iAnywhere SOLUTIONS™
A SYBASE COMPANY

QAnywhere™
ユーザーズ・ガイド

パート番号：DC20048-01-0902-01
改訂：2005 年 3 月

版権

Copyright © 2005 iAnywhere Solutions, Inc., Sybase, Inc. All rights reserved.

ここに記載されている内容を iAnywhere Solutions, Inc.、Sybase, Inc. またはその関連会社の書面による事前許可を得ずに電子的、機械的、手作業、光学的、またはその他のいかなる手段によっても複製、転載、翻訳することを禁じます。

Sybase、SYBASE のロゴ、Adaptive Server、AnswerBase、Anywhere、EIP、Embedded SQL、Enterprise Connect、Enterprise Portal、GainMomentum、iAnywhere、jConnect MASS DEPLOYMENT、Netimpact、ObjectConnect、ObjectCycle、OmniConnect、Open ClientConnect、Open ServerConnect、PowerBuilder、PowerDynamo、Powersoft、Quickstart Datamart、Replication Agent、Replication Driver、SQL Anywhere、SQL Central、SQL Remote、Support Plus、SWAT、Sybase IQ、Sybase System 11、Sybase WAREHOUSE、SyBooks、XA-Library は米国法人 Sybase, Inc. の登録商標です。Backup Server、Client-Library、jConnect for JDBC、MainframeConnect、Net-Gateway、Net-Library、Open Client、Open Client/Server、S-Designor、SQL Advantage、SQL Debug、SQL Server、SQL Server Manager、Sybase Central、Watcom、Web.SQL、XP Server は米国法人 Sybase, Inc. の商標です。

Certicom、MobileTrust および、SSL Plus は Certicom Corp. の商標です。Security Builder は Certicom Corp. の登録商標です。

ここに記載されている上記以外の社名および製品名は、各社の商標または登録商標の場合があります。

目次

はじめに vii
SQL Anywhere Studio のマニュアル viii
表記の規則 xii
詳細情報の検索／フィードバックの提供 xv

1 QAnywhere の概要 1
アプリケーション間メッセージング 2
QAnywhere の機能 4
QAnywhere アーキテクチャ 6
クイック・スタート 13

2 チュートリアル：QAnywhere アプリケーションのサンプル 15
チュートリアルの概要 16
レッスン 1：メッセージング対応 Mobile Link の起動 17
レッスン 2：クライアント・メッセージ・ストアの作成 21
レッスン 3：TestMessage アプリケーションの実行 23
レッスン 4：メッセージの送信 26
レッスン 5：TestMessage クライアントのソース・コードの概要 29
レッスン 6：JMS コネクタの起動 36
チュートリアルのクリーンアップ 39

3 QAnywhere メッセージングの設定 41
サーバ側コンポーネントの設定 42
クライアント側コンポーネントの設定 46
Push 通知の使用方法 53
JMS コネクタの使用方法 55
QAnywhere メッセージングと Mobile Link データ同期の統合 67
フェールオーバ・メカニズムの設定 69

4 QAnywhere クライアント・アプリケーションの作成 71
概要 72
クライアント・アプリケーション作成の概要 74
QAnywhere メッセージ・アドレスについて 75
QAnywhere クライアント API の初期化 77
QAManager のプロパティの設定 83
QAnywhere メッセージの送信 87
同期的なメッセージ受信 89
非同期的なメッセージ受信 91
サイズの大きなメッセージの読み込み 93
Push 通知とネットワーク・ステータスの変化の処理 94
トランザクション志向メッセージングの実装 96
QAnywhere の停止 99
QAnywhere アプリケーションの配備 100

5 QAnywhere Agent 101
QAnywhere Agent の構文 102

6 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 123
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 124
通信ストリームの暗号化 127
Mobile Link に対するパスワード認証の使用 129

7 QAnywhere 転送ルール 131
転送ルール 132
スケジュール構文 136
転送ルール変数 143
削除ルール 153

8 QAnywhere C++ API リファレンス 155
AcknowledgementMode クラス 156
MessageProperties クラス 158
MessageType クラス 161
QABinaryMessage クラス 162
QAError クラス 173
QAManager クラス 178
QAManagerBase クラス 182

QAManagerFactory クラス 198
QAMessage クラス 201
QAMessageListener クラス 217
QATextMessage クラス 218
QATransactionalManager クラス 222

9 iAnywhere.QAnywhere.Client ネームスペース 225
AcknowledgementMode 列挙 226
MessageProperties クラス 227
MessageType 列挙 233
QABinaryMessage クラス 234
QAException クラス 248
QAManager クラス 251
QAManagerBase クラス 260
QAManagerBase.MessageListener 委譲 284
QAManagerFactory クラス 285
QAMessage クラス 290
QAPropertyType 列挙 307
QATextMessage クラス 308
QATransactionalManager クラス 314

索引 323

はじめに

このマニュアルの内容

このマニュアルでは QAnywhere について説明します。QAnywhere を使用すると、従来のデスクトップ・クライアントやラップトップ・クライアント用のメッセージング・プラットフォームのほか、モバイル・クライアントや無線クライアント用のメッセージング・プラットフォームも定義できます。

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、Adaptive Server 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 Studio のマニュアル

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

SQL Anywhere Studio のマニュアル

SQL Anywhere Studio のマニュアルは、各マニュアルを 1 つの大きなヘルプ・ファイルにまとめたオンライン形式、マニュアル別の PDF ファイル、および有料の製本版マニュアルで提供されます。SQL Anywhere Studio のマニュアルは、次の分冊マニュアルで構成されています。

- **『SQL Anywhere Studio の紹介』** このマニュアルでは、SQL Anywhere Studio のデータベース管理と同期テクノロジの概要について説明します。また、SQL Anywhere Studio を構成する各部分について説明するチュートリアルも含まれています。
- **『SQL Anywhere Studio 新機能ガイド』** このマニュアルは、SQL Anywhere Studio のこれまでのリリースのユーザを対象としています。ここでは、製品の今回のリリースと以前のリリースで導入された新機能をリストし、アップグレード手順を説明しています。
- **『Adaptive Server Anywhere データベース管理ガイド』** このマニュアルでは、データベースおよびデータベース・サーバの実行、管理、設定について説明しています。
- **『Adaptive Server Anywhere SQL ユーザーズ・ガイド』** 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。
- **『Adaptive Server Anywhere SQL リファレンス・マニュアル』** このマニュアルは、Adaptive Server Anywhere で使用する SQL 言語の完全なリファレンスです。また、Adaptive Server Anywhere のシステム・テーブルとシステム・プロシージャについても説明しています。
- **『Adaptive Server Anywhere プログラミング・ガイド』** このマニュアルでは、C、C++、Java 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。Visual Basic や PowerBuilder などのツールのユーザは、それらのツールのプログラミング・インタフェースを使用できます。また、Adaptive Server Anywhere ADO.NET データ・プロバイダについても説明します。

- **『Adaptive Server Anywhere SNMP 拡張エージェント ユーザーズ・ガイド』** このマニュアルでは、Adaptive Server Anywhere SNMP 拡張エージェントを SNMP 管理アプリケーションとともに使用できるように設定して、Adaptive Server Anywhere データベースを管理できるようにする方法を説明します。

- **『Adaptive Server Anywhere エラー・メッセージ』** このマニュアルでは、Adaptive Server Anywhere エラー・メッセージの完全なリストを、その診断情報とともに説明します。

- **『SQL Anywhere Studio セキュリティ・ガイド』** このマニュアルでは、Adaptive Server Anywhere データベースのセキュリティ機能について説明します。Adaptive Server Anywhere 7.0 は、米国政府から TCSEC (Trusted Computer System Evaluation Criteria) の C2 セキュリティ評価を授与されています。このマニュアルには、Adaptive Server Anywhere の現在のバージョンを、C2 基準を満たした環境と同等の方法で実行することを望んでいるユーザにとって役に立つ情報が含まれています。

- **『Mobile Link 管理ガイド』** このマニュアルでは、モバイル・コンピューティング用の Mobile Link データ同期システムについてあらゆる角度から説明します。このシステムによって、Oracle、Sybase、Microsoft、IBM の単一データベースと、Adaptive Server Anywhere や Ultra Light の複数データベースの間でのデータ共有が可能になります。

- **『Mobile Link クライアント』** このマニュアルでは、Adaptive Server Anywhere リモート・データベースと Ultra Light リモート・データベースの設定を行い、これらを同期させる方法について説明します。

- **『Mobile Link チュートリアル』** このマニュアルには、Mobile Link テクノロジについて学習するためのチュートリアルがいくつか用意されています。

- **『Mobile Link サーバ起動同期ユーザーズ・ガイド』** このマニュアルでは、Mobile Link のサーバによって開始される同期について説明します。サーバによって開始される同期とは、統合データベースから同期の開始を可能にする Mobile Link の機能です。

- **『QAnywhere ユーザーズ・ガイド』** このマニュアルでは、Mobile Link QAnywhere について説明します。Mobile Link QAnywhere は、従来のデスクトップ・クライアントやラップトップ・クライアントだけでなく、モバイル・クライアントや無線クライアント用のメッセージング・アプリケーションの開発と展開を可能にするメッセージング・プラットフォームです。

- **『Mobile Link およびリモート・データ・アクセスの ODBC ドライバ』** このマニュアルでは、Mobile Link 同期サーバから、または Adaptive Server Anywhere リモート・データ・アクセスによって、Adaptive Server Anywhere 以外の統合データベースにアクセスするための ODBC ドライバの設定方法について説明します。

- **『SQL Remote ユーザーズ・ガイド』** 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、あらゆる角度から説明します。このシステムによって、Adaptive Server Anywhere または Adaptive Server Enterprise の単一データベースと Adaptive Server Anywhere の複数データベースの間で、電子メールやファイル転送などの間接的リンクを使用したデータ共有が可能になります。

- **『SQL Anywhere Studio ヘルプ』** このマニュアルには、Sybase Central や Interactive SQL、その他のグラフィカル・ツールに関するコンテキスト別のヘルプが含まれています。これは、製本版マニュアル・セットには含まれていません。

- **『Ultra Light データベース・ユーザーズ・ガイド』** このマニュアルは、Ultra Light 開発者を対象としています。ここでは、Ultra Light データベース・システムの概要について説明します。また、すべての Ultra Light プログラミング・インタフェースに共通する情報を提供します。

- **Ultra Light のインタフェースに関するマニュアル** 各 Ultra Light プログラミング・インタフェースには、それぞれに対応するマニュアルを用意しています。これらのインタフェースは、RAD (

ラピッド・アプリケーション開発) 用の Ultra Light コンポーネントとして提供されているものと、C、C++、Java 開発用の静的インタフェースとして提供されているものがあります。

このマニュアル・セットの他に、PowerDesigner と InfoMaker には、独自のオンライン・マニュアル (英語版) がそれぞれ用意されています。

マニュアルの形式

SQL Anywhere Studio のマニュアルは、次の形式で提供されています。

- **オンライン・マニュアル** オンライン・マニュアルには、SQL Anywhere Studio の完全なマニュアルがあり、SQL Anywhere ツールに関する印刷マニュアルとコンテキスト別のヘルプの両方が含まれています。オンライン・マニュアルは、製品のメンテナンス・リリースごとに更新されます。これは、最新の情報を含む最も完全なマニュアルです。

  Windows オペレーティング・システムでオンライン・マニュアルにアクセスするには、[スタート] ー [プログラム] ー [SQL Anywhere 9] ー [オンライン・マニュアル] を選択します。オンライン・マニュアルをナビゲートするには、左ウィンドウ枠で HTML ヘルプの目次、索引、検索機能を使用し、右ウィンドウ枠でリンク情報とメニューを使用します。

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

- **PDF 版マニュアル** SQL Anywhere の各マニュアルは、Adobe Acrobat Reader で表示できる PDF ファイルで提供されています。

  PDF 版マニュアルは、オンライン・マニュアルまたは Windows の [スタート] メニューから利用できます。

- **製本版マニュアル** 製本版マニュアルをご希望の方は、ご購入いただいた販売代理店または弊社営業担当までご連絡ください。

表記の規則

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

SQL 構文の表記規則

SQL 構文の表記には、次の規則が適用されます。

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

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

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

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

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

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

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

- **オプション部分** 文のオプション部分は角カッコで囲みます。

  RELEASE SAVEPOINT [*savepoint-name*]

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

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

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

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

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

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

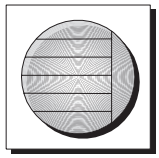
  QUOTES オプションを使用する場合は、ON または OFF のどちらかを選択する必要があります。角カッコと中カッコは入力しないでください。

グラフィック・アイコン

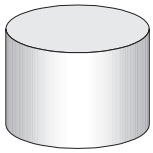
このマニュアルでは、次のアイコンを使用します。

- クライアント・アプリケーション

- Sybase Adaptive Server Anywhere などのデータベース・サーバ



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



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

- プログラミング・インタフェース

詳細情報の検索／フィードバックの提供

詳細情報の検索

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

ご質問がある場合や支援が必要な場合は、次に示す iAnywhere Solutions 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 Studio バージョンのビルド番号を明記し、現在発生している問題について詳しくお知らせくださいますようお願いいたします。バージョン情報は、コマンド・プロンプトで **dbeng9 -v** と入力して確認できます。

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

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

ニュースグループに関するお断り

iAnywhere Solutions は、ニュースグループ上に解決策、情報、または意見を提供する義務を負うものではありません。また、システム・オペレータ以外のスタッフにこのサービスを監視させて、操作状況や可用性を保証する義務もありません。

iAnywhere Solutions のテクニカル・アドバイザとその他のスタッフは、時間のある場合にかぎりニュースグループでの支援を行います。こうした支援は基本的にボランティアで行われるため、解決策や情報を定期的に提供できるとはかぎりません。支援できるかどうかは、スタッフの仕事量に左右されます。

フィードバック

このマニュアルに関するご意見、ご提案、フィードバックをお寄せください。

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

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

第 1 章

QAnywhere の概要

この章の内容

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

アプリケーション間メッセージング

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。メッセージングは、さまざまな状況で利用できる便利なアプリケーション間通信メカニズムであり、以下のことを可能にします。

- 随時接続環境での通信の実現

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

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

- ネットワークに依存しない通信の実現

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

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

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

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

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

QAnywhere の機能

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

- **プログラミング API** オブジェクト指向の QAnywhere API は、Windows デスクトップ向けや Windows CE デバイス向けのメッセージング・アプリケーションを構築するためのインフラストラクチャとして使用できます。
- **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送信が可能になるまで、キュー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、およびリレーショナル・データベースのその他の利点が得られます。

  Adaptive Server 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 128 ビット暗号化キーを使用して暗号化されます。また、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** メッセージは、LZ77 アルゴリズム (デフレーション・アルゴリズムの 1 つ) を使用して圧縮してから格納できます。これによって、広く普及している LZW アルゴリズムでときどき見られる圧縮データの膨張が軽減されます。

- **認証**　組織に導入済みの別のアプリケーションで提供されている既存の認証サービスを使用して、ユーザを認証できます。
- **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 同期サーバを実行し、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。
- **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。
- **メッセージ・ストア管理ルール**　メッセージの送信を実行するタイミングを指定したり、メッセージ・ストア内のメッセージの持続性をカスタマイズしたりするためのルールを作成できます。
- **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重複なく配信されることが保証されます。

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

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

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

  「簡単なメッセージング・シナリオ」7 ページを参照してください。

- **Push 通知によるメッセージング** QAnywhere クライアント間でメッセージを交換する場合です。このシナリオでは、QAnywhere サーバが、クライアント・メッセージ・ストアとサーバ・メッセージ・ストアの間のメッセージ転送を開始できます。

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

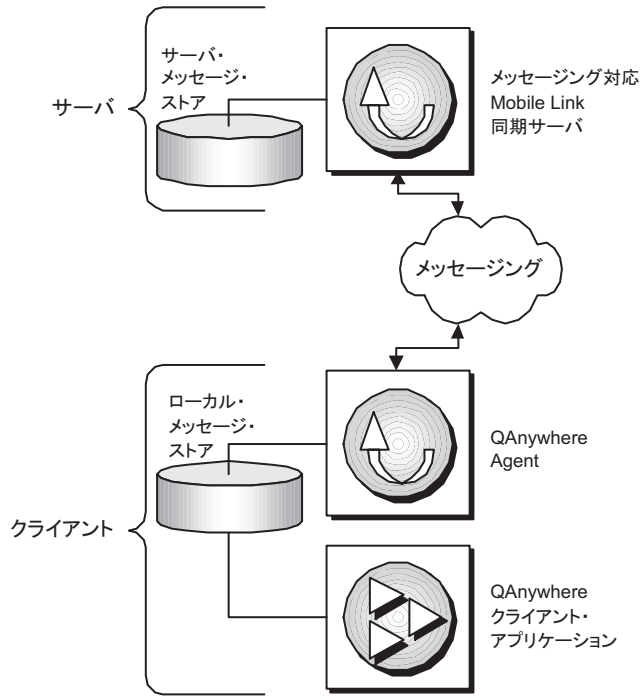
- **外部メッセージング・システムとのメッセージング** QAnywhere クライアント間や、JMS プロバイダを提供する外部システム (BEA WebLogic や Sybase EAServer など) との間でメッセージ交換を行う場合です。

  「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」11 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。



このアーキテクチャは、以下のコンポーネントで構成されています。

- **サーバ・メッセージ・ストア** サーバ側ではメッセージがリレーショナル・データベースに格納されます。このデータベースは、Mobile Link 統合データベースとして設定する必要があります。サポートされている統合データベースのいずれか (Adaptive Server Anywhere、Adaptive Server Enterprise、Microsoft SQL Server、DB2、または Oracle) を使用できます。

- **クライアント・メッセージ・ストア** 各クライアントのメッセージもリレーショナル・データベースに格納されます。使用されているデータベースは Adaptive Server Anywhere です。
- **メッセージング対応 Mobile Link 同期サーバ** 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 メッセージ転送を管理するには、Mobile Link 同期サーバを、メッセージング機能を有効にして起動する必要があります。そのためには、-m コマンド・ライン・オプションを指定して Mobile Link 同期サーバを起動します。

  詳細については、「QAnywhere メッセージング用 Mobile Link 同期サーバの起動」44 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。

  詳細については、「QAnywhere Agent の実行」49 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++ API または QAnywhere .NET API を使用して作成されたアプリケーションによって、メッセージの送受信のために関数が呼び出されます。

  QAnywhere API を使用してアプリケーションを作成する方法の詳細については、「QAnywhere クライアント・アプリケーションの作成」71 ページを参照してください。

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

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」50 ページを参照してください。

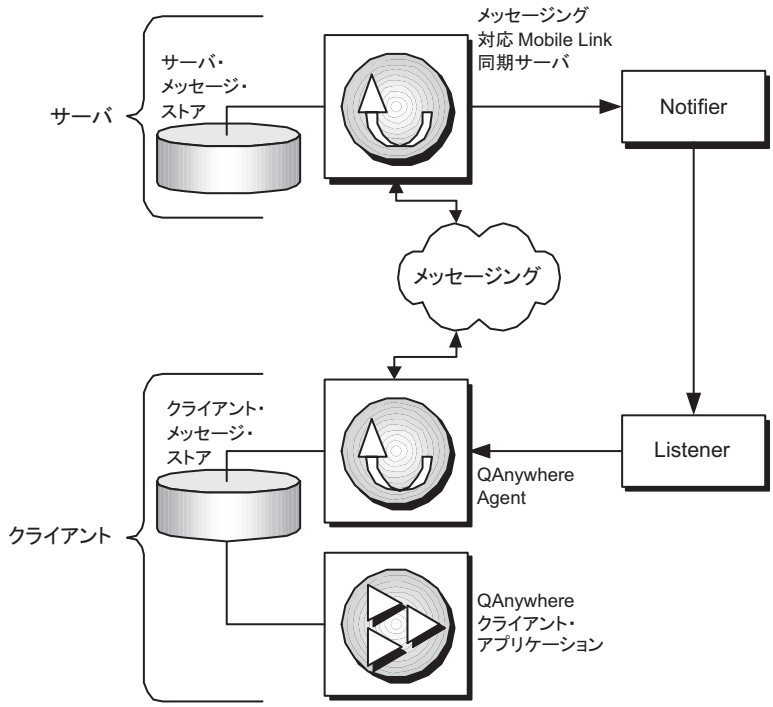
Push 通知によるメッセージングのシナリオ

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。この通知は、サーバに届いているメッセージを取り出すメッセージ転送を開始するようにクライアントに対して要求する役割を果たします。

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、スケジュール転送などを使用してクライアント側からメッセージ転送を開始する必要があります。

Push 通知によるメッセージングのアーキテクチャは、「簡単なメッセージング・シナリオ」7 ページで説明したアーキテクチャを拡張したものです。これは次の図のようになります。



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」7 ページに追加されたコンポーネントは、以下のとおりです。

- **QAnywhere Notifier** QAnywhere Notifier は Mobile Link 同期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。

- **QAnywhere Listener** QAnywhere 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。

参照

詳細については、次の項を参照してください。

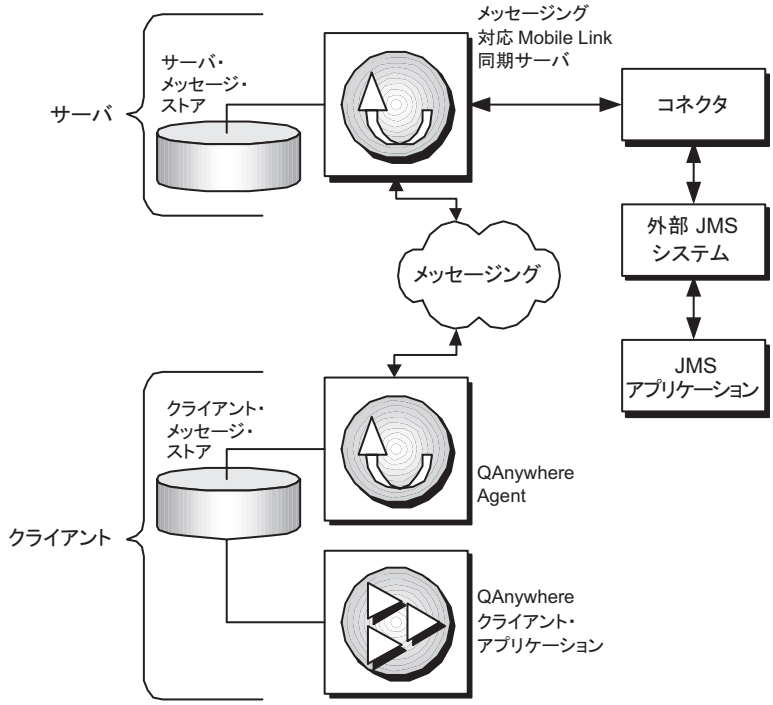
- 「Push 通知の使用方法」53 ページ
- 「非同期的なメッセージ受信」91 ページ
- 『Mobile Link サーバ起動同期ユーザーズ・ガイド』>「サーバ起動同期の概要」

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、固有のアドレスと構成を持ちます。

外部メッセージング・システムを含むメッセージング・アーキテクチャは、「簡単なメッセージング・シナリオ」7 ページで説明したアーキテクチャを拡張したものです。これは次の図のようになります。



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

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

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

参照

詳細については、次の項を参照してください。

- 「JMS コネクタの使用方法」55 ページ
- 「レッスン 6：JMS コネクタの起動」36 ページ

クイック・スタート

QAnywhere メッセージング・システムを設定して起動するには、以下の手順を実行します。

❖ 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」42 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 同期サーバを起動し、サーバ・メッセージ・ストアへの接続を開始します。

   「QAnywhere メッセージング用 Mobile Link 同期サーバの起動」44 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 Adaptive Server 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」46 ページを参照してください。

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

   「JMS コネクタの使用方法」55 ページを参照してください。

5. 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」71 ページを参照してください。

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

「QAnywhere Agent の実行」49 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：QAnywhere アプリケーションのサンプル」15 ページ
- SQL Anywhere Studio インストール・ディレクトリ内の *Samples¥QAnywhere* に、サンプル・アプリケーションがあります。

第 2 章

チュートリアル：QAnywhere アプリケーションのサンプル

この章の内容

このチュートリアルでは、TestMessage という名前のサンプル・クライアント・アプリケーションを使用して、QAnywhere の機能を解説します。QAnywhere アプリケーションは PDA などの多くのデバイスで動作するので、さまざまなデバイスにアプリケーション間メッセージングを展開するために使用できます。ただし、このチュートリアルでは、QAnywhere の機能を説明することが目的なので、Windows コンピュータでクライアントを実行します。

チュートリアルの概要

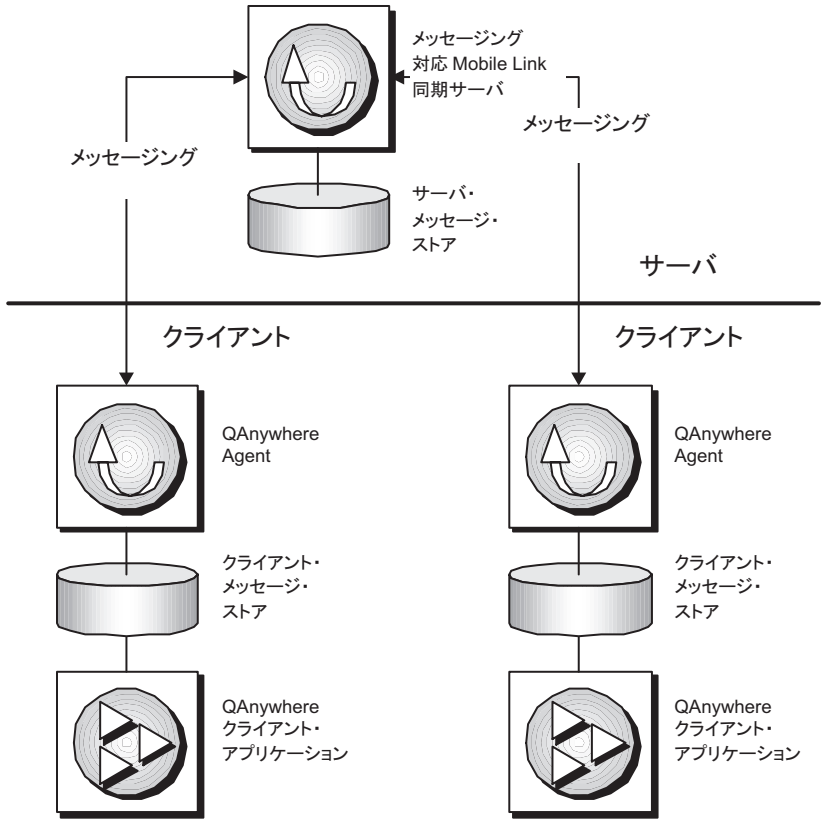
TestMessage は QAnywhere クライアント・アプリケーションのサンプルです。このアプリケーションは、QAnywhere を使用してユーザ独自のメッセージ・クライアント・アプリケーションを作成する方法を示しています。TestMessage では、単一のクライアント間インタフェースを使用して、メッセージを送信、受信、表示します。このサンプルでは、QAnywhere メッセージングの機能をわかりやすく示すため、人間が読めるテキスト・メッセージを使用しています。ただし、QAnywhere はテキスト以外のメッセージも扱うことができます。QAnywhere は、多数のクライアントの間でメッセージ・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 NT/2000/XP システム向けに書かれています。これらのプラットフォームはデモ用に便利です。QAnywhere は、Windows CE デバイスで動作するアプリケーションの作成にも使用できます。

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 同期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 Adaptive Server 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、一連の Mobile Link システム・テーブルだけです。これらのシステム・テーブルは、Adaptive Server Anywhere データベースの作成時に自動的に追加されます。また、これらのシステム・テーブルは、Mobile Link 統合データベースとして設定されているサポート対象データベースのすべてに存在します。

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。また、既存のデータ管理システムと同期システムに、メッセージングを簡単に統合できます。

通常、QAnywhere メッセージングは、複数の異なるマシン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 つのマシンで実行されます。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 同期サーバは、-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 Adaptive Server Anywhere 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 Studio 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と、Mobile Link 同期サーバをメッセージングに対応した状態で起動できます。

❖ メッセージング・サーバを起動する

1　Windows で、[スタート] ー [プログラム] ー [SQL Anywhere 9] ー [Mobile Link] ー [メッセージ・サンプル付き Mobile Link] を選択します。

または、コマンド・プロンプトで、SQL Anywhere Studio インストール環境の *Samples¥QAnywhere¥server* サブディレクトリに移動し、次のコマンドを入力します。

```
dbmlsrv9 -m qanyserv.props -c "dsn=QAnywhere 9.0
Sample" -vcrs -zu+
```

この例では、以下に示す dbmlsrv9 のオプションを使用しています。

オプション	説明
-m	メッセージングを有効にします。この例では、メッセージング・プロパティの設定サンプルが格納されているファイル *qanyserv.props* も同時に指定しています。 『Mobile Link 管理ガイド』>「-m オプション」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 9.0 Sample が接続文字列に含まれています。 『Mobile Link 管理ガイド』>「-c オプション」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。 『Mobile Link 管理ガイド』>「-v オプション」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段階で使用すると便利ですが、運用環境では通常使用しません。 『Mobile Link 管理ガイド』>「-zu オプション」を参照してください。

2 Mobile Link 同期サーバ・ウィンドウを画面左側に移動します。これは、このチュートリアルのサーバ・マシンになります。

Mobile Link 同期サーバが起動し、このサーバのコンソール・ウィンドウに「要求処理の準備ができました」というメッセージが表示されたら、次のレッスンに進んでください。

