ページ	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 string 値に設定します。
「Start メソッド」 316 ページ	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。
「Stop メソッド」 316 ページ	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」 317 ページ	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。

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 列挙体」 226 ページ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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「BrowseMessagesByQueue メソッド」 281 ページ
- 「BrowseMessagesByID メソッド」 280 ページ
- 「BrowseMessages メソッド」 279 ページ

BrowseMessages メソッド

このメソッドは推奨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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「BrowseMessagesByQueue メソッド」 281 ページ
- 「BrowseMessagesByID メソッド」 280 ページ
- 「BrowseMessagesBySelector メソッド」 282 ページ
- 「BrowseMessages メソッド」 279 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「BrowseMessagesByQueue メソッド」 281 ページ
- 「BrowseMessages メソッド」 278 ページ
- 「BrowseMessages メソッド」 279 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「BrowseMessagesByID メソッド」 280 ページ
- 「BrowseMessages メソッド」 278 ページ
- 「BrowseMessages メソッド」 279 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「BrowseMessagesByQueue メソッド」 281 ページ
- 「BrowseMessages メソッド」 278 ページ
- 「BrowseMessages メソッド」 279 ページ
- 「BrowseMessagesByID メソッド」 280 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」 823 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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。

データベース接続エラーが検出された場合は、Close メソッドに続けて Open メソッドを呼び出し、QAManager に再接続することができます。QAManager に再接続するときに、QAManager を再作成したり、プロパティをリセットしたり、メッセージ・リスナをリセットしたりする必要はありません。最初の Open 呼び出し時に QAManager のプロパティを変更することはできません。また、2 度目以降に Open を呼び出すときも、同じ受信確認モードを指定する必要があります。

例外

- 「QAException クラス」 258 ページ - 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。

参照

- 「Open メソッド」 272 ページ

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」 258 ページ - 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「QABinaryMessage インタフェース」 241 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」 258 ページ - 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「QATextMessage インタフェース」 346 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」 238 ページ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」 28 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「MessageStoreProperties クラス」 238 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」 238 ページ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」 28 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「MessageStoreProperties クラス」 238 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」 238 ページ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」 28 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「MessageStoreProperties クラス」 238 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」 238 ページ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」 28 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「MessageStoreProperties クラス」 238 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」 238 ページ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」 28 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「MessageStoreProperties クラス」 238 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。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (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」 258 ページ - 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「QAMessage インタフェース」 324 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。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (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」 258 ページ - 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「QAMessage インタフェース」 324 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。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (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」 258 ページ - 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「QAMessage インタフェース」 324 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。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (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」 258 ページ - 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「QAMessage インタフェース」 324 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。メッセージを同期的に受信する場合は、このメソッドを使用します。メッセージを非同期的に受信する (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」 258 ページ - 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「QAMessage インタフェース」 324 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。メッセージを同期的に受信する場合は、このメソッドを使用します。「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」 258 ページ - 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。

GetProperty メソッド

QAnywhere Manager の設定プロパティを返します。

構文

Visual Basic
```
Public Sub GetProperty( _
  ByVal name As String _
)
```

C#
```
public void GetProperty(
  string  name
);
```

パラメータ

- **name**　QAnywhere Manager の設定プロパティ名。

例外

- 「QAException クラス」 258 ページ - 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。

参照

- 「QAManagerBase インタフェース」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ

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。コミットされていない出力メッセージも含まれます。

構文

Visual Basic
```
Overloads Public Function GetQueueDepth( _
  ByVal queue As String, _
  ByVal filter As QueueDepthFilter _
) As Integer
```

C#
```
public int GetQueueDepth(
  string  address,
  QueueDepthFilter filter
);
```

パラメータ

- **queue**　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。

戻り値

整数 - 受信されていないメッセージ数。

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 メソッドを使用)。

例外

- 「QAException クラス」 258 ページ.

参照

- 「QAManagerBase インタフェース」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「QueueDepthFilter 列挙体」 352 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

整数 - 受信されていないメッセージ数。

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 メソッドを使用)。コミットされていない出力メッセージも含まれます。

例外

- 「QAException クラス」 258 ページ.

参照

- 「QAManagerBase インタフェース」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「QueueDepthFilter 列挙体」 352 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」 238 ページ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」 28 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「MessageStoreProperties クラス」 238 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」 238 ページ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」 28 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「MessageStoreProperties クラス」 238 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」 238 ページ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」 28 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「MessageStoreProperties クラス」 238 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」 28 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 string 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

string 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」 238 ページ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」 28 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「MessageStoreProperties クラス」 238 ページ

PropertyExists メソッド

指定されたプロパティに値が存在するかどうかをチェックします。

構文

Visual Basic
```
Public Sub PropertyExists( _
  ByVal propName As String _
) As Boolean
```

C#
```
public bool PropertyExists(
  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

プロパティにマッピングされた値がメッセージ・ストアに存在する場合は true、そうでない場合は false。

例外

- 「QAException クラス」 258 ページ - プロパティ値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」 273 ページ
- 「QAManagerBase のメンバ」 274 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」 54 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」 69 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「PutMessageTimeToLive メソッド」 302 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」 54 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」 69 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 238 ページ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」 28 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「MessageStoreProperties クラス」 238 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」 238 ページ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」 28 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「MessageStoreProperties クラス」 238 ページ

SetExceptionListener メソッド

QAnywhere メッセージの非同期的な処理時に QAExceptions を受信するように、ExceptionListener デリゲートを設定します。「ExceptionListener デリゲート」 227 ページを参照してください。

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 デリゲートには、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 デリゲート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 メッセージの非同期的な処理時に QAExceptions を受信するように、ExceptionListener2 デリゲートを設定します。「ExceptionListener2 デリゲート」 227 ページを参照してください。

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 デリゲートには、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 デリゲート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」 238 ページ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」 28 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「MessageStoreProperties クラス」 238 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」 238 ページ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」 28 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「MessageStoreProperties クラス」 238 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」 238 ページ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」 28 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「MessageStoreProperties クラス」 238 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 デリゲートを設定します。「MessageListener デリゲート」 228 ページを参照してください。

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的に受信する場合は、このメソッドを使用します。

MessageListener デリゲートには、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。1 つのキューには、1 つのリスナ・デリゲート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 デリゲート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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「MessageListener デリゲート」 228 ページ

SetMessageListener2 メソッド

QAnywhere メッセージを非同期的に受信するように、MessageListener2 デリゲートを設定します。「MessageListener2 デリゲート」 229 ページを参照してください。

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的に受信する場合は、このメソッドを使用します。

MessageListener2 デリゲートには、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。1 つのキューには、1 つのリスナ・デリゲート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 デリゲート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・セレクタを使用して、QAnywhere メッセージを非同期的に受信するように、MessageListener デリゲートを設定します。「MessageListener デリゲート」 228 ページを参照してください。

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的に受信する場合は、このメソッドを使用します。

MessageListener デリゲートには、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。1 つのキューには、1 つのリスナ・デリゲー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。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 デリゲート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」 69 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「MessageListener デリゲート」 228 ページ

SetMessageListenerBySelector2 メソッド

メッセージ・セレクタを使用して、QAnywhere メッセージを非同期的に受信するように、MessageListener2 デリゲートを設定します。「MessageListener2 デリゲート」 229 ページを参照してください。

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的に受信する場合は、このメソッドを使用します。

MessageListener2 デリゲートには、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。1 つのキューには、1 つのリスナ・デリゲー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。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 デリゲート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」 69 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」 98 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」 99 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」 258 ページ - 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「Open メソッド」 272 ページ
- 「Open メソッド」 351 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」 238 ページ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」 28 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「MessageStoreProperties クラス」 238 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」 238 ページ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」 28 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「MessageStoreProperties クラス」 238 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 string 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」 238 ページ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」 28 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「MessageStoreProperties クラス」 238 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 string 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**　string 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」 238 ページ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」 28 ページを参照してください。

参照

- 「QAManagerBase インタフェース」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「MessageStoreProperties クラス」 238 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・リスナをメッセージ受信用に使用することはおすすめしません。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」 258 ページ - 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「Stop メソッド」 316 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」 258 ページ - 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「Start メソッド」 316 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、メッセージ転送をどのようにして実行するかを決定します。たとえば、クライアントが Push 通知を受信した場合や、ユーザが 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を自動的に実行することができます。

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」 54 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 273 ページ
- 「QAManagerBase のメンバ」 274 ページ
- 「PutMessage メソッド」 301 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**

C#
public abstract class **QAManagerFactory**

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」 319 ページ	QAManagerFactory のシングルトン・インスタンスです。

パブリック・コンストラクタ

メンバ名	説明
「QAManagerFactory コンストラクタ」 319 ページ	

パブリック・メソッド

メンバ名	説明
「CreateQAManager メソッド」 320 ページ	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。
「CreateQATransactionalManager メソッド」 322 ページ	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。

QAManagerFactory コンストラクタ

構文

Visual Basic
Public Sub **New()**

C#
public **QAManagerFactory();**

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」 258 ページ - 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。

CreateQAManager メソッド

QAManager インスタンスを作成します。

構文

Visual Basic
Public Function **CreateQAManager()** As QAManager

C#
public QAManager **CreateQAManager();**

戻り値

新しい QAManager インスタンス。

備考

QAManager は、デフォルトのプロパティを使用して作成されます。インスタンスの作成後に QAManagerBase.SetProperty 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。「SetProperty メソッド」 312 ページを参照してください。

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」 98 ページを参照してください。

例外

- 「QAException クラス」 258 ページ.

参照

- 「QAnywhere Manager の設定プロパティをファイルに設定する」 99 ページ
- 「QAManagerFactory クラス」 318 ページ
- 「QAManagerFactory のメンバ」 318 ページ
- 「QAManager インタフェース」 268 ページ

CreateQAManager メソッド

指定されたプロパティを持つ QAManager インスタンスを作成します。

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に使用するプロパティ・ファイルを指定します。

戻り値

新しい QAManager インスタンス。

備考

iniFile が NULL の場合、QAManager はデフォルトのプロパティを使用して作成されます。インスタンスの作成後に QAManagerBase.SetProperty 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。「SetProperty メソッド」 312 ページを参照してください。

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」 98 ページを参照してください。

例外

- 「QAException クラス」 258 ページ.

参照

- 「QAnywhere Manager の設定プロパティをファイルに設定する」 99 ページ
- 「QAManagerFactory クラス」 318 ページ
- 「QAManagerFactory のメンバ」 318 ページ
- 「QAManager インタフェース」 268 ページ

CreateQAManager メソッド

指定されたプロパティを持つ QAManager インスタンスを作成します。

構文

Visual Basic

```
Public Function CreateQAManager( _
  ByVal props As Hashtable _
) As QAManager
```

C#

```
public QAManager CreateQAManager(
  Hashtable  props
);
```

パラメータ

- **props**　QAManager インスタンスを設定するために使用するプロパティのハッシュテーブルを指定します。

戻り値

新しい QAManager インスタンス。

備考

props が NULL の場合、QAManager はデフォルトのプロパティを使用して作成されます。インスタンスの作成後に QAManagerBase.SetProperty 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。「SetProperty メソッド」 312 ページを参照してください。

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」 98 ページを参照してください。

例外

- 「QAException クラス」 258 ページ.

参照

- 「QAManagerFactory クラス」 318 ページ
- 「QAManagerFactory のメンバ」 318 ページ
- 「QAManager インタフェース」 268 ページ

CreateQATransactionalManager メソッド

QATransactionalManager インスタンスを作成します。

構文

Visual Basic
Public Function **CreateQATransactionalManager()** As QATransactionalManager

C#
public QATransactionalManager **CreateQATransactionalManager();**

戻り値

新しい QATransactionalManager インスタンス。

備考

QATransactionalManager は、デフォルトのプロパティを使用して作成されます。インスタンスの作成後に QAManagerBase.SetProperty 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。「SetProperty メソッド」 312 ページを参照してください。

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」 98 ページを参照してください。

例外

- 「QAException クラス」 258 ページ.

参照

- 「QAnywhere Manager の設定プロパティをファイルに設定する」 99 ページ
- 「QAManagerFactory クラス」 318 ページ
- 「QAManagerFactory のメンバ」 318 ページ
- 「QATransactionalManager インタフェース」 349 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 インスタンスを作成します。

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に使用するプロパティ・ファイルを指定します。

戻り値

新しく設定された QATransactionalManager インスタンス。

備考

iniFile が NULL の場合、QATransactionalManager はデフォルトのプロパティを使用して作成されます。インスタンスの作成後に QAManagerBase.SetProperty 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。「SetProperty メソッド」 312 ページを参照してください。

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」 98 ページを参照してください。

例外

- 「QAException クラス」 258 ページ.

参照

- 「QAnywhere Manager の設定プロパティをファイルに設定する」 99 ページ
- 「QAManagerFactory クラス」 318 ページ
- 「QAManagerFactory のメンバ」 318 ページ
- 「QATransactionalManager インタフェース」 349 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 インスタンスを作成します。

構文

Visual Basic

```
Public Function CreateQATransactionalManager( _
  ByVal props As Hashtable _
) As QATransactionalManager
```

C#

```
public QATransactionalManager CreateQATransactionalManager(
  Hashtable  props
);
```

パラメータ

- **props**　QATransactionalManager インスタンスを設定するために使用するプロパティのハッシュテーブルを指定します。

戻り値

新しく設定された QATransactionalManager インスタンス。

備考

props が NULL の場合、QATransactionalManager はデフォルトのプロパティを使用して作成されます。インスタンスの作成後に QAManagerBase.SetProperty 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。「SetProperty メソッド」 312 ページを参照してください。

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」 98 ページを参照してください。

例外

- 「QAException クラス」 258 ページ.

参照

- 「QAManagerFactory クラス」 318 ページ
- 「QAManagerFactory のメンバ」 318 ページ
- 「QATransactionalManager インタフェース」 349 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」 229 ページ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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

- 「QAMessage のメンバ」 325 ページ
- 「QABinaryMessage インタフェース」 241 ページ
- 「QATextMessage インタフェース」 346 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」 327 ページ	QAMessage インスタンスの送信先アドレスです。
「Expiration プロパティ」 327 ページ	メッセージの有効期限を取得します。
「InReplyToID プロパティ」 328 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 です。
「MessageID プロパティ」 328 ページ	メッセージのグローバルなユニークな ID です。
「Priority プロパティ」 329 ページ	メッセージの優先度 (0 から 9) です。
「Redelivered プロパティ」 329 ページ	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」 330 ページ	メッセージの返信先アドレスです。
「Timestamp プロパティ」 330 ページ	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」 331 ページ	メッセージの本文をクリアします。
「ClearProperties メソッド」 331 ページ	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」 331 ページ	boolean 型のメッセージ・プロパティを取得します。
「GetByteProperty メソッド」 332 ページ	byte 型のメッセージ・プロパティを取得します。
「GetDoubleProperty メソッド」 333 ページ	double 型のメッセージ・プロパティを取得します。

メンバ名	説明
「GetFloatProperty メソッド」 333 ページ	float 型のメッセージ・プロパティを取得します。
「GetIntProperty メソッド」 334 ページ	int 型のメッセージ・プロパティを取得します。
「GetLongProperty メソッド」 335 ページ	long 型のメッセージ・プロパティを取得します。
「GetProperty メソッド」 336 ページ	メッセージのプロパティを取得します。
「GetPropertyNames メソッド」 336 ページ	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」 336 ページ	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」 337 ページ	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」 338 ページ	short 型のメッセージ・プロパティを取得します。
「GetStringProperty メソッド」 338 ページ	string 型のメッセージ・プロパティを取得します。
「PropertyExists メソッド」 339 ページ	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」 340 ページ	boolean 型のプロパティを設定します。
「SetByteProperty メソッド」 340 ページ	byte 型のプロパティを設定します。
「SetDoubleProperty メソッド」 341 ページ	double 型のプロパティを設定します。
「SetFloatProperty メソッド」 342 ページ	float 型のプロパティを設定します。
「SetIntProperty メソッド」 342 ページ	int 型のプロパティを設定します。

メンバ名	説明
「SetLongProperty メソッド」 343 ページ	long 型のプロパティを設定します。
「SetProperty メソッド」 344 ページ	プロパティを設定します。
「SetSbyteProperty メソッド」 344 ページ	signed byte 型のプロパティを設定します。
「SetShortProperty メソッド」 345 ページ	short 型のプロパティを設定します。
「SetStringProperty メソッド」 346 ページ	string 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 Readonly Property **Address** As String

C#
public string **Address** {get;}

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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

- 「QAMessage インタフェース」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「PutMessage メソッド」 301 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 の存続時間の引数を現在の時間に追加することで設定されるため、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、「QAnywhere メッセージの概要」 16 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、「QAnywhere メッセージの概要」 16 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なユニークな 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

- 「QAMessage インタフェース」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「PutMessage メソッド」 301 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、「QAnywhere メッセージの概要」 16 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。アプリケーションが次の起動時に同じメッセージを受信すると、Redelivered ヘッダが true になります。

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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

- 「QAMessage インタフェース」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「QAManager インタフェース」 268 ページ
- 「AcknowledgementMode 列挙体」 226 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、「QAnywhere メッセージの概要」 16 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。

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、「QAnywhere メッセージの概要」 16 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「MessageProperties クラス」 229 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「MessageProperties クラス」 229 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「MessageProperties クラス」 229 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「MessageProperties クラス」 229 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「MessageProperties クラス」 229 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「MessageProperties クラス」 229 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、string 型、または DateTime 型でなければなりません。

例外

- 「QAException クラス」 258 ページ - 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「MessageProperties クラス」 229 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

例外

- 「QAException クラス」 258 ページ - 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」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「MessageProperties クラス」 229 ページ

GetStringProperty メソッド

string 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

- 「QAMessage インタフェース」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「MessageProperties クラス」 229 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

- 「QAMessage インタフェース」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「MessageProperties クラス」 229 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

- 「QAMessage インタフェース」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「MessageProperties クラス」 229 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

- 「QAMessage インタフェース」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「MessageProperties クラス」 229 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

- 「QAMessage インタフェース」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「MessageProperties クラス」 229 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

- 「QAMessage インタフェース」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「MessageProperties クラス」 229 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

- 「QAMessage インタフェース」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「MessageProperties クラス」 229 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 string 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

- 「QAMessage インタフェース」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「MessageProperties クラス」 229 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

- 「QAMessage インタフェース」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「MessageProperties クラス」 229 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

- 「QAMessage インタフェース」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「MessageProperties クラス」 229 ページ

SetStringProperty メソッド

string 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

- 「QAMessage インタフェース」 324 ページ
- 「QAMessage のメンバ」 325 ページ
- 「MessageProperties クラス」 229 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た時点でプロバイダは 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」 347 ページ
- 「QABinaryMessage インタフェース」 241 ページ
- 「QAMessage インタフェース」 324 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」 347 ページ	メッセージ・テキストです。
「TextLength プロパティ」 348 ページ	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」 348 ページ	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」 349 ページ	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」 349 ページ	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」 98 ページを参照してください。

参照

- 「QATextMessage インタフェース」 346 ページ
- 「QATextMessage のメンバ」 347 ページ
- 「ReadText メソッド」 348 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」 273 ページ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」 350 ページ
- 「QATransactionalManager インタフェース」 349 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」 350 ページ	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」 351 ページ	QATransactionalManager インスタンスをオープンします。
「Rollback メソッド」 351 ページ	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」 258 ページ - 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」 349 ページ
- 「QATransactionalManager のメンバ」 350 ページ
- 「QATransactionalManager インタフェース」 349 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」 258 ページ - 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」 349 ページ
- 「QATransactionalManager のメンバ」 350 ページ
- 「QATransactionalManager インタフェース」 349 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」 258 ページ - 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」 349 ページ
- 「QATransactionalManager のメンバ」 350 ページ
- 「QATransactionalManager インタフェース」 349 ページ

QueueDepthFilter 列挙体

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。
LOCAL	キューが指定された場合は、そのキューに送信される未受信のローカル・メッセージの数をカウントします。そうでない場合は、メッセージ・ストア内の未受信のローカル・メッセージの合計数 (システム・メッセージを除く) をカウントします。
OUTGOING	発信メッセージのみカウントします。

参照

- 「GetQueueDepth メソッド」 296 ページ
- 「GetQueueDepth メソッド」 295 ページ

StatusCodes 列挙体

この列挙体は、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されました。
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	この定数は、メッセージが最終ステータスになったかどうかを示します。メッセージは、そのステータスがこの定数を超えている場合にのみ最終ステータスになったとみなされます。
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ましたが、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、または受信されまし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。

Web サービス用 QAnywhere .NET API (.NET 2.0)

ネームスペース

iAnywhere.QAnywhere.WS

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」 355 ページ	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」 356 ページ	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」 356 ページ	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」 356 ページ	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」 357 ページ	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」 357 ページ	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。

メンバ名	説明
「SetListener メソッド」 358 ページ	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」 358 ページ	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」 359 ページ	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。
「SetRequestProperty メソッド」 359 ページ	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」 360 ページ	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 ～ 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」 361 ページ - WSBase の構築で問題が発生した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」 361 ページ - WSBase の構築で問題が発生した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」 354 ページ
- 「WSBase のメンバ」 354 ページ
- 「WSStatus 列挙体」 404 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 Listener 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、listener に NULL を指定して SetListener を呼び出します。

このメソッドは、以前に呼び出された SetListener で設定された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 Listener 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、listener に NULL を指定して SetListener を呼び出します。

このメソッドは、以前に呼び出された SetListener で設定された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 ～ 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

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。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」 364 ページ	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」 364 ページ	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」 364 ページ	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」 364 ページ	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」 362 ページ	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」 363 ページ	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」 363 ページ	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」 365 ページ	この例外に関連付けられているエラー・コードです。
HelpLink (Exception から継承)	この例外に関連付けられたヘルプ・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。
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」 366 ページ	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」 365 ページ (WSException から継承)	この例外に関連付けられているエラー・コードです。
HelpLink (Exception から継承)	この例外に関連付けられたヘルプ・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。

メンバ名	説明
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。
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」 367 ページ	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」 367 ページ	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。

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 メソッド」 373 ページ	この WSResult が処理されたことを確認します。
「GetArrayValue メソッド」 373 ページ	この WSResult から複合型の値の配列を取得します。
「GetBoolArrayValue メソッド」 373 ページ	この WSResult から bool 値の配列を取得します。

メンバ名	説明
「GetBooleanArrayValue メソッド」 374 ページ	この WSResult から Boolean 値の配列を取得します。
「GetBooleanValue メソッド」 374 ページ	この WSResult から Boolean 値を取得します。
「GetBoolValue メソッド」 375 ページ	この WSResult から bool 値を取得します。
「GetByteArrayValue メソッド」 376 ページ	この WSResult から byte 値の配列を取得します。
「GetByteValue メソッド」 376 ページ	この WSResult から byte 値を取得します。
「GetCharArrayValue メソッド」 377 ページ	この WSResult から char 値の配列を取得します。
「GetCharValue メソッド」 377 ページ	この WSResult から char 値を取得します。
「GetDecimalArrayValue メソッド」 378 ページ	この WSResult から decimal 値の配列を取得します。
「GetDecimalValue メソッド」 378 ページ	この WSResult から decimal 値を取得します。
「GetDoubleArrayValue メソッド」 379 ページ	この WSResult から double 値の配列を取得します。
「GetDoubleValue メソッド」 379 ページ	この WSResult から double 値を取得します。
「GetErrorMessage メソッド」 380 ページ	エラー・メッセージを取得します。
「GetFloatArrayValue メソッド」 380 ページ	この WSResult から float 値の配列を取得します。
「GetFloatValue メソッド」 380 ページ	この WSResult から float 値を取得します。
「GetInt16ArrayValue メソッド」 381 ページ	この WSResult から Int16 値の配列を取得します。

メンバ名	説明
「GetInt16Value メソッド」 381 ページ	この WSResult から Int16 値を取得します。
「GetInt32ArrayValue メソッド」 382 ページ	この WSResult から Int32 値の配列を取得します。
「GetInt32Value メソッド」 383 ページ	この WSResult から Int32 値を取得します。
「GetInt64ArrayValue メソッド」 383 ページ	この WSResult から Int64 値の配列を取得します。
「GetInt64Value メソッド」 384 ページ	この WSResult から Int64 値を取得します。
「GetIntArrayValue メソッド」 384 ページ	この WSResult から int 値の配列を取得します。
「GetIntValue メソッド」 385 ページ	この WSResult から int 値を取得します。
「GetLongArrayValue メソッド」 385 ページ	この WSResult から long 値の配列を取得します。
「GetLongValue メソッド」 386 ページ	この WSResult から long 値を取得します。
「GetNullableBoolArrayValue メソッド」 386 ページ	この WSResult から bool 値の配列を取得します。
「GetNullableBoolValue メソッド」 387 ページ	この WSResult から bool 値を取得します。
「GetNullableDecimalArrayValue メソッド」 387 ページ	この WSResult から NullableDecimal 値の配列を取得します。
「GetNullableDecimalValue メソッド」 388 ページ	この WSResult から NullableDecimal 値を取得します。
「GetNullableDoubleArrayValue メソッド」 388 ページ	この WSResult から double 値の配列を取得します。
「GetNullableDoubleValue メソッド」 389 ページ	この WSResult から double 値を取得します。

メンバ名	説明
「GetNullableFloatArrayValue メソッド」 389 ページ	この WSResult から float 値の配列を取得します。
「GetNullableFloatValue メソッド」 390 ページ	この WSResult から float 値を取得します。
「GetNullableIntArrayValue メソッド」 390 ページ	この WSResult から int 値の配列を取得します。
「GetNullableIntValue メソッド」 391 ページ	この WSResult から int 値を取得します。
「GetNullableLongArrayValue メソッド」 391 ページ	この WSResult から long 値の配列を取得します。
「GetNullableLongValue メソッド」 392 ページ	この WSResult から Int64 値を取得します。
「GetNullableSByteArrayValue メソッド」 392 ページ	この WSResult から byte 値の配列を取得します。
「GetNullableSByteValue メソッド」 393 ページ	この WSResult から byte 値を取得します。
「GetNullableShortArrayValue メソッド」 393 ページ	この WSResult から short 値の配列を取得します。
「GetNullableShortValue メソッド」 394 ページ	この WSResult から short 値を取得します。
「GetObjectArrayValue メソッド」 394 ページ	この WSResult から object 値の配列を取得します。
「GetObjectValue メソッド」 395 ページ	この WSResult から object 値を取得します。
「GetRequestID メソッド」 395 ページ	この WSResult が表す要求 ID を取得します。
「GetSByteArrayValue メソッド」 396 ページ	この WSResult から sbyte 値の配列を取得します。
「GetSByteValue メソッド」 396 ページ	この WSResult から sbyte 値を取得します。

メンバ名	説明
「GetShortArrayValue メソッド」 397 ページ	この WSResult から short 値の配列を取得します。
「GetShortValue メソッド」 397 ページ	この WSResult から short 値を取得します。
「GetSingleArrayValue メソッド」 398 ページ	この WSResult から single 値の配列を取得します。
「GetSingleValue メソッド」 398 ページ	この WSResult から single 値を取得します。
「GetStatus メソッド」 399 ページ	この WSResult のステータスを取得します。
「GetStringArrayValue メソッド」 399 ページ	この WSResult から string 値の配列を取得します。
「GetStringValue メソッド」 400 ページ	この WSResult から string 値を取得します。
「GetUIntArrayValue メソッド」 400 ページ	この WSResult から unsigned int 値の配列を取得します。
「GetUIntValue メソッド」 401 ページ	この WSResult から unsigned int 値を取得します。
「GetULongArrayValue メソッド」 401 ページ	この WSResult から unsigned long 値の配列を取得します。
「GetULongValue メソッド」 402 ページ	この WSResult から unsigned long 値を取得します。
「GetUShortArrayValue メソッド」 402 ページ	この WSResult から unsigned short 値の配列を取得します。
「GetUShortValue メソッド」 403 ページ	この WSResult から unsigned short 値を取得します。
「GetValue メソッド」 403 ページ	この WSResult から複合型の値を取得します。
「SetLogger メソッド」 404 ページ	デバッグのオン／オフを切り替えます。

Acknowledge メソッド

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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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 decimal 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」 361 ページ - 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 decimal 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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 int 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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 int 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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 single 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」 361 ページ - 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 single 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」 361 ページ - 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」 368 ページ
- 「WSResult のメンバ」 368 ページ
- 「WSStatus 列挙体」 404 ページ

GetStringArrayValue メソッド

この WSResult から string 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」 361 ページ - 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 string 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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」 361 ページ - 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 列挙体

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。

QAnywhere C++ API リファレンス

目次

QAnywhere C++ API では、Ultra Light データベースをサポートしていません。

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、open 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」 62 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、QATransactionalManager インスタンスの commit 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。

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、commit 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、getMode プロパティを使用して判別できます。

参照

「同期的なメッセージ受信」 82 ページ

「非同期的なメッセージ受信」 83 ページ

「QAManager クラス」 442 ページ

「QATransactionalManager クラス」 512 ページ

「QAManagerBase クラス」 447 ページ

メンバ

AcknowledgementMode 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」 407 ページ
- 「IMPLICIT_ACKNOWLEDGEMENT 変数」 407 ページ
- 「TRANSACTIONAL 変数」 407 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受信確認されます。

メッセージを同期的に受信する場合、メッセージは getMessage メソッドが返されると受信確認されます。メッセージを非同期的に受信する場合、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、「QAnywhere メッセージの概要」 16 ページを参照してください。