詳細情報

詳細については、次の項を参照してください。

- 「QAnywhere メッセージング用 Mobile Link 同期サーバの起動」44 ページ
- 『Mobile Link 管理ガイド』>「-m オプション」
- 「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」7 ページ

レッスン 2：クライアント・メッセージ・ストアの作成

背景

QAnywhere Agent は、各クライアント・デバイス上で動作するコンポーネントです。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移動して、メッセージの転送を管理します。クライアント・メッセージ・ストアは、Adaptive Server Anywhere データベースです。

QAnywhere Agent は、デバイスがオンの間は常に動作し続けるように設計されています。一方、QAnywhere アプリケーションは、いつでも起動または停止できます。

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マシン上で動作します。ただし、このレッスンでは、クライアントとサーバを同じマシンに作成します。

このレッスンでは、クライアント・メッセージ・ストアを作成します。

アクティビティ

❖ クライアント・メッセージ・ストアを作成する

1　Adaptive Server Anywhere データベースを作成します。

データベースの作成方法はいくつかありますが、このレッスンでは dbinit コマンド・ライン・ユーティリティを使用します。適切なディレクトリ (*c:¥sample¥qanywhere* など) に移動し、次のように入力します。

```
dbinit -I clientstore.db
```

dbinit -I を使用すると、データベースのサイズが小さくなります。これは、多くのリモート・デバイスに適しています。

2　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。

次のように入力します。

```
qaagent -si -c "DBF=clientstore.db" -id MyclientID
```

この例では、以下に示すオプションを使用しています。

オプション	説明
-si	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 Adaptive Server Anywhere データベースを初期化します。 「-si オプション」118 ページを参照してください。
-c	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。この例では、データベース・ファイル名 *clientstore.db* が接続文字列内に指定されています。 「-c オプション」105 ページを参照してください。
-id	クライアント・メッセージ・ストアの ID を指定します。クライアント・メッセージ・ストアへの接続時には、クライアント・メッセージ・ストアの ID をその都度指定する必要があります。 「-id オプション」107 ページを参照してください。

QAnywhere Agent は、クライアント・メッセージ・ストアを初期化したあと、自動的に停止します。

詳細情報

クライアント・メッセージ・ストアの作成方法の詳細については、以下の項を参照してください。

- 『ASA データベース管理ガイド』>「初期化ユーティリティのオプション」
- 「クライアント・メッセージ・ストアの設定」46 ページ

レッスン 3：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マシン上で動作します。

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 4 では、このメッセージ・ストアを使用して、レッスン 2 で作成したクライアント・メッセージ・ストアにメッセージを送信します。

アクティビティ

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

1. Windows で、[スタート] － [プログラム] － [SQL Anywhere 9] － [QAnywhere] － [QAnywhere エージェント] を選択します。

   TestMessage サンプルのクライアント・メッセージ・ストアが開始されます。

2. QAnywhere Agent ウィンドウに、クライアント・メッセージ・ストア ID が表示されます。この ID は、デフォルトではご使用のマシンの名前になります。この ID をメモしてください。

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

❖ TestMessage を起動する

1. Windows で、[スタート] － [プログラム] － [SQL Anywhere 9] － [QAnywhere] － [TestMessage サンプル・アプリケーション] を選択します。

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

2 TestMessage のウィンドウを、QAnywhere Agent と同じように、画面右側に移動します。これらは、両方ともクライアント上のコンポーネントです。

3 適切な名前を入力して、メッセージ・キューを確認します。

TestMessage サンプル・アプリケーションで、[Tools] － [Options] を選択します。適切な名前を入力します。これは、メッセージが送信されるときに表示される名前です。この名前にはスペースが含まれていてもかまいません。

キュー名として **testmessage** と指定されていることが確認できます。この名前は変更しないでください。

説明

Mobile Link 同期サーバのウィンドウではメッセージがスクロール表示されます。これは、QAnywhere Agent が、サーバ・メッセージ・ストアとクライアント・メッセージ・ストアの間で定期的にメッセージを転送していることを示しています。

一般に、運用環境では、このチュートリアルで行われるように頻繁にメッセージを転送する必要はありません。QAnywhere Agent のメッセージ・モニタリングの方法を設定するには、コマンド・ラインでメッセージ転送ポリシーを設定します。デフォルトのポリシー設定は、**スケジュール済み**です。この設定では、定期的にメッセージが転送されます。転送間隔が指定されなかった場合、デフォルトの 10 秒間隔になります。他にも、**自動** (クライアント・メッセージ・ストアにメッセージが登録されたら送信する)、**オンデマンド** (アプリケーションから指示されたときにだけメッセージを送信する)、**カスタム** (さらに複雑な転送動作を一連のルールとしてルール・ファイルに記述する) の各設定があります。

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 が動作しているマシンの名前です。QAnywhere Agent は、各マシンに 1 つだけ必要です。複数のメッセージング・アプリケーションが動作するマシンでも、QAnywhere Agent は 1 つでかまいません。各アプリケー

ションは、複数のキューで受信することができます。ただし、各キューは 1 つのアプリケーションに割り当てられている必要があります。

詳細情報

- 「QAnywhere Agent の実行」49 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」50 ページ
- 「QAnywhere Agent の構文」102 ページ
- 「QAnywhere 転送ルール」131 ページ
- 「QAnywhere クライアント・アプリケーションの作成」71 ページ
- QAnywhere のサンプルは、SQL Anywhere Studio インストール・ディレクトリ内の *Samples¥QAnywhere* サブディレクトリにあります。

レッスン 4：メッセージの送信

背景

TestMessage サンプル・アプリケーションにはクライアント・メッセージ・ストアが含まれています。このメッセージ・ストアは、レッスン 3 で開始しました。また、レッスン 2 では、クライアント・メッセージ・ストアを作成しました。このレッスンでは、TestMessage サンプル・アプリケーションから、レッスン 2 で作成したクライアント・メッセージ・ストアにメッセージを送信します。

アクティビティ

❖ TestMessage からメッセージを送信する

1 TestMessage サンプル・アプリケーションで、[Message] － [New] を選択します。[New Message] ウィンドウが表示されます。

2 [To] フィールドに、MyclientID と入力します。この ID は、レッスン 2 で作成したクライアント・メッセージ・ストアに対して指定した ID です。

QAnywhere は、この ID の後ろに、[Options] ダイアログで指定されたキュー名を付けて、メッセージ・アドレスを作成します。[Options] ダイアログでキュー名が指定されなかった場合、TestMessage は、testmessage というキュー名をアドレスに付けます。

3 [Subject] フィールドと [Massage] フィールドにサンプル・テキストを入力して、[Send] をクリックします。

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

4 TestMessage と QAnywhere Agent を停止します。10 秒以上待ってから停止してください。

TestMessage のウィンドウで [File] － [Exit] をクリックします。

QAnywhere Agent のウィンドウで、[シャットダウン] をクリックします。

5　QAnywhere Agent を起動し、レッスン 2 で作成したクライアント・メッセージ・ストアに接続します。

そのためには、レッスン 2 でクライアント・メッセージ・ストアを作成したディレクトリに移動して、次のように入力します。

```
qaagent -c "DBF=clientstore.db;eng=qanywhere" -id
MyclientID
```

この例では、以下に示すオプションを使用しています。

オプション	説明
-c	この例では、レッスン 2 で作成したクライアント・メッセージ・ストア *clientstore.db* に接続するための接続文字列を指定しています。ここで指定されている eng=qanywhere は、TestMessage サンプル・アプリケーションが、データベース・サーバ qanywhere を使用したメッセージ・ストアに接続することを示しています。
-id	ID には、MyclientID を指定する必要があります。これは、レッスン 2 でクライアント・メッセージ・ストアに付けた ID です。

6　TestMessage を開始します。

Windows で、[スタート] － [プログラム] － [SQL Anywhere 9] － [QAnywhere] － [TestMessage サンプル・アプリケーション] を選択します。

入力したメッセージが、TestMessage のウィンドウに表示されます (メッセージが表示されない場合、手順 4 でアプリケーションを停止するのが早すぎたことが原因と考えられます)。

7　メッセージを読みます。

メッセージを選択すると、その内容が下部にあるウィンドウ枠に表示されます。

次回 TestMessage を開始したときには、このメッセージは表示されません。これは、読まれたメッセージを削除するように TestMessage が設定されているからです。このデフォルトの動作は、削除ルールを指定することで変更できます。

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 には C++ API と Microsoft .NET API の両方が用意されています。後者は、Microsoft Visual Studio .NET を使用して Visual Basic .NET、C#、C++ アプリケーションを開発するときに使用します。

詳細情報

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・アドレスについて」75 ページ
- 「QAnywhere メッセージの送信」87 ページ
- 「削除ルール」153 ページ

レッスン 5：TestMessage クライアントのソース・コードの概要

背景

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

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

TestMessage のソース・コードは、SQL Anywhere Studio インストール環境のサブディレクトリ *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 および Windows XP 用に次のバージョンが用意されています。

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

Pocket PC 用に次のバージョンが用意されています。

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

必要なソフトウェア

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

C# バージョンまたは Visual Basic .NET バージョンのソースの解説

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

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

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

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

2. ソリューション・エクスプローラが表示されていることを確認します。

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

3. [ソース ファイル] フォルダの内容を確認します。

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

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

5. インクルードされているネームスペースを調べます。

   QAnywhere アプリケーションは、いずれも iAnywhere.QAnywhere.Client ネームスペースをインクルードしている必要があります。このネームスペースを定義しているアセンブリは、SQL Anywhere Studio インストール環境の *ce* または *win32*

サブディレクトリに、DLL ファイル *iAnywhere.QAnywhere.Client.dll* として提供されています。ユーザ独自のプロジェクトでは、この DLL を参照先として追加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. MessageList_Load メソッドを調べます。

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

   - QAManager オブジェクトを作成する。

     ```
     _qaManager =

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

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

     QAnywhere には、QAManager と QATransactionalManager の 2 種類の Manager が用意されています。トランザクション志向の Manager 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 onMessage 関数は、addMessage 関数を呼び出します。この関数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)
          {
              if( addMessage( msg ) ) {
                  String info_msg =
_resources.GetString( "MessageReceived"
);
                  MessageBox.Show( this,
info_msg, "Test Message",
                     MessageBoxButtons.OK,
MessageBoxIcon.Information );
              }
          }
```

- MessageListener を宣言する。

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

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

メッセージ・リスナと通知リスナ

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

---

7. 同じファイルにある startReceiver メソッドを調べます。

   このメソッドはメッセージ・リスナをキューに割り当てます。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 を使用すると、メッセージを**非同期的に**受信できます。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。

このメソッドによって、以下の初期化処理が実行されます。

- QAManager オブジェクトをオープンして開始する。

```
_qaManager.Open(

AcknowledgementMode.EXPLICIT_ACKNOWLEDGEM
ENT );
 _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。

```
_mainWindow.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.GetReceiveQueueName(),
      _receiveListener );
```

  Options.GetReceiveQueueName() 関数は、**testmessage** という文字列を返します。これは、TestMessage アプリケーションの [Options] ダイアログで設定された TestMessage アプリケーション用のキューです。

8. 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と呼び出され、メッセージのプロパティ (ReplyToAddress や PreferredName など) と送信時刻 (Timestamp) 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。次のコードでは、受信メッセージを QATestMessage オブジェクトにキャストし、メッセージの ReplyToAddress を取得しています。

```
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。

9. 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。

10. sendMessage 関数を調べます。

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

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

    ```
    qa_manager = MessageList.GetQAManager();
    msg = qa_manager.CreateTextMessage();
    msg.ReplyToAddress =
    Options.GetReceiveQueueName();
    ```

    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 to は宛先アドレスです。宛先アドレスは、この関数の引数として渡されます。

    ```
    to = BuildQueue( to );
    qa_manager.PutMessage( to, msg );
    ```

詳細情報

詳細については、次の項を参照してください。

- 「QAnywhere C++ API リファレンス」155 ページ
- 「iAnywhere.QAnywhere.Client ネームスペース」225 ページ

- 「QAnywhere クライアント・アプリケーションの作成」71 ページ
- TestMessage のサンプルは、SQL Anywhere Studio インストール・ディレクトリ内の *Samples¥QAnywhere* サブディレクトリにあります。

レッスン 6：JMS コネクタの起動

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。

必要なソフトウェア

このレッスンでは、JMS プロバイダにアクセスできることと、JMS プロバイダの設定方法についての基本的な知識があることが必要です。また、JDK バージョン 1.3.1 以降と、JMS プロバイダの JMS クライアントが使用する JAR ファイルも必要です。

アクティビティ

❖ JMS プロバイダを準備する

1. JMS サーバを起動します。

   詳細については、JMS サーバのマニュアルを参照してください。

2. JMS サーバ内に、qa_testmessage と qa_receive の 2 つのキューを作成します。

   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。キューを作成した後、JMS サーバの再起動が必要になる場合があります。

❖ QAnywhere クライアントとサーバ・コンポーネントを起動する

1. QAnywhere コネクタ・プロパティ・ファイルを作成します。

   SQL Anywhere Studio インストール環境のサブディレクトリ *Samples¥QAnywhere¥connectors* に、コネクタ・プロパティ・ファイルのサンプルがあります。

2. *Samples¥QAnywhere¥server¥qanyserv.props* ファイル内に、QAnywhere コネクタ・プロパティ・ファイルを指定します。

   このサンプル・ファイルには、サンプルのコネクタ・プロパティ・ファイルが指定されている行がコメントとして含まれています。適切な行のコメントを解除してください。

3　「レッスン 1：メッセージング対応 Mobile Link の起動」17 ページの説明に従って、Mobile Link 同期サーバをメッセージングに対応した状態で起動します。

4　「レッスン 2：クライアント・メッセージ・ストアの作成」21 ページの説明に従って、QAnywhere Agent を起動します。

5　「レッスン 3：TestMessage アプリケーションの実行」23 ページの説明に従って、TestMessage を起動します。

❖ TestMessage クライアントを起動する

1　コマンド・プロンプトで、*Samples¥QAnywhere¥connectors¥JMS¥TestMessage* に移動して、次のように入力します。

```
java -cp .;JMS-client-jar-files
ianywhere.message.samples.TestMessage
```

ここで、*JMS-client-jar-files* は、JMS サーバが必要とする jar ファイルのリストです。各ファイル名の間はセミコロンで区切ります。詳細については、JMS サーバのマニュアルを参照してください。Sybase EAServer の場合、上のコマンドは次のようになります。

```
java -cp .; path¥easclient.jar;path¥easj2ee.jar
ianywhere.message.samples.TestMessage
```

ここで、*path* は、jar ファイルが存在するロケーションです。

2　JMS 用の TestMessage のウィンドウを、画面右側にすでに置かれている TestMessage のウィンドウの下に移動します。

3　JMS 用の TestMessage のウィンドウで、[Message] － [New] を選択します。

[New Message] ウィンドウが表示されます。

4　[To] フィールドに、レッスン 2 でメモしたクライアント・メッセージ・ストア ID を入力します。

5 [Subject] フィールドと [Massage] フィールドにサンプル・テキストを入力して、[Send] をクリックします。

少し待つと、前に開始した TestMessage インスタンスによってメッセージが受信されたことを知らせるメッセージが、メッセージ・ボックスに表示されます。

6 もう一方の TestMessage インスタンスに切り替えて、メッセージを受信します。

詳細情報

詳細については、次の項を参照してください。

- 「JMS コネクタの使用方法」55 ページ
- 「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」11 ページ
- JMS プロパティ・ファイルのサンプルとその他の JMS 用サンプルは、SQL Anywhere Studio インストール・ディレクトリ内の *Samples¥QAnywhere¥connectors* サブディレクトリにあります。

チュートリアルのクリーンアップ

TestMessage、QAnywhere Agent、Mobile Link 同期サーバをそれぞれ停止します。

第 3 章

QAnywhere メッセージングの設定

この章の内容

この章では、QAnywhere メッセージングを設定し、実行する方法について説明します。

QAnywhere は、Mobile Link 同期を使用してメッセージを転送します。この章では、Mobile Link 同期サーバをメッセージングに対応できるように設定し、実行する方法について説明します。

サーバ側コンポーネントの設定

❖ QAnywhere サーバ側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」42 ページを参照してください。

2 -m オプションを指定して dbmlsrv9 を起動し、サーバ・メッセージ・ストアへの接続を開始します。

「QAnywhere メッセージング用 Mobile Link 同期サーバの起動」44 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 ID をサーバ・メッセージ・ストアに追加します。

「クライアント・メッセージ・ストア ID の追加」45 ページを参照してください。

サーバ・メッセージ・ストアの設定

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

サーバ・メッセージ・ストアは Mobile Link 統合データベースです。したがって、サーバ・メッセージ・ストアとして使用できるのは、Mobile Link がサポートしているいずれかの RDBMS (Adaptive Server Anywhere、Adaptive Server Enterprise、Microsoft SQL Server、DB2、または Oracle) 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。このデータベースは、サーバ・メッセージ・ストア専用にする必要はないため、他の用途にも使用してかまいません。

Adaptive Server Anywhere データベースは、統合データベースとして使用できるように自動的に設定されます。

Adaptive Server Anywhere データベースの作成方法の詳細については、『ASA データベース管理ガイド』>「初期化ユーティリティ」を参照してください。

9.0.2 よりも前のバージョンで作成された Adaptive Server 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。

アップグレード方法については、『SQL Anywhere Studio 新機能ガイド』>「ソフトウェアとデータベースのアップグレード」を参照してください。

Adaptive Server Anywhere 以外のデータベースを使用する場合、データベースを統合データベースとして使用するための設定スクリプトをまだ実行していなければ、このスクリプトを実行する必要があります。

Adaptive Server Anywhere 以外のデータベースを使用してサーバ・メッセージ・ストアを設定する方法については、『Mobile Link 管理ガイド』>「統合データベースの設定」を参照してください。

注意

- Mobile Link 統合データベースをサーバ・メッセージ・ストアとして使用することで、Mobile Link 同期アプリケーションにメッセージングを統合できます。

  「QAnywhere メッセージングと Mobile Link データ同期の統合」67 ページを参照してください。

例

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

```
dbinit qanysrv.db
```

このデータベースは、サーバ・メッセージ・ストアとして使用できます。

QAnywhere メッセージング用 Mobile Link 同期サーバの起動

QAnywhere は、Mobile Link 同期を使用してメッセージを転送します。QAnywhere メッセージングを使用するには、Mobile Link 同期サーバ (dbmlsrv9) 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。

  『Mobile Link 管理ガイド』> 「-c オプション」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。必要に応じて、設定情報を指定することもできます。

  『Mobile Link 管理ガイド』> 「-m オプション」を参照してください。

- **-zu+** 開発モードで作業するときに、必要に応じて指定してください。クライアント・メッセージ・ストアを作成するときには、クライアント・メッセージ・ストア ID を統合データベースに追加する必要がありますが、-zu+ オプションを指定すると、この操作が自動的に実行されます。

  『Mobile Link 管理ガイド』> 「-zu オプション」を参照してください。

上記以外にも、Mobile Link 同期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、『Mobile Link 管理ガイド』> 「Mobile Link 同期サーバ」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 同期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」56 ページを参照してください。

例

qanysrv.db という名前のサーバ・メッセージ・ストアを使用している場合、QAnywhere メッセージングを開始するには、コマンド・プロンプトで次のように入力します。

```
dbmlsrv9 -m -c "dbf=qanysrv.db"
```

クライアント・メッセージ・ストア ID の追加

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。クライアント・メッセージ・ストア ID は Mobile Link ユーザ名です。この Mobile Link ユーザ名をサーバ・メッセージ・ストアに追加しなければならない場合があります。これを実行するには、いくつかの方法があります。

- dbmluser ユーティリティを使用する。

  詳細については、『Mobile Link 管理ガイド』> 「Mobile Link ユーザ認証ユーティリティ」を参照してください。

- Sybase Central を使用する。

- dbmlsrv9 で -zu+ コマンド・ライン・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。

  詳細については、『Mobile Link 管理ガイド』> 「-zu オプション」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 クライアント』> 「Mobile Link ユーザの概要」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」107 ページを参照してください。

クライアント側コンポーネントの設定

❖ クライアント側コンポーネントの設定の概要

1 Adaptive Server 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」46 ページを参照してください。

2 クライアント・メッセージ・ストアの ID がサーバ・メッセージ・ストアに登録されていることを確認します。

「クライアント・メッセージ・ストア ID の追加」45 ページを参照してください。

3 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」71 ページを参照してください。

4 QAnywhere Agent を起動します。

「QAnywhere Agent の実行」49 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・ストアは、リモート・デバイス上の Adaptive Server Anywhere データベースです。アプリケーションは、QAnywhere クライアント API を使用してこのメッセージ・ストアに接続します。

クライアント・メッセージ・ストア用のデータベースは、他の非メッセージング・アプリケーションから使用しないでください。ただし、Mobile Link リモート・データベースをクライアント・メッセージ・ストアと組み合わせて使用して、メッセージングとデータ同期を統合することは可能です。

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。また、データ同期にメッセージングを簡単に統合できます。

詳細については、「QAnywhere メッセージングと Mobile Link データ同期の統合」67 ページと「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」124 ページを参照してください。

❖ クライアント・メッセージ・ストアを作成するには、次の手順に従います。

1　Adaptive Server Anywhere データベースを作成します。

『ASA SQL ユーザーズ・ガイド』>「データベースの作成」を参照してください。

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

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

  「-c オプション」105 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」118 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」48 ページと「-id オプション」107 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 ID をサーバ・メッセージ・ストアに追加します。-zu+ オプションを指定して dbmlsrv9 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。

「クライアント・メッセージ・ストア ID の追加」45 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」124 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」119 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 の作成

クライアント・メッセージ・ストア ID の設定方法はいくつかあります。

- 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。
- 上記のどちらの方法も使用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る。

詳細については、「QAnywhere Agent」101 ページを参照してください。

クライアント・メッセージ・ストアの作成例

次のコマンドを実行すると、Adaptive Server Anywhere データベース qanyclient.db が作成されます (dbinit -i オプションは、必須ではありませんが、データベースのサイズが削減されるので指定することをおすすめします)。

```
dbinit -i 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"
```

dbinit の詳細については、『ASA データベース管理ガイド』> 「dbinit コマンド・ライン・ユーティリティを使用したデータベースの作成」を参照してください。

QAnywhere Agent の実行

QAnywhere Agent (qaagent) は、クライアント・デバイス上で動作する独立のプロセスです。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。

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。1 つのデバイスで同時に複数の QAnywhere Agent インスタンスを実行することはできません。

qaagent 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。

- **-c "*connection-string*"** クライアント・メッセージ・ストアに接続します。
- **-id *client-message-store-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このクライアント・メッセージ・ストア ID を指定する必要があります。
- **-x *protocol*[(*protocol-options*)]** Mobile Link 同期サーバに接続します (Mobile Link 同期サーバが QAnywhere Agent とは異なるデバイスで動作している場合に指定します)。

これらのオプションの詳細や、qaagent のオプションの完全なリストについては、「QAnywhere Agent の構文」102 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。ポリシーが指定されていない場合、デフォルトの 10 秒間隔でメッセージが転送されます。事前定義済みのポリシーとして、スケジュール済み、自動、オンデマンドの 3 つがあります。これらのポリシーを指定するには、qaagent -policy オプションを使用します。

カスタムのポリシーを定義することもできます。カスタムのポリシーは、メッセージの転送タイミングを決定するルール・セットです。カスタムのポリシーには、転送するメッセージを指定することもできます。転送ルールが保存されているファイルを -policy オプションとともに指定すると、カスタムのポリシーが使用されます。

スケジュール済みポリシー

スケジュール済みポリシーは、指定された時間間隔で転送を実行するように QAnywhere Agent に指示します。スケジュールを開始するには、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。デフォルトは 10 秒です。

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

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

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

時間間隔を無視するには、TriggerSendReceive() 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。

自動ポリシー

自動ポリシーは、クライアント・メッセージ・ストアとサーバ・メッセージ・ストアをできるかぎり最新の状態に維持します。自動メッセージ転送を設定するには、QAnywhere Agent を起動するときに、次のように **automatic** キーワードを指定します。

```
qaagent -policy automatic
```

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。
- 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。
- Push 通知を受信した。
- 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。

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。このモードを開始するには、QAnywhere Agent を起動するときに、次のように **ondemand** キーワードを指定します。

```
qaagent -policy ondemand
```

アプリケーションで TriggerSendReceive() メソッドが呼び出されると、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 ″system″ 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() メソッド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。

カスタム・ポリシー

カスタム・ポリシーを使用すると、メッセージ転送のタイミングと、そのメッセージ転送で送信するメッセージを定義できます。カスタム・ポリシーの内容は、ルール・セットを使用して定義し、ファイルに保存します。カスタム・ポリシーを使用するには、QAnywhere Agent を起動するときに、次のようにルール・ファイルを指定します。

qaagent -policy *rules-file*

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、「スケジュール構文」136 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。
- 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。
- Push 通知を受信した。
- 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。

カスタム・ポリシーの内容をルール・ファイル内に指定する方法の詳細については、「転送ルール」132 ページを参照してください。

ポリシーの作成方法の詳細については、「-policy オプション」113 ページを参照してください。

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 通知が行われるとき、QAnywhere アーキテクチャでは次の 2 つ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。

Push 通知はデフォルトで有効になっています。つまり、qaagent -push_notification オプションは、デフォルトでは有効に設定されます。Push 通知は、UDP プロトコルを介して配信されるか、SMS メッセージとして配信されます。

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。

SMS を使用する場合は、Push 通知を使用するために Listener を起動する必要があります。詳細については、「SMS を介した Push 通知の使用方法」54 ページを参照してください。

Push 通知を無効にする方法については、「-push_notifications オプション」116 ページを参照してください。

参照

Push 通知の詳細については、次の項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」9 ページ
- 「Push 通知とネットワーク・ステータスの変化の処理」94 ページ
- 「-push_notifications オプション」116 ページ
- 「SMS を介した Push 通知の使用方法」54 ページ

SMS を介した Push 通知の使用方法

SMS を介してメッセージを転送している場合も QAnywhere 通知を使用できます。ただし、dblsn 実行プログラムを使用して Listener を起動する必要があります。

詳細については、『Mobile Link サーバ起動同期ユーザーズ・ガイド』>「Listener」を参照してください。

次のコマンドを実行して、通常と同様に QAnywhere Agent を起動します。

```
qaagent -id device-id -c "connect-string"
```

サーバ側で、SMS メッセージを送信するために必要な SMTP プロパティを設定します。

詳細については、『Mobile Link サーバ起動同期ユーザーズ・ガイド』>「SMTP ゲートウェイ・プロパティ」を参照してください。

JMS コネクタの使用方法

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。メッセージの交換は、QAnywhere クライアント・アプリケーション間だけでなく、JMS インタフェースをサポートする外部メッセージング・システムとの間でも行うことができます。これには、コネクタと呼ばれる特殊なクライアントを使用します。QAnywhere アプリケーションでは、外部メッセージング・システムを、QAnywhere クライアントと同様に動作するように設定します。これには、固有のアドレスと構成を使用します。

ここで説明したアプローチを採用したアーキテクチャの詳細については、「外部メッセージング・システムとのメッセージングのシナリオ」11 ページを参照してください。

❖ 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. JMS コネクタ・プロパティ・ファイルを作成し、ianywhere.connector.address プロパティにコネクタ・アドレスを設定します。QAnywhere アプリケーションは、コネクタ経由でメッセージを送信するとき、このアドレスを使用します。

   「JMS コネクタ・プロパティ・ファイルの設定」57 ページを参照してください。

3. Mobile Link メッセージング・プロパティ・ファイルを作成し、ianywhere.qa.server.connectorPropertiesFiles プロパティに JMS コネクタ・プロパティ・ファイルの名前を設定します。

Mobile Link メッセージング・プロパティ・ファイルを指定する方法の詳細については、『Mobile Link 管理ガイド』>「-m オプション」を参照してください。

4 サーバ・メッセージ・ストアに接続するための接続文字列、-m オプション、-sl java オプションを指定して、Mobile Link 同期サーバを起動します。-m オプションを指定すると、前の手順で作成した Mobile Link メッセージング・プロパティ・ファイルが参照されます。

「JMS 統合用 Mobile Link サーバの起動」56 ページを参照してください。

5 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*** に送信します。

「QAnywhere から JMS に送信されるメッセージのアドレス指定」61 ページを参照してください。

6 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、JMS メッセージを作成し、ias_ToAddress プロパティに 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) を設定します。次に、そのメッセージを JMS キューに登録し、送信操作を行います。

「JMS から QAnywhere に送信されるメッセージのアドレス指定」64 ページ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 同期サーバ (dbmlsrv9) 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。

『Mobile Link 管理ガイド』>「-c オプション」を参照してください。

- **-m *message-properties-file*** *message-properties-file* に指定したメッセージ・プロパティ・ファイルには、JMS コネクタ・プロパティ・ファイルを指定する ianywhere.qa.server.connectorPropertiesFiles プロパティを定義しておく必要があります。このメッセージ・プロパティ・ファイルには、以下の文が存在する必要があります。

  ```
  ianywhere.qa.server.connectorPropertiesFiles=file1;
  [ file2; ]...
  ```

  『Mobile Link 管理ガイド』>「-m オプション」を参照してください。

- **-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。

  『Mobile Link 管理ガイド』>「-sl java オプション」を参照してください。

例

次の例では、メッセージ・プロパティ・ファイル *msg.props* (現在の作業ディレクトリにあります)、JMS クライアント・ライブラリ *jmsclient.jar* (これも現在の作業ディレクトリにあります)、QAnywhere 9.0 Sample データベース (メッセージ・ストア) を指定して、Mobile Link 同期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。

```
dbmlsrv9 -sl java(-cp "jmsclient.jar")
         -m msg.props
         -c "QAnywhere 9.0 Sample" ...
```

JMS コネクタ・プロパティ・ファイルの設定

JMS コネクタ・プロパティ・ファイルには、JMS システムとの接続のための情報を指定するエントリを記述します。これで、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することができます。

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。この Java クラスは QAnywhere から提供されています。必ず ianywhere.message.connector.jms.NativeConnectionJMS を指定してください。
- **ianywhere.connector.id** JMS コネクタをユニークに識別する識別子を指定します。指定した識別子は、コネクタの QAnywhere サーバ・コンソールに表示されるすべてのログ出力エラー、警告、情報メッセージの先頭に付けられます。
- **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。

  詳細については、「QAnywhere から JMS に送信されるメッセージのアドレス指定」61 ページを参照してください。
- **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。このアドレスには、たとえば、間違った形式のアドレスや不明なアドレスが含まれるメッセージが送信されます。

  デフォルト値は ianywhere.connector.deadMessage です。
- **ianywhere.connector.logLevel** コンソールとログ・ファイルに出力されるコネクタ情報の詳細度を指定します。次の詳細度レベルがあります。
  - **1** エラー・メッセージを出力します。
  - **2** エラー・メッセージと警告メッセージを出力します。
  - **3** エラー、警告、情報の各メッセージを出力します。
  - **4** デバッグ・モード：最も詳細なログを出力します。

- **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。

  すべてのコネクタに共通の圧縮レベルを設定するには、dbmlsrv9 -m オプションとともに指定する設定ファイルに ianywhere.qa.compressionLevel プロパティを含めます。このように 2 か所にある圧縮レベル設定を両方とも使用した場合、コネクタ別の設定がすべてのコネクタに共通の設定よりも優先されます。詳細については、『Mobile Link 管理ガイド』>「-m オプション」を参照してください。

- **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。
- **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。
- **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。
- **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。
- **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.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。
- **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。
- **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。
- **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。
- **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。

- **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。
- **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。

例

SQL Anywhere Studio インストール環境のサブディレクトリ *Samples¥QAnywhere¥connectors* に、コネクタ・プロパティ・ファイルのサンプルがあります。Sybase EAServer 用のプロパティ・ファイルのサンプルを以下に示します。

```
ianywhere.connector.nativeConnection=ianywhere.message.
connector.jms.NativeConnectionJMS
 ianywhere.connector.id=easerver
 ianywhere.connector.logLevel=4

ianywhere.connector.address=ianywhere.connector.easerve
r
 xjms.jndi.factory=com.sybase.jms.InitialContextFactory
 xjms.jndi.url=iiop://<substitute with host name>:9000
 xjms.jndi.authName=jagadmin
 xjms.jndi.password.e=
 xjms.topicFactory=javax.jms.TopicConnectionFactory
 xjms.topicConnectionAuthName=
 xjms.topicConnectionPassword.e=
 xjms.queueFactory=javax.jms.QueueConnectionFactory
 xjms.queueConnectionAuthName=
 xjms.queueConnectionPassword.e=
 xjms.receiveDestination=qanywhere
```

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・ファイルを定義すると、QAnywhere から複数の JMS メッセージ・システムに接続できます。この場合、JMS メッセージ・システムに接続する各コネクタを、それぞれに専用の JMS コネクタ・プロパティ・ファイルを使用して設定する必要があります。コネクタのプロパティのうち、ianywhere.connector.id と ianywhere.connector.address の 2 つだけは、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.id プロパティはユニークにする必要があります。これは、QAnywhere サーバ・コンソールに表示されるすべてのコネクタ・メッセージのプレフィクスとして使用されます。
- 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。

  QAnywhere クライアントのアドレス指定の詳細については、「QAnywhere から JMS に送信されるメッセージのアドレス指定」61 ページを参照してください。

コネクタ・プロパティ・ファイルの詳細については、「JMS コネクタ・プロパティ・ファイルの設定」57 ページを参照してください。

QAnywhere から JMS に送信されるメッセージのアドレス指定

QAnywhere クライアントは、アドレスに *connector_address¥JMS_destination* と指定することで、JMS システムにメッセージを送信できます。*connector_address* は、コネクタ・プロパティ ianywhere.connector.address の値です。*JMS_destination* 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。

詳細については、「QAnywhere メッセージ・アドレスについて」75 ページを参照してください。

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#
 QAManagerBase mgr;
 QAMessage msg;
 // Initialize the manager
 ...
 msg = mgr.CreateTextMessage();
 // Set the message content
```

```
  ...

mgr.PutMessage(@"ianywhere.connector.easerver¥myqueue",
msg );
```

```
 C++
 QAManagerBase *mgr;
 QATextMessage *msg;
 // Initialize the manager
 ...
 msg = mgr.createTextMessage();
 // Set the message content
 ...
 mgr->putMessage(
"ianywhere.connector.easerver¥¥myqueue", msg );
```

QAnywhere メッセージの JMS メッセージへのマッピング

QAnywhere メッセージは、JMS メッセージに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	JMS Destination と JMS プロパティ ias_ToAddress	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	JMS Expiration	
InReplyToID	JMS CorrelationID	
MessageID	なし	マッピングされない
Priority	JMS Priority	
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	JMS ReplyTo	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

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ます。型も維持されます。ただし、1 つだけ例外があります。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 にマッピングされます。

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。

詳細については、「QAnywhere メッセージ・アドレスについて」75 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 "ianywhere.connector.jms_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(
"ianywhere.connector.jms_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

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。

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 ianywhere.connector.jms_receive に指定されているキューが設定されていなければならない
JMS Expiration	Expiration	
JMS CorrelationID	InReplyToID	
JMS MessageID	なし	マッピングされない
JMS Priority	Priority	
JMS Redelivered	なし	マッピングされない

JMS	QAnywhere	説明
JMS ReplyTo とコネクタの ianywhere.connector.address プロパティの値	ReplyToAddress	コネクタ・アドレスと、JMS ReplyTo に指定されている Destination 名が、間に文字 '¥' を挟んで連結される
JMS DeliveryMode	なし	マッピングされない
JMS Type	QAnywhere メッセージ・プロパティ JMSType	
JMS Timestamp	なし	マッピングされない

JMS のプロパティ

JMS のプロパティは、QAnywhere のプロパティに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。

QAnywhere メッセージングと Mobile Link データ同期の統合

QAnywhere メッセージングは、Adaptive Server Anywhere クライアントを使用した Mobile Link 同期アプリケーションに統合できます。

これには、以下の条件があります。

- QAnywhere クライアント・メッセージ・ストア・データベースと、データ同期アプリケーションが使用するデータベースは、異なるデータベース・ファイルであることが必要です。ただし、小型デバイスでの使用が考慮されているので、両方のデータベースを同じデータベース・サーバで実行することはできます。
- サーバ・メッセージ・ストアには、Mobile Link 統合データベースを使用する必要があります。サポートされている Mobile Link 統合データベースのいずれか (Adaptive Server Anywhere、Adaptive Server Enterprise、Oracle、DB2、または Microsoft SQL Server) を使用できます。
- QAnywhere メッセージング・アプリケーションを別に作成することも、データ同期アプリケーションにメッセージングを統合することもできます。

例

MyServer という名前のサーバと *MyAppData.db* という名前の Adaptive Server Anywhere クライアント・データベース・ファイルを使用する Mobile Link 同期システムに、QAnywhere メッセージングを統合する例を以下に示します。

クライアント・メッセージ・ストア・データベースを作成します。

```
dbinit -i MyAppData.db
```

クライアント・メッセージ・ストアを初期化します。

```
qaagent -id MyID -si -c "dbf=MyAppData.db"
```

MyAppData.db データベース・ファイルと *qanywhere.db* データベース・ファイルを持つリモート・デバイス上のデータベース・サーバを起動します。

```
dbsrv9 -n MyServer MyAppData.db -n MyAppData
qanywhere.db -n qanywhere
```

qanywhere.db をメッセージ・ストアとして、QAnywhere Agent を起動します。

```
qaagent -id MyID -c "eng=MyServer;dbn=qanywhere"
```

QAManager インスタンスを作成するとき、次の行が記述された QAManager プロパティ・ファイルを使用します。

```
CONNECT_PARAMS=eng=MyServer;dbn=qanywhere
```

他の QAnywhere メッセージング・アプリケーションを起動するときと同じようにして、Mobile Link 同期サーバを起動します。ただし、Mobile Link 統合データベースをサーバ・メッセージ・ストアとして使用します。次に例を示します。

```
dbmlsrv9 -m -c "dbf=your_consolidated.db"
```

フェールオーバ・メカニズムの設定

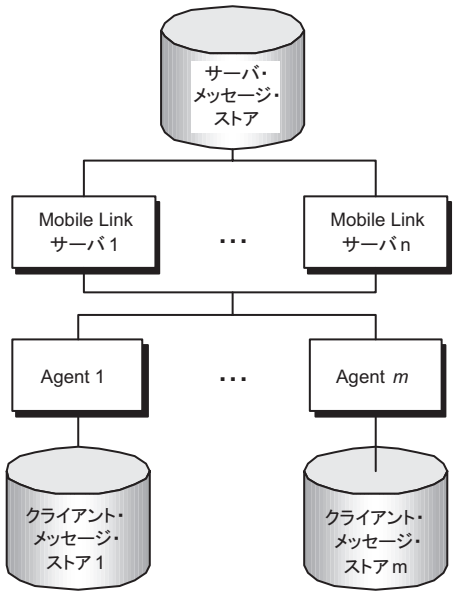
QAnywhere アプリケーションは、フェールオーバ・メカニズムを使用するように設定できます。フェールオーバ・メカニズムを使用すると、Mobile Link サーバで障害が発生しても代替サーバを使用できます。フェールオーバをサポートするには、各 QAnywhere Agent を起動するときに、Mobile Link サーバのリストを指定する必要があります。リストの最初に指定された Mobile Link サーバがプライマリ・サーバになります。残りのサーバは代替サーバになります。

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

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

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

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



この構成には次のような特徴があります。

- メッセージ転送が発生すると、QAnywhere Agent が接続されているサーバに関係なく、サーバ・メッセージ・ストア内のすべてのメッセージがクライアント・メッセージ・ストアに配信される。
- Push 通知は、QAnywhere Agent がプライマリ・サーバに接続されているときだけ、その QAnywhere Agent に送信される。
- 単一障害点 (Single-Point-of-Failure) が存在する。サーバ・メッセージ・ストアが存在するマシンが使用できなくなると、メッセージを転送できない。

第 4 章

QAnywhere クライアント・アプリケーションの作成

この章の内容

この章では、QAnywhere API の使用方法について説明します。QAnywhere API は、C++ API として提供されているほか、iAnywhere.QAnywhere.Client ネームスペースとしても提供されています。

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、次のいずれかのプログラミング・インタフェースを使用して作成できます。

- **QAnywhere .NET API** 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。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 2003 をサポートしています。

  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 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で使用することができます。

  TestMessage サンプル・アプリケーションについては、C# バージョン、Visual Basic .NET バージョン、Managed C++ バージョンがすべて用意されています。詳細については、「レッスン 5：TestMessage クライアントのソース・コードの概要」29 ページを参照してください。

  QAnywhere .NET API の詳細については、「iAnywhere.QAnywhere.Client ネームスペース」225 ページを参照してください。

- **QAnywhere C++ API** Windows コンピュータでの配備に使用するプログラミング・インタフェース。このインタフェースは、C++ プログラマ向けです。

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

  QAnywhere C++ API の構成要素は次のとおりです。

- SQL Anywhere Studio インストール環境の *QAnywhere¥h* サブディレクトリにある一連のヘッダ・ファイル (メインのヘッダ・ファイルは *qa.hpp*)
- SQL Anywhere Studio インストール環境のサブディレクトリ *QAnywhere¥ce¥arm.30¥lib*、*QAnywhere¥lib*、*QAnywhere¥ce¥x86.30¥lib* にあるインポート・ライブラリ (*qany9.lib*)
- SQL Anywhere Studio インストール環境のサブディレクトリ *win32*、*ce¥arm.30*、*ce¥x86.30* にあるランタイム DLL (*qany9.dll*)

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

C++ バージョンの TestMessage サンプル・アプリケーションは、SQL Anywhere Studio インストール環境のサブディレクトリ *Samples¥QAnywhere* にあります。詳細については、「レッスン 5 : TestMessage クライアントのソース・コードの概要」29 ページを参照してください。

C++ API の詳細については、「QAnywhere C++ API リファレンス」155 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概要

❖ **C++ または .NET を使用してクライアント・アプリケーションを構築するには、次の手順に従います。**

1 QAnywhere クライアント API を初期化します。

「QAnywhere クライアント API の初期化」77 ページを参照してください。

2 QAManager のプロパティを設定します。

「QAManager のプロパティの設定」83 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 メッセージの送信」87 ページ
- 「同期的なメッセージ受信」89 ページ
- 「非同期的なメッセージ受信」91 ページ
- 「サイズの大きなメッセージの読み込み」93 ページ
- 「Push 通知とネットワーク・ステータスの変化の処理」94 ページ
- 「トランザクション志向メッセージングの実装」96 ページ
- 「QAnywhere の停止」99 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」100 ページを参照してください。

QAnywhere メッセージ・アドレスについて

QAnywhere メッセージの宛先は、次の 2 つの部分で構成されています。

- 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 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」46 ページを参照してください。

- アプリケーション・キュー名

  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。

送信先アドレスの形式は次のとおりです。

id¥queue-name

エスケープ文字とアドレス
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 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。コネクタ・プロパティ・ファイルに指定されている ianywhere.connector.address プロパティの値です。

  詳細については、「JMS コネクタ・プロパティ・ファイルの設定」57 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。

  詳細については、「JMS コネクタの使用方法」55 ページを参照してください。

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、「QAnywhere から JMS に送信されるメッセージのアドレス指定」61 ページと「JMS から QAnywhere に送信されるメッセージのアドレス指定」64 ページを参照してください。

QAnywhere クライアント 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。

C++ アプリケーションの設定

❖ **QAnywhere クライアント API を初期化するには、次の手順に従います (C++ の場合)。**

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
}
```

3　QAManager オブジェクトを作成します。

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 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、setProperties メソッドを使用する。

  詳細については、「プログラムによるプロパティの設定」84 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 メソッドの引数としてプロパティ・ファイルを指定する。

  ```
  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。

  詳細については、「ファイルによるプロパティの設定」83 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** でなければなりません。前者は暗黙的な受信確認モードを意味します。このモードの場合、メッセージはクライアントで受信されるとすぐに受信確認されます。後者は明示的な受信確認モードを意味します。このモードの場合、QAManager の 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。

詳細については、「QAManager クラス」178 ページを参照してください。

.NET アプリケーションの設定

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

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

これらの作業を行った上で、QAnywhere クライアント API を初期化してください。

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

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

   [参照の追加] ダイアログが表示されます。
3. [.NET] タブで、[参照] をクリックして iAnywhere.QAnywhere.Client.dll を見つけます (デフォルト・ロケーションは、*¥Program Files¥Sybase¥SQL Anywhere 9¥win32* です)。DLL を選択して [開く] をクリックします。

   Windows と Windows CE にはそれぞれ異なるバージョンの DLL があるので注意してください。
4. DLL がプロジェクトに追加されているかどうかを確認できます。[参照の追加] ダイアログを開き、[.NET] タブをクリックします。[選択されたコンポーネント] リストに iAnywhere.QAnywhere.Client.dll が表示されます。[OK] をクリックしてダイアログを閉じます。

プロジェクトの [ソリューション エクスプローラ] ウィンドウの [参照設定] フォルダに DLL が追加されます。ソース・コードのデータ・プロバイダ・クラスを参照します。

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

1 Visual Studio .NET を起動し、プロジェクトを開きます。

2 C# を使用している場合は、プロジェクトの先頭にある using ディレクティブのリストに次の行を追加します。

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

3 Visual Basic .NET を使用している場合は、プロジェクトの先頭で行 Public Class Form1 の前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。この行を追加しない場合は、次のように完全なクラス名を指定してください。

```
iAnywhere.QAnywhere.Client.QAManager
mgr =
  new
iAnywhere.QAnywhere.Client.QAManagerFactory.Instanc
e.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" );
```

このコードではクラス名を省略形で指定しています。

❖ QAnywhere クライアント API を初期化するには、次の手順に従います (.NET の場合)。

1　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。

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

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。

```
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** でなければなりません。前者は暗黙的な受信確認モードを意味します。このモードの場合、メッセージはクライアントで受信されるとすぐに受信確認されます。後者は明示的な受信確認モードを意味します。このモードの場合、QAManager の 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。

詳細については、「QAManager クラス」251 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 のプロパティの設定

QAManager のプロパティを設定するには、次の 2 つの方法があります。

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

ファイルによるプロパティの設定

QAManager 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。特定の QAManager 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。アプリケーションは複数の QAManager を使用できますが、個々の QAManager を複数スレッド間で共有することはできません。

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。

このファイルの名前はアプリケーション内で指定します。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。

詳細については、「QAManager のプロパティ」85 ページを参照してください。

このファイルの # で始まる行はコメントとして処理されます。

例

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

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

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

次は、C++ のコード例です。

```
QAManagerFactory *    qa_factory;
QAManager *        mgr;

qa_factory = QAnywhereFactory_init();
qa_factory->createQAManager( "mymanager.props" );
mgr->open(
AcknowledgementMode::EXPLICIT_ACKNOWLEDGEMENT );
```

次は、C# のコード例です。

```
QAManager        mgr;

 mgr = QAManagerFactory.Instance.CreateQAManager(
"mymanager.props" );
 mgr.Open( AcknowledgeMode.EXPLICIT_ACKNOWLEDGEMENT );
```

プログラムによるプロパティの設定

プログラムでプロパティを設定するには、QAManager の setProperties メソッドを使用します。QAManager の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。

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。

```
QAManagerFactory *    qa_factory;
QAManager *        mgr;

qa_factory = QAnywhereFactory_init();
mgr->createQAManager( NULL );
```

```
mgr->setProperty( "COMPRESSION_LEVEL", "9");
mgr->setProperty("CONNECT_PARAMS", "DBF=mystore.db" );
mgr->open(
AcknowledgementMode::EXPLICIT_ACKNOWLEDGEMENT );
```

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

```
QAManager        mgr;

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

参照

- 「iAnywhere.QAnywhere.Client ネームスペース」225 ページ (.NET を使用する場合)
- 「QAnywhere C++ API リファレンス」155 ページ (C++ を使用する場合)
- 「QAManager のプロパティ」85 ページ

QAManager のプロパティ

次の表は、使用可能なプロパティのすべてを示します。

プロパティ	説明
COMPRESSION_LEVEL=*n*	圧縮レベルを設定します。 *n* は圧縮率です。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。

プロパティ	説明
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ときに使用する接続文字列。接続オプションは、*keyword=value* という形式で指定します。オプションのリストについては、『ASA データベース管理ガイド』> 「接続パラメータ」を参照してください。
LOG_FILE="*filename*"	ログ・メッセージの出力先ファイルの名前。ロギングは暗黙的に有効になっています。
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 100000、Windows CE では 6400 です。

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 同期サーバに送信し、そして Mobile Link 同期サーバが実際に宛先にメッセージを配信します。

❖ 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。

テキスト・メッセージまたはバイナリ・メッセージを作成します。送信するメッセージのタイプに対応するメソッドを使用してください。

```
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。

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。

❖ 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。

テキスト・メッセージまたはバイナリ・メッセージを作成します。送信するメッセージのタイプに対応するメソッドを使用してください。

```
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。

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。

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 (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、メッセージを受信します。

```
if( mgr->start() ) {
     for( ;; ) {
        msg = mgr->getMessageNoWait( "queue_name" );
        if( msg == NULL )  break;
        text_msg = msg->castToTextMessage();
        if( text_msg != NULL ) {
             // display text message using
             // text_msg->getText()
        }
        sleep( time-period );
    }
    mgr->stop();
}
```

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

1　着信メッセージを格納するメッセージ・オブジェクトを宣言します。

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

2 メッセージ・キューをポーリングして、メッセージを収集します。

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

非同期的なメッセージ受信

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、SQL Anywhere Studio とともにインストールされる 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。

❖ 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
{
   private:
   void onMessage( QAMessage * Msg);
};
```

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 msg
}
```

エラー処理には特に注意してください。エラー処理を行わないと、メッセージが失われる可能性があります。

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
);
```

❖ 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・ハンドラを実装します。

```
private void onMessage(QAMessage msg)
 {
     // process msg
 }
```

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・ハンドラ関数を引数として 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。

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

queue-name は、QAManager オブジェクトが待機するキューの名前を表す文字列です。

サイズの大きなメッセージの読み込み

メッセージのサイズが大き過ぎて、QAManager のプロパティ MAX_IN_MEMORY_MESSAGE_SIZE に設定されている制限値、またはデフォルトの制限値 (Windows の場合は 1MB、Windows CE の場合は 64K) を超える場合があります。その場合は、メッセージをメモリから読み込むことができないため、readInt や readString などの通常の読み込み用メソッドは使用できません。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。読み込みが終了すると、-1 が返されます。

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。

Push 通知とネットワーク・ステータスの変化の処理

通知の使用方法の詳細については、「Push 通知によるメッセージングのシナリオ」9 ページを参照してください。

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。

たとえば、次の C# コードではシステム・メッセージと通常のメッセージを処理しています。この例では、メッセージ処理のためのアプリケーション・ロジックを実装したメッセージ処理関数 onMessage と onSystemMessage が、すでに定義されているとものとします。

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

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

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

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

```
if( msg.PropertyExists(MessageProperties.MSG_TYPE) ) {
      msg_type=(MessageType)msg.GetIntProperty(...);
      ...
 } else {
      // Process regular message
 }

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

参照

- 「メッセージ・ヘッダ」143 ページの ias_Status。
- 「クライアント・ストア・プロパティ」146 ページの ias_Network。

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、トランザクション志向マネージャと呼ばれる特殊な QAManager オブジェクトを作成します。

❖ 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 (C++ の場合)。

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まったく同じです。

```
#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 (C++ の場合)。

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() メソッドによって、実際にメッセージが送信されます。

QAnywhere の停止

メッセージの送信と受信を完了したら、次の手順に従って 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 (C++)。

1　QAManager 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 (.NET の場合)。

- QAManager を停止し、クローズします。

```
mgr.Stop();
mgr.Close();
```

QAnywhere アプリケーションの配備

QAnywhere アプリケーションを配備するために必要なファイルの詳細については、『Mobile Link 管理ガイド』> 「QAnywhere アプリケーションの配備」を参照してください。

第 5 章

QAnywhere Agent

この章の内容

この章では、QAnywhere Agent (qaagent) の構文について説明します。

QAnywhere Agent の実行方法の概要については、「QAnywhere Agent の実行」49 ページを参照してください。

QAnywhere Agent の構文

QAnywhere Agent を使用することで、同一クライアント・デバイス上のすべての QAnywhere アプリケーションのメッセージ送受信を処理できます。

使用方法

QAnywhere Agent は、他にも、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制御します。QAnywhere Agent コマンドを起動すると、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。Mobile Link 同期クライアントは、メッセージの送信と受信に使用されます。したがって、dbmlsync 実行プログラムは必須です。
- **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_notifications オプションを無効にして実行する必要があります。
- **データベース・サーバ** クライアント・メッセージ・ストアは、Adaptive Server Anywhere データベースです。QAnywhere Agent は、Adaptive Server Anywhere データベースを実行するために Adaptive Server Anywhere データベース・サーバを必要とします。Windows CE では、*dbsrv9.exe* がデータベース・サーバです。Windows では、パーソナル・データベース・サーバ *dbeng9.exe* がデータベース・サーバです。

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

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。

詳細については、「QAnywhere アプリケーションの配備」100 ページを参照してください。

構文

qaagent [*option* ...]

オプション	説明
@*data*	指定された環境変数または設定ファイルからオプションを読み込む。「@data オプション」105 ページを参照してください。
-c *connection-string*	クライアント・メッセージ・ストアに接続するための接続文字列を指定する。「-c オプション」105 ページを参照してください。
-id *id*	QAnywhere Agent が接続するクライアント・メッセージ・ストアの ID を指定する。「-id オプション」107 ページを参照してください。
-iu *upload-size*	インクリメンタル・アップロードのサイズをバイト単位で制御する。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 k、m を使用します。「-iu オプション」108 ページを参照してください。
-la_port *number*	Listener が Mobile Link 同期サーバからの通知を受信するポートを指定する。デフォルトは 5001 です。「-la_port オプション」109 ページを参照してください。
-mp *password*	同期対象の ID に対応する Mobile Link パスワードを指定する。「-mp オプション」110 ページを参照してください。
-o *logfile*	指定されたファイルに出力メッセージのログを取る。「-o オプション」110 ページを参照してください。
-on *size*	QAnywhere Agent のメッセージ・ログ・ファイルの最大サイズを指定する。ログ・ファイルがこのサイズに達すると、現在のファイルが拡張子 .old を持つ名前に変更され、新しいファイルが作成されます。「-on オプション」111 ページを参照してください。

オプション	説明
-os *size*	QAnywhere Agent のメッセージ・ログ・ファイルの最大サイズを指定する。ログ・ファイルがこのサイズに達すると、新しい名前を持つ新しいログ・ファイルが作成されて使用されます。「-os オプション」112 ページを参照してください。
-ot *logfile*	ファイルをトランケートし、出力メッセージのログを取る。「-ot オプション」112 ページを参照してください。
-policy *policy-type*	転送ポリシーを指定する。指定したポリシーは、QAnywhere Agent で使用されます。「-policy オプション」113 ページを参照してください。
-port *number*	Listener からのメッセージを受信するポートを指定する。デフォルトは 5002 です。「-port オプション」116 ページを参照してください。
-push_notifications *value*	Push 通知を有効または無効にする。デフォルトは有効です。「-push_notifications オプション」116 ページを参照してください。
-q	QAnywhere Agent をクワイエット・モードで起動する。ウィンドウをシステム・トレイ内に最小化します。「-q オプション」117 ページを参照してください。
-qi	QAnywhere Agent をクワイエット・モードで起動する。ウィンドウを完全に非表示にします。「-qi オプション」117 ページを参照してください。
-si	データベースを、クライアント・メッセージ・ストアとして使用できるように初期化する。「-si オプション」118 ページを参照してください。
-su	クライアント・メッセージ・ストアを、Anywhere Studio バージョン 9.0.1 からバージョン 9.0.2 にアップグレードする。「-su オプション」119 ページを参照してください。
-v[*levels*]	ログの冗長レベルを指定する。「-v オプション」120 ページを参照してください。

オプション	説明
-x { **http**|**tcpip** } [(*keyword*=*value*;...)]	Mobile Link 同期サーバとの通信に使用するプロトコルのオプションを指定する。「-x オプション」121 ページを参照してください。

@data オプション

機能　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* } ...

説明　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。

設定ファイルの詳細については、『ASA データベース管理ガイド』>「設定ファイルの使用」を参照してください。

設定ファイル内のパスワードなどの情報を保護する場合は、ファイル非表示ユーティリティを使用して、設定ファイルの内容を難読化できます。

『ASA データベース管理ガイド』>「dbfhide コマンド・ライン・ユーティリティを使用してファイル内容を隠す」を参照してください。

-c オプション

機能　クライアント・メッセージ・ストアに接続するための接続文字列を指定します。

構文　**qaagent -c** *connection-string* ...

デフォルト

- uid - **ml_qa_user**
- pwd - **qanywhere**

- dbn - ***id***（*id* は、-id オプションが指定されていればこのオプションが示すクライアント・メッセージ・ストア ID になり、このオプションが指定されていなければデバイス名になる)
- dbf - ***id*.db**（*id* は、-id オプションが指定されていればこのオプションが示すクライアント・メッセージ・ストア ID になり、このオプションが指定されていなければデバイス名になる)

説明

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。

接続パラメータをコマンド・ラインで指定しない場合は、ODBC データ・ソースの指定に含めてください。RDBMS と ODBC データ・ソースを調べて、必要な接続データを判断してください。

接続パラメータの完全なリストについては、『ASA データベース管理ガイド』> 「接続パラメータ」を参照してください。

以下に、よく使用する接続パラメータの一部を示します。

- **dbf=*filename*** クライアント・メッセージ・ストアを、別のアプリケーションから開始するのではなく QAnywhere Agent に開始させる場合は、データベース・ファイル名を指定する必要があります。データベース・ファイル名のデフォルト値はデータベース名です。データベースのデフォルト名は、デバイス名の後に *.db* を付けたものです。-dbn 接続パラメータを使用して明示的に指定することもできます。

- **dbn=*database-name*** QAnywhere Agent の起動時にクライアント・メッセージ・ストアがすでに開始されている場合は、データベース・ファイル名ではなくデータベース名を指定することでそのクライアント・メッセージ・ストアに接続できます。QAnywhere Agent は、デフォルトの Adaptive Server Anywhere データベース・サーバ上で、その名前を持つデータベースを検索します。

  クライアント・メッセージ・ストアのデフォルトのデータベース名は ID (agent1 など) です。ID は、qaagent -id　オプションを使用して指定することも、デフォルトのクライアント・メッセージ・ストア ID (デバイス名) にすることもできます。

- **eng=*server-name*** すでに起動されているデータベース・サーバを使用する場合は、そのサーバ名をこのオプションで指定します。デフォルト値はデータベース名です。
- **uid=*user*** クライアント・メッセージ・ストアに接続するためのデータベース・ユーザ ID を指定します。デフォルト値を変更する場合、このオプションは必須です。
- **pwd=*password*** データベース・ユーザ ID に対応するパスワードを指定します。デフォルト値を変更する場合、このオプションは必須です。
- **dbkey=*key*** クライアント・メッセージ・ストアを高度な暗号化を使用して暗号化している場合は、データベースへのアクセスに必要な暗号化キーを指定します。

参照

- 『ASA データベース管理ガイド』>「接続パラメータ」
- 『ASA データベース管理ガイド』>「データベースへの接続」
- 『Mobile Link およびリモート・データ・アクセスの ODBC ドライバ』>「iAnywhere Solutions ODBC ドライバの概要」

例