```
QATextMessage * t_msg;
```

次の例では、getIntProperty メソッドを使用して MSG_TYPE に対応する値を取得します。MessageType 列挙体は、int 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、MSG_TYPE と RASNAMES 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」 482 ページ

メンバ

MessageProperties 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。

ADAPTER は、最初のパラメータとして getStringProperty 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」 723 ページを参照してください。

参照

「getStringProperty 関数」 494 ページ

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。

ADAPTERS は、最初のパラメータとして getStringProperty 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」 723 ページと「getStringProperty 関数」 494 ページを参照してください。

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。

DELIVERY_COUNT は、最初のパラメータとして setStringProperty メソッドまたは getStringProperty 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」 504 ページ

「getStringProperty 関数」 494 ページ

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。

IP は、最初のパラメータとして getStringProperty 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」 723 ページを参照してください。

参照

「getStringProperty 関数」 494 ページ

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。

MAC は、最初のパラメータとして getStringProperty 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」 723 ページを参照してください。

参照

「getStringProperty 関数」 494 ページ

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。MSG_TYPE は、最初のパラメータとして setIntProperty メソッドまたは getIntProperty 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」 417 ページ

「setIntProperty 関数」 501 ページ

「getIntProperty 関数」 490 ページ

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」 723 ページを参照してください。

NETWORK_STATUS は、最初のパラメータとして setStringProperty メソッドまたは getStringProperty 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」 504 ページ

「getStringProperty 関数」 494 ページ

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。

ORIGINATOR は、最初のパラメータとして setStringProperty 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」 504 ページ

「getStringProperty 関数」 494 ページ

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。

RAS は、最初のパラメータとして getStringProperty 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」 723 ページを参照してください。

参照

「getStringProperty 関数」 494 ページ

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」 723 ページを参照してください。

RASNAMES は、最初のパラメータとして getStringProperty 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」 504 ページ

「getStringProperty 関数」 494 ページ

「setIntProperty 関数」 501 ページ

「getIntProperty 関数」 490 ページ

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」 518 ページを参照してください。このフィールドの値は "ias_Status" です。

STATUS は、最初のパラメータとして getIntProperty 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」 518 ページ

「setIntProperty 関数」 501 ページ

「getIntProperty 関数」 490 ページ

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。

STATUS_TIME は、最初のパラメータとして getLongProperty 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」 491 ページ

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」 518 ページを参照してください。

このフィールドの値は "ias_TransmissionStatus" です。

TRANSMISSION_STATUS は、最初のパラメータとして setIntProperty メソッドまたは getIntProperty 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」 518 ページ

「setIntProperty 関数」 501 ページ

「getIntProperty 関数」 490 ページ

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。

Properties 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」 28 ページを参照してください。

メンバ

MessageStoreProperties 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」 416 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 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。

参照

「StatusCodes クラス」 518 ページ

MessageType クラス

構文

public **MessageType**

備考

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 メソッドを示します。

メッセージ・タイプは、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 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」 417 ページ
- 「PUSH_NOTIFICATION 変数」 418 ページ
- 「REGULAR 変数」 418 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、ADAPTER、NETWORK、NETWORK_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」 723 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、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。

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」 723 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」 482 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た時点でプロバイダは reset 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 writeString 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();
}
```


受信が終了すると、最初の readString の呼び出しから "Q" が返され、2 番目の 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。

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」 54 ページと「QAnywhere クライアント・アプリケーションの作成」 57 ページを参照してください。

メンバ

QABinaryMessage 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未読部分の先頭から、指定されたバイト数を読み込みます。

参照

「writeBinary 関数」 428 ページ

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未読部分の先頭から、boolean 値を読み込みます。

参照

「writeBoolean 関数」 428 ページ

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未読部分の先頭から、符号付き 8 ビット値を読み込みます。

参照

「writeByte 関数」 429 ページ

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未読部分の先頭から、char 値を読み込みます。

参照

「writeChar 関数」 429 ページ

戻り値

読み込まれた char 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未読部分の先頭から、double 値を読み込みます。

参照

「writeDouble 関数」 429 ページ

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未読部分の先頭から、float 値を読み込みます。

参照

「writeFloat 関数」 430 ページ

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未読部分の先頭から、符号付き 32 ビット int 値を読み込みます。

参照

「writeInt 関数」 430 ページ

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未読部分の先頭から、符号付き 64 ビット int 値を読み込みます。

参照

「writeLong 関数」 431 ページ

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未読部分の先頭から、符号付き 16 ビット値を読み込みます。

参照

「writeShort 関数」 431 ページ

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 ターミネータも含む)。

備考

QABinaryMessage インスタンスのメッセージ本文未読部分の先頭から、string 値を読み込みます。

参照

「writeString 関数」 432 ページ

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 byte 配列値。
- **offset**　byte 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 byte 配列値を追加します。

参照

「readBinary 関数」 422 ページ

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

QABinaryMessage 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」 423 ページ

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 byte 配列値。

備考

QABinaryMessage 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」 423 ページ

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

QABinaryMessage 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」 424 ページ

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

QABinaryMessage 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」 424 ページ

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

QABinaryMessage 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 int に変換され、上位のバイトから追加されます。

参照

「readFloat 関数」 425 ページ

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

QABinaryMessage インスタンスのメッセージ本文に int 値を追加します。

int 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」 425 ページ

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

QABinaryMessage 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」 426 ページ

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

QABinaryMessage 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」 426 ページ

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 string 値。

備考

QABinaryMessage インスタンスのメッセージ本文に string 値を追加します。

受信側アプリケーションは、writeString が呼び出されるたびに readString を呼び出す必要があります。

文字列の UTF-8 表記は、最大で 32767 バイトまで可能です。

参照

「readString 関数」 427 ページ

~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、getLastError 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」 460 ページ

メンバ

QAError 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

構文

const qa_int **QAError::COMMON_ALREADY_OPEN_ERROR**

備考

QAManager はすでに開いています。

COMMON_GETQUEUEDEPTH_ERROR 変数

構文

const qa_int **QAError::COMMON_GETQUEUEDEPTH_ERROR**

備考

キューの深さの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、指定された宛先には 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

クライアントのプロパティ・ファイルにアクセスできません。

COMMON_GET_PROPERTY_ERROR 変数

構文

const qa_int **QAError::COMMON_GET_PROPERTY_ERROR**

備考

メッセージ・ストアからプロパティを取得中にエラーが発生しました。

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・スレッドを初期化中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

クライアントのプロパティ・ファイル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。

すでに送信されたメッセージをキャンセルすることはできません。

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中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

メッセージをクライアント・メッセージ・ストアに格納中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向け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・ストアがデバイスの空きディスク領域と比べて大きすぎます。

COMMON_NOT_OPEN_ERROR 変数

構文

const qa_int **QAError::COMMON_NOT_OPEN_ERROR**

備考

QAManager が開いていません。

COMMON_NO_DEST_ERROR 変数

構文

const qa_int **QAError::COMMON_NO_DEST_ERROR**

備考

宛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

この関数は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・ストアへの接続を開く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・ファイルを開く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

サーバの最大同時要求数が少なすぎるため QAManager を開くことができません。

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_SET_PROPERTY_ERROR 変数

構文

const qa_int **QAError::COMMON_SET_PROPERTY_ERROR**

備考

プロパティをメッセージ・ストアに格納中にエラーが発生しました。

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」 447 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」 447 ページを参照してください。

QAManager は、AcknowledgementMode 列挙体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 クラス」 442 ページを参照してください。

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」 478 ページ

「QAManagerBase クラス」 447 ページ

「QATransactionalManager クラス」 512 ページ

「AcknowledgementMode クラス」 406 ページ

メンバ

QAManager 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

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。

QAMessage が受信確認されると、その STATUS プロパティは RECEIVED に変わります。QAMessage は、ステータスが 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」 823 ページを参照してください。

参照

「acknowledgeAll 関数」 444 ページ

「acknowledgeUntil 関数」 445 ページ

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。

QAMessage が受信確認されると、その STATUS プロパティは RECEIVED に変わります。QAMessage は、ステータスが 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」 823 ページを参照してください。

参照

「acknowledge 関数」 444 ページ

「acknowledgeUntil 関数」 445 ページ

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。

QAMessage が受信確認されると、その STATUS プロパティは RECEIVED に変わります。QAMessage は、ステータスが 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」 823 ページを参照してください。

参照

「acknowledge 関数」 444 ページ

「acknowledgeAll 関数」 444 ページ

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 関数は、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。

データベース接続エラーが検出された場合は、close 関数に続けて open 関数を呼び出し、QAManager に再接続することができます。QAManager に再接続するときに、QAManager を再作成したり、プロパティをリセットしたり、メッセージ・リスナをリセットしたりする必要はありません。最初の open 呼び出し時に QAManager のプロパティを変更することはできません。また、2 度目以降に open を呼び出すときも、同じ受信確認モードを指定する必要があります。

参照

「AcknowledgementMode クラス」 406 ページ

「close 関数」 454 ページ

戻り値

処理が成功した場合は true、失敗した場合は false。

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。

これらのメッセージは、getMessage を使用して再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」 442 ページ
- 「QATransactionalManager クラス」 512 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、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います。適切な QAMessage インスタンスを作成するには、createBinaryMessage メソッドと 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、putMessage 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、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」 54 ページを参照してください。

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、getLastError、getLastErrorMessage、getLastNativeError 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」 28 ページと「MessageStoreProperties クラス」 416 ページを参照してください。

参照

「QATransactionalManager クラス」 512 ページ

「QAManager クラス」 442 ページ

メンバ

QAManagerBase 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

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、nextStorePropertyName メソッドに渡されます。beginEnumStorePropertyNames メソッドと nextStorePropertyName メソッドを使用することで、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。beginEnumStorePropertyNames の呼び出しと endEnumStorePropertyNames 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」 467 ページ

「beginEnumStorePropertyNames 関数」 449 ページ

「endEnumStorePropertyNames 関数」 455 ページ

戻り値

nextStorePropertyName 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、browseNextMessage に渡されます。このメソッドと browseNextMessage 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・ストア内にある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、getMessage を使用します。

参照

「browseNextMessage 関数」 452 ページ

「browseMessagesByQueue 関数」 451 ページ

「browseMessagesByID 関数」 450 ページ

「browseClose 関数」 449 ページ

戻り値

browseNextMessage 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、browseNextMessage に渡されます。このメソッドと browseNextMessage 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・ストア内にある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、getMessage を使用します。

参照

「browseNextMessage 関数」 452 ページ

「browseMessagesByQueue 関数」 451 ページ

「browseMessages 関数」 450 ページ

「browseClose 関数」 449 ページ

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、browseNextMessage に渡されます。このメソッドと browseNextMessage 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・ストア内にある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、getMessage を使用します。

参照

「browseNextMessage 関数」 452 ページ

「browseMessagesByID 関数」 450 ページ

「browseMessages 関数」 450 ページ

「browseClose 関数」 449 ページ

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、browseNextMessage に渡されます。このメソッドと browseNextMessage 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・ストア内にある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、getMessage を使用します。

参照

「browseNextMessage 関数」 452 ページ

「browseMessagesByID 関数」 450 ページ

「browseMessagesByQueue 関数」 451 ページ

「browseMessages 関数」 450 ページ

「browseClose 関数」 449 ページ

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、browseMessages 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」 450 ページ

「browseMessagesByQueue 関数」 451 ページ

「browseMessagesByID 関数」 450 ページ

「browseClose 関数」 449 ページ

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」 823 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。

データベース接続エラーが検出された場合は、close 関数に続けて open 関数を呼び出し、QAManager に再接続することができます。QAManager に再接続するときに、QAManager を再作成したり、プロパティをリセットしたり、メッセージ・リスナをリセットしたりする必要はありません。最初の open 呼び出し時に QAManager のプロパティを変更することはできません。また、2 度目以降に open を呼び出すときも、同じ受信確認モードを指定する必要があります。「open 関数」 446 ページを参照してください。

参照

「open 関数」 514 ページ

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」 419 ページ

戻り値

新しい QABinaryMessage インスタンス。「QABinaryMessage クラス」 419 ページを参照してください。

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、string 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」 507 ページ

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、createTextMessage または createBinaryMessage によって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」 449 ページ

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、getMessage を使用)。

参照

「QueueDepthFilter クラス」 516 ページ.

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」 416 ページ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」 28 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」 416 ページ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」 28 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」 416 ページ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」 28 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」 416 ページ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」 28 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」 416 ページ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」 28 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」 433 ページを参照してください。

参照

「getLastErrorMsg 関数」 460 ページ

「QAError クラス」 433 ページ

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。

getLastError 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 をキャッチした後で取得できます。

参照

「getLastError 関数」 459 ページ

「QAError クラス」 433 ページ

戻り値

エラー・メッセージ。

getLastNativeError 関数

構文

an_sql_code **QAManagerBase::getLastNativeError()**

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。

-1 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」 433 ページを参照してください。

参照

「getLastError 関数」 459 ページ

戻り値

ネイティブのエラー・コード。

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」 416 ページ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」 28 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 を返します。「QAMessage クラス」 482 ページを参照してください。

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。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (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。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (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。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (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。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (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。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (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。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (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。

EXPLICIT_ACKNOWLEDGEMENT と IMPLICIT_ACKNOWLEDGEMENT は QAManager インスタンスで使用されます。TRANSACTIONAL は QATransactionalManager インスタンス用のモードです。

参照

「AcknowledgementMode クラス」 406 ページ

戻り値

受信確認モード。

getQueueDepth 関数

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

qa_int **QAManagerBase::getQueueDepth(**
  qa_const_string *queue*,

```
  qa_short filter
)
```

パラメータ

- **queue**　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。コミットされていない出力メッセージも含まれます。

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、getMessage メソッドを使用)。

参照

「QueueDepthFilter クラス」 516 ページ.

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」 416 ページ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」 28 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 ターミネータも含む)。

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 string 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」 416 ページ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」 28 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 ターミネータ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」 449 ページ

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」 416 ページ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」 28 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」 416 ページ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」 28 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」 416 ページ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」 28 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」 416 ページ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」 28 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」 416 ページ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」 28 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」 416 ページ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」 28 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的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。

listener は、QAMessageListener インタフェースで定義された唯一のメソッドである onMessage を実装するクラスのインスタンスです。onMessage には、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。1 つのキューには、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的に受信する場合は、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」 69 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・セレクタを使用して、QAnywhere メッセージを非同期的に受信するように、メッセージ・リスナ・クラスを設定します。

listener は、QAMessageListener インタフェースで定義された唯一のメソッドである onMessage を実装するクラスのインスタンスです。onMessage には、QAMessage パラメータを 1 つだけ指定できます。

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。1 つのキューには、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的に受信する場合は、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」 69 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」 98 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 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」 99 ページを参照してください。

「注意：」open メソッドを呼び出す前に、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」 416 ページ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」 28 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 string 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」 416 ページ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」 28 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・リスナをメッセージ受信用に使用することはおすすめしません。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 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 関数」 476 ページ

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 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 関数」 476 ページ

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、メッセージ転送をどのようにして実行するかを決定します。たとえば、クライアントが Push 通知を受信した場合や、ユーザが 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を自動的に実行することができます。

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」 54 ページを参照してください。

参照

「putMessage 関数」 468 ページ

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」 442 ページ

「QATransactionalManager クラス」 512 ページ

メンバ

QAManagerFactory 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」 478 ページ
● 「createQATransactionalManager 関数」 479 ページ
● 「deleteQAManager 関数」 479 ページ
● 「deleteQATransactionalManager 関数」 480 ページ
● 「getLastError 関数」 480 ページ
● 「getLastErrorMsg 関数」 481 ページ
● 「getLastNativeError 関数」 481 ページ

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、QAManager はデフォルトのプロパティを使用して作成されます。QAManager の作成後に setProperty() メソッドを使用して、QAnywhere Manager のプロパティをプログラム設定できます。

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」 98 ページを参照してください。

参照

「QAManager クラス」 442 ページ

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、QAManager はデフォルトのプロパティを使用して作成されます。QATransactionalManager の作成後に setProperty メソッドを使用して、QAnywhere Manager のプロパティをプログラム設定できます。

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」 98 ページを参照してください。

参照

「QATransactionalManager クラス」 512 ページ

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」 433 ページを参照してください。

参照

「getLastErrorMsg 関数」 481 ページ

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。

getLastError 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 をキャッチした後で取得できます。

参照

「getLastError 関数」 480 ページ

「QAError クラス」 433 ページ

戻り値

エラー・メッセージ。

getLastNativeError 関数

構文

an_sql_code **QAManagerFactory::getLastNativeError()**

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・コードです。

-1 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」 433 ページを参照してください。

参照

「getLastError 関数」 480 ページ

戻り値

ネイティブのエラー・コード。

QAMessage クラス

構文

public **QAMessage**

派生クラス

- 「QABinaryMessage クラス」 419 ページ
- 「QATextMessage クラス」 507 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」 408 ページ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、「QAnywhere メッセージの概要」 16 ページを参照してください。

メンバ

QAMessage 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。

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 ～ 4 は通常のメッセージ、値 5 ～ 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 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、「QAnywhere メッセージの概要」 16 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

「MessageProperties クラス」 408 ページ

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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

「MessageProperties クラス」 408 ページ

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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

「MessageProperties クラス」 408 ページ

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、putMessageTimeToLive の存続時間の引数を現在の時間に追加することで設定されるため、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、「QAnywhere メッセージの概要」 16 ページを参照してください。

戻り値

有効期限。

参照

「getTimestamp 関数」 495 ページ

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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

「MessageProperties クラス」 408 ページ

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、「QAnywhere メッセージの概要」 16 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

「MessageProperties クラス」 408 ページ

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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

「MessageProperties クラス」 408 ページ

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。

putMessage 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、「QAnywhere メッセージの概要」 16 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 ～ 4 を通常のメッセージ、優先度 5 ～ 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、「QAnywhere メッセージの概要」 16 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、EXPLICIT_ACKNOWLEDGEMENT でオープンされた「QAManager クラス」 442 ページ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。アプリケーションが次の起動時に同じメッセージを受信すると、Redelivered ヘッダが true になります。

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、「QAnywhere メッセージの概要」 16 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、「QAnywhere メッセージの概要」 16 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

「MessageProperties クラス」 408 ページ

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 ターミネータの 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

「MessageProperties クラス」 408 ページ

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 ターミネータの 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

「MessageProperties クラス」 408 ページ

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、「QAnywhere メッセージの概要」 16 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フォーマットされた string として取得します。

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、「QAnywhere メッセージの概要」 16 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 ターミネータ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、「QAnywhere メッセージの概要」 16 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

「MessageProperties クラス」 408 ページ

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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

「MessageProperties クラス」 408 ページ

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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

「MessageProperties クラス」 408 ページ.

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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

「MessageProperties クラス」 408 ページ.

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、「QAnywhere メッセージの概要」 16 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

「MessageProperties クラス」 408 ページ.

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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

「MessageProperties クラス」 408 ページ.

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、「QAnywhere メッセージの概要」 16 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、「QAnywhere メッセージの概要」 16 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、「QAnywhere メッセージの概要」 16 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、「QAnywhere メッセージの概要」 16 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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

「MessageProperties クラス」 408 ページ.

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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

「MessageProperties クラス」 408 ページ.

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、「QAnywhere メッセージの概要」 16 ページを参照してください。

参照

「getTimestamp 関数」 495 ページ

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 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」 506 ページ
- 「~QAMessageListener 関数」 506 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」 482 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た時点でプロバイダは 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。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」 419 ページ

メンバ

QATextMessage 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。

- 「setShortProperty 関数」 503 ページ
- 「setStringProperty 関数」 504 ページ
- 「setText 関数」 510 ページ
- 「setTimestamp 関数」 504 ページ
- 「writeText 関数」 511 ページ
- 「~QATextMessage 関数」 511 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 string 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、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、この関数は NULL を返します。この場合は、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」 98 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 string。

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 0 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 ターミネータも含む)。

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 string。

備考

メッセージのデータが格納されている string 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」 447 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」 447 ページ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。putMessage と getMessage のすべての呼び出しをコミットする場合は、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 クラス」 512 ページ.

メンバ

QATransactionalManager 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。

- 「setMessageListener 関数」 473 ページ
- 「setMessageListenerBySelector 関数」 473 ページ
- 「setProperty 関数」 474 ページ
- 「setShortStoreProperty 関数」 475 ページ
- 「setStringStoreProperty 関数」 475 ページ
- 「start 関数」 476 ページ
- 「stop 関数」 476 ページ
- 「triggerSendReceive 関数」 477 ページ
- 「~QATransactionalManager 関数」 515 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、putMessage() と getMessage() のすべての呼び出しをコミットします。

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 クラス」 512 ページ

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」 512 ページ

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 putMessage と getMessage 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」 512 ページ

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 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」 516 ページ
- 「INCOMING 変数」 516 ページ
- 「OUTGOING 変数」 517 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。

LOCAL 変数

構文

const qa_short **QueueDepthFilter::LOCAL**

備考

LOCAL フィルタを指定して getQueueDepth が呼び出され、キューが指定されている場合、この変数はそのキューに送信される未受信のローカル・メッセージの数を返します。キューが指定されていない場合、LOCAL はメッセージ・ストア内の未受信のローカル・メッセージの合計数 (システム・メッセージを除く) を返します。

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 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されました。

このコードの値は 40 です。このコードは STATUS 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 STATUS で使用されます。

FINAL 変数

構文

const qa_int **StatusCodes::FINAL**

備考

この定数は、メッセージが最終ステータスになったかどうかを示します。

メッセージは、そのステータスがこの定数を超えている場合にのみ最終ステータスになったとみなされます。

このコードの値は 20 です。このコードは STATUS 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 TRANSMISSION_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ましたが、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 STATUS 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 STATUS 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、または受信されまし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 STATUS 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 TRANSMISSION_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 TRANSMISSION_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 STATUS 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 TRANSMISSION_STATUS で使用されます。

QAnywhere Java API リファレンス

目次

クライアント用 QAnywhere Java API

パッケージ

- ianywhere.qanywhere.client (通常のクライアントの場合)
- ianywhere.qanywhere.standaloneclient (スタンドアロン・クライアントの場合)

AcknowledgementMode インタフェース

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。

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。

暗黙的な受信確認の場合、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」 562 ページ

「QATransactionalManager インタフェース」 632 ページ

「QAManagerBase インタフェース」 568 ページ

メンバ

ianywhere.qanywhere.client.AcknowledgementMode 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」 524 ページ
- 「IMPLICIT_ACKNOWLEDGEMENT 変数」 525 ページ
- 「TRANSACTIONAL 変数」 525 ページ

EXPLICIT_ACKNOWLEDGEMENT 変数

構文

final 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。

参照

「QAManager インタフェース」 562 ページ

IMPLICIT_ACKNOWLEDGEMENT 変数

構文

final 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受信確認されます。

メッセージを同期的に受信する場合、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受信確認されます。

参照

「getMessage メソッド」 578 ページ

TRANSACTIONAL 変数

構文

final short **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」 632 ページ

MessageProperties インタフェース

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。

```
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 列挙体は、int 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 (継承されたメンバも含みます) を以下に示します。

ADAPTER 変数

構文

final String **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」 525 ページ

「getStringProperty メソッド」 616 ページ

ADAPTERS 変数

構文

final 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。

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」 525 ページ

「getStringProperty メソッド」 616 ページ

DELIVERY_COUNT 変数

構文

final 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。

IP 変数

構文

final String **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」 525 ページ

「getStringProperty メソッド」 616 ページ

MAC 変数

構文

final String **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」 525 ページ

「getStringProperty メソッド」 616 ページ

MSG_TYPE 変数

構文

final 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。

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」 525 ページ

「getIntProperty メソッド」 611 ページ

NETWORK_STATUS 変数

構文

final 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。

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」 525 ページ

「getIntProperty メソッド」 611 ページ

ORIGINATOR 変数

構文

final 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。

RAS 変数

構文

final String **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」 525 ページ

「getStringProperty メソッド」 616 ページ

RASNAMES 変数

構文

final String **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」 525 ページ

「getStringProperty メソッド」 616 ページ

STATUS 変数

構文

final 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。

参照

「StatusCodes インタフェース」 637 ページ

STATUS_TIME 変数

構文

final 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」 613 ページ

TRANSMISSION_STATUS 変数

構文

final 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。

参照

「StatusCodes インタフェース」 637 ページ

MessageStoreProperties インタフェース

構文

public **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」 568 ページ

メンバ

ianywhere.qanywhere.client.MessageStoreProperties 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。

- 「MAX_DELIVERY_ATTEMPTS 変数」 533 ページ

MAX_DELIVERY_ATTEMPTS 変数

構文

final 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 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」 641 ページ

MessageType インタフェース

構文

public **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 (継承されたメンバも含みます) を以下に示します。

- 「NETWORK_STATUS_NOTIFICATION 変数」 534 ページ
- 「PUSH_NOTIFICATION 変数」 534 ページ
- 「REGULAR 変数」 534 ページ

NETWORK_STATUS_NOTIFICATION 変数

構文

final 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、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 **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 **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 **PropertyType**

備考

QAMessage プロパティ型列挙体。Java の型に自然に対応します。

参照

「QAMessage インタフェース」 604 ページ

メンバ

ianywhere.qanywhere.client.PropertyType 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。

- 「PROPERTY_TYPE_BOOLEAN 変数」 535 ページ
- 「PROPERTY_TYPE_BYTE 変数」 535 ページ
- 「PROPERTY_TYPE_DOUBLE 変数」 536 ページ
- 「PROPERTY_TYPE_FLOAT 変数」 536 ページ
- 「PROPERTY_TYPE_INT 変数」 536 ページ
- 「PROPERTY_TYPE_LONG 変数」 536 ページ
- 「PROPERTY_TYPE_SHORT 変数」 536 ページ
- 「PROPERTY_TYPE_STRING 変数」 537 ページ
- 「PROPERTY_TYPE_UNKNOWN 変数」 537 ページ

PROPERTY_TYPE_BOOLEAN 変数

構文

final short **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 **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 **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 **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 **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 **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 **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 **PropertyType.PROPERTY_TYPE_STRING**

備考

プロパティ型が String であることを示します。

PROPERTY_TYPE_UNKNOWN 変数

構文

final short **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 **QABinaryMessage**

基本クラス

- 「QAMessage インタフェース」 604 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た時点でプロバイダは 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();
}
```

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」 604 ページ

「readString メソッド」 546 ページ

メンバ

ianywhere.qanywhere.client.QABinaryMessage 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 メソッド

構文

long **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 **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 byte 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未読部分の先頭から、指定されたバイト数を読み込みます。

参照

「writeBinary メソッド」 547 ページ

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 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 byte 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未読部分の先頭から、指定されたバイト数を読み込みます。

参照

「writeBinary メソッド」 547 ページ

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 **QABinaryMessage.readBinary(**
  byte[] *dest*,
  int *offset*,
  int *length*
)
throws **QAException**

パラメータ

- **dest**　読み込まれたバイトを保持する byte 配列。
- **offset**　コピー先の配列の開始オフセット。
- **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未読部分の先頭から指定されたバイト数を読み込んで、dest[offset] を開始位置として配列 dest に格納します。

参照

「writeBinary メソッド」 547 ページ

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 **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未読部分の先頭から boolean 値を読み込みます。

参照

「writeBoolean メソッド」 549 ページ

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 **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 メッセージ本文未読部分の先頭から signed byte 値を読み込みます。

参照

「writeByte メソッド」 549 ページ

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 **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 メッセージ本文未読部分の先頭から char 値を読み込みます。

参照

「writeChar メソッド」 550 ページ

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 **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 メッセージ本文未読部分の先頭から double 値を読み込みます。

参照

「writeDouble メソッド」 550 ページ

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 **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 メッセージ本文未読部分の先頭から float 値を読み込みます。

参照

「writeFloat メソッド」 551 ページ

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 **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 メッセージ本文未読部分の先頭から int 値を読み込みます。

参照

「writeInt メソッド」 551 ページ

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 **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 メッセージ本文未読部分の先頭から long 値を読み込みます。

参照

「writeLong メソッド」 552 ページ

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 **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 メッセージ本文未読部分の先頭から short 値を読み込みます。

参照

「writeShort メソッド」 552 ページ

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 **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 メッセージ本文未読部分の先頭から string 値を読み込みます。

参照

「writeString メソッド」 553 ページ

戻り値

メッセージ本文から読み込まれる string 値。

reset メソッド

構文

void **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 **QABinaryMessage.writeBinary(**
byte[] *val*
)
throws **QAException**

パラメータ

- **val** メッセージ本文に書き込む byte 配列値。

スロー

- メッセージ本文への byte 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 byte 配列値を追加します。

参照

「readBinary メソッド」 540 ページ

writeBinary メソッド

構文

void **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 byte 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 byte 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 byte 配列値を追加します。

参照

「readBinary メソッド」 540 ページ

writeBinary メソッド

構文

```
void 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 byte 配列値。
- **offset**　byte 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 byte 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 byte 配列値を追加します。

参照

「readBinary メソッド」 540 ページ

writeBoolean メソッド

構文

void **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

QABinaryMessage 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」 542 ページ

writeByte メソッド

構文

void **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

QABinaryMessage 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」 543 ページ

writeChar メソッド

構文

```
void 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

QABinaryMessage 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」 543 ページ

writeDouble メソッド

構文

```
void 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

QABinaryMessage 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」 544 ページ

writeFloat メソッド

構文

void **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

QABinaryMessage 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 int 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」 544 ページ

writeInt メソッド

構文

void **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 int 値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 int 値を追加します。

int 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」 545 ページ

writeLong メソッド

構文

```
void 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

QABinaryMessage 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」 545 ページ

writeShort メソッド

構文

```
void 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

QABinaryMessage 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」 546 ページ

writeString メソッド

構文

void **QABinaryMessage.writeString(**
String *val*
)
throws **QAException**

パラメータ

- **val** メッセージ本文に書き込む string 値。

スロー

- メッセージ本文への string 値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 string 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」 546 ページ

QAException クラス

構文

public **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 の例外をキャッチできます。

```
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());
    System.err.println("Native error code: " + e.getNativeErrorCode());
    System.err.println("Detailed error message: " + e.getDetailedMessage());
}
```

メンバ

ianywhere.qanywhere.client.QAException 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

構文

final int **QAException.COMMON_ALREADY_OPEN_ERROR**

備考

QAManager はすでに開いています。

参照

「QAManager インタフェース」 562 ページ

COMMON_GETQUEUEDEPTH_ERROR 変数

構文

final int **QAException.COMMON_GETQUEUEDEPTH_ERROR**

備考

キューの深さの取得中にエラーが発生しました。

参照

「getQueueDepth メソッド」 583 ページ

COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG 変数

構文

final int **QAException.COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG**

備考

フィルタが ALL の場合、指定された宛先には QAManagerBase.getQueueDepth を使用できません。

参照

「getQueueDepth メソッド」 583 ページ

COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID 変数

構文

final int **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」 583 ページ

COMMON_GET_INIT_FILE_ERROR 変数

構文

final int **QAException.COMMON_GET_INIT_FILE_ERROR**

備考

クライアントのプロパティ・ファイルにアクセスできません。

COMMON_GET_PROPERTY_ERROR 変数

構文

final int **QAException.COMMON_GET_PROPERTY_ERROR**

備考

メッセージ・ストアからプロパティを取得中にエラーが発生しました。

COMMON_INIT_ERROR 変数

構文

final int **QAException.COMMON_INIT_ERROR**

備考

初期化エラーです。

COMMON_INIT_THREAD_ERROR 変数

構文

final int **QAException.COMMON_INIT_THREAD_ERROR**

備考

バックグラウンド・スレッドを初期化中にエラーが発生しました。

COMMON_INVALID_PROPERTY 変数

構文

final int **QAException.COMMON_INVALID_PROPERTY**

備考

クライアントのプロパティ・ファイル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 **QAException.COMMON_MSG_ACKNOWLEDGE_ERROR**

備考

メッセージの確認中にエラーが発生しました。

COMMON_MSG_CANCEL_ERROR 変数

構文

final int **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 **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。

すでに送信されたメッセージをキャンセルすることはできません。

COMMON_MSG_NOT_WRITEABLE_ERROR 変数

構文

final int **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 **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中にエラーが発生しました。

COMMON_MSG_STORE_ERROR 変数

構文

final int **QAException.COMMON_MSG_STORE_ERROR**

備考

メッセージをクライアント・メッセージ・ストアに格納中にエラーが発生しました。

COMMON_MSG_STORE_NOT_INITIALIZED 変数

構文

final int **QAException.COMMON_MSG_STORE_NOT_INITIALIZED**

備考

メッセージ・ストアがメッセージング向けに初期化されていません。

COMMON_MSG_STORE_TOO_LARGE 変数

構文

final int **QAException.COMMON_MSG_STORE_TOO_LARGE**

備考

メッセージ・ストアがデバイスの空きディスク領域と比べて大きすぎます。

COMMON_NOT_OPEN_ERROR 変数

構文

final int **QAException.COMMON_NOT_OPEN_ERROR**

備考

QAManager が開いていません。

参照

「QAManager インタフェース」 562 ページ

COMMON_NO_DEST_ERROR 変数

構文

final int **QAException.COMMON_NO_DEST_ERROR**

備考

宛先が指定されていません。

COMMON_NO_IMPLEMENTATION 変数

構文

final int **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 **QAException.COMMON_OPEN_ERROR**

備考

メッセージ・ストアへの接続を開くときにエラーが発生しました。

COMMON_OPEN_LOG_FILE_ERROR 変数

構文

final int **QAException.COMMON_OPEN_LOG_FILE_ERROR**

備考

ログ・ファイルを開くときにエラーが発生しました。

COMMON_OPEN_MAXTHREADS_ERROR 変数

構文

final int **QAException.COMMON_OPEN_MAXTHREADS_ERROR**

備考

サーバの最大同時要求数が少なすぎるため QAManager を開くことができません。「-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

final int **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_SET_PROPERTY_ERROR 変数

構文

final int **QAException.COMMON_SET_PROPERTY_ERROR**

備考

プロパティをメッセージ・ストアに格納中にエラーが発生しました。

COMMON_TERMINATE_ERROR 変数

構文

final int **QAException.COMMON_TERMINATE_ERROR**

備考

終了エラーです。

COMMON_UNEXPECTED_EOM_ERROR 変数

構文

final int **QAException.COMMON_UNEXPECTED_EOM_ERROR**

備考

予期しない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 **QAException.COMMON_UNREPRESENTABLE_TIMESTAMP**

備考

タイムスタンプが許容範囲外です。

getDetailedMessage メソッド

構文

string **QAException.getDetailedMessage()**

備考

例外の詳細な説明を返します。

戻り値

例外のエラー・メッセージ。

getErrorCode メソッド

構文

int **QAException.getErrorCode()**

備考

例外のエラー・コードを返します。

戻り値

例外のエラー・コード。

getNativeErrorCode メソッド

構文

int **QAException.getNativeErrorCode()**

備考

例外のネイティブ・エラー・コードを返します。

戻り値

例外のネイティブ・エラー・コード。

QA_NO_ERROR 変数

構文

final int **QAException.QA_NO_ERROR**

備考

エラーはありません。

QAException メソッド

構文

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。

QAManager インタフェース

構文

public **QAManager**

基本クラス

- 「QAManagerBase インタフェース」 568 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」 568 ページ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」 524 ページ

「QAManagerFactory クラス」 600 ページ

「QATransactionalManager インタフェース」 632 ページ

メンバ

ianywhere.qanywhere.client.QAManager 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 メソッド

構文

```
void 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。

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」 640 ページ

「acknowledgeUntil メソッド」 566 ページ

「acknowledgeAll メソッド」 565 ページ

acknowledgeAll メソッド

構文

```
void 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。

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」 640 ページ

「acknowledge メソッド」 565 ページ

「acknowledgeUntil メソッド」 566 ページ

acknowledgeUntil メソッド

構文

```
void 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。

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

「QAMessage インタフェース」 604 ページ

「RECEIVED 変数」 640 ページ

「acknowledge メソッド」 565 ページ

「acknowledgeAll メソッド」 565 ページ

open メソッド

構文

void **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 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。

データベース接続エラーが検出された場合は、close メソッドに続けて open メソッドを呼び出し、QAManager に再接続することができます。QAManager に再接続するときに、QAManager を再作成したり、プロパティをリセットしたり、メッセージ・リスナをリセットしたりする必要はありません。最初の open 呼び出し時に QAManager のプロパティを変更することはできません。また、2 度目以降に open を呼び出すときも、同じ受信確認モードを指定する必要があります。

参照

「AcknowledgementMode インタフェース」 524 ページ

「EXPLICIT_ACKNOWLEDGEMENT 変数」 524 ページ

「IMPLICIT_ACKNOWLEDGEMENT 変数」 525 ページ

「close メソッド」 573 ページ

recover メソッド

構文

void **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」 578 ページ

QAManagerBase インタフェース

構文

public **QAManagerBase**

派生クラス

- 「QAManager インタフェース」 562 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」 632 ページ

「QAManager インタフェース」 562 ページ

メンバ

ianywhere.qanywhere.client.QAManagerBase 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。

- 「setStringStoreProperty メソッド」 598 ページ
- 「start メソッド」 598 ページ
- 「stop メソッド」 599 ページ
- 「triggerSendReceive メソッド」 599 ページ

browseMessages メソッド

構文

java.util.Enumeration **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」 571 ページ

「browseMessagesByID メソッド」 570 ページ

「getMessage メソッド」 578 ページ

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 **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) を使用します。

参照

「browseMessagesByQueue メソッド」 571 ページ

「browseMessages メソッド」 570 ページ

「getMessage メソッド」 578 ページ

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 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」 570 ページ

「browseMessages メソッド」 570 ページ

「getMessage メソッド」 578 ページ

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 **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」 571 ページ

「browseMessages メソッド」 570 ページ

「browseMessagesByID メソッド」 570 ページ

「getMessage メソッド」 578 ページ

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 **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 **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。

データベース接続エラーが検出された場合は、close 関数に続けて open 関数を呼び出し、QAManager に再接続することができます。QAManager に再接続するときに、QAManager を再作成したり、プロパティをリセットしたり、メッセージ・リスナをリセットしたりする必要はありません。最初の open 呼び出し時に QAManager のプロパティを変更することはできません。また、2 度目以降に open を呼び出すときも、同じ受信確認モードを指定する必要があります。

参照

- 「open メソッド」 567 ページ

createBinaryMessage メソッド

構文

QABinaryMessage **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」 537 ページ

戻り値

新しい QABinaryMessage インスタンス。

createTextMessage メソッド

構文

QATextMessage **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、string 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」 626 ページ

戻り値

新しい QATextMessage インスタンス。

getBooleanStoreProperty メソッド

構文

boolean **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」 532 ページを参照してください。

参照

「MessageStoreProperties インタフェース」 532 ページ

戻り値

boolean 型のプロパティの値。

getByteStoreProperty メソッド

構文

```
byte 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」 532 ページを参照してください。

参照

「MessageStoreProperties インタフェース」 532 ページ

戻り値

signed byte 型のプロパティの値。

getDoubleStoreProperty メソッド

構文

```
double 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」 532 ページを参照してください。

参照

「MessageStoreProperties インタフェース」 532 ページ

戻り値

double 型のプロパティの値。

getFloatStoreProperty メソッド

構文

```
float 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」 532 ページを参照してください。

参照

「MessageStoreProperties インタフェース」 532 ページ

戻り値

float 型のプロパティの値。

getIntStoreProperty メソッド

構文

```
int 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」 532 ページを参照してください。

参照

「MessageStoreProperties インタフェース」 532 ページ

戻り値

int 型のプロパティの値。

getLongStoreProperty メソッド

構文

long **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」 532 ページを参照してください。

参照

「MessageStoreProperties インタフェース」 532 ページ

戻り値

long 型のプロパティの値。

getMessage メソッド

構文

QAMessage **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。メッセージを同期的に受信する場合は、このメソッドを使用します。

参照

「QAMessage インタフェース」 604 ページ

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 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。

メッセージを同期的に受信する場合は、このメソッドを使用します。

参照

「QAMessage インタフェース」 604 ページ

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 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。

メッセージを同期的に受信する場合は、このメソッドを使用します。

参照

「QAMessage インタフェース」 604 ページ

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。

getMessageBySelectorTimeout メソッド

構文

```
QAMessage 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。

メッセージを同期的に受信する場合は、このメソッドを使用します。

参照

「QAMessage インタフェース」 604 ページ

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。

getMessageNoWait メソッド

構文

```
QAMessage 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。

メッセージを同期的に受信する場合は、このメソッドを使用します。

参照

「QAMessage インタフェース」 604 ページ

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 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。メッセージを同期的に受信する場合は、このメソッドを使用します。

参照

「QAMessage インタフェース」 604 ページ

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 **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」 524 ページ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」 524 ページ