```
qaagent -id Device1 -c "DBF=qanyclient.db" -x
tcpip(host=hostname) -policy automatic
```

-id オプション

機能

QAnywhere Agent が接続するクライアント・メッセージ・ストアの ID を指定します。

構文

qaagent -id *id ...*

デフォルト

ID のデフォルト値は、QAnywhere Agent が動作しているマシンの名前です。マシン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。

説明

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 ユーザ名です。Mobile Link アプリケーションでは、各リモート・データベースがユニークな Mobile Link ユーザ名を持つ必要があります。そのため、メッセージ・ストア ID は必須です。メッセージ・ストア ID は、Mobile Link ユーザ名と同様、統合データベースの Mobile Link システム・テーブルに追加する必要があります。この作業は、デフォルトの ID を使用する場合も行う必要があります (ただし、カスタムの認証を使用する場合は除きます)。

詳細については、『Mobile Link クライアント』> 「Mobile Link ユーザの作成」を参照してください。

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)

また、次の点にも注意が必要です。

- 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 オプションがオフに設定されている場合 (デフォルトの設定ではありません) は、英数字、アンダースコア (_)、アットマーク (@)、シャープ (#)、ドル記号 ($) のみを ID に含めることができる。

参照

- 『Mobile Link クライアント』> 「Mobile Link ユーザの概要」
- 「クライアント・メッセージ・ストアの設定」46 ページ

-iu オプション

機能

Mobile Link のインクリメンタル・アップロード機能を使用してメッセージの同期を行います。

構文

qaagent -iu *upload-size* [**k** | **m**] ...

デフォルト　アップロードは一括送信されます。

説明　-iu オプションを指定すると、Mobile Link のインクリメンタル・アップロード機能を使用してメッセージの同期を行うことができます。このオプションは、インクリメンタル・スキャンの最小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。

このオプションを指定すると、アップロードが 1 つ以上の部分に分割されて Mobile Link 同期サーバに送信されます。このオプションは、サイトが接続を維持できる時間が、一括アップロードを行うには十分ではない場合に役立ちます。このオプションが設定されていない場合、アップロードは一括送信されます。

インクリメンタル・アップロードでは、分割単位ごとにスキーマが送信されるので、効率が低下する可能性があります。

参照

- 『Mobile Link クライアント』> 「Increment (inc) 拡張オプション」

-la_port オプション

機能　Listener のポートを指定します。

構文　**qaagent -la_port** *number* ...

デフォルト　**5001**

説明　Listener が Mobile Link 同期サーバからの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。

デバイス上で Listener が動作していない場合、qaagent は、指定されたポートで Listener を起動します。Listener が動作している場合は、そのポートを指定する必要があります (または、デフォルトのポートを使用します)。

参照

- 「Push 通知によるメッセージングのシナリオ」9 ページ

- 「-push_notifications オプション」116 ページ

-mp オプション

機能

クライアント・メッセージ・ストア ID に対応する 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

説明

Mobile Link 同期サーバがユーザ認証を必要とする場合は、-mp オプションを使用して、ID に対応する Mobile Link パスワードを指定します。ID は、クライアント・メッセージ・ストアを識別する ID です。この ID は -id オプションを使用して指定します。

参照

- 『Mobile Link クライアント』> 「Mobile Link ユーザの認証」
- 「-id オプション」107 ページ

-o オプション

機能

出力をログ・ファイルに送信します。

構文

qaagent -o *logfile* ...

デフォルト

なし

説明

指定された名前のファイルに出力のログを取ります。Adaptive Server Anywhere 同期クライアント (dbmlsync) は、指定された名前に *_sync* というサフィックスを付けたものを名前とするファイルに、出力のログを取ります。Listener ユーティリティ (dblsn) は、指定された名前に *_lsn* というサフィックスを付けたものを名前とするファイルに、出力のログを取ります。

たとえば、ログ・ファイル名 *c:¥tmp¥mylog.out* を指定した場合、qaagent は *c:¥tmp¥mylog.out* に、dbmlsync は *c:¥tmp¥mylog_sync.out* に、dblsn は *c:¥tmp¥mylog_lsn.out* に、それぞれログを取ります。

参照

- 「-ot オプション」112 ページ
- 「-on オプション」111 ページ
- 「-os オプション」112 ページ
- 「-v オプション」120 ページ

-on オプション

機能

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* を持つ名前に変更され、新しいファイルが作成されます。

構文

qaagent -on *size* [**k** | **m**]...

デフォルト

なし

説明

size には、出力ログの最大ファイル・サイズを、バイト単位で指定します。キロバイトまたはメガバイトの単位を指定するには、それぞれサフィックス k、m を使用します。最小のサイズ制限は 10KB です。

ログ・ファイルが指定されたサイズに達すると、QAnywhere Agent は出力ファイルを拡張子 *.old* を持つ名前に変更し、元の名前で新しいファイルを開始します。

注意
.old ファイルがすでに存在する場合は、そのファイルが上書きされます。古いログ・ファイルを削除しないようにするには、代わりに -os オプションを使用します。

このオプションは、-os オプションと同時に使用できません。

参照

- 「-o オプション」110 ページ
- 「-ot オプション」112 ページ
- 「-os オプション」112 ページ
- 「-v オプション」120 ページ

-os オプション

機能

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

dbmlsrv9 -os *size* [**k** | **m**]...

デフォルト

なし

説明

size には、出力メッセージのログを取るファイルの最大サイズを指定します。デフォルトの単位はバイトです。キロバイトまたはメガバイトの単位を指定するには、それぞれサフィックス k、m を使用します。最小のサイズ制限は 10KB です。

QAnywhere Agent は、現在のファイル・サイズを確認してから、ファイルに出力メッセージのログを取ります。ファイルにログ・メッセージを追加するとファイル・サイズが指定されたサイズを超える場合、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。この場合、*xx* は、00 から 99 までの連続する文字で、*yymmdd* は現在の年、月、日を示します。

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。

このオプションは、-on オプションと同時に使用できません。

参照

- 「-o オプション」110 ページ
- 「-ot オプション」112 ページ
- 「-on オプション」111 ページ
- 「-v オプション」120 ページ

-ot オプション

機能

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

説明　指定された名前のファイルに出力のログを取ります。Adaptive Server Anywhere 同期クライアント (dbmlsync) は、指定された名前に *_sync* というサフィックスを付けたものを名前とするファイルに、出力のログを取ります。Listener ユーティリティ (dblsn) は、指定された名前に *_lsn* というサフィックスを付けたものを名前とするファイルに、出力のログを取ります。

たとえば、ログ・ファイル名 *c:¥tmp¥mylog.out* を指定した場合、qaagent は *c:¥tmp¥mylog.out* に、dbmlsync は *c:¥tmp¥mylog_sync.out* に、dblsn は *c:¥tmp¥mylog_lsn.out* に、それぞれログを取ります。

参照
- 「-o オプション」110 ページ
- 「-on オプション」111 ページ
- 「-os オプション」112 ページ
- 「-v オプション」120 ページ

-policy オプション

機能　メッセージの送受信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*] | **automatic** | *rules-file*

デフォルト
- scheduled ポリシーが選択され、転送間隔が 10 秒に設定される。
- 転送ルール・ファイル内に永続性ルールが指定されていない場合、メッセージの最終ステータスが受信済みまたは期限切れと判定された時点でメッセージが削除される。

説明　QAnywhere は、ポリシーに基づいて、メッセージを送受信するタイミングを決定します。*policy-type* には、次のいずれかを指定できます。

- **ondemand** QAnywhere クライアント・アプリケーションによって適切な関数が呼び出されたときだけ、メッセージが送信されます。

QAManager.PutMessage() メソッドが呼び出されると、メッセージがローカルのキューに登録されます。これらのメッセージは、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() メソッドが呼び出されるまで、クライアントに送信されません。

ondemand ポリシーを使用して通知に応答することもできます。ただし、クライアント・アプリケーションにこの操作を追加する必要があります。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。

通知の詳細については、「Push 通知によるメッセージングのシナリオ」9 ページを参照してください。

- **scheduled** 指定された間隔でメッセージを送受信します。デフォルトは 10 秒です。

  クライアント／サーバ間のメッセージ転送が、指定された間隔で実行されます。

  QAManager.PutMessage() メソッドが呼び出されると、メッセージがローカルのキューに登録されます。これらのメッセージは、指定された時間間隔が経過するまで転送されません。サーバ上のキューに登録された配信対象のメッセージも、指定された時間間隔が経過した時点でクライアントに転送されます。

  Push 通知が有効になっている場合、サーバ上のキューに登録された、クライアントを配信先とするメッセージは、次の時間間隔が経過したときに転送されます。

  TriggerSendReceive() メソッドが呼び出されると、指定された時間間隔は無視されます。指定の時間間隔が経過する前にメッセージ転送が強制的に実行されます。

interval 引数 (オプション) には、送受信操作を実行する間隔を秒単位で指定します。たとえば、次のコマンドを実行すると、15 分ごとにメッセージを同期するように QA Agent がスケジュール設定されます。

```
qaagent.exe -policy scheduled[900]
```

- **automatic** Push 通知を受信したとき、または次のいずれかのイベントが発生したときに、メッセージを受信します。

  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() メソッドの呼び出し
  - TriggerSendReceive() メソッドの呼び出し
  - 通知

    通知の詳細については、「Push 通知によるメッセージングのシナリオ」9 ページを参照してください。

  - クライアント上のメッセージ・ステータスの変化。たとえば、アプリケーションがローカルのキューからメッセージを取り出すとステータスの変化が発生します。この場合、メッセージ・ステータスは保留から配信済みに変化します。

- ***rules-file*** 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、メッセージを送信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。

  転送ルールの詳細については、「転送ルール」132 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」50 ページ
- 「QAnywhere C++ API リファレンス」155 ページ
- 「iAnywhere.QAnywhere.Client ネームスペース」225 ページ

-port オプション

機能 QAnywhere Agent のポートを指定します。

構文 **qaagent -port** *number* ...

デフォルト **5002**

説明 QAnywhere Agent が Listener からの通信を受信するポート番号です。

デフォルト値は **5002** です。デフォルト以外のポートを指定すると、QAnywhere Agent は、そのポート上で通信するように Listener を設定する処理を自動的に行います。

-push_notifications オプション

機能 サーバ通知をサポートするかどうかを指定します。

構文 **qaagent -push_notifications** { **enabled** | **disabled** } ...

デフォルト 有効 (enabled)。

説明 通知を使用しない場合は、このオプションを無効 (disabled) 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。

この設定の詳細については、「簡単なメッセージング・シナリオ」7 ページを参照してください。

dblsn.exe 実行プログラム (Listener) は、Windows 95 および Windows NT ではサポートされていません。

UDP を使用している場合、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。

SMS を介して Push 通知を使用する方法については、「SMS を介した Push 通知の使用方法」54 ページを参照してください。

-q オプション

機能　QAnywhere Agent をクワイエット・モードで起動します。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

説明　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」117 ページ

-qi オプション

QAnywhere Agent をクワイエット・モードで起動します。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

説明　QAnywhere Agent をクワイエット・モードで起動すると、Windows デスクトップ上ではメイン・ウィンドウがシステム・トレイ内に最小化され、Windows CE 上ではメイン・ウィンドウが非表示になります。また、メッセージ・ストア用のデータベース・サーバが、-qi オプションを指定した状態で起動されます。

Windows CE では、同時実行可能なプロセス数が 32 個に制限されており、この制限に達するとアプリケーションがクローズされます。クワイエット・モードを使用すると、このような状態になるのを回避できるので、一部の Windows CE アプリケーションでは有用です。クワイエット・モードでは、QAnywhere Agent をサービスのように実行できます。

-qi クワイエット・モードで実行中の QAnywhere Agent を停止するには、**qastop** を呼び出すしかありません。

参照

- 「-q オプション」117 ページ

-si オプション

機能

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。

説明

このオプションを使用するには、まず、Adaptive Server Anywhere データベースを作成する必要があります。このデータベースは、メッセージ・ストア以外の用途に使用しないでください。-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、Mobile Link 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。

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。詳細については、『ASA データベース管理ガイド』>「パスワードの変更」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・ストアは設定されているが ID は割り当てら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」46 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」124 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

機能

クライアント・メッセージ・ストアを、SQL Anywhere Studio バージョン 9.0.1 からバージョン 9.0.2 にアップグレードします。

構文

qaagent -su -c *"connection-string"* ...

説明

このオプションは、SQL Anywhere Studio 9.0.1 を使用して作成されたクライアント・メッセージ・ストアにメッセージが格納されている場合に役に立ちます。

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。処理が完了すると終了します。この操作は取り消せません。

-v オプション

機能

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、同期ウィンドウに表示する情報を指定できます。冗長レベルを上げ過ぎるとパフォーマンスに影響する可能性があるので、通常は冗長レベルを上げるのは開発段階だけにしてください。

構文

qaagent -v *levels ...*

デフォルト

最低の冗長レベル

説明

-v オプションは、ログ・ファイルと同期ウィンドウに影響を及ぼします。qaagent コマンド・ラインで -o または -ot を指定すると、メッセージ・ログだけが記録されます。

−v を単独で指定すると、少量の情報のみがログに出力されます。

levels の値は次のとおりです。-vlm のように、一度に 1 つ以上のオプションを指定することもできます。

- **+** すべてのロギング・オプションを有効にします。
- **l** Mobile Link Listener のロギングをすべて表示します。このオプションを指定すると、Mobile Link Listener (dblsn) が冗長レベル -v3 で起動されます。

  詳細については、『Mobile Link サーバ起動同期ユーザーズ・ガイド』>「Listener ユーティリティ」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、Adaptive Server Anywhere 同期サーバ (dbmlsync) が冗長レベル -v+ で起動されます。

  詳細については、dbmlsync の『Mobile Link クライアント』>「-v オプション」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、QAnywhere 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」110 ページ
- 「-ot オプション」112 ページ
- 「-on オプション」111 ページ
- 「-os オプション」112 ページ

-x オプション

機能

Mobile Link 同期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**、または **https_fips**

protocol-options：*keyword=value*

説明

-x オプションは、Mobile Link 同期サーバが QAnywhere Agent と同じデバイス上に存在しない場合、必須です。

-x オプションは複数個指定できます。これによって、複数の Mobile Link 同期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 同期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、サーバ上に待機中のメッセージが存在することを知らせる通知を Mobile Link 同期サーバから受信します。Listener は、コマンド・ラインに最初に指定された Mobile Link 同期サーバだけを使用し、他の同期サーバにフェールオーバすることはありません。

参照

- Mobile Link 同期サーバとの通信用に設定可能なプロトコル・オプションの完全なリストについては、dbmlsrv9 の『Mobile Link 管理ガイド』>「-x オプション」を参照してください。
- 「通信ストリームの暗号化」127 ページ
- 『Mobile Link 管理ガイド』>「Mobile Link トランスポート・レイヤ・セキュリティ」
- 「フェールオーバ・メカニズムの設定」69 ページ

第 6 章

セキュリティ保護されたメッセージング・アプリケーションの作成

この章の内容

この章では、セキュリティ保護されたメッセージング・ソリューションの実装方法について説明します。

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

セキュリティ保護されたクライアント・メッセージ・ストアを作成するには、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」125 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」125 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 Adaptive Server Anywhere データベースを作成します。

```
dbinit mystore.db -ek key
```

次に、データベースをクライアント・メッセージ・ストアとして初期化します。

```
qaagent -id mystore -si -c
"dbf=mystore.db;dbkey=some_phrase"
```

次に、DBA 権限を持つ新規のリモート・ユーザを作成し、そのユーザのパスワードを設定します。デフォルトの QAnywhere ユーザを削除し、デフォルトの DBA ユーザのパスワードを変更します。ユーザ DBA、パスワード SQL でログインし、次の SQL 文を実行します。

```
GRANT CONNECT TO secure_user IDENTIFIED BY
secure_password
     GRANT MEMBERSHIP IN GROUP ml_qa_user_group TO
secure_user
     GRANT REMOTE dba TO secure_user
     REVOKE CONNECT FROM ml_qa_user
     GRANT CONNECT TO dba IDENTIFIED BY new_dba_password
     COMMIT
```

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=s
ecure_password"
```

リモート・アプリケーション用の接続パラメータも設定しなければなりません。たとえば、QAManager プロパティ・ファイルには次の行を含める必要があります。

```
CONNECT_PARAMS=dbn=mystore;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 Adaptive Server 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。

詳細については、『ASA データベース管理ガイド』> 「パスワードの変更」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 -ek encryption-key database-file
```

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 クライアント 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 クライアント 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
```

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 同期サーバに接続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。

詳細については、「-x オプション」121 ページを参照してください。

Mobile Link 同期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。詳細については、『Mobile Link 管理ガイド』>「Mobile Link トランスポート・レイヤ・セキュリティ」を参照してください。

別途ライセンスを取得できるオプションが必要

トランスポート・レイヤ・セキュリティを使用するには、別途ライセンスの SQL Anywhere Studio セキュリティ・オプションを入手する必要があります。このセキュリティ・オプションは、輸出規制対象品目です。

このコンポーネントの注文方法については、『SQL Anywhere Studio の紹介』>「別途ライセンスが入手可能なコンポーネント」を参照してください。

例

次の例では、QAnywhere Agent と Mobile Link 同期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 Studio セキュリティ・オプションといっしょにインストールされるサンプルの証明書を使用しています。

RSA を使用してセキュリティ保護された TCP/IP：

```
dbmlsrv9 -x
tcpip(security=rsa_tls(certificate=rsaserver.crt;certif
icate_password=test))
 qaagent -x
tcpip(security=rsa_tls(trusted_certificates=rsaroot.crt
))
```

ECC を使用してセキュリティ保護された TCP/IP：

```
dbmlsrv9 -x
tcpip(security=ecc_tls(certificate=sample.crt;certifica
te_password=tJ1#m6+W))
 qaagent -x
tcpip(security=ecc_tls(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) :

```
dbmlsrv9 -x
https(certificate=rsaserver.crt;certificate_password=te
st)
 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。

詳細については、次の項を参照してください。

- 「-mp オプション」110 ページ
- 『Mobile Link クライアント』>「Mobile Link ユーザの認証」

第 7 章

QAnywhere 転送ルール

この章の内容

この章では、転送ルールを作成する方法について説明します。サーバ上に転送ルールを作成すると、クライアントにどのメッセージをダウンロードするのかを定義できます。クライアント上に転送ルールを作成すると、サーバにどのメッセージをアップロードするのかを定義できます。

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 同期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 同期サーバとの間でメッセージを転送します。
- Mobile Link 同期サーバは、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 同期サーバに接続されているときしか実行されません。

転送ルールでは、メッセージ転送のタイミングと転送するメッセージを指定できます。メッセージ・ストアからメッセージを削除するタイミングについても、デフォルトの動作を使わずに別に指定できます。

転送ルールは、サーバ側とクライアント側に作成できます。次の各項を参照してください。

- 「クライアント側の転送ルール」133 ページ
- 「サーバ側の転送ルール」134 ページ

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。

詳細については、「スケジュール構文」136 ページと「条件構文」138 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。

詳細については、「削除ルール」153 ページを参照してください。

クライアント側の転送ルール

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。

QAnywhere メッセージは、デフォルトでは、10 秒ごとに転送されます。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、「スケジュール構文」136 ページを参照してください。

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」50 ページを参照してください。

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送信し、営業時間外は任意のメッセージを送信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 ias_CurrentDayOfWeek in ( 'Saturday', 'Sunday' ) ¥
  or ias_CurrentTime < '8:00AM' or ias_CurrentTime >
'6:00PM'
```

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 同期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・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。

サーバ転送ルールは、クライアントごとに指定します。ルールの対象となる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。

メッセージ・プロパティ・ファイルの詳細については、『Mobile Link 管理ガイド』> 「-m オプション」を参照してください。

例

次は、サーバ側の転送ルール・ファイルの例です。この例には、sample_store_id というクライアント・メッセージ・ストア ID によって識別されるクライアントだけに適用されるルールが示されています。

```
[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, that is messages less than 100
characters or
; bytes in size, should always be transmitted.
;
;    ias_CurrentTime < '8:00am' or ias_CurrentTime >
'6:00pm'
;
; Outside of business hours, messages should always be
; transmitted

auto = ias_Priority >= 7 or ias_ContentSize < 100 ¥
    or ias_CurrentTime < '8:00am' or ias_CurrentTime >
6:00pm'



[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 < '8:00am' or ias_CurrentTime >
'6:00pm'
;
; Outside of business hours, messages should always be
; transmitted

auto = tm_Subject not like '%non-business%' ¥
     or ias_CurrentTime < '8:00am' or ias_CurrentTime >
'6:00pm'
```

スケジュール構文

スケジュールは、条件が評価される時刻を指定するために使用します。指定された時刻になると、対応する条件が送信前のすべてのメッセージについて評価され、条件を満たすメッセージだけが送信されます。

構文

各ルールの形式は次のとおりです。

schedules=condition

スケジュールされた時刻になると、各メッセージに条件が適用されます。メッセージが条件を満たしていれば、送信されます。

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** AUTOMATIC を指定すると、転送可能なメッセージが存在するときは常に条件が評価されます。対応する条件を満たすメッセージが転送されます。
- **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、対応する条件が送信前のすべてのメッセージについて評価され、条件を満たすメッセージだけが送信されます。
- **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、Adaptive Server 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

- 『ASA SQL リファレンス・マニュアル』> 「CREATE EVENT 文」

条件構文

QAnywhere 条件構文は、SQL の構文に似ています。各式は、true、false、unknown のいずれかに評価されます。メッセージは、条件が true に評価された場合だけ送信されます。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。

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 operator expression*
| *arithmetic-expr* [**NOT**] **BETWEEN** *start-expr* **AND** *end-expr*
| *rule-variable* [**NOT**] **IN (** *string-literal*, ... **)**
| *rule-variable* [**NOT**] **LIKE** *pattern* [**ESCAPE** *escape-character*]
| *rule-variable* **IS** [**NOT**] **NULL**

パラメータ

- **condition** true、false、または unknown の評価を行う式。条件が true に評価されたメッセージだけが条件を満たすものと判定されます。
- **expression** 算術式または条件式。標準的なカッコの使用方法によって、式内での評価順を示します。

- 算術式は、算術式自体、算術演算子、数値識別子、算術リテラルで構成されます。

  算術演算子は次のとおりです (優先度の高い順)。

  - **+**、**-** (単項符合標識)
  - *****、**/** (乗算と除算)
  - **+**、**-** (加算と減算)

- 条件式は、条件式自体、比較演算子、論理演算子で構成されます。

  比較演算子は次のとおりです (優先度の高い順)。

  - **=** (等号)
  - **>** (より大きい)
  - **>=** (以上)
  - **<** (より小さい)
  - **<=** (以下)
  - **<>** (等しくない)

  論理演算子は次のとおりです (優先度の高い順)。

  - NOT
  - AND
  - OR

- **rule-variable** QAnywhere のルール変数は、メッセージ・ヘッダ、メッセージ・プロパティ、クライアント・ストア・プロパティのいずれかです。

  条件内のプロパティ値の型は、プロパティを設定するときに使用する型と一致します。メッセージ内に存在しないプロパティが参照された場合、その値は NULL になります。

  詳細については、「転送ルール変数」143 ページを参照してください。

- **string-literal** 文字列リテラルは、一重引用符で囲まれた文字シーケンスであり、QAnywhere Agent によって指定された文字列エンコードを使用します。

文字列リテラルの中で一重引用符を使用する場合は、一重引用符を重ね書きします。

たとえば、次の例は有効な文字列リテラルです。

```
'literal'

'literal''s'
```

- **numeric-literal** 真数値リテラルとは、小数点なしの数値のことです (57、-957、+62 など)。値の範囲は、-2^{63} ～ 2^{63} - 1、すなわち -9223372036854775808 ～ 9223372036854775807 です。

  概数値リテラルとして、指数表記の数値 (7E3、-57.9E2 など)、または小数点付の数値 (7.、-95.7、+6.2 など) も使用できます。概数値の範囲は、2.22507385850721e-308 ～ 1.79769313486231e+308 です。

- **boolean-literal** boolean 値リテラルは、TRUE または FALSE のどちらかです。

- **BETWEEN** BETWEEN 条件は、true、false、または unknown のいずれかに評価されます。NOT キーワードを指定しない場合で、*arithmetic-expr* が *start-expr* と *end-expr* の間にあれば、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**] **(** *arithmetic-expr* **>=** *start-expr* **AND** *arithmetic-expr* **<=** *end-expr* **)**

例：

- `age BETWEEN 15 AND 19` は、`age >=15 AND age <= 19` と等価です。
- `age NOT BETWEEN 15 AND 19` は、`age < 15 OR age > 19` と等価です。

- **IN** IN 条件は、次の規則に従って評価されます。

  - *rule-variable* が null 以外で、少なくとも 1 つの値と等しい場合は true。

  - *rule-variable* 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 で式が他のいずれの値とも一致しない場合は、unknown。

  - どの値も null 以外で、*rule-variable* がどの値とも一致しない場合は、FALSE。

  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。

  例：

  - `Country IN ( 'UK', 'US', 'France' )` は、Country の値が `'UK'` の場合は true、`'Peru'` の場合は false になります。これは、次の条件と同義です。

    ```
    ( Country = 'UK' )        ¥
     OR ( Country = 'US' )     ¥
     OR ( Country = 'France' )
    ```

  - `Country NOT IN ( 'UK', 'US', 'France' )` は、Country の値が `'UK'` の場合は false、`'Peru'` の場合は true になります。これは、次の条件と同義です。

    ```
    NOT (     ( Country = 'UK' )       ¥
            OR ( Country = 'US' )       ¥
            OR ( Country = 'France' ) )
    ```

  -

- **LIKE** LIKE 条件は、true、false、または unknown として評価されます。

  NOT キーワードを指定しない場合、*expression* が *pattern* に一致すれば、TRUE に評価されます。*expression* または *pattern* が NULL 値の場合、unknown 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が、UNKNOWN は変わりません。

pattern 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

例：

- `phone LIKE 12%3` は、phone が `'123'` または `'12993'` の場合は true、`'1234'` の場合は false になります。
- `word LIKE 's_d'` は、word が `'sad'` の場合は true、`'said'` の場合は false になります。
- `phone NOT LIKE '12%3'` は、phone が `'123'` または `'12993'` のときは false、`'1234'` のときは true になります。

- **エスケープ文字** *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。
例：
  - `underscored LIKE '¥_%' ESCAPE '¥'` は、underscored が `'_myvar'` の場合は true、`'myvar'` の場合は false になります。

- **IS NULL** IS NULL 条件は、rule-variable が不定の場合は true、それ以外の場合は false に評価されます。NOT キーワードを使用すると条件の意味が逆になります。IS NULL 条件が unknown に評価されることはありません。

転送ルール変数

QAnywhere 転送ルール変数は、転送ルール・ファイル内の条件構文で使用します。転送ルール変数を使用すると、転送ルールを定義および削除できます。ルール変数には、次の 3 種類があります。

- メッセージ・ヘッダ
- メッセージ・プロパティ
- メッセージ・ストア・プロパティ

メッセージ・ヘッダ

次のメッセージ・ヘッダが事前に定義されています。

- **ias_Address** メッセージのアドレス。myclient¥myqueue など。
- **ias_Originator** メッセージの送信元に対応するクライアント・メッセージ・ストア ID。
- **ias_Status** メッセージのステータス。次のいずれかの値を取ります。
  - **ias_ExpireState** メッセージが宛先に受信される前に期限切れになった。
  - **ias_FinalState** メッセージが受信されたか、期限切れになった。したがって、**>= ias_FinalState** 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_FinalState** 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_PendingState** メッセージが宛先に受信されていない。
  - **ias_Received** メッセージが宛先に受信された。
- **ias_StatusTime** メッセージが現在のステータスになった日付と時刻。

- **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_Priority** 0 ～ 9 の数字で表されたメッセージの優先度。
- **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。

メッセージ・プロパティ

QAnywhere では、C++ または .NET QAnywhere API を使用してメッセージ・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・ストアとも同期がとられます。これにより、同じクライアント・メッセージ・ストアに接続された QAnywhere Agent によって使用される転送ルールで、これらのプロパティを使用できます。開発者は、メッセージ内にメッセージ・プロパティを定義し、転送ルール内で参照します。

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。文字と数字を使用できますが、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・プロパティの管理には、次の QAManager メソッドを使用できます。

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 )
```

詳細については、「QAnywhere C++ API リファレンス」155 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#)

```
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 )
```

詳細については、「iAnywhere.QAnywhere.Client ネームスペース」225 ページを参照してください。

例