「IMPLICIT_ACKNOWLEDGEMENT 変数」 525 ページ

「QAManager インタフェース」 562 ページ

「QATransactionalManager インタフェース」 632 ページ

戻り値

受信メッセージの QAManager 受信確認モード。

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

```
int QAManagerBase.getQueueDepth(
  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。

スロー

- 「QAException クラス」 553 ページ.

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 メソッドを使用)。コミットされていない出力メッセージも含まれます。

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」 636 ページを参照してください。

参照

「getMessage メソッド」 578 ページ

戻り値

整数 - 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

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

```
int QAManagerBase.getQueueDepth(
  String queue,
  short filter
)
```

パラメータ

- **queue**　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。

スロー

- 「QAException クラス」 553 ページ.

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 メソッドを使用)。コミットされていない出力メッセージも含まれます。

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」 636 ページを参照してください。

戻り値

整数 - 受信されていないメッセージ数。

getShortStoreProperty メソッド

構文

```
short 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」 532 ページを参照してください。

参照

「MessageStoreProperties インタフェース」 532 ページ

戻り値

short 型のプロパティの値。

getStoreProperty メソッド

構文

Object **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」 532 ページを参照してください。

参照

「MessageStoreProperties インタフェース」 532 ページ

戻り値

プロパティの値。

getStorePropertyNames メソッド

構文

java.util.Enumeration **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 **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

- string 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 string 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」 532 ページを参照してください。

戻り値

string 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。

参照

「MessageStoreProperties インタフェース」 532 ページ

propertyExists メソッド

構文

boolean **QAManagerBase.propertyExists(**
  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に値が存在するかどうかをテストします。

このメソッドを使用して、指定されたプロパティ名にメッセージ・ストアによって現在マッピングされている値があるかどうかを判断できます。

参照

「MessageStoreProperties インタフェース」 532 ページ

putMessage メソッド

構文

```
void 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」 588 ページ

putMessageTimeToLive メソッド

構文

```
void 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 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」 532 ページを参照してください。

参照

「MessageStoreProperties インタフェース」 532 ページ

setByteStoreProperty メソッド

構文

```
void 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」 532 ページを参照してください。

参照

「MessageStoreProperties インタフェース」 532 ページ

setDoubleStoreProperty メソッド

構文

```
void 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」 532 ページを参照してください。

参照

「MessageStoreProperties インタフェース」 532 ページ

setFloatStoreProperty メソッド

構文

```
void 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」 532 ページを参照してください。

参照

「MessageStoreProperties インタフェース」 532 ページ

setIntStoreProperty メソッド

構文

```
void 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」 532 ページを参照してください。

参照

「MessageStoreProperties インタフェース」 532 ページ

setLongStoreProperty メソッド

構文

```
void 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」 532 ページを参照してください。

参照

「MessageStoreProperties インタフェース」 532 ページ

setMessageListener メソッド

構文

```
void 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的に受信します。

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。1 つのキューには、1 つの Listener 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的に受信する場合は、このメソッドを使用します。

参照

「QAMessageListener インタフェース」 623 ページ

setMessageListener2 メソッド

構文

```
void 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的に受信します。

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。1 つのキューには、1 つの Listener 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的に受信する場合は、このメソッドを使用します。

参照

「QAMessageListener2 インタフェース」 625 ページ

setMessageListenerBySelector メソッド

構文

```
void 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 Listener オブジェクトがすでに割り当てられてい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

QAMessageListener オブジェクトを登録し、メッセージ・セレクタを使用して QAnywhere メッセージを非同期的に受信します。

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。1 つのキューには、1 つの Listener 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的に受信する場合は、このメソッドを使用します。

setMessageListenerBySelector2 メソッド

構文

```
void 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

QAMessageListener2 オブジェクトを登録し、メッセージ・セレクタを使用して QAnywhere メッセージを非同期的に受信します。

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。1 つのキューには、1 つの Listener 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的に受信する場合は、このメソッドを使用します。

参照

「QAMessageListener2 インタフェース」 625 ページ

setProperty メソッド

QAnywhere Manager の設定プロパティを設定します。

構文

```
void QAManagerBase.setProperty(
  string name,
  string val
)
  throws QAException
```

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」 98 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」 99 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。

スロー

- 「QAException クラス」 553 ページ

参照

- 「QAManagerBase インタフェース」 568 ページ
- 「open メソッド」 567 ページ
- 「open メソッド」 635 ページ

setShortStoreProperty メソッド

構文

```
void 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」 532 ページを参照してください。

参照

「MessageStoreProperties インタフェース」 532 ページ

setStoreProperty メソッド

構文

```
void 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 String 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」 532 ページを参照してください。

参照

「MessageStoreProperties インタフェース」 532 ページ

setStringStoreProperty メソッド

構文

void **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**　String 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 string 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 String 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」 532 ページを参照してください。

参照

「MessageStoreProperties インタフェース」 532 ページ

start メソッド

構文

void **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」 599 ページ

stop メソッド

構文

void **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」 598 ページ

triggerSendReceive メソッド

構文

void **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、メッセージ転送をどのようにして実行するかを決定します。たとえば、クライアントが Push 通知を受信した場合や、ユーザが 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を自動的に実行することができます。

参照

「putMessage メソッド」 588 ページ

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 インタフェース」 562 ページ

「QATransactionalManager インタフェース」 632 ページ

メンバ

ianywhere.qanywhere.client.QAManagerFactory 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 メソッド」 601 ページ
● 「createQAManager メソッド」 601 ページ
● 「createQAManager メソッド」 602 ページ
● 「createQATransactionalManager メソッド」 602 ページ
● 「createQATransactionalManager メソッド」 603 ページ
● 「createQATransactionalManager メソッド」 603 ページ
● 「getInstance メソッド」 604 ページ

createQAManager メソッド

構文

```
abstract QAManager 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を使用して作成されます。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設定できます。

参照

「QAManager インタフェース」 562 ページ

戻り値

新しい QAManager インスタンス。

createQAManager メソッド

構文

```
abstract QAManager 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」 562 ページ

戻り値

新しい QAManager インスタンス。

createQAManager メソッド

構文

```
abstract QAManager 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」 562 ページ

戻り値

新しい QAManager インスタンス。

createQATransactionalManager メソッド

構文

```
abstract QATransactionalManager 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を使用して作成されます。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設定できます。

参照

「QATransactionalManager インタフェース」 632 ページ

戻り値

設定された QATransactionalManager。

createQATransactionalManager メソッド

構文

```
abstract QATransactionalManager 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」 632 ページ

戻り値

設定された QATransactionalManager。

createQATransactionalManager メソッド

構文

```
abstract QATransactionalManager 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」 632 ページ

戻り値

新しい QATransactionalManager。

getInstance メソッド

構文

QAManagerFactory **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 **QAMessage**

派生クラス

- 「QABinaryMessage インタフェース」 537 ページ
- 「QATextMessage インタフェース」 626 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」 525 ページを参照してください。

参照

「QABinaryMessage インタフェース」 537 ページ

「QATextMessage インタフェース」 626 ページ

メンバ

ianywhere.qanywhere.client.QAMessage 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。

DEFAULT_PRIORITY 変数

構文

final int **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 **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 **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 **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 **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」 525 ページ

戻り値

プロパティの値。

getByteProperty メソッド

構文

byte **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」 525 ページ

戻り値

プロパティの値。

getDoubleProperty メソッド

構文

```
double 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」 525 ページ

戻り値

プロパティの値。

getExpiration メソッド

構文

```
java.util.Date 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することで設定されるため、このプロパティは読み込み専用です。

参照

「putMessageTimeToLive メソッド」 588 ページ

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 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」 525 ページ

戻り値

プロパティの値。

getInReplyToID メソッド

構文

```
String 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 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」 525 ページ

戻り値

プロパティの値。

getLongProperty メソッド

構文

```
long 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」 525 ページ

戻り値

プロパティの値。

getMessageID メソッド

構文

String **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な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」 588 ページ

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 **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 **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 **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 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」 535 ページ

戻り値

プロパティ型。

getRedelivered メソッド

構文

```
boolean 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。アプリケーションが次の起動時に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 **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 **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」 525 ページ

戻り値

プロパティの値。

getStringProperty メソッド

構文

```
String 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

String 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」 525 ページ

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 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 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。

setBooleanProperty メソッド

構文

```
void 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」 525 ページ

setByteProperty メソッド

構文

```
void 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」 525 ページ

setDoubleProperty メソッド

構文

```
void 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」 525 ページ

setFloatProperty メソッド

構文

```
void 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」 525 ページ

setInReplyToID メソッド

構文

```
void 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 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」 525 ページ

setLongProperty メソッド

構文

```
void 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」 525 ページ

setPriority メソッド

構文

```
void 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 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 String 型でなければなりません。

参照

「MessageProperties インタフェース」 525 ページ

setReplyToAddress メソッド

構文

```
void 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 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」 525 ページ

setStringProperty メソッド

構文

```
void 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

string 型のプロパティを設定します。

参照

「MessageProperties インタフェース」 525 ページ

QAMessageListener インタフェース

構文

public **QAMessageListener**

備考

メッセージを受信するには、QAManagerBase.setMessageListener(String,QAMessageListener) を呼び出すことで、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」 593 ページ

メンバ

ianywhere.qanywhere.client.QAMessageListener 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。

- 「onException メソッド」 624 ページ
- 「onMessage メソッド」 624 ページ

onException メソッド

構文

```
void 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」 568 ページ

「close メソッド」 573 ページ

onMessage メソッド

構文

```
void 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 **QAMessageListener2**

備考

メッセージを受信するには、QAManagerBase を呼び出すことで、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」 593 ページ

メンバ

ianywhere.qanywhere.client.QAMessageListener2 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。

- 「onException メソッド」 625 ページ
- 「onMessage メソッド」 626 ページ

onException メソッド

構文

void **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」 568 ページ

「close メソッド」 573 ページ

「onMessage メソッド」 624 ページ

onMessage メソッド

構文

```
void 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」 568 ページ

QATextMessage インタフェース

構文

public **QATextMessage**

基本クラス

- 「QAMessage インタフェース」 604 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た時点でプロバイダは 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

「onMessage メソッド」 624 ページ

メンバ

ianywhere.qanywhere.client.QATextMessage 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。

getText メソッド

構文

String **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」 629 ページ

戻り値

メッセージ・テキストまたは NULL。

getTextLength メソッド

構文

long **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 **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 **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 **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 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 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 **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 **QATransactionalManager**

基本クラス

- 「QAManagerBase インタフェース」 568 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」 568 ページ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」 635 ページ

「putMessage メソッド」 588 ページ

「getMessage メソッド」 578 ページ

メンバ

ianywhere.qanywhere.client.QATransactionalManager 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。

- 「setStoreProperty メソッド」 597 ページ
- 「setStringStoreProperty メソッド」 598 ページ
- 「start メソッド」 598 ページ
- 「stop メソッド」 599 ページ
- 「triggerSendReceive メソッド」 599 ページ

commit メソッド

構文

void **QATransactionalManager.commit()**
throws **QAException**

スロー

- 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。

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 **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 **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 **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」 584 ページ

「getQueueDepth メソッド」 583 ページ

メンバ

ianywhere.qanywhere.client.QueueDepthFilter 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」 636 ページ
- 「INCOMING 変数」 637 ページ
- 「LOCAL 変数」 637 ページ
- 「OUTGOING 変数」 637 ページ

ALL 変数

構文

final short **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 **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。

LOCAL 変数

構文

final short **QueueDepthFilter.LOCAL**

備考

LOCAL フィルタを指定して getQueueDepth が呼び出され、キューが指定されている場合、この変数はそのキューに送信される未受信のローカル・メッセージの数を返します。キューが指定されていない場合、LOCAL はメッセージ・ストア内の未受信のローカル・メッセージの合計数 (システム・メッセージを除く) を返します。

OUTGOING 変数

構文

final short **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 **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 (継承されたメンバも含みます) を以下に示します。

CANCELLED 変数

構文

final int **StatusCodes.CANCELLED**

備考

メッセージはキャンセルされました。

このコードは MessageProperties.STATUS で使用されます。

参照

「STATUS 変数」 531 ページ

EXPIRED 変数

構文

final 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。

このコードは MessageProperties.STATUS で使用されます。

参照

「STATUS 変数」 531 ページ

FINAL 変数

構文

final int **StatusCodes.FINAL**

備考

この定数は、メッセージが最終ステータスになったかどうかを示します。

メッセージは、そのステータスがこの定数を超えている場合にのみ最終ステータスになったとみなされます。

このコードは MessageProperties.STATUS で使用されます。

参照

「STATUS 変数」 531 ページ

LOCAL 変数

構文

final 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。

このコードは MessageProperties.TRANSMISSION_STATUS で使用されます。

参照

「TRANSMISSION_STATUS 変数」 532 ページ

PENDING 変数

構文

final int **StatusCodes.PENDING**

備考

メッセージは送信されましたが、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」 531 ページ

RECEIVED 変数

構文

final 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。

このコードは MessageProperties.STATUS で使用されます。

参照

「STATUS 変数」 531 ページ

RECEIVING 変数

構文

final int **StatusCodes.RECEIVING**

備考

メッセージは受信中であるか、または受信されまし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」 531 ページ

TRANSMITTED 変数

構文

final int **StatusCodes.TRANSMITTED**

備考

メッセージはサーバに送信されました。

このコードは MessageProperties.TRANSMISSION_STATUS で使用されます。

参照

「TRANSMISSION_STATUS 変数」 532 ページ

TRANSMITTING 変数

構文

final int **StatusCodes.TRANSMITTING**

備考

メッセージはサーバに送信中です。

このコードは MessageProperties.TRANSMISSION_STATUS で使用されます。

参照

「TRANSMISSION_STATUS 変数」 532 ページ

UNRECEIVABLE 変数

構文

final 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。

このコードは MessageProperties.STATUS で使用されます。

参照

「STATUS 変数」 531 ページ

UNTRANSMITTED 変数

構文

final int **StatusCodes.UNTRANSMITTED**

備考

メッセージはサーバに送信されていません。

このコードは MessageProperties.TRANSMISSION_STATUS で使用されます。

参照

「TRANSMISSION_STATUS 変数」 532 ページ

Web サービス用 QAnywhere Java API

パッケージ

ianywhere.qanywhere.ws

WSBase クラス

構文

public **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 (継承されたメンバも含みます) を以下に示します。

- 「clearRequestProperties メソッド」 643 ページ
- 「getResult メソッド」 643 ページ
- 「getServiceID メソッド」 644 ページ
- 「setListener メソッド」 644 ページ
- 「setListener メソッド」 645 ページ
- 「setProperty メソッド」 645 ページ
- 「setQAManager メソッド」 646 ページ
- 「setRequestProperty メソッド」 646 ページ
- 「setServiceID メソッド」 647 ページ
- 「WSBase メソッド」 643 ページ
- 「WSBase メソッド」 642 ページ

WSBase メソッド

構文

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。

WSBase メソッド

構文

```
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 ～ 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。

clearRequestProperties メソッド

構文

```
void 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 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」 674 ページ

getServiceID メソッド

構文

String **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 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 Listener 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、listener に NULL を指定して 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 で設定されたリスナを置き換えます。

setListener メソッド

構文

```
void 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 Listener 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、listener に NULL を指定して setListener を呼び出します。

このメソッドは、以前に呼び出された setListener で設定されたリスナを置き換えます。

setProperty メソッド

構文

```
void 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 ～ 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。

setQAManager メソッド

構文

```
void 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。

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 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 のインスタンスが作成した 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 **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。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 **WSException**

派生クラス

- 「WSFaultException クラス」 649 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 (継承されたメンバも含みます) を以下に示します。

- 「getErrorCode メソッド」 649 ページ
- 「WSException メソッド」 648 ページ
- 「WSException メソッド」 648 ページ
- 「WSException メソッド」 648 ページ

WSException メソッド

構文

```
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

```
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

```
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 **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 **WSFaultException**

基本クラス

- 「WSException クラス」 647 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 (継承されたメンバも含みます) を以下に示します。

- 「getErrorCode メソッド」 649 ページ
- 「WSException メソッド」 648 ページ
- 「WSException メソッド」 648 ページ
- 「WSException メソッド」 648 ページ
- 「WSFaultException メソッド」 649 ページ

WSFaultException メソッド

構文

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 **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 (継承されたメンバも含みます) を以下に示します。

- 「onException メソッド」 650 ページ
- 「onResult メソッド」 651 ページ

onException メソッド

構文

```
void 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」 647 ページ

「WSResult クラス」 651 ページ

onResult メソッド

構文

void **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」 651 ページ

WSResult クラス

構文

public **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 (継承されたメンバも含みます) を以下に示します。

- 「getStringArrayValue メソッド」 673 ページ
- 「getStringValue メソッド」 673 ページ
- 「getValue メソッド」 674 ページ

acknowledge メソッド

構文

void **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[] **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[] **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 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[] 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 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[] 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 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[] 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 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[] 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 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[] 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 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 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[] 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 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[] 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 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[] 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 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[] 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 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[] 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 **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[] **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 **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[] **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 **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[] **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 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[] 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 **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[] **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 int 配列値を取得します。

戻り値

値。

getPrimitiveIntValue メソッド

構文

```
int 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 int 値を取得します。

戻り値

値。

getPrimitiveLongArrayValue メソッド

構文

```
long[] 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 **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[] **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 **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 **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[] **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 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 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」 674 ページ

getStringArrayValue メソッド

構文

```
String[] 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 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 **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 **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 (継承されたメンバも含みます) を以下に示します。

- 「STATUS_ERROR 変数」 674 ページ
- 「STATUS_QUEUED 変数」 675 ページ
- 「STATUS_RESULT_AVAILABLE 変数」 675 ページ
- 「STATUS_SUCCESS 変数」 675 ページ

STATUS_ERROR 変数

構文

final int **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 **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 **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 **WSStatus.STATUS_SUCCESS**

備考

要求は正常に処理されました。

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」 720 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」 67 ページ
- 「QAnywhere メッセージ・アドレス」 69 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ

例

次の例では、メッセージを受信して、そのアドレスをデータベース・サーバ・メッセージ・ウィンドウに出力します。

```
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」 67 ページ
- 「メッセージの削除ルール」 823 ページ
- 「QAnywhere メッセージの送信」 73 ページ
- 「ml_qa_setexpiration」 685 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ

例

次の例では、メッセージを受信して、その有効期限をデータベース・サーバ・メッセージ・ウィンドウに出力します。

```
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」 67 ページ
- 「ml_qa_setinreplytoid」 685 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ

例

次の例では、メッセージを受信して、その in-reply-to ID をデータベース・サーバ・メッセージ・ウィンドウに出力します。

```
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 ～ 4 を通常のメッセージ、優先度 5 ～ 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」 67 ページ
- 「ml_qa_setpriority」 686 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ

例

次の例では、メッセージを受信して、その優先度をデータベース・サーバ・メッセージ・ウィンドウに出力します。

```
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」 67 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ

例

次の例では、メッセージを受信し、そのメッセージが以前に配信されたが受信されていなかった場合に、メッセージ ID をデータベース・サーバ・メッセージ・ウィンドウに出力します。

```
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」 67 ページ
- 「ml_qa_setreplytoaddress」 687 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 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」 67 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ

例

次の例では、メッセージを受信して、その作成時刻をデータベース・サーバ・メッセージ・ウィンドウに出力します。

```
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。
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」 67 ページ
- 「ml_qa_getexpiration」 679 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 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。

項目	説明	備考
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」 67 ページ
- 「ml_qa_getinreplytoid」 679 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 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 ～ 4 を通常のメッセージ、優先度 5 ～ 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」 67 ページ
- 「ml_qa_getpriority」 680 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 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」 67 ページ
- 「ml_qa_getreplytoaddress」 683 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 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」 723 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」 67 ページ
- 「ml_qa_setbooleanproperty」 697 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ
- 「カスタムのメッセージ・プロパティ」 725 ページ

例

次の例では、メッセージを受信して、boolean 型プロパティ mybooleanproperty の値をデータベース・サーバ・メッセージ・ウィンドウに出力します。

```
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」 67 ページ
- 「ml_qa_setbyteproperty」 697 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ
- 「カスタムのメッセージ・プロパティ」 725 ページ

例

次の例では、メッセージを受信して、byte 型プロパティ mybyteproperty の値をデータベース・サーバ・メッセージ・ウィンドウに出力します。

```
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」 67 ページ
- 「ml_qa_setdoubleproperty」 698 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ
- 「カスタムのメッセージ・プロパティ」 725 ページ

例

次の例では、メッセージを受信して、double 型プロパティ mydoubleproperty の値をデータベース・サーバ・メッセージ・ウィンドウに出力します。

```
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」 67 ページ
- 「ml_qa_setfloatproperty」 699 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ
- 「カスタムのメッセージ・プロパティ」 725 ページ

例

次の例では、メッセージを受信して、float 型プロパティ myfloatproperty の値をデータベース・サーバ・メッセージ・ウィンドウに出力します。

```
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」 67 ページ
- 「ml_qa_setintproperty」 700 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ
- 「カスタムのメッセージ・プロパティ」 725 ページ

例

次の例では、メッセージを受信して、integer 型プロパティ myintproperty の値をデータベース・サーバ・メッセージ・ウィンドウに出力します。

```
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」 67 ページ
- 「ml_qa_setlongproperty」 701 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ
- 「カスタムのメッセージ・プロパティ」 725 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」 67 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ
- 「カスタムのメッセージ・プロパティ」 725 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」 67 ページ
- 「ml_qa_setshortproperty」 702 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ
- 「カスタムのメッセージ・プロパティ」 725 ページ

例

次の例では、メッセージを受信して、short 型プロパティ myshortproperty の値をデータベース・サーバ・メッセージ・ウィンドウに出力します。

```
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」 67 ページ
- 「ml_qa_setstringproperty」 702 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ
- 「カスタムのメッセージ・プロパティ」 725 ページ

例

次の例では、メッセージを受信して、string 型プロパティ mystringproperty の値をデータベース・サーバ・メッセージ・ウィンドウに出力します。

```
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」 67 ページ
- 「ml_qa_getbooleanproperty」 688 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ
- 「カスタムのメッセージ・プロパティ」 725 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」 67 ページ
- 「ml_qa_getbyteproperty」 689 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ
- 「カスタムのメッセージ・プロパティ」 725 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)

項目	説明	備考
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」 67 ページ
- 「ml_qa_getdoubleproperty」 690 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ
- 「カスタムのメッセージ・プロパティ」 725 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」 67 ページ
- 「ml_qa_getfloatproperty」 691 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ
- 「カスタムのメッセージ・プロパティ」 725 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」 67 ページ
- 「ml_qa_getintproperty」 692 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ
- 「カスタムのメッセージ・プロパティ」 725 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」 67 ページ
- 「ml_qa_getlongproperty」 693 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ
- 「カスタムのメッセージ・プロパティ」 725 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」 67 ページ
- 「ml_qa_getshortproperty」 695 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ
- 「カスタムのメッセージ・プロパティ」 725 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。
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」 67 ページ
- 「ml_qa_getstringproperty」 696 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ
- 「カスタムのメッセージ・プロパティ」 725 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」 67 ページ
- 「ml_qa_setbinarycontent」 706 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ
- 「ml_qa_getcontentclass」 704 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・サーバ・メッセージ・ウィンドウに出力します。

```
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」 67 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ
- 「ml_qa_getbinarycontent」 703 ページ
- 「ml_qa_gettextcontent」 705 ページ

例

次の例では、メッセージを受信して、その内容をデータベース・サーバ・メッセージ・ウィンドウに出力します。

```
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」 67 ページ
- 「ml_qa_settextcontent」 707 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 ページ
- 「ml_qa_getcontentclass」 704 ページ

例

次の例では、メッセージの内容をデータベース・サーバ・メッセージ・ウィンドウに出力します。

```
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」 67 ページ
- 「ml_qa_getbinarycontent」 703 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 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」 67 ページ
- 「ml_qa_gettextcontent」 705 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 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」 28 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」 67 ページ
- 「ml_qa_setstoreproperty」 709 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・サーバ・メッセージ・ウィンドウに出力します。

```
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」 67 ページ
- 「ml_qa_getstoreproperty」 709 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」 67 ページ
- 「メッセージ・ヘッダ」 678 ページ
- 「メッセージ・プロパティ」 688 ページ
- 「メッセージの内容」 703 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 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」 713 ページまたは「ml_qa_getmessagetimeout」 714 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」 67 ページ
- 「ml_qa_getmessagenowait」 713 ページ
- 「ml_qa_getmessagetimeout」 714 ページ
- 「メッセージ・ヘッダ」 678 ページ
- 「メッセージ・プロパティ」 688 ページ
- 「メッセージの内容」 703 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 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」 711 ページと「ml_qa_getmessagetimeout」 714 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」 67 ページ
- 「QAnywhere メッセージ・アドレス」 69 ページ
- 「Listener」 『Mobile Link - サーバ起動同期』
- 「ml_qa_getmessagetimeout」 714 ページ
- 「メッセージ・ヘッダ」 678 ページ
- 「メッセージ・プロパティ」 688 ページ
- 「メッセージの内容」 703 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 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」 67 ページ
- 「ml_qa_getmessage」 711 ページ
- 「ml_qa_getmessagenowait」 713 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・サーバ・メッセージ・ウィンドウに出力します。メッセージが受信されなかった場合は、データベース・サーバ・メッセージ・ウィンドウを 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」 67 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的に受信するために、**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」 69 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」 67 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」 711 ページ
- 「ml_qa_getmessage」 711 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」 67 ページ
- 「ml_qa_createmessage」 711 ページ
- 「ml_qa_getmessage」 711 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」 67 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
```

メッセージ・ヘッダとメッセージ・プロパティ

目次

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」 328 ページ
  - C++ API：「getMessageID 関数」 491 ページと「setMessageID 関数」 502 ページ
  - Java API：「getMessageID メソッド」 612 ページ
  - SQL API：「ml_qa_createmessage」 711 ページと「ml_qa_getmessage」 711 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」 330 ページ
  - C++ API：「getTimestamp 関数」 495 ページと「setTimestamp 関数」 504 ページ
  - Java API：「getTimestamp メソッド」 616 ページ
  - SQL API：「ml_qa_gettimestamp」 684 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」 330 ページ
  - C++ API：「getReplyToAddress 関数」 493 ページと「setReplyToAddress 関数」 503 ページ
  - Java API：「getReplyToAddress メソッド」 615 ページと「setReplyToAddress メソッド」 622 ページ
  - SQL API：「ml_qa_getreplytoaddress」 683 ページと「ml_qa_setreplytoaddress」 687 ページ

- **メッセージ・アドレス**　読み込み専用。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」 327 ページ
- ○ C++ API：「getAddress 関数」 486 ページ
- ○ Java API：「getAddress メソッド」 607 ページ
- ○ SQL API：「ml_qa_getaddress」 678 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」 329 ページ
- ○ C++ API：「getRedelivered 関数」 493 ページと「setRedelivered 関数」 502 ページ
- ○ Java API：「getRedelivered メソッド」 614 ページ
- ○ SQL API：「ml_qa_getredelivered」 681 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」 327 ページ
  - C++ API：「getExpiration 関数」 488 ページ
  - Java API：「getExpiration メソッド」 609 ページ
  - SQL API：「ml_qa_getexpiration」 679 ページと「ml_qa_setexpiration」 685 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 ～ 4 を通常のメッセージ、優先度 5 ～ 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」 329 ページ
  - C++ API：「getPriority 関数」 492 ページ
  - Java API：「getPriority メソッド」 613 ページ
  - SQL API：「ml_qa_getpriority」 680 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」 328 ページ
  - C++ API：「getInReplyToID 関数」 490 ページ
  - Java API：「getInReplyToID メソッド」 610 ページ
  - SQL API：「ml_qa_getinreplytoid」 679 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」 816 ページを参照してください。

参照

- .NET API：「QAMessage のメンバ」 325 ページ
- C++ API：「QAMessage クラス」 482 ページ
- Java API：「QAMessage インタフェース」 604 ページ
- SQL API：「メッセージ・ヘッダ」 678 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」 726 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、各文字列はパイプ記号で区切られます。
- **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ります。

値	メッセージ・タイプ	説明
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・キューに送信されます。「Push 通知の通知」 71 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・キューに送信されます。「ネットワーク・ステータス通知」 70 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。各文字列はパイプ記号で区切られます。
- **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ましたが、まだ受信されていません。
10	受信中 - メッセージは受信中であるか、または受信されましたがまだ確認されていません。
20	最終 - メッセージは最終ステータスになりました。
30	失効 - 有効期限が切れる前にメッセージが受信されませんでした。
40	キャンセル済み - メッセージはキャンセルされました。
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ませんでした。
60	受信済み - メッセージは受信され、確認されました。

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 列挙体」 353 ページ
- C++ API：「StatusCodes クラス」 518 ページ
- Java API：「StatusCodes インタフェース」 637 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++	string
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」 230 ページ
- C++ API：「MessageProperties クラス」 408 ページ
- Java API：「MessageProperties インタフェース」 525 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」 324 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」 482 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」 604 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」 688 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
```

サーバ管理要求のリファレンス

目次

サーバ管理要求の親タグ

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、値は論理 AND 結合されます。

<condition> タグのサブタグ	**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。
<archived>	アーカイブ・メッセージ・ストアにあるメッセージの詳細情報を返します。
<customRule>	ルールに基づいて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。
<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>	指定され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。

CustomRule タグ

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」 812 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 以上であるという探索条件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>
```

Schedule タグ

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します。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します。<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>
<everyhour>	後続のレポートとの間隔を時間単位で定義します。<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、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>

<schedule> タグのサブタグ	説明
<everyminute>	後続のレポートとの間隔を分単位で定義します。<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します。<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 つ指定します。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 つ指定します。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>
```

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、選択条件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>	要求の選択条件を満たすメッセージの数。

<MessageDetailsReport> タグのサブタグ	説明
<requestId>	レポートを生成した要求の ID。
<statusDescription>	レポートが生成された理由の簡単な説明。
<UTCDateline>	レポートが生成された時刻と日付。

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 などです。
<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です。
<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」 720 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 ～ 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 は最も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」 723 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」 723 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ません。 ● 1 - メッセージは、目的の受信側メッセージ・ストアに転送済みです。 ● 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、転送は不要です。 ● 3 - メッセージは目的の受信先に転送されましたが、まだ受信確認されていません。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、 (msgId=ID:144... OR msgId=ID225...) AND (status=pending) AND (kind=textmessage) AND (値が 'myVal' のプロパティ 'myProp' を含む) という探索条件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、1 つの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>
```


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 (actionsResponseId?,(CloseConnector|OpenConnector|RestartRules|SetProperty
 |ClientStatusRequest|MessageDetailsRequest|CancelMessageRequest
 |GetClientList)+)>

<!ELEMENT actionsResponseId(requestId)>

<!-- 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?, archived?
 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|archived|
 customRule|property)+)>

<!ELEMENT archived (#PCDATA)>
<!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)>
```


QAnywhere Agent ユーティリティのリファレンス

目次

qaagent ユーティリティ

QAnywhere Agent (qaagent) を使用すると、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。このユーティリティは、クライアント・メッセージ・ストアが SQL Anywhere データベースである場合のみ使用してください。

dbmlsync など、メッセージ・ストアの同期に使用される外部ユーティリティは、QAnywhere ではサポートされていません。

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みます。「@data オプション」 746 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します。「-c オプション」 747 ページを参照してください。
-fd *seconds*	プライマリ・サーバへの再試行を実行する間の遅延時間を指定します。「-fd オプション」 748 ページを参照してください。
-fr *number-of-retries*	接続がエラーになってから、プライマリ・サーバへの接続を再試行する回数を指定します。「-fr オプション」 749 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 を指定します。「-id オプション」 750 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します。「-idl オプション」 751 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します。「-iu オプション」 751 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します。デフォルトは 5001 です。「-lp オプション」 752 ページを参照してください。
-mn *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」 753 ページを参照してください。

オプション	説明
-mp *password*	Mobile Link ユーザのパスワードを指定します。「-mp オプション」 753 ページを参照してください。
-mu *username*	Mobile Link ユーザを指定します。「-mu オプション」 754 ページを参照してください。
-o *logfile*	メッセージのログを出力するファイルを指定します。「-o オプション」 754 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します。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 の付いた名前に変更され、新しいファイルが作成されます。「-on オプション」 755 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。「-os オプション」 756 ページを参照してください。
-ot *logfile*	メッセージのログを出力するファイルを指定します。「-ot オプション」 756 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します。「-pc オプション」 757 ページを参照してください。
-policy *policy-type*	QAnywhere Agent によって使用される転送ポリシーを指定します。「-policy オプション」 758 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します。デフォルトは有効です。「-push オプション」 759 ページを参照してください。
-q	QAnywhere Agent をクワイエット・モードで起動し、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」 761 ページを参照してください。
-qi	QAnywhere Agent をクワイエット・モードで起動し、ウィンドウを完全に非表示にします。「-qi オプション」 761 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します。「-si オプション」 762 ページを参照してください。

オプション	説明
-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します。dbunload/reload は実行しません。「-su オプション」 763 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します。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」 764 ページを参照してください。
-sv	SQL Anywhere ネットワーク・データベース・サーバをデータベース・サーバとして使用します。「-sv オプション」 764 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します。「-v オプション」 765 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します。「-x オプション」 766 ページを参照してください。
-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します。「-xd オプション」 766 ページを参照してください。

参照

- 「QAnywhere Agent の起動」 51 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、ファイル難読化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。「ファイル難読化ユーティリティ (dbfhide)」 『SQL Anywhere サーバ - データベース管理』を参照してください。

Windows Mobile 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。

Sybase Central での同等機能

Sybase Central の QAnywhere プラグインには、**[エージェント・コマンド・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]** ウィンドウが表示され、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。

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*** データベース・ファイルではなくデータベース名を指定して、すでに実行されているクライアント・メッセージ・ストアに接続します。「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*** すでに実行されているデータベース・サーバの名前を指定します。デフォルト値はデータベース名です。「ServerName 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。このパラメータは、デフォルトの UID または PWD 接続パラメータを変更す

る場合に必要となります。「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。このパラメータは、デフォルトの UID または PWD 接続パラメータを変更する場合に必要となります。「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*** データベースにアクセスするための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。開始行を指定しない場合、Windows Mobile の場合のデフォルトは start=dbsrv11 -m -gn 5、他の Windows プラットフォームの場合のデフォルトは start=dbsrv11 -m です。-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』
- 「SQL Anywhere データベース接続」『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」 749 ページ
- 「-x オプション」 766 ページ
- 「フェールオーバ・メカニズムの設定」 41 ページ

-fr オプション

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 がプライマリ Mobile Link サーバへの接続を再試行しないことを意味します。

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」 748 ページ
- 「-x オプション」 766 ページ
- 「フェールオーバ・メカニズムの設定」 41 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、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)

この他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」 25 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・サイズが指定されていないことを意味します。

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。

キューに登録された配信対象メッセージのうちでダウンロード・スレッショルドを超えたものは、複数の小さいメッセージ・パーツに分割されます。各メッセージ・パーツは別個にダウンロードできるため、何度か同期を行いながらメッセージを段階的にダウンロードできます。送信先ですべてのメッセージ・パーツが受信されると、完全なメッセージが受信されたことになります。

インクリメンタル・ダウンロードのサイズは概算値です。実際のダウンロード・サイズは、メッセージのサイズ以外にも多くの要素によって異なります。

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」 792 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。

アップロード・スレッショルドを超えたメッセージは、複数の小さいメッセージ・パーツに分割されます。各メッセージ・パーツは別個にアップロードできるため、何度か同期を行いながらメッセージを段階的にアップロードできます。送信先ですべてのメッセージ・パーツが受信されると、完全なメッセージが受信されたことになります。

インクリメンタル・アップロードのサイズは概算値です。実際のアップロード・サイズは、メッセージのサイズ以外にも多くの要素によって異なります。

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」 792 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 ポートは、QAnywhere Agent が制御コマンドを Listener に送信する際にも使用されます。

参照

- 「Push 通知によるメッセージングのシナリオ」 7 ページ
- 「-push オプション」 759 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」 753 ページ
- 「-mu オプション」 754 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」 754 ページ

-mu オプション

Mobile Link ユーザ名を指定します。

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 ユーザ名は、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」 33 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」 750 ページ
- 「-mp オプション」 753 ページ
- 「リモート ID」『Mobile Link - クライアント管理』

-o オプション

指定されたログ・ファイルに出力を送信します。

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ログを出力します。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、ログを出力します。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、ログを出力します。

たとえば、*c:¥tmp¥mylog.out* というログ・ファイルを指定した場合、qaagent は *c:¥tmp¥mylog.out* に、dbmlsync は *c:¥tmp¥mylog_sync.out* に、dblsn は *c:¥tmp¥mylog_lsn.out* に、それぞれログを出力します。

参照

- 「-ot オプション」 756 ページ
- 「-on オプション」 755 ページ
- 「-os オプション」 756 ページ
- 「-v オプション」 765 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します。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* の付いた名前に変更され、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、メッセージ・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* の付いた名前に変更し、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」 754 ページ
- 「-ot オプション」 756 ページ
- 「-os オプション」 756 ページ
- 「-v オプション」 765 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、メッセージのログを出力す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 10KB です。

QAnywhere Agent は、現在のファイル・サイズを確認してから、ファイルにメッセージのログを出力します。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。ここで、*xx* は 00 ～ 99 の連続した数字で、*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、-on 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」 754 ページ
- 「-ot オプション」 756 ページ
- 「-on オプション」 755 ページ
- 「-v オプション」 765 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ログを出力します。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、ログを出力します。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、ログを出力します。

たとえば、*c:¥tmp¥mylog.out* というログ・ファイルを指定した場合、qaagent は *c:¥tmp¥mylog.out* に、dbmlsync は *c:¥tmp¥mylog_sync.out* に、dblsn は *c:¥tmp¥mylog_lsn.out* に、それぞれログを出力します。

参照

- 「-o オプション」 754 ページ
- 「-on オプション」 755 ページ
- 「-os オプション」 756 ページ
- 「-v オプション」 765 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」 759 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。

  on demand 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」 70 ページを参照してください。

- **scheduled** スケジュール設定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を受信した。

  ○ 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 列挙体」 226 ページ
  - C++：「AcknowledgementMode クラス」 406 ページ
  - Java：「AcknowledgementMode インタフェース」 524 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・ステータス変更通知を受信した。

○ 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」 317 ページ
○ C++：「triggerSendReceive 関数」 477 ページ
○ Java：「triggerSendReceive メソッド」 599 ページ
○ SQL：「ml_qa_triggersendreceive」 718 ページ

● **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」 7 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」 819 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」 54 ページ
● 「Push 通知によるメッセージングのシナリオ」 7 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。

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」 5 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」 37 ページ
- 「-pc オプション」 757 ページ
- 「QAnywhere Agent の起動」 51 ページ
- 「Push 通知の通知」 71 ページ
- 「-lp オプション」 752 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」 761 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 Mobile 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 Mobile 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 Mobile 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・モードで動作している QAnywhere Agent は、**qastop** と入力することによってのみ停止できます。

参照

- 「qastop ユーティリティ」 789 ページ
- 「-q オプション」 761 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」 25 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」 140 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。

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。

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。

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」 764 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」 763 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 11 にアップグレードします。

```
qaagent -q -sur -c "UID=dba;PWD=sql;DBF=qanywhere.db"
```

-sv オプション

パーソナル・データベース・サーバの代わりに SQL Anywhere ネットワーク・データベース・サーバをデータベース・サーバとして使用するように、エージェントに通知します。

構文

qaagent -sv -c "*connection-string*" ...

備考

デフォルトでは、qaagent は dbeng11 アプリケーションを使用してパーソナル・データベース・サーバに接続します。-sv を指定すると、qaagent は dbsrv11 アプリケーションを使用してネットワーク・データベース・サーバに接続します。

参照

- 「SQL Anywhere データベース・サーバ」 『SQL Anywhere サーバ - データベース管理』

-v オプション

QAnywhere Agent のメッセージ・ログ・ファイルにログを出力し、QAnywhere Agent のメッセージ・ウィンドウに表示する情報を指定できます。

構文

qaagent -v *levels ...*

デフォルト

最低の冗長レベル

備考

-v オプションはメッセージ・ログ・ファイルとメッセージ・ウィンドウに影響します。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・ファイルだけが作成されます。

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。

-v を単独で指定すると、最小限の情報のログが取られます。

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。「Windows デバイス用の Listener ユーティリティ」 『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。dbmlsync に関する「-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」 754 ページ
- 「-ot オプション」 756 ページ
- 「-on オプション」 755 ページ
- 「-os オプション」 756 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・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、必須となります。

-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・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」 142 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」 41 ページ
- 「-fd オプション」 748 ページ
- 「-fr オプション」 749 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・プロパティを初期化し、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確立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」 28 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" );
```


qauagent ユーティリティ

QAnywhere Ultra Light Agent (qauagent) を使用すると、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。このユーティリティは、クライアント・メッセージ・ストアが Ultra Light データベースである場合のみ使用してください。

注意
dbmlsync ユーティリティは、SQL Anywhere メッセージ・ストアを同期するように設計されていますが、Ultra Light メッセージ・ストアをサポートしていません。

構文

qauagent [*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みます。「@data オプション」 770 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します。「-c オプション」 770 ページを参照してください。
-fd *seconds*	プライマリ・サーバへの再試行を実行する間の遅延時間を指定します。「-fd オプション」 772 ページを参照してください。
-fr *number-of-retries*	接続がエラーになってから、プライマリ・サーバへの接続を再試行する回数を指定します。「-fr オプション」 772 ページを参照してください。
-id *id*	QAnywhere Ultra Light 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。「-id オプション」 773 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します。「-idl オプション」 774 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します。「-iu オプション」 775 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します。デフォルトは 5001 です。「-lp オプション」 776 ページを参照してください。
-mn *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」 776 ページを参照してください。

オプション	説明
-mp *password*	Mobile Link ユーザのパスワードを指定します。「-mp オプション」 776 ページを参照してください。
-mu *username*	Mobile Link ユーザを指定します。「-mp オプション」 776 ページを参照してください。
-o *logfile*	メッセージのログを出力するファイルを指定します。「-o オプション」 777 ページを参照してください。
-on *size*	QAnywhere Ultra Light 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、現在のファイルが拡張子 .old の付いた名前に変更され、新しいファイルが作成されます。「-on オプション」 778 ページを参照してください。
-os *size*	QAnywhere Ultra Light 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。「-os オプション」 779 ページを参照してください。
-ot *logfile*	メッセージのログを出力するファイルを指定します。「-ot オプション」 779 ページを参照してください。
-policy *policy-type*	QAnywhere Ultra Light Agent によって使用される転送ポリシーを指定します。「-policy オプション」 780 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します。デフォルトは有効です。「-push オプション」 782 ページを参照してください。
-q	QAnywhere Ultra Light 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。「-q オプション」 783 ページを参照してください。
-qi	QAnywhere Ultra Light 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。「-qi オプション」 783 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します。「-si オプション」 784 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します。「-v オプション」 785 ページを参照してください。

オプション	説明
-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します。「-x オプション」 786 ページを参照してください。
-xd	QAnywhere Ultra Light 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。「-xd オプション」 787 ページを参照してください。

参照

- 「QAnywhere Agent の起動」 51 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、ファイル難読化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。「ファイル難読化ユーティリティ (dbfhide)」 『SQL Anywhere サーバ - データベース管理』を参照してください。

Windows Mobile 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。

Sybase Central での同等機能

Sybase Central の QAnywhere プラグインには、**[エージェント・コマンド・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]** ウィンドウが表示され、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文字列を指定します。

構文

qauagent -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 qauagent で使用されません。

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。「Ultra Light DBF 接続パラメータ」『Ultra Light データベース管理とリファレンス』を参照してください。
- **dbn=*database-name*** データベース・ファイルではなくデータベース名を指定して、すでに実行されているクライアント・メッセージ・ストアに接続します。「Ultra Light DBN 接続パラメータ」『Ultra Light データベース管理とリファレンス』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。このパラメータは、デフォルトの UID または PWD 接続パラメータを変更する場合に必要となります。「Ultra Light UID 接続パラメータ」『Ultra Light データベース管理とリファレンス』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。このパラメータは、デフォルトの UID または PWD 接続パラメータを変更する場合に必要となります。「Ultra Light PWD 接続パラメータ」『Ultra Light データベース管理とリファレンス』を参照してください。
- **dbkey=*key*** データベースにアクセスするための暗号化キーを指定します。「Ultra Light DBKEY 接続パラメータ」『Ultra Light データベース管理とリファレンス』を参照してください。

参照

- 「接続パラメータ」『SQL Anywhere サーバ - データベース管理』
- 「SQL Anywhere データベース接続」『SQL Anywhere サーバ - データベース管理』

例

```
qauagent -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

qauagent -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、qauagent -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 Ultra Light 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 Ultra Light 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、qauagent -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」 772 ページ
- 「-x オプション」 786 ページ
- 「フェールオーバ・メカニズムの設定」 41 ページ

-fr オプション

QAnywhere Ultra Light 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

qauagent -fr *number-of-retries* ...

デフォルト

0 - QAnywhere Ultra Light Agent がプライマリ Mobile Link サーバへの接続を再試行しないことを意味します。

備考

デフォルトでは、QAnywhere Ultra Light 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 Ultra Light 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 Ultra Light 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 Ultra Light 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 Ultra Light 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、qauagent -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」 772 ページ
- 「-x オプション」 786 ページ
- 「フェールオーバ・メカニズムの設定」 41 ページ

-id オプション

QAnywhere Ultra Light 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

qauagent -id *id* ...

デフォルト

ID のデフォルト値は、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 qauagent 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)

この他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」 25 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

qauagent -idl *download-size* [**K** | **M**] ...

デフォルト

-1 - 最大ダウンロード・サイズが指定されていないことを意味します。

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 Ultra Light 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。

キューに登録された配信対象メッセージのうちでダウンロード・スレッショルドを超えたものは、複数の小さいメッセージ・パーツに分割されます。各メッセージ・パーツは別個にダウンロードできるため、何度か同期を行いながらメッセージを段階的にダウンロードできます。送信

先ですべてのメッセージ・パーツが受信されると、完全なメッセージが受信されたことになります。

インクリメンタル・ダウンロードのサイズは概算値です。実際のダウンロード・サイズは、メッセージのサイズ以外にも多くの要素によって異なります。

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」 792 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

qauagent -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 Ultra Light 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。

アップロード・スレッショルドを超えたメッセージは、複数の小さいメッセージ・パーツに分割されます。各メッセージ・パーツは別個にアップロードできるため、何度か同期を行いながらメッセージを段階的にアップロードできます。送信先ですべてのメッセージ・パーツが受信されると、完全なメッセージが受信されたことになります。

インクリメンタル・アップロードのサイズは概算値です。実際のアップロード・サイズは、メッセージのサイズ以外にも多くの要素によって異なります。

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」 792 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

qauagent -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 Ultra Light Agent に知らせるために使用されます。UDP ポートは、QAnywhere Ultra Light Agent が制御コマンドを Listener に送信する際にも使用されます。

参照

- 「Push 通知によるメッセージングのシナリオ」 7 ページ
- 「-push オプション」 782 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

qauagent -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」 776 ページ
- 「-mu オプション」 777 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

qauagent -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」 777 ページ

-mu オプション

Mobile Link ユーザ名を指定します。

構文

qauagent -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 ユーザ名は、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」 33 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」 773 ページ
- 「-mp オプション」 776 ページ
- 「リモート ID」 『Mobile Link - クライアント管理』

-o オプション

指定されたログ・ファイルに出力を送信します。

構文

qauagent -o *logfile* ...