```
c++
 QAManager *mgr = ...; // Init QAManager
 QAMessage *msg = mgr->createTextMessage();
 msg->setStringProperty( "tm_Subject", "Some message
subject" );
 mgr->putMessage( "myqueue", mgr );

 c#
 QAManager mgr = ...; // init QAManager
 QAMessage msg = mgr.createTextMessage();
 msg.setStringProperty( "tm_Subject", "Some message
subject" );
 mgr.putMessage( "myqueue", msg );
```

クライアント・ストア・プロパティ

クライアント・ストア・プロパティには、次の 2 種類があります。

- 事前に定義されたプロパティ
- ユーザ定義のプロパティ

事前に定義されたクライアント・ストア・プロパティ

次のクライアント・ストア・プロパティが事前に定義されています。

- **ias_Network** 現在使用中のネットワークに関する情報。ias_Network は特殊なプロパティです。このプロパティには、デバイスで使用されている現在のネットワークに関する情報を提供する多数の組み込み属性が定義されています。これらの属性は、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、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** RAS ダイヤルアップを使用している場合、現在のダイヤルアップ名。

  - **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_CurrentDayOfWeek** 現在の曜日。

- **ias_CurrentDayOfMonth** 現在の日付 (1 ～ 31)。

- **ias_CurrentMonth** 現在の月 (1 ～ 12)。

- **ias_CurrentYear** 現在の年。

- **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。

  - yyyy/mm/dd または yyyy-mm-dd のどちらかの形式の文字列。これらの形式は解釈のときにあいまいさがありません。

  - 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_ORDER オプションに従った文字列。

詳細については、『ASA データベース管理ガイド』> 「オプションの設定」と『ASA データベース管理ガイド』> 「DATE_ORDER オプション [互換性]」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 時間形式が使用されます。

カスタムのクライアント・ストア・プロパティ

QAnywhere では、C++ または .NET QAnywhere API を使用して、ユーザ独自のクライアント・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・ストアとも同期がとられます。これにより、同じクライアント・メッセージ・ストアに接続された QAnywhere Agent によって使用される転送ルールで、これらのプロパティを使用できます。

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。文字と数字を使用できますが、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、クライアント・ストア・プロパティ名として使用することはできません。

- NULL
- TRUE
- FALSE
- NOT
- AND
- OR
- BETWEEN
- LIKE
- IN
- IS
- ESCAPE
- **ias_** で始まるすべての名前

クライアント・ストア・プロパティを管理するために、次の QAManager メソッドを使用できます。

クライアント・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 )
```

詳細については、「QAnywhere C++ API リファレンス」155 ページを参照してください。

クライアント・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
)
```

詳細については、「iAnywhere.QAnywhere.Client ネームスペース」225 ページを参照してください。

クライアント・ストア・プロパティにも属性があります。属性を指定するには、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。次の例の 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++
   setStoreStringProperty( "Object.Shape", "Round" );
   setStoreStringProperty( "Object.Color", "Blue" );
C#
   SetStoreProperty( "Object.Shape", "Round" );
   SetStoreProperty( "Object.Color", "Blue" );
```

すべてのクライアント・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++
   setStoreStringProperty( "Circle.Shape", "Round" );
   setStoreStringProperty( "Circle.Color", "Blue" );
   setStoreStringProperty( "Object.Type", "Circle" );
C#
   SetStoreProperty( "Circle.Shape", "Round" );
   SetStoreProperty( "Circle.Color", "Blue" );
   SetStoreProperty( "Object.Type", "Circle" );
```

例

この項では、転送ルール内でクライアント・ストア・プロパティを使用する方法を、C# のサンプル・コードを用いて説明します。

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。

注意 : 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`*` !...'`

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、LAN、WLAN、WWAN の 3 種類のネットワークに対応するクライアント・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・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 の Type 属性に、″My LAN Card″、″My Wireless LAN Card″、または ″My Wireless WAN Card″ のいずれかを設定して、ias_Network プロパティが、使用しているネットワークの属性を継承するようにします。

最後に、つぎの転送ルールに従って転送ルール・ファイルを作成します。

```
ias_Network.Cost < 3 or ias_Priority >= 7
```

削除ルール

削除ルールは、クライアント・メッセージ・ストアおよびサーバ・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。削除ルールは、転送ルール・ファイル内に指定します。

クライアント側の削除ルール

デフォルトでは、メッセージの最終ステータスが受信済みまたは期限切れであることが確定すると、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、受信確認後も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。

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」133 ページを参照してください。

クライアント側の転送ルール・ファイル内の削除ルール・セクションの例を以下に示します。

```
[system:delete]

 ; This rule governs when messages are deleted from the
client
 ; store
 ;
 ;    start time '1:00am' on ( 'Sunday' )
 ;
 ; Messages are deleted every Sunday at 1:00AM.
 ;
 ;    ias_Status >= ias_FinalState
 ;
 ; Typically, messages are deleted when they reach a
final
 ; state: received, unreceivable, expired, or
cancelled.

 start time '1:00am' on ( 'Sunday' ) = ias_Status >=
ias_FinalState
```

ias_Status の詳細については、「メッセージ・ヘッダ」143 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、メッセージが配信され、配信が確認されると、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。それには、サーバ側の転送ルール・ファイル内に削除セクションを記述します。

サーバ側の削除ルールは、すべての QAnywhere クライアントに適用されます。

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」134 ページを参照してください。

サーバ側の転送ルール・ファイル内の削除ルール・セクションの例を以下に示します。

```
[system:delete]

 ; This rule governs when messages are deleted from the
server
 ; store
 ;
 ;    start time '1:00am' on ( 'Sunday' )
 ;
 ; Messages are deleted every Sunday at 1:00AM.
 ;
 ;    ias_Status >= ias_FinalState
 ;
 ; Typically messages are deleted when they reach a
final
 ; status: received, unreceivable, expired or
cancelled.

 start time '1:00am' on ( 'Sunday' ) = ias_Status >=
ias_FinalState
```

ias_Status の詳細については、「メッセージ・ヘッダ」143 ページを参照してください。

第 8 章

QAnywhere C++ API リファレンス

この章の内容　この章では QAnywhere C++ API について説明します。

AcknowledgementMode クラス

構文 public **AcknowledgementMode**

説明 QAnywhere でサポートされている受信確認モードには、トランザクション志向、暗黙的、明示的の 3 つがあります。クライアント・アプリケーションでは、「QAManager クラス」のインスタンスを作成するときに受信確認モードを指定します。

メンバ AcknowledgementMode クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「EXPLICIT_ACKNOWLEDGEMENT 変数」
- 「IMPLICIT_ACKNOWLEDGEMENT 変数」
- 「TRANSACTIONAL 変数」

EXPLICIT_ACKNOWLEDGEMENT 変数

構文 const qa_short **AcknowledgementMode::EXPLICIT_ACKNOWLEDGEMENT**

説明 明示的な受信確認では、メッセージは、`QAManager` のいずれかの受信確認用メソッドが呼び出されたときに受信確認されます。

IMPLICIT_ACKNOWLEDGEMENT 変数

構文 const qa_short **AcknowledgementMode::IMPLICIT_ACKNOWLEDGEMENT**

説明 暗黙的な受信確認では、メッセージは、クライアント・アプリケーションで受信されるとすぐに受信確認されます。

TRANSACTIONAL 変数

構文 const qa_short **AcknowledgementMode::TRANSACTIONAL**

説明

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。したがって、`QAManager` の commit メソッドが呼び出されてはじめて、すべてのメッセージが受信確認されます。

MessageProperties クラス

構文

public **MessageProperties**

説明

このクラスは、QAnywhere サーバにメッセージが送信されるときに利用されるメッセージ・プロパティの定数値を定義します。

メンバ

MessageProperties クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「ABS_RETRY_TIMEOUT 変数」
- 「ADAPTER 変数」
- 「FROM_ADDR 変数」
- 「MSG_TYPE 変数」
- 「NETWORK 変数」
- 「NETWORK_STATUS 変数」
- 「RETRY_FAILED 変数」
- 「RETRY_FAILED_ADDR 変数」
- 「RETRY_FAILED_PRIORITY 変数」
- 「RETRY_MAX 変数」
- 「RETRY_TIMEOUT 変数」

ABS_RETRY_TIMEOUT 変数

構文

const qa_string **MessageProperties::ABS_RETRY_TIMEOUT**

説明

コネクタ経由で送信されるメッセージのオプションのプロパティ。リトライ・タイムアウト時間です。この時間になると、コネクタ経由の送信リトライが中止され、送信エラーになります。

ADAPTER 変数

構文

const qa_string **MessageProperties::ADAPTER**

説明

"system" キュー・メッセージのプロパティ。QAnywhere サーバに接続するときに使用できるネットワーク・アダプタを示す、デリミタ付きリストです。

FROM_ADDR 変数

構文　const qa_string **MessageProperties::FROM_ADDR**

説明　送信者のアドレスを示すオプションのプロパティ。

MSG_TYPE 変数

構文　const qa_string **MessageProperties::MSG_TYPE**

説明　メッセージのタイプを示します。このプロパティが設定されていないメッセージは、通常のデータ・メッセージ (「REGULAR 変数」) です。

参照　「MessageType クラス」

NETWORK 変数

構文　const qa_string **MessageProperties::NETWORK**

説明　"system" キュー・メッセージのプロパティ。QAnywhere サーバに接続するときに使用できるネットワーク名を示す、デリミタ付きリストです。

NETWORK_STATUS 変数

構文　const qa_string **MessageProperties::NETWORK_STATUS**

説明　"system" キュー・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。

RETRY_FAILED 変数

構文　const qa_string **MessageProperties::RETRY_FAILED**

説明

メッセージが RetryFailedAddress に送信されるときに、コネクタによって設定されます。受信側のクライアントは、このプロパティを参照して、再送信に失敗したメッセージを特定できます。

RETRY_FAILED_ADDR 変数

構文

const qa_string **MessageProperties::RETRY_FAILED_ADDR**

説明

コネクタ経由で送信されるメッセージのオプションのプロパティ。このプロパティが設定されている場合、RetryMax または RetryTimeout に設定された値を超えると、設定されているアドレスにメッセージが送信されます。

RETRY_FAILED_PRIORITY 変数

構文

const qa_string **MessageProperties::RETRY_FAILED_PRIORITY**

説明

コネクタ経由で送信されるメッセージのオプションのプロパティ。メッセージが RetryFailedAddress に送信されるとき、メッセージ優先度はこの値に設定されます。

RETRY_MAX 変数

構文

const qa_string **MessageProperties::RETRY_MAX**

説明

コネクタ経由で送信されるメッセージのオプションのプロパティ。コネクタでの送信リトライ回数の最大値です。この回数を超えると送信は失敗します。

RETRY_TIMEOUT 変数

構文

const qa_string **MessageProperties::RETRY_TIMEOUT**

説明

コネクタ経由で送信されるメッセージのオプションのプロパティ。リトライ・タイムアウト時間です。この時間が経過すると、送信リトライが中止され、送信は失敗します。

MessageType クラス

構文　public **MessageType**

説明　このクラスは、メッセージ・プロパティ "ias_MessageType" の定数値を定義します。

参照　「MSG_TYPE 変数」

メンバ　MessageType クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「NETWORK_STATUS_NOTIFICATION 変数」
- 「PUSH_NOTIFICATION 変数」
- 「REGULAR 変数」

NETWORK_STATUS_NOTIFICATION 変数

構文　const qa_int **MessageType::NETWORK_STATUS_NOTIFICATION**

説明　ネットワーク・ステータス通知メッセージ。ネットワーク・ステータスの変化を示します。

PUSH_NOTIFICATION 変数

構文　const qa_int **MessageType::PUSH_NOTIFICATION**

説明　Push 通知メッセージ。QAnywhere サーバから送信されるのを待機しているメッセージが存在することを示します。

REGULAR 変数

構文　const qa_int **MessageType::REGULAR**

説明　通常のデータ・メッセージ。

QABinaryMessage クラス

構文

public **QABinaryMessage**

基本クラス

- 「QAMessage クラス」

説明

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。このバイト・ストリームを解釈するのはメッセージの受信側です。

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。reset が最初に呼び出されたときに、メッセージ本文が読み込み専用モードになります。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。プロバイダは、メッセージが受信された時点で reset を呼び出して、メッセージ本文をクライアントに対し読み込み専用にします。

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。

メンバ

QABinaryMessage クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「castToBinaryMessage 関数」
- 「castToTextMessage 関数」
- 「clearProperties 関数」
- 「DEFAULT_PRIORITY 変数」
- 「DEFAULT_TIME_TO_LIVE 変数」
- 「getAddress 関数」
- 「getBodyLength 関数」
- 「getBooleanProperty 関数」
- 「getByteProperty 関数」
- 「getDoubleProperty 関数」
- 「getExpiration 関数」
- 「getFloatProperty 関数」

- 「getInReplyToID 関数」
- 「getIntProperty 関数」
- 「getLongProperty 関数」
- 「getMessageID 関数」
- 「getPriority 関数」
- 「getPropertyNames 関数」
- 「getPropertyType 関数」
- 「getRedelivered 関数」
- 「getReplyToAddress 関数」
- 「getShortProperty 関数」
- 「getStringProperty 関数」
- 「getStringProperty 関数」
- 「getTimestamp 関数」
- 「getTimestampAsString 関数」
- 「propertyExists 関数」
- 「readBinary 関数」
- 「readBoolean 関数」
- 「readByte 関数」
- 「readChar 関数」
- 「readDouble 関数」
- 「readFloat 関数」
- 「readInt 関数」
- 「readLong 関数」
- 「readShort 関数」
- 「readString 関数」
- 「reset 関数」
- 「setAddress 関数」
- 「setBooleanProperty 関数」
- 「setByteProperty 関数」
- 「setDoubleProperty 関数」
- 「setFloatProperty 関数」
- 「setInReplyToID 関数」
- 「setIntProperty 関数」
- 「setLongProperty 関数」
- 「setMessageID 関数」
- 「setPriority 関数」
- 「setRedelivered 関数」
- 「setReplyToAddress 関数」

- 「setShortProperty 関数」
- 「setStringProperty 関数」
- 「setTimestamp 関数」
- 「writeBinary 関数」
- 「writeBoolean 関数」
- 「writeByte 関数」
- 「writeChar 関数」
- 「writeDouble 関数」
- 「writeFloat 関数」
- 「writeInt 関数」
- 「writeLong 関数」
- 「writeShort 関数」
- 「writeString 関数」
- 「~QABinaryMessage 関数」
- 「~QAMessage 関数」

getBodyLength 関数

構文

virtual qa_long **QABinaryMessage::getBodyLength()**

説明

メッセージ本文の qa_bytes のサイズを返します。

readBinary 関数

構文

virtual 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。

説明

バイト・メッセージ・ストリームの一部を読み込みます。

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

virtual 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。

説明　バイト・メッセージ・ストリームから `qa_bool` 値を読み込みます。

戻り値　処理が正常終了した場合のみ true。

readByte 関数

構文

virtual 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。

説明　バイト・メッセージ・ストリームから符合付き 8 ビット値を読み込みます。

戻り値　処理が正常終了した場合のみ true。

readChar 関数

構文

virtual 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。

説明

バイト・メッセージ・ストリームから文字値を読み込みます。

戻り値

処理が正常終了した場合のみ true。

readDouble 関数

構文

virtual 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 qa_double 値の格納先。

説明

バイト・メッセージ・ストリームから double 値を読み込みます。

戻り値

処理が正常終了した場合のみ true。

readFloat 関数

構文

virtual 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 qa_float 値の格納先。

説明　バイト・メッセージ・ストリームから float 値を読み込みます。

戻り値　処理が正常終了した場合のみ true。

readInt 関数

構文