デフォルト

なし

備考

指定された名前のファイルにログを出力します。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。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、ログを出力します。

たとえば、*c:¥tmp¥mylog.out* というログ・ファイルを指定した場合、qauagent は *c:¥tmp¥mylog.out* に、dblsn は *c:¥tmp¥mylog_lsn.out* に、それぞれログを出力します。

参照

- 「-ot オプション」 779 ページ
- 「-on オプション」 778 ページ
- 「-os オプション」 779 ページ
- 「-v オプション」 785 ページ

-on オプション

QAnywhere Ultra Light 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、現在のファイルが拡張子 *.old* の付いた名前に変更され、新しいファイルが作成されます。

構文

qauagent -on *size* [**k** | **m**] ...

デフォルト

なし

備考

size には、メッセージ・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 Ultra Light Agent は出力ファイルを拡張子 *.old* の付いた名前に変更し、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」 777 ページ
- 「-ot オプション」 779 ページ
- 「-os オプション」 779 ページ
- 「-v オプション」 785 ページ

-os オプション

QAnywhere Ultra Light 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

qauagent -os *size* [**k** | **m**] ...

デフォルト

なし

備考

size には、メッセージのログを出力す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 10KB です。

QAnywhere Ultra Light Agent は、現在のファイル・サイズを確認してから、ファイルにメッセージのログを出力します。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 Ultra Light 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。ここで、*xx* は 00 ～ 99 の連続した数字で、*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、-on 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」 777 ページ
- 「-ot オプション」 779 ページ
- 「-on オプション」 778 ページ
- 「-v オプション」 785 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

qauagent -ot *logfile* ...

デフォルト

なし

備考

指定された名前のファイルにログを出力します。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。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、ログを出力します。

たとえば、*c:¥tmp¥mylog.out* というログ・ファイルを指定した場合、qauagent は *c:¥tmp¥mylog.out* に、dblsn は *c:¥tmp¥mylog_lsn.out* に、それぞれログを出力します。

参照

- 「-o オプション」 777 ページ
- 「-on オプション」 778 ページ
- 「-os オプション」 779 ページ
- 「-v オプション」 785 ページ

-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

qauagent -policy *policy-type* ...

policy-type: **ondemand** | **scheduled**[*interval-in-seconds*] | **automatic** | *rules-file*

デフォルト値

- 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。

on demand 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」 70 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。

  スケジュール設定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を受信した。
  - 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。
  - 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・ステータス変更通知を受信した。
  - 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。

- **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 Ultra Light 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」 7 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」 819 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」 54 ページ
- 「Push 通知によるメッセージングのシナリオ」 7 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

qauagent -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 qauagent 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 qauagent 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。

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」 5 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」 37 ページ
- 「QAnywhere Agent の起動」 51 ページ
- 「Push 通知の通知」 71 ページ
- 「-lp オプション」 776 ページ

-q オプション

QAnywhere Ultra Light 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

qauagent -q ...

デフォルト

なし

備考

QAnywhere Ulra Light 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」 783 ページ

-qi オプション

QAnywhere Ultra Light 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

qauagent -qi ...

デフォルト

なし

備考

QAnywhere Ultra Light 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 Mobile 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 Mobile 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 Mobile 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 Ultra Light Agent をサービスのように実行できます。

-qi クワイエット・モードで動作している QAnywhere Ultra Light Agent は、**qastop** と入力することによってのみ停止できます。

参照

- 「qastop ユーティリティ」 789 ページ
- 「-q オプション」 783 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

qauagent -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、Ultra Light 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 Ultra Light 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 Ultra Light 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、qauagent 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、qauagent 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 qauagent 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」 25 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」 140 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 Ultra Light Agent は、初期化が完了するとすぐに終了します。

```
qauagent -si -c "DBF=qaclient.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 Ultra Light Agent コンソールに表示する情報を指定できます。

構文

qauagent -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。qauagent 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。

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。

-v を単独で指定すると、最小限の情報のログが取られます。

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。「Windows デバイス用の Listener ユーティリティ」 『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** 同期のロギングをすべて表示します。
- **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 Ultra Light 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 Ultra Light 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 Ultra Light 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」 777 ページ
- 「-ot オプション」 779 ページ
- 「-on オプション」 778 ページ
- 「-os オプション」 779 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

qauagent -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・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 Ultra Light Agent と同じデバイス上に存在しない場合、必須となります。

-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 Ultra Light 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 Ultra Light 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・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」 142 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」 41 ページ
- 「-fd オプション」 772 ページ
- 「-fr オプション」 772 ページ

-xd オプション

QAnywhere Ultra Light 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

qauagent -xd

備考

-xd を指定すると、QAnywhere Ultra Light 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 Ultra Light 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・プロパティを初期化し、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確立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 Ultra Light 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」 28 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" );
```

qastop ユーティリティ

エージェントがクワイエット・モードで動作しているときに QAnywhere Agent または QAnywhere Ultra Light Agent を停止するには、QAnywhere 停止ユーティリティを使用します。

構文

qastop [*option* ...]

オプション	説明
-id *id*	QAnywhere Agent または QAnywhere Ultra Light Agent を停止するクライアント・メッセージ・ストアの ID を指定します。
-wc *name*	QAnywhere Agent または QAnywhere Ultra Light Agent を停止するウィンドウ・クラス名を指定します。

参照

- 「-q オプション」 761 ページ
- 「-q オプション」 783 ページ
- 「QAnywhere Agent の停止」 52 ページ

QAnywhere のプロパティ

目次

クライアント・メッセージ・ストア・プロパティ

以下の項では、クライアント・メッセージ・ストア・プロパティについて説明します。

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、いくつかのクライアント・メッセージ・ストア・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、各文字列はパイプ記号で区切られます。
- **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。デフォルトでは、QAnywhere が使用する最大ダウンロード・サイズは -1 です。これは最大サイズの指定がないことを意味します。サーバ・コネクタまたは送信先エイリアスから送信されたメッセージが、指定されたダウンロードのインクリメント・サイズを超えた場合、メッセージは小さなメッセージ・パーツに分割され、別個のダウンロードとして送信されます。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」 751 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 256 K 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。メッセージが指定されたアップロードのインクリメント・サイズを超えた場合、メッセージは小さなメッセージ・パーツに分割され、別個のアップロードとして送信されます。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」 751 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いくつか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。各文字列はパイプ記号で区切られます。

- **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」 762 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」 416 ページ
- .NET API：「MessageStoreProperties クラス」 238 ページ
- Java API：「MessageStoreProperties インタフェース」 532 ページ
- SQL API：「メッセージ・ストア・プロパティ」 709 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 (SQL Anywhere メッセージ・ストアのみ)
- **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。

Ultra Light をクライアント・メッセージ・ストアとして使用する際のプロパティ属性は、限定的にサポートされています。Ultra Light メッセージ・ストアでは、事前定義の ias_Network プロパティのみをサポートしています。

例 (SQL Anywhere のみ)

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 ～ 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」 809 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・プロパティの列挙体

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」 299 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」 449 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」 586 ページを参照してください。

```
// qaManager is a QAManager instance.
Enumeration propertyNames = qaManager.getStorePropertyNames();
```

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」 447 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」 273 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」 568 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」 709 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」 198 ページ
- 「Sybase Central でのサーバ・プロパティの設定」 801 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.server.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 ～ 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・プロパティの設定」 165 ページを参照してください。

- **ianywhere.qa.server.connectorPropertiesFiles**

  廃止予定機能
  Sybase Central 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。

  「コネクタ」 161 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・メッセージ・ウィンドウに出力されます。mlsrv11 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 サーバのメッセージ・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。このプロパティを設定しなかった場合、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」 182 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 ～ 23 の値)	0
k	1 日の特定の時刻 (1 ～ 24 の値)	24
K	午前／午後の特定の時刻 (0 ～ 11 の値)	0
h	午前／午後の特定の時刻 (1 ～ 12 の値)	12
m	分	30
s	秒	55

- **ianywhere.qa.server.transmissionRulesFile**

> **廃止予定機能**
> Sybase Central 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 11] - [Sybase Central] を選択します。
2. **[接続] - [QAnywhere 11 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。**[OK]** をクリックします。
4. 左ウィンドウ枠の **[サーバ・メッセージ・ストア]** の下で、データ・ソースの名前を選択します。
5. **[ファイル] - [プロパティ]** を選択します。

JMS コネクタ・プロパティ

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 Mobile Link サーバ・メッセージ・ウィンドウ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。

  「JMS コネクタへの QAnywhere メッセージの送信」 167 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**の **[コネクタ名]** ページにある **[コネクタ名]** フィールドで設定します。
- **ianywhere.connector.incoming.priority** すべての受信メッセージに割り当てられている優先度を、整数で指定します。この値が指定されていないか負の数である場合、そのコネクタのタイプでデフォルトの値が指定されます。JMS では、JMS メッセージの優先度がデフォルトで使用されます。Web サービスでは、デフォルトは 4 です。
- **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あることを意味します。
- **ianywhere.connector.incoming.ttl** すべての受信メッセージに割り当てられている存続期間 (ミリ秒単位) を、整数で指定します。この値が指定されていないか負の数である場合、そのコネクタのタイプでデフォルトの値が指定されます。値が 0 の場合、メッセージは期限切れになりません。JMS では、JMS メッセージの有効期限を使用してデフォルト値が計算されます。Web サービスでは、デフォルトは 0 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]** ウィンドウの **[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・メッセージ・ウィンドウとメッセージ・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]** ウィンドウの **[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・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 ～ 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]** ウィンドウの **[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・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]** ウィンドウの **[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]** ウィンドウの **[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**の **[JNDI の設定]** ページにある **[ユーザ名]** フィールド、または **[コネクタ・プロパティ]** ウィンドウの **[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**の **[JNDI の設定]** ページにある **[JMDI ファクトリ]** フィールド、または **[コネクタ・プロパティ]** ウィンドウの **[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**の **[JNDI の設定]** ページにある **[パスワード]** フィールド、または **[コネクタ・プロパティ]** ウィンドウの **[JMS]** タブにある **[JNDI]** 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.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**の **[JNDI の設定]** ページにある **[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]** ウィンドウの **[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**の **[JMS キューの設定]** ページにある **[ユーザ名]** フィールド、または **[コネクタ・プロパティ]** ウィンドウの **[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**の **[JMS キューの設定]** ページにある **[パスワード]** フィールド、または **[コネクタ・プロパティ]** ウィンドウの **[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**の **[JMS キューの設定]** ページにある **[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]** ウィンドウの **[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**の **[コネクタ名]** ページにある **[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**の **[JMS トピックの設定]** ページにある **[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]** ウィンドウ

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**の **[JMS トピックの設定]** ページにある **[ユーザ名]** フィールド、または **[コネクタ・プロパティ]** ウィンドウの **[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**の **[JMS トピックの設定]** ページにある **[パスワード]** フィールド、または **[コネクタ・プロパティ]** ウィンドウの **[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。

- **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 Mobile Link サーバ・メッセージ・ウィンドウ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。

  「JMS コネクタへの QAnywhere メッセージの送信」 167 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**の **[コネクタ名]** ページにある **[コネクタ名]** フィールドで設定します。

- **ianywhere.connector.incoming.priority**　すべての受信メッセージに割り当てられている優先度を、整数で指定します。この値が指定されていないか負の数である場合、そのコネクタのタイプでデフォルトの値が指定されます。JMS では、JMS メッセージの優先度がデフォルトで使用されます。Web サービスでは、デフォルトは 4 です。

- **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あることを意味します。

- **ianywhere.connector.incoming.ttl**　すべての受信メッセージに割り当てられている存続期間 (ミリ秒単位) を、整数で指定します。この値が指定されていないか負の数である場合、そのコネクタのタイプでデフォルトの値が指定されます。値が 0 の場合、メッセージは期限切れになりません。JMS では、JMS メッセージの有効期限を使用してデフォルト値が計算されます。Web サービスでは、デフォルトは 0 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]** ウィンドウの **[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・メッセージ・ウィンドウとメッセージ・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]** ウィンドウの **[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・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 ～ 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]** ウィンドウの **[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・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]** ウィンドウの **[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]** ウィンドウの **[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**の **[JNDI の設定]** ページにある **[ユーザ名]** フィールド、または **[コネクタ・プロパティ]** ウィンドウの **[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**の **[JNDI の設定]** ページにある **[JMDI ファクトリ]** フィールド、または **[コネクタ・プロパティ]** ウィンドウの **[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**の **[JNDI の設定]** ページにある **[パスワード]** フィールド、または **[コネクタ・プロパティ]** ウィンドウの **[JMS]** タブにある **[JNDI]** 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.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**の **[JNDI の設定]** ページにある **[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]** ウィンドウの **[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**の **[JMS キューの設定]** ページにある **[ユーザ名]** フィールド、または **[コネクタ・プロパティ]** ウィンドウの **[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**の **[JMS キューの設定]** ページにある **[パスワード]** フィールド、または **[コネクタ・プロパティ]** ウィンドウの **[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**の **[JMS キューの設定]** ページにある **[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]** ウィンドウの **[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**の **[コネクタ名]** ページにある **[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**の **[JMS トピックの設定]** ページにある **[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]** ウィンドウの **[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**の **[JMS トピックの設定]** ページにある **[ユーザ名]** フィールド、または **[コネクタ・プロパティ]** ウィンドウの **[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**の **[JMS トピックの設定]** ページにある **[パスワード]** フィールド、または **[コネクタ・プロパティ]** ウィンドウの **[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。

QAnywhere 転送ルールと削除ルール

目次

ルールの構文

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 ～ 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.
;
   START TIME '06:00:00' EVERY 1 hours = ias_Priority = 9
   BETWEEN '08:00:00' AND '09:00:00' EVERY 1 minutes = ias_Priority = 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** *pattern* [**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*, ... **)**

constant: *integer* | *floating point number* | *string* | *boolean*

integer: An integer in the range -2**63 to 2**63-1.

floating point number: A number 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.

boolean: A statement that is **TRUE** or **FALSE**, **T** or **F**, **Y** or **N**, **1** or **0**.

operator: **+** | **-** | ***** | */*

rule-variable:

「ルール変数」 816 ページを参照してください。

rule-function:

「ルール関数」 814 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* **)**

  次に例を示します。

  - age BETWEEN 15 AND 19 この文は次の文と同義です。 age >=15 AND age <= 19
  - age NOT BETWEEN 15 AND 19 は age < 15 OR age > 19 と同義です。

- **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、'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、'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、'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、'sad' の場合は true、'said' の場合は false になります。
- phone NOT LIKE '12%3' は、'123' または '12993' の場合は false、'1234' の場合は 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、'_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* **)**	日時にいくつかの日時の単位を加算した日時を返します。*datepart* は、year、quarter、month、week、day、hour、minute、second 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、year、quarter、month、week、day、dayofyear、weekday、hour、minute、second 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。
LENGTH(*string* **)**	文字列の文字数を返します。

構文	説明
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。

```
START TIME '06:00:00' every 1 hours = ias_Status >= ias_FinalState ¥
  AND ias_StatusTime < DATEADD( day, -10, ias_CurrentTimestamp) ¥
  AND ias_TransmissionStatus = ias_Transmitted
```

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」 723 ページ
- 「クライアント・メッセージ・ストア・プロパティ」 28 ページ
- 「ルール・エンジンで定義される変数」 816 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 に設定されている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、Unicode 文字で構成されます。
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ます。

- **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。

「データベース・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 [データベース]」『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 ～ 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されました。
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りました。
IAS_FINAL_STATE	メッセージが受信されたか、期限切れになりました。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ません。
IAS_RECEIVED_STATE	メッセージが宛先に受信されました。
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ます。

- **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ません。

IAS_TRANSMITTED	メッセージは、目的の受信側メッセージ・ストアに転送済みです。
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、転送は不要です。
IAS_TRANSMITTING	メッセージは目的の受信先に転送されましたが、まだ受信確認されていません。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」 793 ページを参照してください。