```
virtual 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。

説明　バイト・メッセージ・ストリームから符合付き 32 ビット整数を読み込みます。

戻り値　処理が正常終了した場合のみ true。

readLong 関数

構文

```
virtual 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 `qa_long` 値の格納先。

説明　バイト・メッセージ・ストリームから符合付き 64 ビット整数を読み込みます。

戻り値　処理が正常終了した場合のみ true。

readShort 関数

構文

virtual 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。

説明

バイト・メッセージ・ストリームから符合付き 16 ビットの数値を読み込みます。

戻り値

処理が正常終了した場合のみ true。

readString 関数

構文

virtual 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 qa_char の最大数 (null 終端子も含みます)。

説明

バイト・メッセージ・ストリームから文字列を読み込みます。

戻り値

バッファに読み込まれた null 以外の qa_char の合計バイト数。読み込めるデータが残っていない場合、またはバッファが小さ過ぎる場合は -1。

reset 関数

構文

virtual void **QABinaryMessage::reset()**

説明　メッセージ本文を読み込み専用モードにし、読み込み位置をバイト・ストリームの先頭に戻します。

writeBinary 関数

構文

virtual void **QABinaryMessage::writeBinary(**
qa_const_bytes *value*
qa_int *offset*
qa_int *length*
)

パラメータ

- **value** 書き出されるバイト配列値。
- **offset** バイト配列内の初期オフセット。
- **length** 書き出されるバイト数。

説明　バイト配列の一部をバイト・メッセージ・ストリームに書き出します。

writeBoolean 関数

構文

virtual void **QABinaryMessage::writeBoolean(**
qa_bool *value*
)

パラメータ

- **value** 書き出される `qa_bool` 値。

説明　`qa_bool` 値を、1 バイトの値としてバイト・メッセージ・ストリームに書き出します。値 `true` は値 `(qa_byte)1` として、値 `false` は値 `(qa_byte)0` として書き出されます。

writeByte 関数

構文

virtual void **QABinaryMessage::writeByte(**
qa_byte *value*
)

パラメータ

- **value** 書き出される qa_byte 値。

説明

qa_byte 値を、1 バイトの値としてバイト・メッセージ・ストリームに書き出します。

writeChar 関数

構文

virtual void **QABinaryMessage::writeChar(**
qa_char *value*
)

パラメータ

- **value** 書き出される qa_char 値。

説明

qa_char 値を、2 バイトの値として、高位バイトから順にバイト・メッセージ・ストリームに書き出します。

writeDouble 関数

構文

virtual void **QABinaryMessage::writeDouble(**
qa_double *value*
)

パラメータ

- **value** 書き出される qa_double 値。

説明

引数に指定された qa_double 値を qa_long 値に変換し、得られた qa_long 値を、8 バイトの値として、高位バイトから順にバイト・メッセージ・ストリームに書き出します。

writeFloat 関数

構文

virtual void **QABinaryMessage::writeFloat(**
qa_float *value*
)

パラメータ

- **value** 書き出される qa_float 値。

説明　引数に指定された qa_float 値を qa_int 値に変換し、得られた qa_int 値を、4 バイトの値として、高位バイトから順にバイト・メッセージ・ストリームに書き出します。

writeInt 関数

構文

virtual void **QABinaryMessage::writeInt(**
    qa_int *value*
)

パラメータ

- **value** 書き出される qa_int 値。

説明　qa_int 値を、4 バイトの値として、高位バイトから順にバイト・メッセージ・ストリームに書き出します。

writeLong 関数

構文

virtual void **QABinaryMessage::writeLong(**
    qa_long *value*
)

パラメータ

- **value** 書き出される qa_long 値。

説明　qa_long 値を、8 バイトの値として、高位バイトから順にバイト・メッセージ・ストリームに書き出します。

writeShort 関数

構文

virtual void **QABinaryMessage::writeShort(**
    qa_short *value*
)

パラメータ

- **value** 書き出される qa_short 値。

説明　qa_short 値を、2 バイトの値として、高位バイトから順にバイト・メッセージ・ストリームに書き出します。

writeString 関数

構文

virtual void **QABinaryMessage::writeString(**
qa_const_string *value*
)

パラメータ

- **value** 書き出される文字列。

説明

文字列をバイト・メッセージ・ストリームに書き出します。

~QABinaryMessage 関数

構文

virtual **QABinaryMessage::~QABinaryMessage()**

説明

仮想デストラクタ。

QAError クラス

構文

public **QAError**

説明

このクラスは、QAnywhere クライアント関連の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 クラス 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、getLastError 関数 によって返されたエラー・コードを解釈するときに、これらのエラー定数を使用します。

参照

「getLastErrorMsg 関数」

メンバ

QAError クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「COMMON_GET_INIT_FILE_ERROR 変数」
- 「COMMON_INIT_ERROR 変数」
- 「COMMON_INIT_THREAD_ERROR 変数」
- 「COMMON_INVALID_PROPERTY 変数」
- 「COMMON_MSG_NOT_WRITEABLE_ERROR 変数」
- 「COMMON_MSG_RETRIEVE_ERROR 変数」
- 「COMMON_MSG_STORE_ERROR 変数」
- 「COMMON_MSG_STORE_NOT_INITIALIZED 変数」
- 「COMMON_MSG_STORE_TOO_LARGE 変数」
- 「COMMON_NO_DEST_ERROR 変数」
- 「COMMON_NO_IMPLEMENTATION 変数」
- 「COMMON_OPEN_ERROR 変数」
- 「COMMON_OPEN_LOG_FILE_ERROR 変数」
- 「COMMON_TERMINATE_ERROR 変数」
- 「COMMON_UNEXPECTED_EOM_ERROR 変数」
- 「QA_NO_ERROR 変数」
- 「~QAError 関数」

COMMON_GET_INIT_FILE_ERROR 変数

構文 const qa_int **QAError::COMMON_GET_INIT_FILE_ERROR**

説明 クライアント・プロパティ・ファイルにアクセスできません。

COMMON_INIT_ERROR 変数

構文 const qa_int **QAError::COMMON_INIT_ERROR**

説明 初期化エラーです。

COMMON_INIT_THREAD_ERROR 変数

構文 const qa_int **QAError::COMMON_INIT_THREAD_ERROR**

説明 バックグラウンド・スレッドを初期化するときにエラーが発生しました。

COMMON_INVALID_PROPERTY 変数

構文 const qa_int **QAError::COMMON_INVALID_PROPERTY**

説明 クライアント・プロパティ・ファイル内に無効なプロパティが存在します。

COMMON_MSG_NOT_WRITEABLE_ERROR 変数

構文 const qa_int **QAError::COMMON_MSG_NOT_WRITEABLE_ERROR**

説明 メッセージは書き込み禁止です。

COMMON_MSG_RETRIEVE_ERROR 変数

構文　const qa_int **QAError::COMMON_MSG_RETRIEVE_ERROR**

説明　メッセージ・ストアからメッセージを取り出すときにエラーが発生しました。

COMMON_MSG_STORE_ERROR 変数

構文　const qa_int **QAError::COMMON_MSG_STORE_ERROR**

説明　メッセージ・ストアにメッセージを格納するときにエラーが発生しました。

COMMON_MSG_STORE_NOT_INITIALIZED 変数

構文　const qa_int **QAError::COMMON_MSG_STORE_NOT_INITIALIZED**

説明　メッセージ・ストアがメッセージング用に初期化されていません。

COMMON_MSG_STORE_TOO_LARGE 変数

構文　const qa_int **QAError::COMMON_MSG_STORE_TOO_LARGE**

説明　メッセージ・ストアが大き過ぎて、デバイス上のディスクの空き領域に収まりません。

COMMON_NO_DEST_ERROR 変数

構文　const qa_int **QAError::COMMON_NO_DEST_ERROR**

説明　送信先が指定されていません。

COMMON_NO_IMPLEMENTATION 変数

構文 const qa_int **QAError::COMMON_NO_IMPLEMENTATION**

説明 関数が実装されていません。

COMMON_OPEN_ERROR 変数

構文 const qa_int **QAError::COMMON_OPEN_ERROR**

説明 メッセージ・ストアへの接続をオープンするときにエラーが発生しました。

COMMON_OPEN_LOG_FILE_ERROR 変数

構文 const qa_int **QAError::COMMON_OPEN_LOG_FILE_ERROR**

説明 ログ・ファイルをオープンするときにエラーが発生しました。

COMMON_TERMINATE_ERROR 変数

構文 const qa_int **QAError::COMMON_TERMINATE_ERROR**

説明 終了エラーです。

COMMON_UNEXPECTED_EOM_ERROR 変数

構文 const qa_int **QAError::COMMON_UNEXPECTED_EOM_ERROR**

説明 予期しないところでメッセージの終わりに到達しました。

QA_NO_ERROR 変数

構文 const qa_int **QAError::QA_NO_ERROR**

説明　　エラーはありません。

~QAError 関数

構文　　virtual **QAError::~QAError()**

説明　　仮想デストラクタ。

QAManager クラス

構文

public **QAManager**

基本クラス

- 「QAManagerBase クラス」

説明

このクラスは、非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。

メンバ

QAManager クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「acknowledge 関数」
- 「acknowledgeAll 関数」
- 「acknowledgeUntil 関数」
- 「close 関数」
- 「createBinaryMessage 関数」
- 「createTextMessage 関数」
- 「deleteMessage 関数」
- 「getBooleanStoreProperty 関数」
- 「getByteStoreProperty 関数」
- 「getDoubleStoreProperty 関数」
- 「getFloatStoreProperty 関数」
- 「getIntStoreProperty 関数」
- 「getLastError 関数」
- 「getLastErrorMsg 関数」
- 「getLongStoreProperty 関数」
- 「getMessage 関数」
- 「getMessageNoWait 関数」
- 「getMessageTimeout 関数」
- 「getMode 関数」
- 「getShortStoreProperty 関数」
- 「getStringStoreProperty 関数」
- 「open 関数」
- 「peekFirstMessage 関数」
- 「peekNextMessage 関数」
- 「publishMessage 関数」
- 「putMessage 関数」

- 「putMessageTimeToLive 関数」
- 「recover 関数」
- 「setBooleanStoreProperty 関数」
- 「setByteStoreProperty 関数」
- 「setDoubleStoreProperty 関数」
- 「setFloatStoreProperty 関数」
- 「setIntStoreProperty 関数」
- 「setLongStoreProperty 関数」
- 「setMessageListener 関数」
- 「setProperty 関数」
- 「setShortStoreProperty 関数」
- 「setStringStoreProperty 関数」
- 「start 関数」
- 「stop 関数」
- 「triggerSendReceive 関数」
- 「~QAManager 関数」
- 「~QAManagerBase 関数」

acknowledge 関数

構文

virtual qa_bool **QAManager::acknowledge(**
QAMessage * *msg*
)

パラメータ

- **msg** メッセージ。

説明

指定されたメッセージを受信確認します。

戻り値

処理が正常終了した場合のみ true。

acknowledgeAll 関数

構文

virtual qa_bool **QAManager::acknowledgeAll()**

説明

すべてのメッセージを受信確認します。

戻り値

処理が正常終了した場合のみ true。

acknowledgeUntil 関数

構文

virtual qa_bool **QAManager::acknowledgeUntil(**
QAMessage * *msg*
)

パラメータ

- **msg** メッセージ。

説明

指定されたメッセージと、それより前のメッセージを受信確認します。

戻り値

処理が正常終了した場合のみ true。

open 関数

構文

virtual qa_bool **QAManager::open(**
qa_short *mode*
)

パラメータ

- **mode** 受信確認モード。

説明

指定された受信確認モードを使用して QAManager をオープンします。

戻り値

処理が正常終了した場合のみ true。

参照

「AcknowledgementMode クラス」

recover 関数

構文

virtual qa_bool **QAManager::recover()**

説明

受信確認されていないメッセージのすべてを未受信状態に戻します。

戻り値

処理が正常終了した場合のみ true。

~QAManager 関数

構文　　virtual **QAManager::~QAManager()**

説明　　仮想デストラクタ。

QAManagerBase クラス

構文

public **QAManagerBase**

派生クラス

- 「QAManager クラス」
- 「QATransactionalManager クラス」

説明

このクラスは、QATransactionalManager クラス と QAManager クラス 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。アプリケーションのスレッドごとに 1 つの QAManagerBase インスタンスが存在している必要があります。このクラスは、メッセージ作成用ファクトリです。getMessage 関数 メソッドはメッセージの作成も行うため、このクラスは、ユーザがメソッドを呼び出した時点または QAManagerBase がクローズされた時点でメモリからメッセージが解放されるように、すべてのメッセージを管理します。パブリッシュ／サブスクライブ・モデルはサポートされていないため、publishMessage 関数 メソッドは常に false を返します。

メンバ

QAManagerBase クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「close 関数」
- 「createBinaryMessage 関数」
- 「createTextMessage 関数」
- 「deleteMessage 関数」
- 「getBooleanStoreProperty 関数」
- 「getByteStoreProperty 関数」
- 「getDoubleStoreProperty 関数」
- 「getFloatStoreProperty 関数」
- 「getIntStoreProperty 関数」
- 「getLastError 関数」
- 「getLastErrorMsg 関数」
- 「getLongStoreProperty 関数」
- 「getMessage 関数」
- 「getMessageNoWait 関数」
- 「getMessageTimeout 関数」
- 「getMode 関数」

- 「getShortStoreProperty 関数」
- 「getStringStoreProperty 関数」
- 「peekFirstMessage 関数」
- 「peekNextMessage 関数」
- 「publishMessage 関数」
- 「putMessage 関数」
- 「putMessageTimeToLive 関数」
- 「setBooleanStoreProperty 関数」
- 「setByteStoreProperty 関数」
- 「setDoubleStoreProperty 関数」
- 「setFloatStoreProperty 関数」
- 「setIntStoreProperty 関数」
- 「setLongStoreProperty 関数」
- 「setMessageListener 関数」
- 「setProperty 関数」
- 「setShortStoreProperty 関数」
- 「setStringStoreProperty 関数」
- 「start 関数」
- 「stop 関数」
- 「triggerSendReceive 関数」
- 「~QAManagerBase 関数」

close 関数

構文　virtual qa_bool **QAManagerBase::close()**

説明　QAManagerBase をクローズします。これにより、QAManagerBase インスタンスに関連付けられているリソースがすべて解放されます。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。新規のインスタンスを作成し、オープン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。

createBinaryMessage 関数

構文　virtual QABinaryMessage * **QAManagerBase::createBinaryMessage()**

説明 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・ストリームが含まれるメッセージの送信に使用します。

戻り値 作成されたメッセージ。

createTextMessage 関数

構文 virtual QATextMessage * **QAManagerBase::createTextMessage()**

説明 QATextMessage オブジェクトを作成します。QATextMessage オブジェクトは、qa_string 値が含まれるメッセージの送信に使用します。

戻り値 作成されたメッセージ。

deleteMessage 関数

構文

virtual void **QAManagerBase::deleteMessage(**
QAMessage * *msg*
)

パラメータ

- **msg** 削除されるメッセージ。

説明 QAMessage オブジェクトを削除します。デフォルトでは、前述の作成用メソッド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。

getBooleanStoreProperty 関数

構文

virtual qa_bool **QAManagerBase::getBooleanStoreProperty(**
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。

説明　指定された名前を持つ qa_bool 型メッセージ・ストア・プロパティの値を取得します。

戻り値　処理が正常終了した場合のみ true。

getByteStoreProperty 関数

構文

virtual qa_bool **QAManagerBase::getByteStoreProperty(**
    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。

説明　指定された名前を持つ qa_byte 型メッセージ・ストア・プロパティの値を取得します。

戻り値　処理が正常終了した場合のみ true。

getDoubleStoreProperty 関数

構文

virtual qa_bool **QAManagerBase::getDoubleStoreProperty(**
    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。

説明　指定された名前を持つ qa_double 型メッセージ・ストア・プロパティの値を取得します。

戻り値　処理が正常終了した場合のみ true。

getFloatStoreProperty 関数

構文

virtual qa_bool **QAManagerBase::getFloatStoreProperty(**
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。

説明

指定された名前を持つ `qa_float` 型メッセージ・ストア・プロパティの値を取得します。

戻り値

処理が正常終了した場合のみ true。

getIntStoreProperty 関数

構文

virtual qa_bool **QAManagerBase::getIntStoreProperty(**
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。

説明

指定された名前を持つ `qa_int` 型メッセージ・ストア・プロパティの値を取得します。

戻り値

処理が正常終了した場合のみ true。

getLastError 関数

構文

virtual qa_int **QAManagerBase::getLastError()**

説明

一番最後に失敗したメソッド呼び出しのエラー・コードを取得します。

戻り値　エラー・コード。

getLastErrorMsg 関数

構文　virtual qa_string **QAManagerBase::getLastErrorMsg()**

説明　エラー・コードに対応するエラー・メッセージを取得します。

戻り値　エラー・メッセージ。

getLongStoreProperty 関数

構文

virtual qa_bool **QAManagerBase::getLongStoreProperty(**
    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。

説明　指定された名前を持つ `qa_long` 型メッセージ・ストア・プロパティの値を取得します。

戻り値　処理が正常終了した場合のみ true。

getMessage 関数

構文

virtual QAMessage * **QAManagerBase::getMessage(**
    qa_const_string *dest*
)

パラメータ

- **dest** 送信先。

説明　キューに登録されている、指定された送信先を持つメッセージのうち、次に取り出せるものを取り出します。キュー内にメッセージがない場合は、メッセージが登録されるまで待機します。

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 null。

getMessageNoWait 関数

構文

virtual QAMessage * **QAManagerBase::getMessageNoWait(**
qa_const_string *dest*
)

パラメータ

- **dest** 送信先。

説明　キューに登録されている、指定された送信先を持つメッセージのうち、次に取り出せるものを取り出します。キュー内にメッセージがない場合は、qa_null を返します。

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 null。

getMessageTimeout 関数

構文

virtual QAMessage * **QAManagerBase::getMessageTimeout(**
qa_const_string *dest*
qa_long *timeout*
)

パラメータ

- **dest** 送信先。
- **timeout** 最大待ち時間 (ミリ秒)。

説明　キューに登録されている、指定された送信先を持つメッセージのうち、次に取り出せるものを取り出します。キュー内にメッセージがない場合は、timeout に指定された時間の範囲内で待機します。

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 null。

getMode 関数

構文

virtual qa_short **QAManagerBase::getMode()**

説明　QAManagerBase インスタンスの受信確認モードを取得します。受信確認モードは、IMPLICIT_ACKNOWLEDGE、EXPLICIT_ACKNOWLEDGE、TRANSACTIONAL のいずれかです。

戻り値　受信確認モード。

参照　「AcknowledgementMode クラス」

getShortStoreProperty 関数

構文

virtual qa_bool **QAManagerBase::getShortStoreProperty(**
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。

説明　指定された名前を持つ `qa_short` 型メッセージ・ストア・プロパティの値を取得します。

戻り値　処理が正常終了した場合のみ true。

getStringStoreProperty 関数

構文

virtual qa_int **QAManagerBase::getStringStoreProperty(**
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** 取得した値からコピーする qa_char の最大数 (null 終端子も含みます)。

説明 指定された名前を持つメッセージ・ストア・プロパティの値を取得します。

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。

peekFirstMessage 関数

構文

virtual QAMessage * **QAManagerBase::peekFirstMessage(**
qa_const_string *dest*
)

パラメータ

- **dest** 送信先。

説明 キューに登録されている、指定された送信先を持つメッセージのうち、最初のものを参照します。キュー内にメッセージがない場合は、qa_null を返します。このメソッドは peekNextMessage メソッドよりも前に呼び出します。peekNextMessage メソッドを呼び出すと、peekFirstMessage メソッドが呼び出された時点でキューに登録されていた、指定された送信先を持つメッセージが列挙されます。

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 null。

peekNextMessage 関数

構文

virtual QAMessage * **QAManagerBase::peekNextMessage()**

説明 キューに登録されている、指定された送信先を持つメッセージのうち、次のものを参照します。キュー内にメッセージがない場合は、qa_null を返します。このメソッドは peekFirstMessage メソッドの後に呼び出します。このメソッドを呼び出すと、peekFirstMessage が呼び出された時点でキューに登録されていた、指定された送信先を持つメッセージが列挙されます。

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 null。

publishMessage 関数

構文

virtual qa_bool **QAManagerBase::publishMessage(**
qa_const_string *dest*
QAMessage * *msg*
)

パラメータ

- **dest** 送信先。
- **msg** メッセージ。

説明

未実装。

putMessage 関数

構文

virtual qa_bool **QAManagerBase::putMessage(**
qa_const_string *dest*
QAMessage * *msg*
)

パラメータ

- **dest** 送信先。
- **msg** メッセージ。

説明

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

virtual qa_bool **QAManagerBase::putMessageTimeToLive(**
qa_const_string *dest*
QAMessage * *msg*
qa_long *ttl*
)

パラメータ

- **dest** 送信先。
- **msg** メッセージ。

- **ttl** 存続時間 (ミリ秒)。

説明 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

virtual qa_bool **QAManagerBase::setBooleanStoreProperty(**
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。

説明 指定された名前を持つ `qa_bool` 型メッセージ・ストア・プロパティの値を設定します。

戻り値 処理が正常終了した場合のみ true。

setByteStoreProperty 関数

構文

virtual qa_bool **QAManagerBase::setByteStoreProperty(**
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。

説明 指定された名前を持つ `qa_byte` 型メッセージ・ストア・プロパティの値を設定します。

戻り値 処理が正常終了した場合のみ true。

setDoubleStoreProperty 関数

構文

virtual qa_bool **QAManagerBase::setDoubleStoreProperty(**
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。

説明

指定された名前を持つ `qa_double` 型メッセージ・ストア・プロパティの値を設定します。

戻り値

処理が正常終了した場合のみ true。

setFloatStoreProperty 関数

構文

virtual qa_bool **QAManagerBase::setFloatStoreProperty(**
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。

説明

指定された名前を持つ `qa_float` 型メッセージ・ストア・プロパティの値を設定します。

戻り値

処理が正常終了した場合のみ true。

setIntStoreProperty 関数

構文

virtual qa_bool **QAManagerBase::setIntStoreProperty(**
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。

説明 指定された名前を持つ qa_int 型メッセージ・ストア・プロパティの値を設定します。

戻り値 処理が正常終了した場合のみ true。

setLongStoreProperty 関数

構文

virtual qa_bool **QAManagerBase::setLongStoreProperty(**
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。

説明 指定された名前を持つ qa_long 型メッセージ・ストア・プロパティの値を設定します。

戻り値 処理が正常終了した場合のみ true。

setMessageListener 関数

構文

virtual void **QAManagerBase::setMessageListener(**
qa_const_string *dest*
QAMessageListener * *listener*
)

パラメータ

- **dest** リスナに適用される送信先アドレス。
- **listener** 送信先アドレス dest に関連付けられるメッセージ・リスナ。

説明 指定されたメッセージ・リスナを、特定の送信先に関連付けます。

setProperty 関数

構文

virtual qa_bool **QAManagerBase::setProperty(**
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** プロパティの値。

説明

指定されたプロパティに、指定された値を設定します。QAManagerBase の作成時にプロパティ・ファイルを指定する代わりに、このメソッドを使用して QAManagerBase の各プロパティを設定できます。各プロパティは、派生クラスの open() メソッドを呼び出す前に設定しておく必要があります。

戻り値

処理が正常終了した場合のみ true。

setShortStoreProperty 関数

構文

virtual qa_bool **QAManagerBase::setShortStoreProperty(**
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。

説明

指定された名前を持つ `qa_short` 型メッセージ・ストア・プロパティの値を設定します。

戻り値

処理が正常終了した場合のみ true。

setStringStoreProperty 関数

構文

virtual qa_bool **QAManagerBase::setStringStoreProperty(**
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。

説明

指定されたメッセージ・ストア・プロパティに、指定された値を設定します。

戻り値

処理が正常終了した場合のみ true。

start 関数

構文

virtual qa_bool **QAManagerBase::start()**

説明

着信メッセージを受信するための QAManagerBase を開始します。

戻り値

処理が正常終了した場合のみ true。

stop 関数

構文

virtual qa_bool **QAManagerBase::stop()**

説明

QAManagerBase による着信メッセージの受信を停止します。

戻り値

処理が正常終了した場合のみ true。

triggerSendReceive 関数

構文

virtual qa_bool **QAManagerBase::triggerSendReceive()**

説明　保留中のメッセージの送信または受信を開始します。ローカルのキューに登録されているメッセージとサーバのキューに登録されているローカル宛のメッセージの両方が対象になります。

戻り値　処理が正常終了した場合のみ true。

~QAManagerBase 関数

構文　virtual **QAManagerBase::~QAManagerBase()**

説明　仮想デストラクタ。

QAManagerFactory クラス

構文

public **QAManagerFactory**

説明

このクラスは、QATransactionalManager クラス オブジェクトまたは QAManager クラス オブジェクトを作成するためのファクトリ・クラスです。

メンバ

QAManagerFactory クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- createQAManager 関数
- 「createQATransactionalManager 関数」
- 「deleteQAManager 関数」
- 「deleteQATransactionalManager 関数」
- 「getLastError 関数」
- 「getLastErrorMsg 関数」
- ~QAManagerFactory 関数

createQAManager 関数

構文

virtual 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。

説明

指定されたプロパティを持つ新規 QAManager クラス インスタンスを返します。

戻り値

QAManager クラス インスタンス。

createQATransactionalManager 関数

構文

virtual 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。

説明

指定されたプロパティを持つ新規 QATransactionalManager クラス インスタンスを返します。

戻り値

QATransactionalManager クラス インスタンス。

deleteQAManager 関数

構文

virtual void **QAManagerFactory::deleteQAManager(**
QAManager * *mgr*
)

パラメータ

- **mgr** 破棄される QAManager クラス。

説明

QAManager クラス 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。

deleteQATransactionalManager 関数

構文

virtual void **QAManagerFactory::deleteQATransactionalManager(**
QATransactionalManager * *mgr*
)

パラメータ

- **mgr** 破棄される QATransactionalManager クラス。

説明

「QATransactionalManager クラス」を破棄し、そのリソースを解放します。作成済みの QATransactionalManager 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。

getLastError 関数

構文

virtual qa_int **QAManagerFactory::getLastError()**

説明

一番最後に失敗したメソッド呼び出しのエラー・コードを取得します。

戻り値

エラー・コード。

getLastErrorMsg 関数

構文

virtual qa_string **QAManagerFactory::getLastErrorMsg()**

説明

エラー・コードに対応するエラー・メッセージを取得します。

戻り値

エラー・メッセージ。

~QAManagerFactory 関数

構文

virtual **QAManagerFactory::~QAManagerFactory()**

説明

仮想デストラクタ。

QAMessage クラス

構文　public **QAMessage**

派生クラス

- 「QABinaryMessage クラス」
- 「QATextMessage クラス」

説明

QAMessage インタフェースは、すべての QAnywhere クライアント・メッセージのルート・インタフェースです。

メンバ

QAMessage クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「castToBinaryMessage 関数」
- 「castToTextMessage 関数」
- 「clearProperties 関数」
- 「DEFAULT_PRIORITY 変数」
- 「DEFAULT_TIME_TO_LIVE 変数」
- 「getAddress 関数」
- 「getBooleanProperty 関数」
- 「getByteProperty 関数」
- 「getDoubleProperty 関数」
- 「getExpiration 関数」
- 「getFloatProperty 関数」
- 「getInReplyToID 関数」
- 「getIntProperty 関数」
- 「getLongProperty 関数」
- 「getMessageID 関数」
- 「getPriority 関数」
- 「getPropertyNames 関数」
- 「getPropertyType 関数」
- 「getRedelivered 関数」
- 「getReplyToAddress 関数」
- 「getShortProperty 関数」
- 「getStringProperty 関数」
- 「getStringProperty 関数」
- 「getTimestamp 関数」
- 「getTimestampAsString 関数」

- 「propertyExists 関数」
- 「setAddress 関数」
- 「setBooleanProperty 関数」
- 「setByteProperty 関数」
- 「setDoubleProperty 関数」
- 「setFloatProperty 関数」
- 「setInReplyToID 関数」
- 「setIntProperty 関数」
- 「setLongProperty 関数」
- 「setMessageID 関数」
- 「setPriority 関数」
- 「setRedelivered 関数」
- 「setReplyToAddress 関数」
- 「setShortProperty 関数」
- 「setStringProperty 関数」
- 「setTimestamp 関数」
- 「~QAMessage 関数」

DEFAULT_PRIORITY 変数

構文 const qa_int **QAMessage::DEFAULT_PRIORITY**

説明 デフォルトのメッセージ優先度。

DEFAULT_TIME_TO_LIVE 変数

構文 const qa_long **QAMessage::DEFAULT_TIME_TO_LIVE**

説明 メッセージのデフォルトの存続時間。

castToBinaryMessage 関数

構文 virtual QABinaryMessage * **QAMessage::castToBinaryMessage()**

説明 QAMessage を QABinaryMessage クラス にキャストします。

戻り値　「QABinaryMessage クラス」へのポインタ。処理後のメッセージが「QABinaryMessage クラス」のインスタンスではない場合は NULL。

castToTextMessage 関数

構文　virtual QATextMessage * **QAMessage::castToTextMessage()**

説明　QAMessage を 「QATextMessage クラス」にキャストします。

戻り値　「QATextMessage クラス」へのポインタ。処理後のメッセージが「QATextMessage クラス」のインスタンスではない場合は NULL。

clearProperties 関数

構文　virtual void **QAMessage::clearProperties()**

説明　メッセージのプロパティをクリアします。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。

getAddress 関数

構文　virtual qa_const_string **QAMessage::getAddress()**

説明　メッセージ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。このフィールドは、send (または publish) メソッドが完了した後、このメソッドによって指定された送信先を保持します。

戻り値　送信先アドレス。

getBooleanProperty 関数

構文

```
virtual 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。

説明 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。

戻り値 処理が正常終了した場合のみ true。

getByteProperty 関数

構文

```
virtual 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。

説明 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。

戻り値 処理が正常終了した場合のみ true。

getDoubleProperty 関数

構文

```
virtual 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。

説明 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。

戻り値 処理が正常終了した場合のみ true。

getExpiration 関数

構文　virtual qa_long **QAMessage::getExpiration()**

説明　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。このフィールドは、send (または publish) 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。これは、send または publish の開始時刻 (GMT) に、クライアントによって指定された存続時間値を加えた時刻です。存続時間としてゼロを指定すると、Expiration ヘッダもゼロに設定され、メッセージは無期限になります。

戻り値　有効期限。

getFloatProperty 関数

構文

```
virtual 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。

説明　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。

戻り値　処理が正常終了した場合のみ true。

getInReplyToID 関数

構文　virtual qa_const_string **QAMessage::getInReplyToID()**

説明　メッセージの In-Reply-To ID を取得します。

戻り値　In-Reply-To ID。

getIntProperty 関数

構文

virtual 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。

説明

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。

戻り値

処理が正常終了した場合のみ true。

getLongProperty 関数

構文

virtual 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。

説明

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。

戻り値

処理が正常終了した場合のみ true。

getMessageID 関数

構文

virtual qa_const_string **QAMessage::getMessageID()**

説明

メッセージ ID を取得します。メッセージの MessageID ヘッダ・フィールドは、QAnywhere クライアントによって送信されたメッセージをユニークに識別するための値を保持します。MessageID は、メッセージの送信時には無視できます。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。

戻り値　メッセージ ID。

getPriority 関数

構文

```
virtual qa_int QAMessage::getPriority()
```

説明　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。

戻り値　優先度。

getPropertyNames 関数

構文

```
virtual qa_const_string * QAMessage::getPropertyNames()
```

説明　QAMessage に現在設定されているプロパティ名のリストを返します。

戻り値　NULL で終了されているプロパティ名の配列。

getPropertyType 関数

構文

```
virtual qa_short QAMessage::getPropertyType(
    qa_const_string name
    )
```

パラメータ

- **name** プロパティ名。

説明 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 virtual qa_bool **QAMessage::getRedelivered()**

説明 メッセージが再配信中かどうかを示す値を取得します。

戻り値 メッセージが再配信されたかどうかを表す値。

getReplyToAddress 関数

構文 virtual qa_const_string **QAMessage::getReplyToAddress()**

説明 メッセージの返信先アドレスを取得します。

戻り値 返信先アドレス。

getShortProperty 関数

構文

virtual 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。

説明　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。

戻り値　処理が正常終了した場合のみ true。

getStringProperty 関数

構文

virtual 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** 取得した値からコピーする qa_char の最大数 (null 終端子も含みます)。

説明　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。

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

virtual 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** 取得した値からコピーする qa_char の最大数 (null 終端子も含みます)。

説明 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。

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。指定された名前を持つ qa_string 型プロパティの、オフセット位置からの値を返します。

getTimestamp 関数

構文

virtual qa_long **QAMessage::getTimestamp()**

説明 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。

戻り値 メッセージのタイムスタンプ。

getTimestampAsString 関数

構文

virtual 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。

説明　メッセージのタイムスタンプを文字列として取得します。この文字列にはフォーマットがあります。フォーマットは、"dow, MMM dd, yyyy hh:mm:ss.nnn GMT" です。

戻り値　バッファに書き込まれた null 以外の qa_char の数。

propertyExists 関数

構文

virtual qa_bool **QAMessage::propertyExists(**
qa_const_string *name*
)

パラメータ

- **name** プロパティ名。

説明　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

virtual void **QAMessage::setAddress(**
qa_const_string *destination*
)

パラメータ

- **destination**　送信先アドレス。

説明　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。

setBooleanProperty 関数

構文

virtual 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。

説明

指定された名前を持つ `qa_bool` 型プロパティの値をメッセージに設定します。

setByteProperty 関数

構文

virtual 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。

説明

指定された名前を持つ `qa_byte` 型プロパティの値をメッセージに設定します。

setDoubleProperty 関数

構文

virtual 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。

説明　　指定された名前を持つ qa_double 型プロパティの値をメッセージに設定します。

setFloatProperty 関数

構文

virtual 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。

説明　　指定された名前を持つ qa_float 型プロパティの値をメッセージに設定します。

setInReplyToID 関数

構文

virtual void **QAMessage::setInReplyToID(**
qa_const_string *id*
)

パラメータ

- **id** In-Reply-To ID。

説明　　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。

setIntProperty 関数

構文

virtual 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。

説明 指定された名前を持つ `qa_int` 型プロパティの値をメッセージに設定します。

setLongProperty 関数

構文

virtual 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。

説明 指定された名前を持つ `qa_long` 型プロパティの値をメッセージに設定します。

setMessageID 関数

構文

virtual void **QAMessage::setMessageID(**
qa_const_string *id*
)

パラメータ

- **id** メッセージ ID。

説明 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。

setPriority 関数

構文

virtual void **QAMessage::setPriority(**
qa_int *priority*
)

パラメータ

- **priority** 優先度。

説明　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。

setRedelivered 関数

構文

virtual 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。

説明　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。

setReplyToAddress 関数

構文

virtual void **QAMessage::setReplyToAddress(**
qa_const_string *replyTo*
)

パラメータ

- **replyTo** 返信先アドレス。

説明　メッセージの返信先アドレスを設定します。

setShortProperty 関数

構文

virtual 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。

説明　指定された名前を持つ `qa_short` 型プロパティの値をメッセージに設定します。

setStringProperty 関数

構文

virtual 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。

説明

指定された名前を持つ `qa_string` 型プロパティの値をメッセージに設定します。

setTimestamp 関数

構文

virtual 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。

説明

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。

~QAMessage 関数

構文

virtual **QAMessage::~QAMessage()**

説明

仮想デストラクタ。

QAMessageListener クラス

構文　public **QAMessageListener**

説明　QAMessageListener オブジェクトは、配信されたメッセージを非同期的に受信するために使用します。

メンバ　QAMessageListener クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「onMessage 関数」
- 「~QAMessageListener 関数」

onMessage 関数

構文

virtual void **QAMessageListener::onMessage(**
QAMessage * *message*
)

パラメータ

- **message** リスナに渡されるメッセージ。

説明　メッセージをリスナに渡します。

~QAMessageListener 関数

構文　virtual **QAMessageListener::~QAMessageListener()**

説明　仮想デストラクタ。

QATextMessage クラス

構文

public **QATextMessage**

基本クラス

- 「QAMessage クラス」

説明

QATextMessage オブジェクトは、qa_string 値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、メッセージ本文にテキストが追加されます。

クライアントが QATextMessage を受信するとき、メッセージは読み込み専用モードになっています。この状態でクライアントが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。

メンバ

QATextMessage クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「castToBinaryMessage 関数」
- 「castToTextMessage 関数」
- 「clearProperties 関数」
- 「DEFAULT_PRIORITY 変数」
- 「DEFAULT_TIME_TO_LIVE 変数」
- 「getAddress 関数」
- 「getBooleanProperty 関数」
- 「getByteProperty 関数」
- 「getDoubleProperty 関数」
- 「getExpiration 関数」
- 「getFloatProperty 関数」
- 「getInReplyToID 関数」
- 「getIntProperty 関数」
- 「getLongProperty 関数」
- 「getMessageID 関数」
- 「getPriority 関数」
- 「getPropertyNames 関数」
- 「getPropertyType 関数」
- 「getRedelivered 関数」
- 「getReplyToAddress 関数」
- 「getShortProperty 関数」

- 「getStringProperty 関数」
- 「getStringProperty 関数」
- 「getText 関数」
- 「getTextLength 関数」
- 「getTimestamp 関数」
- 「getTimestampAsString 関数」
- 「propertyExists 関数」
- 「readText 関数」
- 「setAddress 関数」
- 「setBooleanProperty 関数」
- 「setByteProperty 関数」
- 「setDoubleProperty 関数」
- 「setFloatProperty 関数」
- 「setInReplyToID 関数」
- 「setIntProperty 関数」
- 「setLongProperty 関数」
- 「setMessageID 関数」
- 「setPriority 関数」
- 「setRedelivered 関数」
- 「setReplyToAddress 関数」
- 「setShortProperty 関数」
- 「setStringProperty 関数」
- 「setText 関数」
- 「setTimestamp 関数」
- 「writeText 関数」
- 「~QAMessage 関数」
- 「~QATextMessage 関数」

getText 関数

構文　virtual qa_string **QATextMessage::getText()**

説明　メッセージのデータが格納されている文字列を取得します。デフォルト値は null です。

戻り値　メッセージのデータが格納されている `qa_string`。

getTextLength 関数

構文

virtual qa_long **QATextMessage::getTextLength()**

説明

テキスト長を返します。テキスト長がゼロ以外であるのに「getText 関数」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 関数」を使用して、テキストを分割して読み込む必要があります。

readText 関数

構文

virtual qa_int **QATextMessage::readText(**
qa_string *string*
qa_int *length*
)

パラメータ

- **string** テキストの格納先。
- **length** 格納先バッファに読み込む qa_char の最大数 (null 終端子も含みます)。

説明

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。現在のテキスト位置がテキストの終わりを過ぎている場合は -1。

setText 関数

構文

virtual void **QATextMessage::setText(**
qa_const_string *string*
)

パラメータ

- **string** メッセージのデータが格納されている `qa_string`。

説明

メッセージのデータが格納されている文字列を設定します。

writeText 関数

構文

virtual 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。

説明

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()**

説明

仮想デストラクタ。

QATransactionalManager クラス

構文

public **QATransactionalManager**

基本クラス

- 「QAManagerBase クラス」

説明

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。

メンバ

QATransactionalManager クラスのすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「close 関数」
- 「commit 関数」
- 「createBinaryMessage 関数」
- 「createTextMessage 関数」
- 「deleteMessage 関数」
- 「getBooleanStoreProperty 関数」
- 「getByteStoreProperty 関数」
- 「getDoubleStoreProperty 関数」
- 「getFloatStoreProperty 関数」
- 「getIntStoreProperty 関数」
- 「getLastError 関数」
- 「getLastErrorMsg 関数」
- 「getLongStoreProperty 関数」
- 「getMessage 関数」
- 「getMessageNoWait 関数」
- 「getMessageTimeout 関数」
- 「getMode 関数」
- 「getShortStoreProperty 関数」
- 「getStringStoreProperty 関数」
- 「open 関数」
- 「peekFirstMessage 関数」
- 「peekNextMessage 関数」
- 「publishMessage 関数」
- 「putMessage 関数」
- 「putMessageTimeToLive 関数」
- 「rollback 関数」

- 「setBooleanStoreProperty 関数」
- 「setByteStoreProperty 関数」
- 「setDoubleStoreProperty 関数」
- 「setFloatStoreProperty 関数」
- 「setIntStoreProperty 関数」
- 「setLongStoreProperty 関数」
- 「setMessageListener 関数」
- 「setProperty 関数」
- 「setShortStoreProperty 関数」
- 「setStringStoreProperty 関数」
- 「start 関数」
- 「stop 関数」
- 「triggerSendReceive 関数」
- 「~QAManagerBase 関数」
- 「~QATransactionalManager 関数」

commit 関数

構文　virtual qa_bool **QATransactionalManager::commit()**

説明　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。最初のトランザクションは、「open 関数」の呼び出しで開始されます。

戻り値　処理が正常終了した場合のみ true。

open 関数

構文　virtual qa_bool **QATransactionalManager::open()**

説明　QATransactionalManager をオープンします。

戻り値　処理が正常終了した場合のみ true。

rollback 関数

構文 virtual qa_bool **QATransactionalManager::rollback()**

説明 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。

戻り値 処理が正常終了した場合のみ true。

~QATransactionalManager 関数

構文 virtual **QATransactionalManager::~QATransactionalManager()**

説明 仮想デストラクタ。

第 9 章

iAnywhere.QAnywhere.Client ネームスペース

この章について

iAnywhere.QAnywhere.Client ネームスペースには、QAnywhere メッセージを処理するアプリケーションを構築するためのクラスと列挙が含まれています。

AcknowledgementMode 列挙

QAManager インスタンスの受信確認モード。

プロトタイプ

' Visual Basic
Public Enum **AcknowledgementMode**

// C#
public enum **AcknowledgementMode**

メンバ

メンバ	説明
EXPLICIT_ACKNOWLEDGEMENT	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は受信確認されません。
IMPLICIT_ACKNOWLEDGEMENT	getMessage が呼び出し元に戻り次第、すべてのメッセージが受信確認されます。同様に、メッセージ・リスナでも、メッセージ・リスナ・デリゲートの呼び出しが戻り次第、メッセージが受信確認されます。
TRANSACTIONAL	このモードでは、メッセージのキューへの登録とキューからの取り出しのすべてがトランザクション志向で実行されます。つまり、キューへの登録とキューからの取り出しはいずれもトランザクション内で実行され、まとめてコミットまたはロールバックされます。トランザクションは常に 1 つだけ存在します。トランザクションがコミットまたはロールバックされると、新しいトランザクションが暗黙的に開始されます。

MessageProperties クラス

標準のメッセージ・プロパティ名。

プロトタイプ

```
' Visual Basic
Public Class MessageProperties

// C#
public class MessageProperties
```

メンバ MessageProperties

パブリック静的フィールド (共有)

メンバ	説明
ABS_RETRY_TIMEOUT フィールド	コネクタ経由で送信されるメッセージのオプションのプロパティ。リトライ・タイムアウト時刻。この時刻になると、送信リトライが中止され、送信は失敗します。
ADAPTER フィールド	"system" キュー・メッセージのプロパティ。QAnywhere サーバに接続するときに使用できるネットワーク・アダプタを示す、デリミタ付きリスト。
COMPRESSED フィールド	メッセージの内容が圧縮されているかどうかを示します。
FROM_ADDR フィールド	送信者のアドレスを示すオプションのプロパティ。
MSG_TYPE フィールド	メッセージのタイプを示すオプションのプロパティ。
NETWORK フィールド	"system" キュー・メッセージのプロパティ。QAnywhere サーバに接続するときに使用できるネットワーク名を示す、デリミタ付きリスト。
NETWORK_STATUS フィールド	"system" キュー・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。値 1 は 接続済み、0 はそれ以外を意味します。

メンバ	説明
RETRY_FAILED フィールド	メッセージが RetryFailedAddress に送信されるときに、コネクタによって設定されます。受信側のクライアントは、このプロパティを参照して、再送信に失敗したメッセージを特定できます。
RETRY_FAILED_ADDR フィールド	コネクタ経由で送信されるメッセージのオプションのプロパティ。このプロパティが設定されている場合、RetryMax または RetryTimeout に設定された値を超えると、設定されているアドレスにメッセージが送信されます。
RETRY_FAILED_PRIORITY フィールド	コネクタ経由で送信されるメッセージのオプションのプロパティ。メッセージが RetryFailedAddress に送信されるとき、メッセージ優先度はこの値に設定されます。
RETRY_MAX フィールド	コネクタ経由で送信されるメッセージのオプションのプロパティ。コネクタでの送信リトライ回数の最大値。この回数を超えると送信は失敗します。
RETRY_TIMEOUT フィールド	コネクタ経由で送信されるメッセージのオプションのプロパティ。リトライ・タイムアウト時間。この時間が経過すると、送信リトライが中止され、送信は失敗します。

パブリック・インスタンス・コンストラクタ

メンバ	説明
MessageProperties コンストラクタ	MessageProperties クラス の新しいインスタンスを初期化する。

MessageProperties コンストラクタ

MessageProperties クラスの新しいインスタンスを初期化する。

プロトタイプ

```
' Visual Basic
    Public Sub New()
```

```
// C#
    public MessageProperties();
```

ABS_RETRY_TIMEOUT フィールド

コネクタ経由で送信されるメッセージのオプションのプロパティ。リトライ・タイムアウト時刻。この時刻になると、送信リトライが中止され、送信は失敗します。

プロトタイプ

```
' Visual Basic
    Public Shared ABS_RETRY_TIMEOUT As String

// C#
    public const string ABS_RETRY_TIMEOUT;
```

ADAPTER フィールド

"system" キュー・メッセージのプロパティ。QAnywhere サーバに接続するときに使用できるネットワーク・アダプタを示す、デリミタ付きリスト。

プロトタイプ

```
' Visual Basic
    Public Shared ADAPTER As String

// C#
    public const string ADAPTER;
```

COMPRESSED フィールド

メッセージの内容が圧縮されているかどうかを示します。

プロトタイプ

```
' Visual Basic
    Public Shared COMPRESSED As String

// C#
    public const string COMPRESSED;
```

FROM_ADDR フィールド

送信者のアドレスを示すオプションのプロパティ。

プロトタイプ

```
' Visual Basic
    Public Shared FROM_ADDR As String

// C#
    public const string FROM_ADDR;
```

MSG_TYPE フィールド

メッセージのタイプを示すオプションのプロパティ。

プロトタイプ

```
' Visual Basic
    Public Shared MSG_TYPE As String

// C#
    public const string MSG_TYPE;
```

NETWORK フィールド

"system" キュー・メッセージのプロパティ。QAnywhere サーバに接続するときに使用できるネットワーク名を示す、デリミタ付きリスト。

プロトタイプ

```
' Visual Basic
    Public Shared NETWORK As String

// C#
    public const string NETWORK;
```

NETWORK_STATUS フィールド

"system" キュー・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。値 1 は 接続済み、0 はそれ以外を意味します。

プロトタイプ

```
' Visual Basic
    Public Shared NETWORK_STATUS As String
```

```
// C#
    public const string NETWORK_STATUS;
```

RETRY_FAILED フィールド

メッセージが RetryFailedAddress に送信されるときに、コネクタによって設定されます。受信側のクライアントは、このプロパティを参照して、再送信に失敗したメッセージを特定できます。

プロトタイプ

```
' Visual Basic
    Public Shared RETRY_FAILED As String

// C#
    public const string RETRY_FAILED;
```

RETRY_FAILED_ADDR フィールド

コネクタ経由で送信されるメッセージのオプションのプロパティ。このプロパティが設定されている場合、RetryMax または RetryTimeout に設定された値を超えると、設定されているアドレスにメッセージが送信されます。

プロトタイプ

```
' Visual Basic
    Public Shared RETRY_FAILED_ADDR As String

// C#
    public const string RETRY_FAILED_ADDR;
```

RETRY_FAILED_PRIORITY フィールド

コネクタ経由で送信されるメッセージのオプションのプロパティ。メッセージが RetryFailedAddress に送信されるとき、メッセージ優先度はこの値に設定されます。

プロトタイプ

```
' Visual Basic
    Public Shared RETRY_FAILED_PRIORITY As String

// C#
    public const string RETRY_FAILED_PRIORITY;
```

RETRY_MAX フィールド

コネクタ経由で送信されるメッセージのオプションのプロパティ。コネクタでの送信リトライ回数の最大値。この回数を超えると送信は失敗します。

プロトタイプ

```
' Visual Basic
    Public Shared RETRY_MAX As String

// C#
    public const string RETRY_MAX;
```

RETRY_TIMEOUT フィールド

コネクタ経由で送信されるメッセージのオプションのプロパティ。リトライ・タイムアウト時間。この時間が経過すると、送信リトライが中止され、送信は失敗します。

プロトタイプ

```
' Visual Basic
    Public Shared RETRY_TIMEOUT As String

// C#
    public const string RETRY_TIMEOUT;
```

MessageType 列挙

メッセージのメッセージ・タイプ・プロパティの有効値。

プロトタイプ

' Visual Basic
Public Enum **MessageType**

// C#
public enum **MessageType**

メンバ

メンバ	説明
NETWORK_STATUS_NOTIFICATION	現在のデバイスのネットワーク・ステータスが変化したことを受信側に通知するメッセージであることを示すメッセージ・タイプ。
PUSH_NOTIFICATION	同期可能なメッセージがメッセージ・サーバに 1 つ以上存在することを受信側に通知するためのメッセージであることを示すメッセージ・タイプ。
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。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

バイナリ・メッセージのカプセル化。

プロトタイプ

```
' Visual Basic
    Public Class QABinaryMessage
    Inherits QAMessage

// C#
    public class QABinaryMessage :
    QAMessage
```

メンバ QABinaryMessage

パブリック・インスタンス・プロパティ

メンバ	説明
Address プロパティ (継承 QAMessage)	メッセージのアドレス。null の場合もありますが、getMessage またはメッセージ・リスナによって受信したメッセージの場合は null になりません。
BodyLength プロパティ	メッセージ本文のバイト数。
Expiration プロパティ (継承 QAMessage)	メッセージの有効期限を示します。この期限を過ぎても受信されなかったメッセージは、メッセージ・システムから削除されます。
InReplyToID プロパティ (継承 QAMessage)	どのメッセージへの返信であるかを示すメッセージ ID。null の場合もあります。
MessageID プロパティ (継承 QAMessage)	メッセージのグローバルにユニークな ID。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。
Priority プロパティ (継承 QAMessage)	メッセージの優先度 (0 ～ 9)。
Redelivered プロパティ (継承 QAMessage)	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。

メンバ	説明
ReplyToAddress プロパティ (継承 QAMessage)	メッセージの replyTo アドレス。null の場合もあります。
Timestamp プロパティ (継承 QAMessage)	メッセージのタイムスタンプ。

パブリック・インスタンス・メソッド

メンバ	説明
ClearProperties メソッド (継承 QAMessage)	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 (継承 QAMessage)	boolean 型のメッセージ・プロパティを取得します。
GetDoubleProperty メソッド (継承 QAMessage)	double 型のメッセージ・プロパティを取得します。
GetFloatProperty メソッド (継承 QAMessage)	float 型のメッセージ・プロパティを取得します。
GetIntProperty メソッド (継承 QAMessage)	int 型のメッセージ・プロパティを取得します。
GetLongProperty メソッド (継承 QAMessage)	long 型のメッセージ・プロパティを取得します。
GetProperty メソッド (継承 QAMessage)	メッセージのプロパティを取得します。
GetPropertyNames メソッド (継承 QAMessage)	メッセージのプロパティ名の列挙子を取得します。

メンバ	説明
GetPropertyType メソッド (継承 QAMessage)	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 (継承 QAMessage)	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 (継承 QAMessage)	short 型のメッセージ・プロパティを取得します。
GetStringProperty メソッド (継承 QAMessage)	文字列型のメッセージ・プロパティを取得します。
PropertyExists メソッド (継承 QAMessage)	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。
ReadBinary メソッド	バイナリ値の未読部分の先頭から、指定されたバイト数だけ、byte 型配列に読み込みます。
ReadBoolean メソッド	バイナリ値の未読部分の先頭から boolean 値として読み込みます。
ReadChar メソッド	バイナリ値の未読部分の先頭から char 値として読み込みます。
ReadDouble メソッド	バイナリ値の未読部分の先頭から double 値として読み込みます。
ReadFloat メソッド	バイナリ値の未読部分の先頭から float 値として読み込みます。
ReadInt メソッド	バイナリ値の未読部分の先頭から int 値として読み込みます。
ReadLong メソッド	バイナリ値の未読部分の先頭から long 値として読み込みます。
ReadSbyte メソッド	バイナリ値の未読部分の先頭から signed byte 値として読み込みます。

メンバ	説明
ReadShort メソッド	バイナリ値の未読部分の先頭から short 値として読み込みます。
ReadString メソッド	バイナリ値の未読部分の先頭から文字列値として読み込みます。
Reset メソッド	メッセージ・バイトの先頭から値の読み込みが開始されるようにメッセージをリセットします。
SetBooleanProperty メソッド (継承 QAMessage)	boolean 型のプロパティを設定します。
SetDoubleProperty メソッド (継承 QAMessage)	double 型のプロパティを設定します。
SetFloatProperty メソッド (継承 QAMessage)	float 型のプロパティを設定します。
SetIntProperty メソッド (継承 QAMessage)	int 型のプロパティを設定します。
SetLongProperty メソッド (継承 QAMessage)	long 型のプロパティを設定します。
SetProperty メソッド (継承 QAMessage)	プロパティを設定します。プロパティ型は、使用可能ないずれかのプリミティブ型、または文字列型でなければなりません。
SetSbyteProperty メソッド (継承 QAMessage)	byte 型のプロパティを設定します。
SetShortProperty メソッド (継承 QAMessage)	short 型のプロパティを設定します。
SetStringProperty メソッド (継承 QAMessage)	文字列型のプロパティを設定します。

メンバ	説明
WriteBinary メソッド	byte 型配列値をメッセージのバイト列の末尾に追加します。
WriteBoolean メソッド	boolean 値をバイナリ・コード化して、メッセージのバイト列の末尾に追加します。
WriteChar メソッド	char 値をバイナリ・コード化して、メッセージのバイト列の末尾に追加します。
WriteDouble メソッド	double 値をバイナリ・コード化して、メッセージのバイト列の末尾に追加します。
WriteFloat メソッド	float 値をバイナリ・コード化して、メッセージのバイト列の末尾に追加します。
WriteInt メソッド	int 値をバイナリ・コード化して、メッセージのバイト列の末尾に追加します。
WriteLong メソッド	long 値をバイナリ・コード化して、メッセージのバイト列の末尾に追加します。
WriteSbyte メソッド	signed byte 値をバイナリ・コード化して、メッセージのバイト列の末尾に追加します。
WriteShort メソッド	short 値をバイナリ・コード化して、メッセージのバイト列の末尾に追加します。
WriteString メソッド	文字列値をバイナリ・コード化して、メッセージのバイト列の末尾に追加します。

プロテクト・インスタンス・メソッド

メンバ	説明
Dispose メソッド (継承 QAMessage)	使用されているリソースをクリーンアップします。

BodyLength プロパティ

メッセージ本文のバイト数。

プロトタイプ

```
' Visual Basic
    Public Readonly Property BodyLength As Long

// C#
    public long BodyLength {get;}
```

ReadBinary メソッド

バイナリ値の未読部分の先頭から、指定されたバイト数だけ、byte 型配列に読み込みます。

プロトタイプ

```
' 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** 読み込んだバイトを格納する byte 型配列。
- **len** 読み込むバイト数の最大値。

戻り値

読み込まれたバイト数。

例外

- QAException クラス - 値の読み込み時に変換エラーが発生した場合、または読み込める入力が残っていない場合。

ReadBoolean メソッド

バイナリ値の未読部分の先頭から boolean 値として読み込みます。

プロトタイプ

```
' Visual Basic
    Public Function ReadBoolean() As Boolean
```

```
// C#
    public bool ReadBoolean();
```

戻り値　読み込まれた boolean 値。

例外
- QAException クラス - 値の読み込み時に変換エラーが発生した場合、または読み込める入力が残っていない場合。

ReadChar メソッド

バイナリ値の未読部分の先頭から char 値として読み込みます。

プロトタイプ

```
' Visual Basic
    Public Function ReadChar() As Char

// C#
    public char ReadChar();
```

戻り値　読み込まれた char 値。

例外
- QAException クラス - 値の読み込み時に変換エラーが発生した場合、または読み込める入力が残っていない場合。

ReadDouble メソッド

バイナリ値の未読部分の先頭から double 値として読み込みます。

プロトタイプ

```
' Visual Basic
    Public Function ReadDouble() As Double

// C#
    public double ReadDouble();
```

戻り値　読み込まれた double 値。

例外
- QAException クラス - 値の読み込み時に変換エラーが発生した場合、または読み込める入力が残っていない場合。

ReadFloat メソッド

バイナリ値の未読部分の先頭から float 値として読み込みます。

プロトタイプ

```
' Visual Basic
    Public Function ReadFloat() As Single

// C#
    public float ReadFloat();
```

戻り値

読み込まれた float 値。

例外

- QAException クラス - 値の読み込み時に変換エラーが発生した場合、または読み込める入力が残っていない場合。

ReadInt メソッド

バイナリ値の未読部分の先頭から int 値として読み込みます。

プロトタイプ

```
' Visual Basic
    Public Function ReadInt() As Integer

// C#
    public int ReadInt();
```

戻り値

読み込まれた int 値。

例外

- QAException クラス - 値の読み込み時に変換エラーが発生した場合、または読み込める入力が残っていない場合。

ReadLong メソッド

バイナリ値の未読部分の先頭から long 値として読み込みます。

プロトタイプ

```
' Visual Basic
    Public Function ReadLong() As Long

// C#
    public long ReadLong();
```

戻り値 読み込まれた long 値。

例外
- QAException クラス - 値の読み込み時に変換エラーが発生した場合、または読み込める入力が残っていない場合。

ReadSbyte メソッド

バイナリ値の未読部分の先頭から signed byte 値として読み込みます。

プロトタイプ

```
' Visual Basic
    Public Function ReadSbyte() As System.SByte

// C#
    public System.Sbyte ReadSbyte();
```

戻り値 読み込まれた signed byte 値。

例外
- QAException クラス - 値の読み込み時に変換エラーが発生した場合、または読み込める入力が残っていない場合。

ReadShort メソッド

バイナリ値の未読部分の先頭から short 値として読み込みます。

プロトタイプ

```
' Visual Basic
    Public Function ReadShort() As Short

// C#
    public short ReadShort();
```

戻り値 読み込まれた short 値。

例外
- QAException クラス - 値の読み込み時に変換エラーが発生した場合、または読み込める入力が残っていない場合。

ReadString メソッド

バイナリ値の未読部分の先頭から文字列値として読み込みます。

プロトタイプ

```
' Visual Basic
    Public Function ReadString() As String

// C#
    public string ReadString();
```

戻り値

読み込まれた文字列値。

例外

- QAException クラス - 値の読み込み時に変換エラーが発生した場合、または読み込める入力が残っていない場合。

Reset メソッド

メッセージ・バイトの先頭から値の読み込みが開始されるようにメッセージをリセットします。

プロトタイプ

```
' Visual Basic
    Public Sub Reset()

// C#
    public void Reset();
```

WriteBinary メソッド

byte 型配列値をメッセージのバイト列の末尾に追加します。

プロトタイプ

```
' 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** byte 型配列値。

- **len** 書き込むバイトの数。
- **offset** byte 型配列のオフセット (書き込むバイトの先頭の位)。

WriteBoolean メソッド

boolean 値をバイナリ・コード化して、メッセージのバイト列の末尾に追加します。

プロトタイプ

```
' Visual Basic
    Public Sub WriteBoolean( _
    ByVal val As Boolean _
    )

// C#
    public void WriteBoolean(
    bool val
    );
```

パラメータ

- **val** boolean 値。

WriteChar メソッド

char 値をバイナリ・コード化して、メッセージのバイト列の末尾に追加します。

プロトタイプ

```
' Visual Basic
    Public Sub WriteChar( _
    ByVal val As Char _
    )

// C#
    public void WriteChar(
    char val
    );
```

パラメータ

- **val** char 値。

WriteDouble メソッド

double 値をバイナリ・コード化して、メッセージのバイト列の末尾に追加します。

プロトタイプ

```
' Visual Basic
    Public Sub WriteDouble( _
    ByVal val As Double _
    )

// C#
    public void WriteDouble(
    double val
    );
```

パラメータ

- **val** double 値。

WriteFloat メソッド

float 値をバイナリ・コード化して、メッセージのバイト列の末尾に追加します。

プロトタイプ

```
' Visual Basic
    Public Sub WriteFloat( _
    ByVal val As Single _
    )

// C#
    public void WriteFloat(
    float val
    );
```

パラメータ

- **val** float 値。

WriteInt メソッド

int 値をバイナリ・コード化して、メッセージのバイト列の末尾に追加します。

プロトタイプ

```
' Visual Basic
    Public Sub WriteInt( _
    ByVal val As Integer _
    )

// C#
    public void WriteInt(
    int val
    );
```

パラメータ

- **val** int 値。

WriteLong メソッド

long 値をバイナリ・コード化して、メッセージのバイト列の末尾に追加します。

プロトタイプ

```
' Visual Basic
    Public Sub WriteLong( _
    ByVal val As Long _
    )

// C#
    public void WriteLong(
    long val
    );
```

パラメータ

- **val** long 値。

WriteSbyte メソッド

signed byte 値をバイナリ・コード化して、メッセージのバイト列の末尾に追加します。

プロトタイプ

```
' Visual Basic
    Public Sub WriteSbyte( _
    ByVal val As System.SByte _
    )
```

```
// C#
    public void WriteSbyte(
    System.Sbyte val
    );
```

パラメータ

- **val** signed byte 値。

WriteShort メソッド

short 値をバイナリ・コード化して、メッセージのバイト列の末尾に追加します。

プロトタイプ

```
' Visual Basic
    Public Sub WriteShort( _
    ByVal val As Short _
    )

// C#
    public void WriteShort(
    short val
    );
```

パラメータ

- **val** short 値。

WriteString メソッド

文字列値をバイナリ・コード化して、メッセージのバイト列の末尾に追加します。

プロトタイプ

```
' Visual Basic
    Public Sub WriteString( _
    ByVal val As String _
    )

// C#
    public void WriteString(
    string val
    );
```

パラメータ

- **val** 文字列値。

QAException クラス

QAnywhere によってスローされる例外。

プロトタイプ

```
' Visual Basic
    Public Class QAException
    Inherits ApplicationException

// C#
    public class QAException :
    ApplicationException
```

メンバ QAException

パブリック・インスタンス・コンストラクタ

メンバ	説明
QAException コンストラクタ	QAException を生成します。
QAException コンストラクタ	QAException を生成します。

パブリック・インスタンス・プロパティ

メンバ	説明
ErrorCode プロパティ	例外のエラー・コード。
HelpLink (継承 Exception)	例外に関連付けられているヘルプ ファイルへのリンクを取得または設定します。
InnerException (継承 Exception)	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 (継承 Exception)	現在の例外を説明するメッセージを取得します。
Source (継承 Exception)	エラーの原因となったアプリケーションまたはオブジェクトの名前を取得または設定します。

メンバ	説明
StackTrace (継承 Exception)	現在の例外がスローされたときにコール スタックにあったフレームの文字列形式を取得します。
TargetSite (継承 Exception)	現在の例外をスローするメソッドを取得します。

パブリック・インスタンス・メソッド

メンバ	説明
GetBaseException (継承 Exception)	派生クラスでオーバーライドされた場合、それ以後に発生する 1 つ以上の例外の主要な原因である System.Exception を返します。
GetObjectData (継承 Exception)	派生クラスでオーバーライドされた場合は、その例外に関する情報を使用して System.Runtime.Serialization.SerializationInfo を設定します。
ToString (継承 Exception)	現在の例外の文字列形式を作成して返します。

プロテクト・インスタンス・プロパティ

メンバ	説明
HResult (継承 Exception)	特定の例外に割り当てられているコード化数値である HRESULT を取得または設定します。

QAException コンストラクタ

QAException を生成します。

プロトタイプ

```
' Visual Basic
    Overloads Public Sub New( _
    ByVal msg As String _
    )
```

```
// C#
    public QAException(
    string msg
    );
```

パラメータ

- **msg** 例外の説明。

QAException コンストラクタ

QAException を生成します。

プロトタイプ

```
' 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** 例外の説明。
- **errCode** エラー・コード。

ErrorCode プロパティ

例外のエラー・コード。

プロトタイプ

```
' Visual Basic
    Public Readonly Property ErrorCode As Integer

// C#
    public int ErrorCode {get;}
```

QAManager クラス

QA メッセージング操作のマネージャ。トランザクション志向のマネージャとは異なり、メッセージのキューへの登録がただちに実行され、クライアントは特に何もしなくてもそのメッセージをキューから取り出すことができます。メッセージの受信確認には、2 つのモードがあります。暗黙の受信確認モードでは、getMessage が呼び出し元に戻り次第、すべてのメッセージが受信確認されます。同様に、メッセージ・リスナでも、メッセージ・リスナ・デリゲートの呼び出しが戻り次第、メッセージが受信確認されます。明示的な受信確認モードでは、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は受信確認されません。

プロトタイプ

```
' Visual Basic
    Public Class QAManager
    Inherits QAManagerBase

// C#
    public class QAManager :
    QAManagerBase
```

メンバ QAManager

パブリック・インスタンス・プロパティ

メンバ	説明
LastError プロパティ (継承 QAManagerBase)	最後に実行されたメソッドのエラー・コード。0 は正常終了を意味します。
LastErrorMessage プロパティ (継承 QAManagerBase)	最後に実行されたメソッドに対応するエラー・テキスト。最後に実行されたメソッドの LastError が 0 の場合は null になります。
Mode プロパティ (継承 QAManagerBase)	この Manager を介してすべてのメッセージを受信する場合の受信確認モード。

パブリック・インスタンス・メソッド

メンバ	説明
Acknowledge メソッド	指定されたメッセージの受信を確認します。
AcknowledgeAll メソッド	受信確認されていないメッセージをすべて受信確認します。
AcknowledgeUntil メソッド	指定されたメッセージとそれ以前に受信したメッセージのうち、受信確認されていないものをすべて受信確認します。
BrowseMessages メソッド (継承 QAManagerBase)	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。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。同じ Manager から返された各列挙子の間で、メソッドを交互に呼び出すことはできません。交互に呼び出しを行うと、ある呼び出しで参照されるべきメッセージが、別の呼び出しで参照される可能性があります。
Close メソッド (継承 QAManagerBase)	QA メッセージ・システムへの接続をクローズして、すべてのリソースを解放します。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 以外のメソッドを呼び出すと、例外が発生します。
CreateBinaryMessage メソッド (継承 QAManagerBase)	送信用の BinaryMessage インスタンスを作成します。
CreateTextMessage メソッド (継承 QAManagerBase)	送信用の TextMessage インスタンスを作成します。
GetBooleanStoreProperty メソッド (継承 QAManagerBase)	boolean 型のメッセージ・ストア・プロパティを取得します。
GetDoubleStoreProperty メソッド (継承 QAManagerBase)	double 型のメッセージ・ストア・プロパティを取得します。

メンバ	説明
GetFloatStoreProperty メソッド (継承 QAManagerBase)	float 型のメッセージ・ストア・プロパティを取得します。
GetIntStoreProperty メソッド (継承 QAManagerBase)	int 型のメッセージ・ストア・プロパティを取得します。
GetLongStoreProperty メソッド (継承 QAManagerBase)	long 型のメッセージ・ストア・プロパティを取得します。
GetMessage メソッド (継承 QAManagerBase)	指定されたアドレスに送信された受信待機中のメッセージのうち、次に取得可能なものを取得します。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。
GetMessageNoWait メソッド (継承 QAManagerBase)	指定されたアドレスに送信された受信待機中のメッセージのうち、次に取得可能なものを取得します。該当するメッセージが存在しない場合は、ブロックされることなくただちに null を返します。
GetMessageTimeout メソッド (継承 QAManagerBase)	指定されたアドレスに送信された受信待機中のメッセージのうち、次に取得可能なものを取得します。該当するメッセージが存在しない場合、タイムアウト時間になるまで新しいメッセージの着信を待機します。
GetSbyteStoreProperty メソッド (継承 QAManagerBase)	signed byte 型のメッセージ・ストア・プロパティを取得します。
GetShortStoreProperty メソッド (継承 QAManagerBase)	short 型のメッセージ・ストア・プロパティを取得します。
GetStoreProperty メソッド (継承 QAManagerBase)	メッセージ・ストアのプロパティを取得します。

メンバ	説明
GetStringStoreProperty メソッド (継承 QAManagerBase)	文字列型のメッセージ・ストア・プロパティを取得します。
Open メソッド	指定された受信確認モードを使用して Manager 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。
PutMessage メソッド (継承 QAManagerBase)	指定されたアドレスのメッセージ・システムにメッセージを登録します。
PutMessageTimeToLive メソッド (継承 QAManagerBase)	指定されたアドレスのメッセージ・システムに、有効期限付きでメッセージを登録します。
Recover メソッド	受信確認されていないメッセージを、すべて強制的に未受信に戻します。つまり、該当するメッセージは、getMessage を使用して再度受信する必要があります。
SetBooleanStoreProperty メソッド (継承 QAManagerBase)	boolean 型のメッセージ・ストア・プロパティを設定します。
SetDoubleStoreProperty メソッド (継承 QAManagerBase)	double 型のメッセージ・ストア・プロパティを設定します。
SetFloatStoreProperty メソッド (継承 QAManagerBase)	float 型のメッセージ・ストア・プロパティを設定します。
SetIntStoreProperty メソッド (継承 QAManagerBase)	int 型のメッセージ・ストア・プロパティを設定します。
SetLongStoreProperty メソッド (継承 QAManagerBase)	long 型のメッセージ・ストア・プロパティを設定します。

メンバ	説明
SetMessageListener メソッド (継承 QAManagerBase)	指定されたアドレスで使用可能なメッセージ・リスナを設定します。指定されたアドレスで設定できるリスナは 1 つだけです。指定されたアドレスに NULL リスナを割り当てると、そのアドレスへのリスナの割り当てが解除されます。
SetProperty メソッド (継承 QAManagerBase)	指定されたプロパティに、指定された値を設定します。QAManagerBase の作成時にプロパティ・ファイルを指定する代わりに、このメソッドを使用して QAManagerBase の各プロパティを設定できます。各プロパティは、派生クラスの open() メソッドを呼び出す前に設定しておく必要があります。
SetSbyteStoreProperty メソッド (継承 QAManagerBase)	byte 型のメッセージ・ストア・プロパティを設定します。
SetShortStoreProperty メソッド (継承 QAManagerBase)	short 型のメッセージ・ストア・プロパティを設定します。
SetStoreProperty メソッド (継承 QAManagerBase)	メッセージ・ストアのプロパティを設定します。プロパティ型は、使用可能ないずれかのプリミティブ型、または文字列型でなければなりません。
SetStringStoreProperty メソッド (継承 QAManagerBase)	文字列型のメッセージ・ストア・プロパティを設定します。
Start メソッド (継承 QAManagerBase)	Manager は、開始されると、すべての着信メッセージを受信します。この Start メソッドを繰り返し呼び出そうとしても、間に Stop メソッドを挟まないかぎり 2 回目以降の呼び出しは無視されます。

メンバ	説明
Stop メソッド (継承 QAManagerBase)	Manager は、停止されると、着信メッセージを一切受信しなくなります。ただし、メッセージが失われることはありません。Manager が開始されるまで受信されないだけです。この Stop メソッドを繰り返し呼び出そうとしても、間に Start メソッドを挟まないかぎり 2 回目以降の呼び出しは無視されます。
TriggerSendReceive メソッド (継承 QAManagerBase)	QA メッセージ・サーバとの同期処理を発生させて、このクライアント宛ではないメッセージをアップロードし、このクライアント宛のメッセージをダウンロードします。

プロテクト・インスタンス・フィールド

メンバ	説明
isOpen フィールド (継承 QAManagerBase)	インスタンスがオープンされているかどうかを示します。
mgrBase フィールド (継承 QAManagerBase)	基になっている C++ の QA Manager のハンドル。

プロテクト・インスタンス・メソッド

メンバ	説明
Dispose メソッド	使用されているリソースをクリーンアップします。

Acknowledge メソッド

指定されたメッセージの受信を確認します。

プロトタイプ

```
' 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。

例外

- QAException クラス - メッセージの受信確認時に問題が発生した場合。

AcknowledgeAll メソッド

受信確認されていないメッセージをすべて受信確認します。

プロトタイプ