メッセージ転送ルール

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」 819 ページ
- 「サーバ側の転送ルール」 820 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」 810 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」 54 ページ
- 「-policy オプション」 758 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」 823 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」 810 ページと「条件構文」 812 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」 823 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、営業時間外はどのメッセージでも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 < datetime('8:00:00') ¥
 OR ias_CurrentTime > datetime('18:00:00')
```

サーバ側の転送ルール

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。

スケジュールされたサーバ側の転送ルール

スケジュールされたサーバ側の転送ルールを指定するときは、次の点に注意してください。

- クライアントに対する自動ルールの評価は、そのクライアントが同期されるたび、および自動ルールの評価時に行われる。
- クライアントに対するスケジュールされたルールの評価は、指定されたスケジュールで行われる。
- ルールの評価により、現在ルール条件を満たしているメッセージを持つクライアントに対して、Push 通知が送信される。
- クライアントが同期されるたびに、そのクライアントの自動ルールの条件を満たすメッセージがクライアントに転送される。
- クライアントが前回同期された後でスケジュールされたルールによってそのクライアントに Push 通知が送信された場合に限り、次回の同期時にスケジュールされたルールの条件を満たすすべてのメッセージがクライアントに転送される。

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」 823 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'
;
; Messages outside business hours should always be
; transmitted

auto = ias_Priority >= 7 OR ias_ContentSize < 100 ¥
   OR ias_CurrentTime < datetime('8:00:00') ¥
   OR ias_CurrentTime > datetime('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'
;
; Messages outside business hours should always be
; transmitted

auto = tm_Subject NOT LIKE '%non-business%' ¥
    OR ias_CurrentTime < datetime('8:00am') OR ias_CurrentTime > datetime('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 回以上受信されていても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 列挙体」 226 ページ
- C++：「AcknowledgementMode クラス」 406 ページ
- Java：「AcknowledgementMode インタフェース」 524 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」 819 ページを参照してください。

次に、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 canceled.

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」 816 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」 820 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」 816 ページを参照してください。

用語解説

用語解説

Adaptive Server Anywhere (ASA)

SQL Anywhere Studio のリレーショナル・データベース・サーバ・コンポーネントであり、主に、モバイル環境と埋め込み環境、または小規模および中規模のビジネス用のサーバとして使用されます。バージョン 10.0.0 で、Adaptive Server Anywhere は SQL Anywhere サーバに、SQL Anywhere Studio は SQL Anywhere にそれぞれ名前が変更されました。

参照：「SQL Anywhere」 832 ページ。

Carrier

Mobile Link システム・テーブルまたは Notifier プロパティ・ファイルに保存される Mobile Link オブジェクトで、システム起動同期で使用される通信業者に関する情報が含まれます。

参照：「サーバ起動同期」 837 ページ。

DB 領域

データ用の領域をさらに作成する追加のデータベース・ファイルです。1 つのデータベースは 13 個までのファイルに保管されます (初期ファイル 1 つと 12 の DB 領域)。各テーブルは、そのインデックスとともに、単一のデータベース・ファイルに含まれている必要があります。CREATE DBSPACE という SQL コマンドで、新しいファイルをデータベースに追加できます。

参照：「データベース・ファイル」 841 ページ。

DBA 権限

ユーザに、データベース内の管理作業を許可するレベルのパーミッションです。DBA ユーザにはデフォルトで DBA 権限が与えられています。

参照：「データベース管理者 (DBA)」 841 ページ。

EBF

Express Bug Fix の略です。Express Bug Fix は、1 つ以上のバグ・フィックスが含まれる、ソフトウェアのサブセットです。これらのバグ・フィックスは、更新のリリース・ノートにリストされます。バグ・フィックス更新を適用できるのは、同じバージョン番号を持つインストール済みのソフトウェアに対してだけです。このソフトウェアについては、ある程度のテストが行われているとはいえ、完全なテストが行われたわけではありません。自分自身でソフトウェアの妥当性を確かめるまでは、アプリケーションとともにこれらのファイルを配布しないでください。

Embedded SQL

C プログラム用のプログラミング・インタフェースです。SQL Anywhere の Embedded SQL は ANSI と IBM 規格に準拠して実装されています。

FILE

SQL Remote のレプリケーションでは、レプリケーション・メッセージのやりとりのために共有ファイルを使うメッセージ・システムのことです。これは特定のメッセージ送信システムに頼らずにテストやインストールを行うのに便利です。

参照：「レプリケーション」 849 ページ。

grant オプション

他のユーザにパーミッションを許可できるレベルのパーミッションです。

iAnywhere JDBC ドライバ

iAnywhere JDBC ドライバでは、pure Java である jConnect JDBC ドライバに比べて何らかの有利なパフォーマンスや機能を備えた JDBC ドライバが提供されます。ただし、このドライバは pure Java ソリューションではありません。iAnywhere JDBC ドライバは一般に推奨されるドライバです。

参照：

- 「JDBC」 829 ページ
- 「jConnect」 829 ページ

InfoMaker

レポート作成とデータ管理用のツールです。洗練されたフォーム、レポート、グラフ、クロスタブ、テーブルを作成できます。また、これらを基本的な構成要素とするアプリケーションも作成できます。

Interactive SQL

データベース内のデータの変更や問い合わせ、データベース構造の修正ができる、SQL Anywhere のアプリケーションです。Interactive SQL では、SQL 文を入力するためのウィンドウ枠が表示されます。また、クエリの進捗情報や結果セットを返すウィンドウ枠も表示されます。

JAR ファイル

Java アーカイブ・ファイルです。Java のアプリケーションで使用される 1 つ以上のパッケージの集合からなる圧縮ファイルのフォーマットです。Java プログラムをインストールしたり実行したりするのに必要なリソースが 1 つの圧縮ファイルにすべて収められています。

Java クラス

Java のコードの主要な構造単位です。これはプロシージャや変数の集まりで、すべてがある一定のカテゴリに関連しているためグループ化されたものです。

jConnect

JavaSoft JDBC 標準を Java で実装したものです。これにより、Java 開発者は多層／異機種環境でもネイティブなデータベース・アクセスができます。iAnywhere JDBC ドライバは一般に推奨されるドライバです。

参照：

- 「JDBC」 829 ページ
- 「iAnywhere JDBC ドライバ」 828 ページ

JDBC

Java Database Connectivity の略です。Java アプリケーションからリレーショナル・データにアクセスすることを可能にする SQL 言語プログラミング・インタフェースです。推奨 JDBC ドライバは、iAnywhere JDBC ドライバです。

参照：

- 「jConnect」 829 ページ
- 「iAnywhere JDBC ドライバ」 828 ページ

Listener

Mobile Link サーバ起動同期に使用される、dblsn という名前のプログラムです。Listener はリモート・デバイスにインストールされ、Push 通知を受け取ったときにデバイス上でアクションが開始されるように設定されます。

参照：「サーバ起動同期」 837 ページ。

LTM

LTM (Log Transfer Manager) は、Replication Agent とも呼ばれます。Replication Server と併用することで、LTM はデータベース・トランザクション・ログを読み込み、コミットされた変更を Sybase Replication Server に送信します。

参照：「Replication Server」 832 ページ。

Mobile Link

Ultra Light と SQL Anywhere のリモート・データベースを統合データベースと同期させるために設計された、セッションベース同期テクノロジです。

参照：

- 「統合データベース」 856 ページ
- 「同期」 856 ページ
- 「Ultra Light」 833 ページ

Mobile Link クライアント

2 種類の Mobile Link クライアントがあります。SQL Anywhere リモート・データベース用の Mobile Link クライアントは、dbmlsync コマンド・ライン・ユーティリティです。Ultra Light リモート・データベース用の Mobile Link クライアントは、Ultra Light ランタイム・ライブラリに組み込まれています。

Mobile Link サーバ

Mobile Link 同期を実行する、mlsrv11 という名前のコンピュータ・プログラムです。

Mobile Link システム・テーブル

Mobile Link の同期に必要なシステム・テーブルです。Mobile Link 設定スクリプトによって、Mobile Link 統合データベースにインストールされます。

Mobile Link モニタ

Mobile Link の同期をモニタするためのグラフィカル・ツールです。

Mobile Link ユーザ

Mobile Link ユーザは、Mobile Link サーバに接続するのに使用されます。Mobile Link ユーザをリモート・データベースに作成し、統合データベースに登録します。Mobile Link ユーザ名はデータベース・ユーザ名から完全に独立しています。

Notifier

Mobile Link サーバ起動同期に使用されるプログラムです。Notifier は Mobile Link サーバに統合されており、統合データベースに Push 要求がないか確認し、Push 通知を送信します。

参照：

- 「サーバ起動同期」 837 ページ
- 「Listener」 829 ページ

ODBC

Open Database Connectivity の略です。データベース管理システムに対する Windows の標準的なインタフェースです。ODBC は、SQL Anywhere がサポートするインタフェースの 1 つです。

ODBC アドミニストレータ

Windows オペレーティング・システムに付属している Microsoft のプログラムです。ODBC データ・ソースの設定に使用します。

ODBC データ・ソース

ユーザが ODBC からアクセスするデータと、そのデータにアクセスするために必要な情報の仕様です。

PDB

Palm のデータベース・ファイルです。

PowerDesigner

データベース・モデリング・アプリケーションです。これを使用すると、データベースやデータ・ウェアハウスの設計に対する構造的なアプローチが可能となります。SQL Anywhere には、PowerDesigner の Physical Data Model コンポーネントが付属します。

PowerJ

Java アプリケーション開発に使用する Sybase 製品です。

Push 通知

QAnywhere では、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。Mobile Link サーバ起動同期では、Push 要求データや内部情報を含むデバイスに Notifer から配信される特殊なメッセージです。

参照：

- 「QAnywhere」 831 ページ
- 「サーバ起動同期」 837 ページ

Push 要求

Mobile Link サーバ起動同期において、Push 通知をデバイスに送信する必要があるかどうかを判断するために Notifier が確認する、結果セット内の値のローです。

参照：「サーバ起動同期」 837 ページ。

QAnywhere

アプリケーション間メッセージング (モバイル・デバイス間メッセージングやモバイル・デバイスとエンタープライズの間のメッセージングなど) 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 Agent

QAnywhere では、クライアント・デバイス上で動作する独立のプロセスのことです。クライアント・メッセージ・ストアをモニタリングし、メッセージを転送するタイミングを決定します。

REMOTE DBA 権限

SQL Remote では、Message Agent (dbremote) で必要なパーミッションのレベルを指します。Mobile Link では、SQL Anywhere 同期クライアント (dbmlsync) で必要なパーミッションのレベルを指します。Message Agent (dbremote) または同期クライアントがこの権限のあるユーザとして接続した場合、DBA のフル・アクセス権が与えられます。Message Agent (dbremote) または同期クライアント (dbmlsync) から接続しない場合、このユーザ ID にはパーミッションは追加されません。

参照：「DBA 権限」 827 ページ。

Replication Agent

参照：「LTM」 829 ページ。

Replication Server

SQL Anywhere と Adaptive Server Enterprise で動作する、Sybase による接続ベースのレプリケーション・テクノロジです。Replication Server は、少数のデータベース間でほぼリアルタイムのレプリケーションを行うことを目的に設計されています。

参照：「LTM」 829 ページ。

SQL

リレーショナル・データベースとの通信に使用される言語です。SQL は ANSI により標準が定義されており、その最新版は SQL-2003 です。SQL は、公認されてはいませんが、Structured Query Language の略です。

SQL Anywhere

SQLAnywhere のリレーショナル・データベース・サーバ・コンポーネントであり、主に、モバイル環境と埋め込み環境、または小規模および中規模のビジネス用のサーバとして使用されます。SQL Anywhere は、SQL Anywhere RDBMS、Ultra Light RDBMS、Mobile Link 同期ソフトウェア、その他のコンポーネントを含むパッケージの名前でもあります。

SQL Remote

統合データベースとリモート・データベース間で双方向レプリケーションを行うための、メッセージベースのデータ・レプリケーション・テクノロジです。統合データベースとリモート・データベースは、SQL Anywhere である必要があります。

SQL ベースの同期

Mobile Link では、Mobile Link イベントを使用して、テーブル・データを Mobile Link でサポートされている統合データベースに同期する方法のことです。SQL ベースの同期では、SQL を直接使用したり、Java と .NET 用の Mobile Link サーバ API を使用して SQL を返すことができます。

SQL 文

DBMS に命令を渡すために設計された、SQL キーワードを含む文字列です。

参照：

- 「スキーマ」 839 ページ
- 「SQL」 832 ページ
- 「データベース管理システム (DBMS)」 841 ページ

Sybase Central

SQL Anywhere データベースのさまざまな設定、プロパティ、ユーティリティを使用できる、グラフィカル・ユーザ・インタフェースを持つデータベース管理ツールです。Mobile Link などの他の iAnywhere 製品を管理する場合にも使用できます。

SYS

システム・オブジェクトの大半を所有する特別なユーザです。一般のユーザは SYS でログインできません。

Ultra Light

小型デバイス、モバイル・デバイス、埋め込みデバイス用に最適化されたデータベースです。対象となるプラットフォームとして、携帯電話、ポケットベル、パーソナル・オーガナイザなどが挙げられます。

Ultra Light ランタイム

組み込みの Mobile Link 同期クライアントを含む、インプロセス・リレーショナル・データベース管理システムです。Ultra Light ランタイムは、Ultra Light の各プログラミング・インタフェースで使用されるライブラリと、Ultra Light エンジンの両方に含まれます。

Windows

Windows Vista、Windows XP、Windows 200x などの、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 CE

「Windows Mobile」 833 ページを参照してください。

Windows Mobile

Microsoft がモバイル・デバイス用に開発したオペレーティング・システムのファミリです。

アーティクル

Mobile Link または SQL Remote では、テーブル全体もしくはテーブル内のカラムとローのサブセットを表すデータベース・オブジェクトを指します。アーティクルの集合がパブリケーションです。

参照：

- 「レプリケーション」 849 ページ
- 「パブリケーション」 844 ページ

アップロード

同期中に、リモート・データベースから統合データベースにデータが転送される段階です。

アトミックなトランザクション

完全に処理されるかまったく処理されないことが保証される 1 つのトランザクションです。エラーによってアトミックなトランザクションの一部が処理されなかった場合は、データベースが一貫性のない状態になるのを防ぐために、トランザクションがロールバックされます。

アンロード

データベースをアンロードすると、データベースの構造かデータ、またはその両方がテキスト・ファイルにエクスポートされます (構造は SQL コマンド・ファイルに、データはカンマ区切りの ASCII ファイルにエクスポートされます)。データベースのアンロードには、アンロード・ユーティリティを使用します。

また、UNLOAD 文を使って、データから抜粋した部分だけをアンロードできます。

イベント・モデル

Mobile Link では、同期を構成する、begin_synchronization や download_cursor などの一連のイベントのことです。イベントは、スクリプトがイベント用に作成されると呼び出されます。

インクリメンタル・バックアップ

トランザクション・ログ専用のバックアップです。通常、フル・バックアップとフル・バックアップの間に使用します。

参照：「トランザクション・ログ」 843 ページ。

インデックス

ベース・テーブルにある 1 つ以上のカラムに関連付けられた、キーとポインタのソートされたセットです。テーブルの 1 つ以上のカラムにインデックスが設定されていると、パフォーマンスが向上します。

ウィンドウ

分析関数の実行対象となるローのグループです。ウィンドウには、ウィンドウ定義内のグループ化指定に従って分割されたデータの、1 つ、複数、またはすべてのローが含まれます。ウィンドウは、入力の現在のローについて計算を実行する必要があるローの数や範囲を含むように移動します。ウィンドウ構成の主な利点は、追加のクエリを実行しなくても、結果をグループ化して分析する機会が増えることです。

エージェント ID

参照：「クライアント・メッセージ・ストア ID」 836 ページ。

エンコード

文字コードとも呼ばれます。エンコードは、文字セットの各文字が情報の 1 つまたは複数のバイトにマッピングされる方法のことで、一般的に 16 進数で表現されます。UTF-8 はエンコードの例です。

参照：

- 「文字セット」 857 ページ
- 「コード・ページ」 837 ページ
- 「照合」 854 ページ

オブジェクト・ツリー

Sybase Central では、データベース・オブジェクトの階層を指します。オブジェクト・ツリーの最上位には、現在使用しているバージョンの Sybase Central がサポートするすべての製品が表示されます。それぞれの製品を拡張表示すると、オブジェクトの下位ツリーが表示されます。

参照：「Sybase Central」 833 ページ。

カーソル

結果セットへの関連付けに名前を付けたもので、プログラミング・インタフェースからローにアクセスしたり更新したりするときに使用します。SQL Anywhere では、カーソルはクエリ結果内で前方や後方への移動をサポートします。カーソルは、カーソル結果セット (通常 SELECT 文で定義される) とカーソル位置の 2 つの部分から構成されます。

参照：

- 「カーソル結果セット」 836 ページ
- 「カーソル位置」 835 ページ

カーソル位置

カーソル結果セット内の 1 つのローを指すポインタ。

参照：

- 「カーソル」 835 ページ
- 「カーソル結果セット」 836 ページ

カーソル結果セット

カーソルに関連付けられたクエリから生成されるローのセットです。

参照：

- 「カーソル」 835 ページ
- 「カーソル位置」 835 ページ

クエリ

データベースのデータにアクセスしたり、そのデータを操作したりする SQL 文や SQL 文のグループです。

参照：「SQL」 832 ページ。

クライアント／サーバ

あるアプリケーション (クライアント) が別のアプリケーション (サーバ) に対して情報を送受信するソフトウェア・アーキテクチャのことです。通常この 2 種類のアプリケーションは、ネットワークに接続された異なるコンピュータ上で実行されます。

クライアント・メッセージ・ストア

QAnywhere では、メッセージを保管するリモート・デバイスにある SQL Anywhere データベースのことです。

クライアント・メッセージ・ストア ID

QAnywhere では、Mobile Link リモート ID のことです。これによって、クライアント・メッセージ・ストアがユニークに識別されます。

グローバル・テンポラリ・テーブル

明示的に削除されるまでデータ定義がすべてのユーザに表示されるテンポラリ・テーブルです。グローバル・テンポラリ・テーブルを使用すると、各ユーザが、1 つのテーブルのまったく同じインスタンスを開くことができます。デフォルトでは、コミット時にローが削除され、接続終了時にもローが削除されます。

参照：

- 「テンポラリ・テーブル」 842 ページ
- 「ローカル・テンポラリ・テーブル」 850 ページ

ゲートウェイ

Mobile Link システム・テーブルまたは Notifier プロパティ・ファイルに保存される Mobile Link オブジェクトで、システム起動同期用のメッセージの送信方法に関する情報が含まれます。

参照：「サーバ起動同期」 837 ページ。

コード・ページ

コード・ページは、文字セットの文字を数値表示 (通常 0 ～ 255 の整数) にマッピングするエンコードです。Windows Code Page 1252 などのコード・ページがあります。このマニュアルの目的上、コード・ページとエンコードは同じ意味で使用されます。

参照：

- 「文字セット」 857 ページ
- 「エンコード」 835 ページ
- 「照合」 854 ページ

コマンド・ファイル

SQL 文で構成されたテキスト・ファイルです。コマンド・ファイルは手動で作成できますが、データベース・ユーティリティによって自動的に作成することもできます。たとえば、dbunload ユーティリティを使うと、指定されたデータベースの再構築に必要な SQL 文で構成されたコマンド・ファイルを作成できます。

サーバ・メッセージ・ストア

QAnywhere では、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。

サーバ管理要求

XML 形式の QAnywhere メッセージです。サーバ・メッセージ・ストアを管理したり、QAnywhere アプリケーションをモニタリングするために QAnywhere システム・キューに送信されます。

サーバ起動同期

Mobile Link サーバから Mobile Link 同期を開始する方法です。

サービス

Windows オペレーティング・システムで、アプリケーションを実行するユーザ ID がログオンしていないときにアプリケーションを実行する方法です。

サブクエリ

別の SELECT 文、INSERT 文、UPDATE 文、DELETE 文、または別のサブクエリの中にネストされた SELECT 文です。

相関とネストの 2 種類のサブクエリがあります。

サブスクリプション

Mobile Link 同期では、パブリケーションと Mobile Link ユーザ間のクライアント・データベース内のリンクであり、そのパブリケーションが記述したデータの同期を可能にします。

SQL Remote レプリケーションでは、パブリケーションとリモート・ユーザ間のリンクのことで、これによりリモート・ユーザはそのパブリケーションの更新内容を統合データベースとの間で交換できます。

参照：

- 「パブリケーション」 844 ページ
- 「Mobile Link ユーザ」 830 ページ

システム・オブジェクト

SYS または dbo が所有するデータベース・オブジェクトです。

システム・テーブル

SYS または dbo が所有するテーブルです。メタデータが格納されています。システム・テーブル (データ辞書テーブルとしても知られています) はデータベース・サーバが作成し管理します。

システム・ビュー

すべてのデータベースに含まれているビューです。システム・テーブル内に格納されている情報をわかりやすいフォーマットで示します。

ジョイン

指定されたカラムの値を比較することによって 2 つ以上のテーブルにあるローをリンクする、リレーショナル・システムでの基本的な操作です。

ジョイン・タイプ

SQL Anywhere では、クロス・ジョイン、キー・ジョイン、ナチュラル・ジョイン、ON 句を使ったジョインの 4 種類のジョインが使用されます。

参照：「ジョイン」 838 ページ。

ジョイン条件

ジョインの結果に影響を及ぼす制限です。ジョイン条件は、JOIN の直後に ON 句か WHERE 句を挿入して指定します。ナチュラル・ジョインとキー・ジョインについては、SQL Anywhere がジョイン条件を生成します。

参照：

- 「ジョイン」 838 ページ
- 「生成されたジョイン条件」 855 ページ

スキーマ

テーブル、カラム、インデックス、それらの関係などを含んだデータベース構造です。

スクリプト

Mobile Link では、Mobile Link のイベントを処理するために記述されたコードです。スクリプトは、業務上の要求に適合するように、データ交換をプログラム的に制御します。

参照：「イベント・モデル」 834 ページ。

スクリプト・バージョン

Mobile Link では、同期を作成するために同時に適用される、一連の同期スクリプトです。

スクリプトベースのアップロード

Mobile Link では、ログ・ファイルを使用した方法の代わりとなる、アップロード処理のカスタマイズ方法です。

ストアド・プロシージャ

ストアド・プロシージャは、データベースに保存され、データベース・サーバに対する一連の操作やクエリを実行するために使用される SQL 命令のグループです。

スナップショット・アイソレーション

読み込み要求を発行するトランザクション用のデータのコミットされたバージョンを返す、独立性レベルの種類です。SQL Anywhere では、スナップショット、文のスナップショット、読み込み専用文のスナップショットの 3 つのスナップショットの独立性レベルがあります。スナップショット・アイソレーションが使用されている場合、読み込み処理は書き込み処理をブロックしません。

参照：「独立性レベル」 857 ページ。

セキュア機能

データベース・サーバが起動されたときに、そのデータベース・サーバで実行されているデータベースでは使用できないように -sf オプションによって指定される機能です。

セッション・ベースの同期

統合データベースとリモート・データベースの両方でデータ表現の一貫性が保たれる同期です。Mobile Link はセッション・ベースです。

ダイレクト・ロー・ハンドリング

Mobile Link では、テーブル・データを Mobile Link でサポートされている統合データベース以外のソースに同期する方法のことです。アップロードとダウンロードの両方をダイレクト・ロー・ハンドリングで実装できます。

参照：

- 「統合データベース」 856 ページ
- 「SQL ベースの同期」 833 ページ

ダウンロード

同期中に、統合データベースからリモート・データベースにデータが転送される段階です。

チェックサム

データベース・ページを使用して記録されたデータベース・ページのビット数の合計です。チェックサムを使用すると、データベース管理システムは、ページがディスクに書き込まれるときに数が一致しているかを確認することで、ページの整合性を検証できます。数が一致した場合は、ページが正常に書き込まれたとみなされます。

チェックポイント

データベースに加えたすべての変更内容がデータベース・ファイルに保存されるポイントです。通常、コミットされた変更内容はトランザクション・ログだけに保存されます。

データ・キューブ

同じ結果を違う方法でグループ化およびソートされた内容を各次元に反映した、多次元の結果セットです。データ・キューブは、セルフジョイン・クエリと相関サブクエリを必要とするデータの複雑な情報を提供します。データ・キューブは OLAP 機能の一部です。

データベース

プライマリ・キーと外部キーによって関連付けられているテーブルの集合です。これらのテーブルでデータベース内の情報が保管されます。また、テーブルとキーによってデータベースの構造が定義されます。データベース管理システムでこの情報にアクセスします。

参照：

- 「外部キー」 851 ページ
- 「プライマリ・キー」 846 ページ
- 「データベース管理システム (DBMS)」 841 ページ
- 「リレーショナル・データベース管理システム (RDBMS)」 849 ページ

データベース・オブジェクト

情報を保管したり受け取ったりするデータベース・コンポーネントです。テーブル、インデックス、ビュー、プロシージャ、トリガはデータベース・オブジェクトです。

データベース・サーバ

データベース内にある情報へのすべてのアクセスを規制するコンピュータ・プログラムです。SQL Anywhere には、ネットワーク・サーバとパーソナル・サーバの 2 種類のサーバがあります。

データベース・ファイル

データベースは 1 つまたは複数のデータベース・ファイルに保持されます。まず、初期ファイルがあり、それに続くファイルは DB 領域と呼ばれます。各テーブルは、それに関連付けられているインデックスとともに、単一のデータベース・ファイルに含まれている必要があります。

参照：「DB 領域」 827 ページ。

データベース管理システム (DBMS)

データベースを作成したり使用したりするためのプログラムの集合です。

参照：「リレーショナル・データベース管理システム (RDBMS)」 849 ページ。

データベース管理者 (DBA)

データベースの管理に必要なパーミッションを持つユーザです。DBA は、データベース・スキーマのあらゆる変更や、ユーザやグループの管理に対して、全般的な責任を負います。データベース管理者のロールはデータベース内に自動的に作成されます。その場合、ユーザ ID は DBA であり、パスワードは sql です。

データベース所有者 (dbo)

SYS が所有しないシステム・オブジェクトを所有する特別なユーザです。

参照：

- 「データベース管理者 (DBA)」 841 ページ
- 「SYS」 833 ページ

データベース接続

クライアント・アプリケーションとデータベース間の通信チャネルです。接続を確立するためには有効なユーザ ID とパスワードが必要です。接続中に実行できるアクションは、そのユーザ ID に付与された権限によって決まります。

データベース名

サーバがデータベースをロードするとき、そのデータベースに指定する名前です。デフォルトのデータベース名は、初期データベース・ファイルのルート名です。

参照：「データベース・ファイル」 841 ページ。

データ型

CHAR や NUMERIC などのデータのフォーマットです。ANSI SQL 規格では、サイズ、文字セット、照合に関する制限もデータ型に組み込めます。

参照：「ドメイン」 842 ページ。

データ操作言語 (DML)

データベース内のデータの操作に使う SQL 文のサブセットです。DML 文は、データベース内のデータを検索、挿入、更新、削除します。

データ定義言語 (DDL)

データベース内のデータの構造を定義するときに使う SQL 文のサブセットです。DDL 文は、テーブルやユーザなどのデータベース・オブジェクトを作成、変更、削除できます。

デッドロック

先へ進めない場所に一連のトランザクションが到達する状態です。

デバイス・トラッキング

Mobile Link サーバ起動同期において、デバイスを特定する Mobile Link のユーザ名を使用して、メッセージのアドレスを指定できる機能です。

参照：「サーバ起動同期」 837 ページ。

テンポラリ・テーブル

データを一時的に保管するために作成されるテーブルです。グローバルとローカルの 2 種類があります。

参照：

- 「ローカル・テンポラリ・テーブル」 850 ページ
- 「グローバル・テンポラリ・テーブル」 836 ページ

ドメイン

適切な位置に精度や小数点以下の桁数を含み、さらにオプションとしてデフォルト値や CHECK 条件などを含んでいる、組み込みデータ型のエイリアスです。ドメインには、通貨データ型のように SQL Anywhere が事前に定義したものもあります。ユーザ定義データ型とも呼ばれます。

参照：「データ型」 842 ページ。

トランザクション

作業の論理単位を構成する一連の SQL 文です。1 つのトランザクションは完全に処理されるかまったく処理されないかのどちらかです。SQL Anywhere は、ロック機能のあるトランザクション処理をサポートしているので、複数のトランザクションが同時にデータベースにアクセスしてもデータを壊すことがありません。トランザクションは、データに加えた変更を永久なものにする COMMIT 文か、トランザクション中に加えられたすべての変更を元に戻す ROLLBACK 文のいずれかで終了します。

トランザクション・ログ

データベースに対するすべての変更内容が、変更された順に格納されるファイルです。パフォーマンスを向上させ、データベース・ファイルが破損した場合でもデータをリカバリできます。

トランザクション・ログ・ミラー

オプションで設定できる、トランザクション・ログ・ファイルの完全なコピーのことで、トランザクション・ログと同時に管理されます。データベースの変更がトランザクション・ログへ書き込まれると、トランザクション・ログ・ミラーにも同じ内容が書き込まれます。

ミラー・ファイルは、トランザクション・ログとは別のデバイスに置いてください。一方のデバイスに障害が発生しても、もう一方のログにリカバリのためのデータが確保されます。

参照：「トランザクション・ログ」 843 ページ。

トランザクション単位の整合性

Mobile Link で、同期システム全体でのトランザクションの管理を保証します。トランザクション全体が同期されるか、トランザクション全体がまったく同期されないかのどちらかになります。

トリガ

データを修正するクエリをユーザが実行すると、自動的に実行されるストアド・プロシージャの特別な形式です。

参照：

- 「ロー・レベルのトリガ」 850 ページ
- 「文レベルのトリガ」 857 ページ
- 「整合性」 854 ページ

ネットワーク・サーバ

共通ネットワークを共有するコンピュータからの接続を受け入れるデータベース・サーバです。

参照：「パーソナル・サーバ」 844 ページ。

ネットワーク・プロトコル

TCP/IP や HTTP などの通信の種類です。

パーソナル・サーバ

クライアント・アプリケーションが実行されているコンピュータと同じマシンで実行されているデータベース・サーバです。パーソナル・データベース・サーバは、単一のコンピュータ上で単一のユーザが使用しますが、そのユーザからの複数の同時接続をサポートできます。

パッケージ

Java では、それぞれが互いに関連のあるクラスの集合を指します。

ハッシュ

ハッシュは、インデックスのエントリをキーに変換する、インデックスの最適化のことです。インデックスのハッシュの目的は、必要なだけの実際のロー・データをロー ID に含めることで、インデックスされた値を特定するためのローの検索、ロード、アンパックという負荷の高い処理を避けることです。

パフォーマンス統計値

データベース・システムのパフォーマンスを反映する値です。たとえば、CURRREAD 統計値は、データベース・サーバが要求したファイル読み込みのうち、現在まだ完了していないものの数を表します。

パブリケーション

Mobile Link または SQL Remote では、同期されるデータを識別するデータベース・オブジェクトのことです。Mobile Link では、クライアント上にのみ存在します。1 つのパブリケーションは複数のアーティクルから構成されています。SQL Remote ユーザは、パブリケーションに対してサブスクリプションを作成することによって、パブリケーションを受信できます。Mobile Link ユーザは、パブリケーションに対して同期サブスクリプションを作成することによって、パブリケーションを同期できます。

参照：

- 「レプリケーション」 849 ページ
- 「アーティクル」 834 ページ
- 「パブリケーションの更新」 844 ページ

パブリケーションの更新

SQL Remote レプリケーションでは、単一のデータベース内の 1 つまたは複数のパブリケーションに対して加えられた変更のリストを指します。パブリケーションの更新は、レプリケーション・メッセージの一部として定期的にリモート・データベースへ送られます。

参照：

- 「レプリケーション」 849 ページ
- 「パブリケーション」 844 ページ

パブリッシャ

SQL Remote レプリケーションでは、レプリケートできる他のデータベースとレプリケーション・メッセージを交換できるデータベースの単一ユーザを指します。

参照：「レプリケーション」 849 ページ。

ビジネス・ルール

実世界の要求に基づくガイドラインです。通常ビジネス・ルールは、検査制約、ユーザ定義データ型、適切なトランザクションの使用により実装されます。

参照：

- 「制約」 854 ページ
- 「ユーザ定義データ型」 848 ページ

ヒストグラム

ヒストグラムは、カラム統計のもっとも重要なコンポーネントであり、データ分散を表します。SQL Anywhere は、ヒストグラムを維持して、カラムの値の分散に関する統計情報をオプティマイザに提供します。

ビット配列

ビット配列は、一連のビットを効率的に保管するのに使用される配列データ構造の種類です。ビット配列は文字列に似てますが、使用される要素は文字ではなく 0 (ゼロ) と 1 になります。ビット配列は、一般的にブール値の文字列を保持するのに使用されます。

ビュー

データベースにオブジェクトとして格納される SELECT 文です。ビューを使用すると、ユーザは 1 つまたは複数のテーブルのローやカラムのサブセットを参照できます。ユーザが特定のテーブルやテーブルの組み合わせのビューを使うたびに、テーブルに保持されているデータから再計算されます。ビューは、セキュリティの目的に有用です。またデータベース情報の表示を調整して、データへのアクセスが簡単になるようにする場合も役立ちます。

ファイルベースのダウンロード

Mobile Link では、ダウンロードがファイルとして配布されるデータの同期方法であり、同期変更のオフライン配布を可能にします。

ファイル定義データベース

Mobile Link では、ダウンロード・ファイルの作成に使用される SQL Anywhere データベースのことです。

参照：「ファイルベースのダウンロード」 845 ページ。

フェールオーバ

アクティブなサーバ、システム、またはネットワークで障害や予定外の停止が発生したときに、冗長な (スタンバイ) サーバ、システム、またはネットワークに切り替えることです。フェールオーバは自動的に発生します。

プライマリ・キー

テーブル内のすべてのローをユニークに識別する値を持つカラムまたはカラムのリストです。

参照：「外部キー」 851 ページ。

プライマリ・キー制約

プライマリ・キーのカラムに対する一意性制約です。テーブルにはプライマリ・キー制約を 1 つしか設定できません。

参照：

- 「制約」 854 ページ
- 「検査制約」 853 ページ
- 「外部キー制約」 852 ページ
- 「一意性制約」 851 ページ
- 「整合性」 854 ページ

プライマリ・テーブル

外部キー関係でプライマリ・キーを含むテーブルです。

プラグイン・モジュール

Sybase Central で、製品にアクセスしたり管理したりする方法です。プラグインは、通常、インストールすると Sybase Central にもインストールされ、自動的に登録されます。プラグインは、多くの場合、Sybase Central のメイン・ウィンドウに最上位のコンテナとして、その製品名 (たとえば SQL Anywhere) で表示されます。

参照：「Sybase Central」 833 ページ。

フル・バックアップ

データベース全体をバックアップすることです。オプションでトランザクション・ログのバックアップも可能です。フル・バックアップには、データベース内のすべての情報が含まれており、システム障害やメディア障害が発生した場合の保護として機能します。

参照：「インクリメンタル・バックアップ」 834 ページ。

プロキシ・テーブル

メタデータを含むローカル・テーブルです。リモート・データベース・サーバのテーブルに、ローカル・テーブルであるかのようにアクセスするときに使用します。

参照：「メタデータ」 847 ページ。

ベース・テーブル

データを格納する永久テーブルです。テーブルは、テンポラリ・テーブルやビューと区別するために、「ベース・テーブル」と呼ばれることがあります。

参照：

- 「テンポラリ・テーブル」 842 ページ
- 「ビュー」 845 ページ

ポーリング

Mobile Link サーバ起動同期において、Mobile Link Listerner などのライト・ウェイト・ポーラが Notifier から Push 通知を要求する方法です。

参照：「サーバ起動同期」 837 ページ。

ポリシー

QAnywhere では、メッセージ転送の発生時期を指定する方法のことです。

マテリアライズド・ビュー

計算され、ディスクに保存されたビューのことです。マテリアライズド・ビューは、ビュー (クエリ指定を使用して定義される) とテーブル (ほとんどのテーブルの操作をそのテーブル上で実行できる) の両方の特性を持ちます。

参照：

- 「ベース・テーブル」 847 ページ
- 「ビュー」 845 ページ

ミラー・ログ

参照：「トランザクション・ログ・ミラー」 843 ページ。

メタデータ

データについて説明したデータです。メタデータは、他のデータの特質と内容について記述しています。

参照：「スキーマ」 839 ページ。

メッセージ・システム

SQL Remote のレプリケーションでは、統合データベースとリモート・データベースの間でのメッセージのやりとりに使用するプロトコルのことです。SQL Anywhere では、FILE、FTP、SMTP のメッセージ・システムがサポートされています。

参照：

- 「レプリケーション」 849 ページ
- 「FILE」 828 ページ

メッセージ・ストア

QAnywhere では、メッセージを格納するクライアントおよびサーバ・デバイスのデータベースのことです。

参照：

- 「クライアント・メッセージ・ストア」 836 ページ
- 「サーバ・メッセージ・ストア」 837 ページ

メッセージ・タイプ

SQL Remote のレプリケーションでは、リモート・ユーザと統合データベースのパブリッシャとの通信方法を指定するデータベース・オブジェクトのことを指します。統合データベースには、複数のメッセージ・タイプが定義されていることがあります。これによって、リモート・ユーザはさまざまなメッセージ・システムを使って統合データベースと通信できることになります。

参照：

- 「レプリケーション」 849 ページ
- 「統合データベース」 856 ページ

メッセージ・ログ

データベース・サーバや Mobile Link サーバなどのアプリケーションからのメッセージを格納できるログです。この情報は、メッセージ・ウィンドウに表示されたり、ファイルに記録されたりすることもあります。メッセージ・ログには、情報メッセージ、エラー、警告、MESSAGE 文からのメッセージが含まれます。

メンテナンス・リリース

メンテナンス・リリースは、同じメジャー・バージョン番号を持つ旧バージョンのインストール済みソフトウェアをアップグレードするための完全なソフトウェア・セットです (バージョン番号のフォーマットは、*メジャー.マイナー.パッチ.ビルド*です)。バグ・フィックスとその他の変更については、アップグレードのリリース・ノートにリストされます。

ユーザ定義データ型

参照：「ドメイン」 842 ページ。

ライト・ウェイト・ポーラ

Mobile Link サーバ起動同期において、Mobile Link サーバからの Push 通知をポーリングするデバイス・アプリケーションです。

参照：「サーバ起動同期」 837 ページ。

リダイレクタ

クライアントと Mobile Link サーバ間で要求と応答をルート指定する Web サーバ・プラグインです。このプラグインによって、負荷分散メカニズムとフェールオーバ・メカニズムも実装されます。

リファレンス・データベース

Mobile Link では、Ultra Light クライアントの開発に使用される SQL Anywhere データベースです。開発中は、1 つの SQL Anywhere データベースをリファレンス・データベースとしても統合データベースとしても使用できます。他の製品によって作成されたデータベースは、リファレンス・データベースとして使用できません。

リモート ID

SQL Anywhere と Ultra Light データベース内のユニークな識別子で、Mobile Link によって使用されます。リモート ID は NULL に初期設定されていますが、データベースの最初の同期時に GUID に設定されます。

リモート・データベース

Mobile Link または SQL Remote では、統合データベースとデータを交換するデータベースを指します。リモート・データベースは、統合データベース内のすべてまたは一部のデータを共有できます。

参照：

- 「同期」 856 ページ
- 「統合データベース」 856 ページ

リレーショナル・データベース管理システム (RDBMS)

関連するテーブルの形式でデータを格納するデータベース管理システムです。

参照：「データベース管理システム (DBMS)」 841 ページ。

レプリケーション

物理的に異なるデータベース間でデータを共有することです。Sybase では、Mobile Link、SQL Remote、Replication Server の 3 種類のレプリケーション・テクノロジを提供しています。

レプリケーション・メッセージ

SQL Remote または Replication Server では、パブリッシュするデータベースとサブスクリプションを作成するデータベース間で送信される通信内容を指します。メッセージにはデータを含み、レプリケーション・システムで必要なパススルー文、情報があります。

参照：

- 「レプリケーション」 849 ページ
- 「パブリケーションの更新」 844 ページ

レプリケーションの頻度

SQL Remote レプリケーションでは、リモート・ユーザに対する設定の 1 つで、パブリッシャの Message Agent がレプリケーション・メッセージを他のリモート・ユーザに送信する頻度を定義します。

参照：「レプリケーション」 849 ページ。

ロー・レベルのトリガ

変更されているローごとに一回実行するトリガです。

参照：

- 「トリガ」 843 ページ
- 「文レベルのトリガ」 857 ページ

ローカル・テンポラリ・テーブル

複合文を実行する間だけ存在したり、接続が終了するまで存在したりするテンポラリ・テーブルです。データのセットを 1 回だけロードする必要がある場合にローカル・テンポラリ・テーブルが便利です。デフォルトでは、COMMIT を実行するとローが削除されます。

参照：

- 「テンポラリ・テーブル」 842 ページ
- 「グローバル・テンポラリ・テーブル」 836 ページ

ロール

概念データベース・モデルで、ある視点からの関係を説明する動詞またはフレーズを指します。各関係は 2 つのロールを使用して表すことができます。"contains (A は B を含む)" や "is a member of (B は A のメンバ)" などのロールがあります。

ロールバック・ログ

コミットされていない各トランザクションの最中に行われた変更のレコードです。ROLLBACK 要求やシステム障害が発生した場合、コミットされていないトランザクションはデータベースから破棄され、データベースは前の状態に戻ります。各トランザクションにはそれぞれロールバック・ログが作成されます。このログは、トランザクションが完了すると削除されます。

参照：「トランザクション」 843 ページ。

ロール名

外部キーの名前です。この外部キーがロール名と呼ばれるのは、外部テーブルとプライマリ・テーブル間の関係に名前を指定するためです。デフォルトでは、テーブル名がロール名になります。ただし、別の外部キーがそのテーブル名を使用している場合、デフォルトのロール名はテーブル名に 3 桁のユニークな数字を付けたものになります。ロール名は独自に作成することもできます。

参照：「外部キー」 851 ページ。

ログ・ファイル

SQL Anywhere によって管理されているトランザクションのログです。ログ・ファイルを使用すると、システム障害やメディア障害が発生してもデータベースを回復させることができます。また、データベースのパフォーマンスを向上させたり、SQL Remote を使用してデータをレプリケートしたりする場合にも使用できます。

参照：

- 「トランザクション・ログ」 843 ページ
- 「トランザクション・ログ・ミラー」 843 ページ
- 「フル・バックアップ」 846 ページ

ロック

複数のトランザクションを同時に実行しているときにデータの整合性を保護する同時制御メカニズムです。SQL Anywhere では、2 つの接続によって同じデータが同時に変更されないようにするために、また変更処理の最中に他の接続によってデータが読み込まれないようにするために、自動的にロックが適用されます。

ロックの制御は、独立性レベルを設定して行います。

参照：

- 「独立性レベル」 857 ページ
- 「同時性 (同時実行性)」 857 ページ
- 「整合性」 854 ページ

ワーク・テーブル

クエリの最適化の最中に中間結果を保管する内部保管領域です。

一意性制約

NULL 以外のすべての値が重複しないことを要求するカラムまたはカラムのセットに対する制限です。テーブルには複数の一意性制約を指定できます。

参照：

- 「外部キー制約」 852 ページ
- 「プライマリ・キー制約」 846 ページ
- 「制約」 854 ページ

解析ツリー

クエリを代数で表現したものです。

外部キー

別のテーブルにあるプライマリ・キーの値を複製する、テーブルの 1 つ以上のカラムです。テーブル間の関係は、外部キーによって確立されます。

参照：

- 「プライマリ・キー」 846 ページ
- 「外部テーブル」 852 ページ

外部キー制約

カラムまたはカラムのセットに対する制約で、テーブルのデータが別のテーブルのデータとどのように関係しているかを指定するものです。カラムのセットに外部キー制約を加えると、それらのカラムが外部キーになります。

参照：

- 「制約」 854 ページ
- 「検査制約」 853 ページ
- 「プライマリ・キー制約」 846 ページ
- 「一意性制約」 851 ページ

外部ジョイン

テーブル内のすべてのローを保護するジョインです。SQL Anywhere では、左外部ジョイン、右外部ジョイン、全外部ジョインがサポートされています。左外部ジョインは JOIN 演算子の左側にあるテーブルのローを保護し、右側にあるテーブルのローがジョイン条件を満たさない場合には NULL を返します。全外部ジョインは両方のテーブルに含まれるすべてのローを保護します。

参照：

- 「ジョイン」 838 ページ
- 「内部ジョイン」 857 ページ

外部テーブル

外部キーを持つテーブルです。

参照：「外部キー」 851 ページ。

外部ログイン

リモート・サーバとの通信に使用される代替のログイン名とパスワードです。デフォルトでは、SQL Anywhere は、クライアントに代わってリモート・サーバに接続するときは、常にそのクライアントの名前とパスワードを使用します。外部ログインを作成することによって、このデフォルトを上書きできます。外部ログインは、リモート・サーバと通信するときに使用する代替のログイン名とパスワードです。

競合

リソースについて対立する動作のことです。たとえば、データベース用語では、複数のユーザがデータベースの同じローを編集しようとした場合、そのローの編集権についての競合が発生します。

競合解決

Mobile Link では、競合解決は 2 人のユーザが別々のリモート・データベースの同じローを変更した場合にどう処理するかを指定するロジックのことです。

検査制約

指定された条件をカラムやカラムのセットに課す制約です。

参照：

- 「制約」 854 ページ
- 「外部キー制約」 852 ページ
- 「プライマリ・キー制約」 846 ページ
- 「一意性制約」 851 ページ

検証

データベース、テーブル、またはインデックスについて、特定のタイプのファイル破損をテストすることです。

作成者 ID

Ultra Light の Palm OS アプリケーションでは、アプリケーションが作成されたときに割り当てられる ID のことです。

参照元オブジェクト

テーブルなどのデータベースの別のオブジェクトをオブジェクト定義が直接参照する、ビューなどのオブジェクトです。

参照：「外部キー」 851 ページ。

参照整合性

データの整合性、特に異なるテーブルのプライマリ・キー値と外部キー値との関係を管理する規則を厳守することです。参照整合性を備えるには、それぞれの外部キーの値が、参照テーブルにあるローのプライマリ・キー値に対応するようにします。

参照：

- 「プライマリ・キー」 846 ページ
- 「外部キー」 851 ページ

参照先オブジェクト

ビューなどの別のオブジェクトの定義で直接参照される、テーブルなどのオブジェクトです。

参照：「プライマリ・キー」 846 ページ。

識別子

テーブルやカラムなどのデータベース・オブジェクトを参照するときに使う文字列です。A ～ Z、a ～ z、0 ～ 9、アンダースコア (_)、アットマーク (@)、シャープ記号 (#)、ドル記号 ($) のうち、任意の文字を識別子として使用できます。

述部

条件式です。オプションで論理演算子 AND や OR と組み合わせて、WHERE 句または HAVING 句に条件のセットを作成します。SQL では、unknown と評価される述部が false と解釈されます。

照合

データベース内のテキストのプロパティを定義する文字セットとソート順の組み合わせのことです。SQL Anywhere データベースでは、サーバを実行しているオペレーティング・システムと言語によって、デフォルトの照合が決まります。たとえば、英語版 Windows システムのデフォルトの照合は 1252LATIN1 です。照合は、照合順とも呼ばれ、文字列の比較とソートに使用します。

参照：

- 「文字セット」 857 ページ
- 「コード・ページ」 837 ページ
- 「エンコード」 835 ページ

世代番号

Mobile Link では、リモート・データベースがデータをアップロードしてからダウンロード・ファイルを適用するようにするためのメカニズムのことです。

参照：「ファイルベースのダウンロード」 845 ページ。

制約

テーブルやカラムなど、特定のデータベース・オブジェクトに含まれた値に関する制約です。たとえば、一意性制約があるカラム内の値は、すべて異なっている必要があります。テーブルに、そのテーブルの情報と他のテーブルのデータがどのように関係しているのかを指定する外部キー制約が設定されていることもあります。

参照：

- 「検査制約」 853 ページ
- 「外部キー制約」 852 ページ
- 「プライマリ・キー制約」 846 ページ
- 「一意性制約」 851 ページ

整合性

データが適切かつ正確であり、データベースの関係構造が保たれていることを保証する規則を厳守することです。

参照：「参照整合性」 853 ページ。

正規化

データベース・スキーマを改善することです。リレーショナル・データベース理論に基づく規則に従って、冗長性を排除したり、編成を改良します。

正規表現

正規表現は、文字列内で検索するパターンを定義する、一連の文字、ワイルドカード、演算子です。

生成されたジョイン条件

自動的に生成される、ジョインの結果に対する制限です。キーとナチュラルの 2 種類があります。キー・ジョインは、KEY JOIN を指定したとき、またはキーワード JOIN を指定したが、CROSS、NATURAL、または ON を使用しなかった場合に生成されます。キー・ジョインの場合、生成されたジョイン条件はテーブル間の外部キー関係に基づいています。ナチュラル・ジョインは NATURAL JOIN を指定したときに生成され、生成されたジョイン条件は、2 つのテーブルの共通のカラム名に基づきます。

参照：

- 「ジョイン」 838 ページ
- 「ジョイン条件」 839 ページ

接続 ID

クライアント・アプリケーションとデータベース間の特定の接続に付けられるユニークな識別番号です。現在の接続 ID を確認するには、次の SQL 文を使用します。

```
SELECT CONNECTION_PROPERTY( 'Number' );
```

接続プロファイル

ユーザ名、パスワード、サーバ名などの、データベースに接続するために必要なパラメータのセットです。便宜的に保管され使用されます。

接続起動同期

Mobile Link のサーバ起動同期の 1 つの形式で、接続が変更されたときに同期が開始されます。

参照：「サーバ起動同期」 837 ページ。

相関名

クエリの FROM 句内で使用されるテーブルやビューの名前です。テーブルやビューの元の名前か、FROM 句で定義した代替名のいずれかになります。

抽出

SQL Remote レプリケーションでは、統合データベースから適切な構造とデータをアンロードする動作を指します。この情報は、リモート・データベースを初期化するときに使用されます。

参照：「レプリケーション」 849 ページ。

通信ストリーム

Mobile Link では、Mobile Link クライアントと Mobile Link サーバ間での通信にネットワーク・プロトコルが使用されます。

転送ルール

QAnywhere では、メッセージの転送を発生させる時期、転送するメッセージ、メッセージを削除する時期を決定する論理のことです。

統合データベース

分散データベース環境で、データのマスタ・コピーを格納するデータベースです。競合や不一致が発生した場合、データのプライマリ・コピーは統合データベースにあるとみなされます。

参照：

- 「同期」 856 ページ
- 「レプリケーション」 849 ページ

統合化ログイン

オペレーティング・システムへのログイン、ネットワークへのログイン、データベースへの接続に、同一のユーザ ID とパスワードを使用するログイン機能の 1 つです。

動的 SQL

実行される前に作成したプログラムによって生成される SQL です。Ultra Light の動的 SQL は、占有容量の小さいデバイス用に設計された変形型です。

同期

Mobile Link テクノロジを使用してデータベース間でデータをレプリケートする処理です。

SQL Remote では、同期はデータの初期セットを使ってリモート・データベースを初期化する処理を表すために特に使用されます。

参照：

- 「Mobile Link」 829 ページ
- 「SQL Remote」 832 ページ

同時性 (同時実行性)

互いに独立し、場合によっては競合する可能性のある 2 つ以上の処理を同時に実行することです。SQL Anywhere では、自動的にロックを使用して各トランザクションを独立させ、同時に稼働するそれぞれのアプリケーションが一貫したデータのセットを参照できるようにします。

参照：

- 「トランザクション」 843 ページ
- 「独立性レベル」 857 ページ

独立性レベル

あるトランザクションの操作が、同時に処理されている別のトランザクションの操作からどの程度参照できるかを示します。独立性レベルには 0 から 3 までの 4 つのレベルがあります。最も高い独立性レベルには 3 が設定されます。デフォルトでは、レベルは 0 に設定されています。SQL Anywhere では、スナップショット、文のスナップショット、読み込み専用文のスナップショットの 3 つのスナップショットの独立性レベルがあります。

参照：「スナップショット・アイソレーション」 839 ページ。

内部ジョイン

2 つのテーブルがジョイン条件を満たす場合だけ、結果セットにローが表示されるジョインです。内部ジョインがデフォルトです。

参照：

- 「ジョイン」 838 ページ
- 「外部ジョイン」 852 ページ

物理インデックス

インデックスがディスクに保存されるときの実際のインデックス構造です。

文レベルのトリガ

トリガ付きの文の処理が完了した後に実行されるトリガです。

参照：

- 「トリガ」 843 ページ
- 「ロー・レベルのトリガ」 850 ページ

文字セット

文字セットは記号、文字、数字、スペースなどから成ります。"ISO-8859-1" は文字セットの例です。Latin1 とも呼ばれます。

参照：

- 「コード・ページ」 837 ページ
- 「エンコード」 835 ページ
- 「照合」 854 ページ

文字列リテラル

文字列リテラルとは、一重引用符 (') で囲まれ、シーケンスで並べられた文字のことです。

論理インデックス

物理インデックスへの参照 (ポインタ) です。ディスクに保存される論理インデックス用のインデックス構造はありません。

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

う

え

お

か

き

く

け

こ

さ

し

す

せ

そ

な

に

ね

は

ひ

ふ

へ

ほ

ま

も

ゆ

よ

ら

り

る

れ

ろ

わ