```
' Visual Basic
    Public Sub AcknowledgeAll()
```

```
// C#
    public void AcknowledgeAll();
```

例外

- QAException クラス - メッセージの受信確認時に問題が発生した場合。

AcknowledgeUntil メソッド

指定されたメッセージとそれ以前に受信したメッセージのうち、受信確認されていないものをすべて受信確認します。

プロトタイプ

```
' 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。

例外

- QAException クラス - メッセージの受信確認時に問題が発生した場合。

Dispose メソッド

使用されているリソースをクリーンアップします。

プロトタイプ

```
' Visual Basic
    Overloads Overrides Protected Sub Dispose( _
    ByVal disposing As Boolean _
    )

// C#
    protected override void Dispose(
    bool disposing
    );
```

パラメータ

- **disposing** マネージ・リソースとアンマネージ・リソースの両方を解放する場合は true、アンマネージ・リソースだけを解放する場合は false。

Open メソッド

指定された受信確認モードを使用して Manager 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。

プロトタイプ

```
' Visual Basic
    Public Sub Open( _
    ByVal mode As AcknowledgementMode _
    )

// C#
    public void Open(
    AcknowledgementMode mode
    );
```

パラメータ

- **mode**
  AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT または AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT。

例外

- QAException クラス - Manager のオープン時に問題が発生した場合。

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。つまり、該当するメッセージは、getMessage を使用して再度受信する必要があります。

プロトタイプ

```
' Visual Basic
    Public Sub Recover()

// C#
    public void Recover();
```

例外

- QAException クラス - メッセージのステータスを戻すときに問題が発生した場合。

QAManagerBase クラス

このクラスは、QA Manager の抽象基本クラスです。メッセージの作成、送信、参照、受信の各サービスを提供します。シングル・スレッドで動作します。つまり、Manager を生成したスレッドだけが Manager の機能を呼び出すことができます。

プロトタイプ

```
' Visual Basic
    Public Class QAManagerBase
    Inherits Component

// C#
    public class QAManagerBase :
    Component
```

メンバ QAManagerBase

パブリック・インスタンス・プロパティ

メンバ	説明
LastError プロパティ	最後に実行されたメソッドのエラー・コード。0 は正常終了を意味します。
LastErrorMessage プロパティ	最後に実行されたメソッドに対応するエラー・テキスト。最後に実行されたメソッドの LastError が 0 の場合は null になります。
Mode プロパティ	この Manager を介してすべてのメッセージを受信する場合の受信確認モード。

パブリック・インスタンス・メソッド

メンバ	説明
BrowseMessages 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。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。同じ Manager から返された各列挙子の間で、メソッドを交互に呼び出すことはできません。交互に呼び出しを行うと、ある呼び出しで参照されるべきメッセージが、別の呼び出しで参照される可能性があります。
Close メソッド	QA メッセージ・システムへの接続をクローズして、すべてのリソースを解放します。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 以外のメソッドを呼び出すと、例外が発生します。
CreateBinaryMessage メソッド	送信用の BinaryMessage インスタンスを作成します。
CreateTextMessage メソッド	送信用の TextMessage インスタンスを作成します。
GetBooleanStoreProperty メソッド	boolean 型のメッセージ・ストア・プロパティを取得します。
GetDoubleStoreProperty メソッド	double 型のメッセージ・ストア・プロパティを取得します。
GetFloatStoreProperty メソッド	float 型のメッセージ・ストア・プロパティを取得します。
GetIntStoreProperty メソッド	int 型のメッセージ・ストア・プロパティを取得します。
GetLongStoreProperty メソッド	long 型のメッセージ・ストア・プロパティを取得します。
GetMessage メソッド	指定されたアドレスに送信された受信待機中のメッセージのうち、次に取得可能なものを取得します。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。

メンバ	説明
GetMessageNoWait メソッド	指定されたアドレスに送信された受信待機中のメッセージのうち、次に取得可能なものを取得します。該当するメッセージが存在しない場合は、ブロックされることなくただちに null を返します。
GetMessageTimeout メソッド	指定されたアドレスに送信された受信待機中のメッセージのうち、次に取得可能なものを取得します。該当するメッセージが存在しない場合、タイムアウト時間になるまで新しいメッセージの着信を待機します。
GetSbyteStoreProperty メソッド	signed byte 型のメッセージ・ストア・プロパティを取得します。
GetShortStoreProperty メソッド	short 型のメッセージ・ストア・プロパティを取得します。
GetStoreProperty メソッド	メッセージ・ストアのプロパティを取得します。
GetStringStoreProperty メソッド	文字列型のメッセージ・ストア・プロパティを取得します。
PutMessage メソッド	指定されたアドレスのメッセージ・システムにメッセージを登録します。
PutMessageTimeToLive メソッド	指定されたアドレスのメッセージ・システムに、有効期限付きでメッセージを登録します。
SetBooleanStoreProperty メソッド	boolean 型のメッセージ・ストア・プロパティを設定します。
SetDoubleStoreProperty メソッド	double 型のメッセージ・ストア・プロパティを設定します。
SetFloatStoreProperty メソッド	float 型のメッセージ・ストア・プロパティを設定します。
SetIntStoreProperty メソッド	int 型のメッセージ・ストア・プロパティを設定します。
SetLongStoreProperty メソッド	long 型のメッセージ・ストア・プロパティを設定します。

メンバ	説明
SetMessageListener メソッド	指定されたアドレスで使用可能なメッセージ・リスナを設定します。指定されたアドレスで設定できるリスナは 1 つだけです。指定されたアドレスに NULL リスナを割り当てると、そのアドレスへのリスナの割り当てが解除されます。
SetProperty メソッド	指定されたプロパティに、指定された値を設定します。QAManagerBase の作成時にプロパティ・ファイルを指定する代わりに、このメソッドを使用して QAManagerBase の各プロパティを設定できます。各プロパティは、派生クラスの open() メソッドを呼び出す前に設定しておく必要があります。
SetSbyteStoreProperty メソッド	byte 型のメッセージ・ストア・プロパティを設定します。
SetShortStoreProperty メソッド	short 型のメッセージ・ストア・プロパティを設定します。
SetStoreProperty メソッド	メッセージ・ストアのプロパティを設定します。プロパティ型は、使用可能ないずれかのプリミティブ型、または文字列型でなければなりません。
SetStringStoreProperty メソッド	文字列型のメッセージ・ストア・プロパティを設定します。
Start メソッド	Manager は、開始されると、すべての着信メッセージを受信します。この Start メソッドを繰り返し呼び出そうとしても、間に Stop メソッドを挟まないかぎり 2 回目以降の呼び出しは無視されます。
Stop メソッド	Manager は、停止されると、着信メッセージを一切受信しなくなります。ただし、メッセージが失われることはありません。Manager が開始されるまで受信されないだけです。この Stop メソッドを繰り返し呼び出そうとしても、間に Start メソッドを挟まないかぎり 2 回目以降の呼び出しは無視されます。

メンバ	説明
TriggerSendReceive メソッド	QA メッセージ・サーバとの同期処理を発生させて、このクライアント宛ではないメッセージをアップロードし、このクライアント宛のメッセージをダウンロードします。

プロテクト・インスタンス・フィールド

メンバ	説明
isOpen フィールド	インスタンスがオープンされているかどうかを示します。
mgrBase フィールド	基になっている C++ の QA Manager のハンドル。

プロテクト・インスタンス・メソッド

メンバ	説明
Dispose メソッド	使用されているリソースをクリーンアップします。

isOpen フィールド

インスタンスがオープンされているかどうかを示します。

プロトタイプ

```
' Visual Basic
    Family isOpen As Boolean

// C#
    family bool isOpen;
```

mgrBase フィールド

基になっている C++ の QA Manager のハンドル。

プロトタイプ

```
' Visual Basic
    Family mgrBase As IntPtr

// C#
    family IntPtr mgrBase;
```

LastError プロパティ

最後に実行されたメソッドのエラー・コード。0 は正常終了を意味します。

プロトタイプ

```
' Visual Basic
    Public Readonly Property LastError As Integer

// C#
    public int LastError {get;}
```

LastErrorMessage プロパティ

最後に実行されたメソッドに対応するエラー・テキスト。最後に実行されたメソッドの LastError が 0 の場合は null になります。

プロトタイプ

```
' Visual Basic
    Public Readonly Property LastErrorMessage As String

// C#
    public string LastErrorMessage {get;}
```

Mode プロパティ

この Manager を介してすべてのメッセージを受信する場合の受信確認モード。

プロトタイプ

```
' Visual Basic
    Public Readonly Property Mode As AcknowledgementMode

// C#
    public AcknowledgementMode Mode {get;}
```

BrowseMessages 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。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。同じ Manager から返された各列挙子の間で、メソッドを交互に呼び出すことはできません。交互に呼び出しを行うと、ある呼び出しで参照されるべきメッセージが、別の呼び出しで参照される可能性があります。

プロトタイプ

```
' Visual Basic
    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。

Close メソッド

QA メッセージ・システムへの接続をクローズして、すべてのリソースを解放します。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 以外のメソッドを呼び出すと、例外が発生します。

プロトタイプ

```
' Visual Basic
    Public Sub Close()

// C#
    public void Close();
```

例外

- QAException クラス - Manager のクローズ時に問題が発生した場合。

CreateBinaryMessage メソッド

送信用の BinaryMessage インスタンスを作成します。

プロトタイプ

```
' Visual Basic
    Public Function CreateBinaryMessage() As QABinaryMessage

// C#
    public QABinaryMessage CreateBinaryMessage();
```

戻り値

新しい BinaryMessage インスタンス。

例外

- QAException クラス - メッセージの作成時に問題が発生した場合。

CreateTextMessage メソッド

送信用の TextMessage インスタンスを作成します。

プロトタイプ

```
' Visual Basic
    Public Function CreateTextMessage() As QATextMessage

// C#
    public QATextMessage CreateTextMessage();
```

戻り値

新しい TextMessage インスタンス。

例外

- QAException クラス - メッセージの作成時に問題が発生した場合。

Dispose メソッド

使用されているリソースをクリーンアップします。

プロトタイプ

```
' Visual Basic
    Overloads Overrides Protected Sub Dispose( _
    ByVal disposing As Boolean _
    )
```

```
// C#
protected override void Dispose(
bool disposing
);
```

パラメータ

- **disposing** マネージ・リソースとアンマネージ・リソースの両方を解放する場合は true、アンマネージ・リソースだけを解放する場合は false。

GetBooleanStoreProperty メソッド

boolean 型のメッセージ・ストア・プロパティを取得します。

プロトタイプ

```
' Visual Basic
Public Function GetBooleanStoreProperty( _
ByVal propName As String _
) As Boolean
```

```
// C#
public bool GetBooleanStoreProperty(
string propName
);
```

パラメータ

- **propName** プロパティ名。

戻り値

プロパティの値

例外

- QAException クラス - プロパティ値の取得時に変換エラーが発生した場合、または該当するプロパティが存在しない場合。

GetDoubleStoreProperty メソッド

double 型のメッセージ・ストア・プロパティを取得します。

プロトタイプ

```
' Visual Basic
Public Function GetDoubleStoreProperty( _
ByVal propName As String _
) As Double
```

```
// C#
    public double GetDoubleStoreProperty(
    string propName
    );
```

パラメータ

- **propName** プロパティ名。

戻り値

プロパティの値。

例外

- QAException クラス - プロパティ値の取得時に変換エラーが発生した場合、または該当するプロパティが存在しない場合。

GetFloatStoreProperty メソッド

float 型のメッセージ・ストア・プロパティを取得します。

プロトタイプ

```
' Visual Basic
    Public Function GetFloatStoreProperty( _
    ByVal propName As String _
    ) As Single

// C#
    public float GetFloatStoreProperty(
    string propName
    );
```

パラメータ

- **propName** プロパティ名。

戻り値

プロパティの値。

例外

- QAException クラス - プロパティ値の取得時に変換エラーが発生した場合、または該当するプロパティが存在しない場合。

GetIntStoreProperty メソッド

int 型のメッセージ・ストア・プロパティを取得します。

プロトタイプ

```
' Visual Basic
    Public Function GetIntStoreProperty( _
    ByVal propName As String _
    ) As Integer

// C#
    public int GetIntStoreProperty(
    string propName
    );
```

パラメータ

- **propName** プロパティ名。

戻り値

プロパティの値。

例外

- QAException クラス - プロパティ値の取得時に変換エラーが発生した場合、または該当するプロパティが存在しない場合。

GetLongStoreProperty メソッド

long 型のメッセージ・ストア・プロパティを取得します。

プロトタイプ

```
' Visual Basic
    Public Function GetLongStoreProperty( _
    ByVal propName As String _
    ) As Long

// C#
    public long GetLongStoreProperty(
    string propName
    );
```

パラメータ

- **propName** プロパティ名。

戻り値

プロパティの値。

例外

- QAException クラス - プロパティ値の取得時に変換エラーが発生した場合、または該当するプロパティが存在しない場合。

GetMessage メソッド

指定されたアドレスに送信された受信待機中のメッセージのうち、次に取得可能なものを取得します。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。

プロトタイプ

```
' Visual Basic
    Public Function GetMessage( _
    ByVal address As String _
    ) As QAMessage

// C#
    public QAMessage GetMessage(
    string address
    );
```

パラメータ

- **address** メッセージのアドレス。

戻り値

次の取得可能なメッセージ。

例外

- QAException クラス - メッセージの取得時に問題が発生した場合。

GetMessageNoWait メソッド

指定されたアドレスに送信された受信待機中のメッセージのうち、次に取得可能なものを取得します。該当するメッセージが存在しない場合は、ブロックされることなくただちに null を返します。

プロトタイプ

```
' Visual Basic
    Public Function GetMessageNoWait( _
    ByVal address As String _
    ) As QAMessage

// C#
    public QAMessage GetMessageNoWait(
    string address
    );
```

パラメータ

- **address** メッセージのアドレス。

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。

例外

- QAException クラス - メッセージの取得時に問題が発生した場合。

GetMessageTimeout メソッド

指定されたアドレスに送信された受信待機中のメッセージのうち、次に取得可能なものを取得します。該当するメッセージが存在しない場合、タイムアウト時間になるまで新しいメッセージの着信を待機します。

プロトタイプ

```
' 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** メッセージのアドレス。
- **timeout** メッセージ着信タイムアウト (ミリ秒単位)。

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。

例外

- QAException クラス - メッセージの取得時に問題が発生した場合。

GetSbyteStoreProperty メソッド

signed byte 型のメッセージ・ストア・プロパティを取得します。

プロトタイプ

```
' 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** プロパティ名。

戻り値

プロパティの値。

例外

- QAException クラス - プロパティ値の取得時に変換エラーが発生した場合、または該当するプロパティが存在しない場合。

GetShortStoreProperty メソッド

short 型のメッセージ・ストア・プロパティを取得します。

プロトタイプ

```
' Visual Basic
    Public Function GetShortStoreProperty( _
    ByVal propName As String _
    ) As Short

// C#
    public short GetShortStoreProperty(
    string propName
    );
```

パラメータ

- **propName** プロパティ名。

戻り値

プロパティの値。

例外

- QAException クラス - プロパティ値の取得時に変換エラーが発生した場合、または該当するプロパティが存在しない場合。

GetStoreProperty メソッド

メッセージ・ストアのプロパティを取得します。

プロトタイプ

```
' Visual Basic
    Public Function GetStoreProperty( _
    ByVal propName As String _
    ) As Object

// C#
    public object GetStoreProperty(
    string propName
    );
```

パラメータ

- **propName** プロパティ名。

戻り値

プロパティの値。

例外

- QAException クラス - 該当するプロパティが存在しない場合。

GetStringStoreProperty メソッド

文字列型のメッセージ・ストア・プロパティを取得します。

プロトタイプ

```
' Visual Basic
    Public Function GetStringStoreProperty( _
    ByVal propName As String _
    ) As String

// C#
    public string GetStringStoreProperty(
    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。

PutMessage メソッド

指定されたアドレスのメッセージ・システムにメッセージを登録します。

プロトタイプ

```
' 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** メッセージのアドレス。
- **msg** 登録するメッセージ。

例外

- QAException クラス - メッセージの登録時に問題が発生した場合。

PutMessageTimeToLive メソッド

指定されたアドレスのメッセージ・システムに、有効期限付きでメッセージを登録します。

プロトタイプ

```
' 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** 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。

例外

- QAException クラス - メッセージの登録時に問題が発生した場合。

SetBooleanStoreProperty メソッド

boolean 型のメッセージ・ストア・プロパティを設定します。

プロトタイプ

```
' 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** プロパティ名。
- **val** プロパティの値。

SetDoubleStoreProperty メソッド

double 型のメッセージ・ストア・プロパティを設定します。

プロトタイプ

```
' 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** プロパティ名。
- **val** プロパティの値。

SetFloatStoreProperty メソッド

float 型のメッセージ・ストア・プロパティを設定します。

プロトタイプ

```
' 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** プロパティ名。
- **val** プロパティの値。

SetIntStoreProperty メソッド

int 型のメッセージ・ストア・プロパティを設定します。

プロトタイプ

```
' 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** プロパティ名。
- **val** プロパティの値。

SetLongStoreProperty メソッド

long 型のメッセージ・ストア・プロパティを設定します。

プロトタイプ

```
' 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** プロパティ名。
- **val** プロパティの値。

SetMessageListener メソッド

指定されたアドレスで使用可能なメッセージ・リスナを設定します。指定されたアドレスで設定できるリスナは 1 つだけです。指定されたアドレスに NULL リスナを割り当てると、そのアドレスへのリスナの割り当てが解除されます。

プロトタイプ

```
' Visual Basic
    Public Sub SetMessageListener( _
    ByVal address As String, _
    ByVal listener As QAManagerBase.MessageListener _
    )

// C#
    public void SetMessageListener(
    string address,
    QAManagerBase.MessageListener listener
    );
```

パラメータ

- **address** メッセージのアドレス。
- **listener** リスナ。

SetProperty メソッド

指定されたプロパティに、指定された値を設定します。QAManagerBase の作成時にプロパティ・ファイルを指定する代わりに、このメソッドを使用して QAManagerBase の各プロパティを設定できます。各プロパティは、派生クラスの open() メソッドを呼び出す前に設定しておく必要があります。

プロトタイプ

```
' 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** プロパティ名。
- **val** プロパティの値。

例外

- QAException クラス - プロパティの設定時に問題が発生した場合。

SetSbyteStoreProperty メソッド

byte 型のメッセージ・ストア・プロパティを設定します。

プロトタイプ

```
' 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** プロパティ名。
- **val** プロパティの値。

SetShortStoreProperty メソッド

short 型のメッセージ・ストア・プロパティを設定します。

プロトタイプ

```
' 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** プロパティ名。
- **val** プロパティの値。

SetStoreProperty メソッド

メッセージ・ストアのプロパティを設定します。プロパティ型は、使用可能ないずれかのプリミティブ型、または文字列型でなければなりません。

プロトタイプ

```
' 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** プロパティ名。
- **val** プロパティの値。

SetStringStoreProperty メソッド

文字列型のメッセージ・ストア・プロパティを設定します。

プロトタイプ

```
' 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** プロパティ名。
- **val** プロパティの値。

Start メソッド

Manager は、開始されると、すべての着信メッセージを受信します。この Start メソッドを繰り返し呼び出そうとしても、間に Stop メソッドを挟まないかぎり 2 回目以降の呼び出しは無視されます。

プロトタイプ

```
' Visual Basic
    Public Sub Start()

// C#
    public void Start();
```

例外

- QAException クラス - Manager の開始時に問題が発生した場合。

Stop メソッド

Manager は、停止されると、着信メッセージを一切受信しなくなります。ただし、メッセージが失われることはありません。Manager が開始されるまで受信されないだけです。この Stop メソッドを繰り返し呼び出そうとしても、間に Start メソッドを挟まないかぎり 2 回目以降の呼び出しは無視されます。

プロトタイプ

```
' Visual Basic
    Public Sub Stop()

// C#
    public void Stop();
```

例外

- QAException クラス - Manager の停止時に問題が発生した場合。

TriggerSendReceive メソッド

QA メッセージ・サーバとの同期処理を発生させて、このクライアント宛ではないメッセージをアップロードし、このクライアント宛のメッセージをダウンロードします。

プロトタイプ

```
' Visual Basic
    Public Sub TriggerSendReceive()
```

```
// C#
    public void TriggerSendReceive();
```

例外

- QAException クラス - 送信／受信のトリガ時に問題が発生した場合。

QAManagerBase.MessageListener 委譲

MessageListener デリゲートの定義。

プロトタイプ

```
' Visual Basic
 Delegate Sub QAManagerBase.MessageListener( _
ByVal msg As QAMessage _
)

// C#
 delegate void QAManagerBase.MessageListener(
QAMessage msg
);
```

パラメータ

- **msg** 受信したメッセージ。

QAManagerFactory クラス

QAManager オブジェクトと QATransactionalManager オブジェクトを作成するためのファクトリ。QAManagerFactory のインスタンスは常に 1 つしか存在しません。

プロトタイプ

' Visual Basic

```
Public Class QAManagerFactory
Inherits Component
```

// C#

```
public class QAManagerFactory :
Component
```

メンバ QAManagerFactory

パブリック静的プロパティ (共有)

メンバ	説明
Instance プロパティ	QAManagerFactory のシングルトン・インスタンス。
InstanceCount プロパティ	ファクトリ・インスタンスの数を示します。

パブリック・インスタンス・フィールド

メンバ	説明
InstanceID フィールド	ファクトリ ID。

パブリック・インスタンス・プロパティ

メンバ	説明
LastError プロパティ	最後に実行されたメソッドのエラー・コード。0 は正常終了を意味します。

メンバ	説明
LastErrorMessage プロパティ	最後に実行されたメソッドに対応するエラー・テキスト。最後に実行されたメソッドの LastError が 0 の場合は null になります。

パブリック・インスタンス・メソッド

メンバ	説明
CreateQAManager メソッド	INI ファイルから設定を読み込んで QA Manager を生成します。
CreateQATransactional Manager メソッド	INI ファイルから設定を読み込んでトランザクション志向の QA Manager を生成します。

プロテクト・インスタンス・メソッド

メンバ	説明
Dispose メソッド	使用されているリソースをクリーンアップします。
Finalize メソッド	インスタンスをクリーンアップします。

InstanceID フィールド

ファクトリ ID。

プロトタイプ

```
' Visual Basic
    PublicInstanceID As Integer

// C#
    public int InstanceID;
```

Instance プロパティ

QAManagerFactory のシングルトン・インスタンス。

プロトタイプ

```
' Visual Basic
    Public Shared Readonly Property Instance As QAManagerFactory

// C#
    public const QAManagerFactory Instance {get;}
```

例外

- QAException クラス - マネージャ・ファクトリの生成時に問題が発生した場合。

InstanceCount プロパティ

ファクトリ・インスタンスの数を示します。

プロトタイプ

```
' Visual Basic
    Public Shared Readonly Property InstanceCount As Long

// C#
    public const long InstanceCount {get;}
```

LastError プロパティ

最後に実行されたメソッドのエラー・コード。0 は正常終了を意味します。

プロトタイプ

```
' Visual Basic
    Public Readonly Property LastError As Integer

// C#
    public int LastError {get;}
```

LastErrorMessage プロパティ

最後に実行されたメソッドに対応するエラー・テキスト。最後に実行されたメソッドの LastError が 0 の場合は null になります。

プロトタイプ

```
' Visual Basic
    Public Readonly Property LastErrorMessage As String
```

```
// C#
    public string LastErrorMessage {get;}
```

CreateQAManager メソッド

INI ファイルから設定を読み込んで QA Manager を生成します。

プロトタイプ

```
' Visual Basic
    Public Function CreateQAManager( _
    ByVal iniFile As String _
    ) As QAManager

// C#
    public QAManager CreateQAManager(
    string iniFile
    );
```

パラメータ

- **iniFile** QAManager インスタンスを設定するためのプロパティ・ファイル。

戻り値

設定された QA Manager。

例外

- QAException クラス - Manager の生成時に問題が発生した場合。

CreateQATransactionalManager メソッド

INI ファイルから設定を読み込んでトランザクション志向の QA Manager を生成します。

プロトタイプ

```
' Visual Basic
    Public Function CreateQATransactionalManager( _
    ByVal iniFile As String _
    ) As QATransactionalManager

// C#
    public QATransactionalManager CreateQATransactionalManager(
    string iniFile
    );
```

パラメータ

- **iniFile** QATransactionalManager インスタンスを設定するためのプロパティ・ファイル。

戻り値

設定された QATransactionalManager。

例外

- QAException クラス - Manager の生成時にエラーが発生した場合。

Dispose メソッド

使用されているリソースをクリーンアップします。

プロトタイプ

```
' Visual Basic
    Overloads Overrides Protected Sub Dispose( _
    ByVal disposing As Boolean _
    )

// C#
    protected override void Dispose(
    bool disposing
    );
```

パラメータ

- **disposing** マネージ・リソースとアンマネージ・リソースの両方を解放する場合は true、アンマネージ・リソースだけを解放する場合は false。

Finalize メソッド

インスタンスをクリーンアップします。

プロトタイプ

```
' Visual Basic
    Overrides Protected Sub Finalize()

// C#
    protected override void Finalize();
```

QAMessage クラス

QA メッセージのカプセル化。

プロトタイプ

```
' Visual Basic
    Public Class QAMessage
    Inherits Component

// C#
    public class QAMessage :
    Component
```

メンバ QAMessage

パブリック・インスタンス・プロパティ

メンバ	説明
Address プロパティ	メッセージのアドレス。null の場合もありますが、getMessage またはメッセージ・リスナによって受信したメッセージの場合は null になりません。
Expiration プロパティ	メッセージの有効期限を示します。この期限を過ぎても受信されなかったメッセージは、メッセージ・システムから削除されます。
InReplyToID プロパティ	どのメッセージへの返信であるかを示すメッセージ ID。null の場合もあります。
MessageID プロパティ	メッセージのグローバルにユニークな ID。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。
Priority プロパティ	メッセージの優先度 (0 ～ 9)。
Redelivered プロパティ	受信されたが受信確認されていないメッセージであるかどうかを示します。
ReplyToAddress プロパティ	メッセージの replyTo アドレス。null の場合もあります。
Timestamp プロパティ	メッセージのタイムスタンプ。

パブリック・インスタンス・メソッド

メンバ	説明
ClearProperties メソッド	メッセージのすべてのプロパティをクリアします。
GetBooleanProperty メソッド	boolean 型のメッセージ・プロパティを取得します。
GetDoubleProperty メソッド	double 型のメッセージ・プロパティを取得します。
GetFloatProperty メソッド	float 型のメッセージ・プロパティを取得します。
GetIntProperty メソッド	int 型のメッセージ・プロパティを取得します。
GetLongProperty メソッド	long 型のメッセージ・プロパティを取得します。
GetProperty メソッド	メッセージのプロパティを取得します。
GetPropertyNames メソッド	メッセージのプロパティ名の列挙子を取得します。
GetPropertyType メソッド	指定されたプロパティのプロパティ型を返します。
GetSbyteProperty メソッド	signed byte 型のメッセージ・プロパティを取得します。
GetShortProperty メソッド	short 型のメッセージ・プロパティを取得します。
GetStringProperty メソッド	文字列型のメッセージ・プロパティを取得します。
PropertyExists メソッド	指定されたプロパティがこのメッセージに設定されているかどうかを示します。
SetBooleanProperty メソッド	boolean 型のプロパティを設定します。

メンバ	説明
SetDoubleProperty メソッド	double 型のプロパティを設定します。
SetFloatProperty メソッド	float 型のプロパティを設定します。
SetIntProperty メソッド	int 型のプロパティを設定します。
SetLongProperty メソッド	long 型のプロパティを設定します。
SetProperty メソッド	プロパティを設定します。プロパティ型は、使用可能ないずれかのプリミティブ型、または文字列型でなければなりません。
SetSbyteProperty メソッド	byte 型のプロパティを設定します。
SetShortProperty メソッド	short 型のプロパティを設定します。
SetStringProperty メソッド	文字列型のプロパティを設定します。

プロテクト・インスタンス・メソッド

メンバ	説明
Dispose メソッド	使用されているリソースをクリーンアップします。

Address プロパティ

メッセージのアドレス。null の場合もありますが、getMessage またはメッセージ・リスナによって受信したメッセージの場合は null になりません。

プロトタイプ

```
' Visual Basic
    Public Property Address As String

// C#
    public string Address {get;set;}
```

Expiration プロパティ

メッセージの有効期限を示します。この期限を過ぎても受信されなかったメッセージは、メッセージ・システムから削除されます。

プロトタイプ

```
' Visual Basic
    Public Readonly Property Expiration As Long

// C#
    public long Expiration {get;}
```

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。null の場合もあります。

プロトタイプ

```
' Visual Basic
    Public Property InReplyToID As String

// C#
    public string InReplyToID {get;set;}
```

MessageID プロパティ

メッセージのグローバルにユニークな ID。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。

プロトタイプ

```
' Visual Basic
    Public Readonly Property MessageID As String

// C#
    public string MessageID {get;}
```

Priority プロパティ

メッセージの優先度 (0 ～ 9)。

プロトタイプ

```
' Visual Basic
    Public Property Priority As Integer

// C#
    public int Priority {get;set;}
```

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。

プロトタイプ

```
' Visual Basic
    Public Readonly Property Redelivered As Boolean

// C#
    public bool Redelivered {get;}
```

ReplyToAddress プロパティ

メッセージの replyTo アドレス。null の場合もあります。

プロトタイプ

```
' Visual Basic
    Public Property ReplyToAddress As String

// C#
    public string ReplyToAddress {get;set;}
```

Timestamp プロパティ

メッセージのタイムスタンプ。

プロトタイプ

```
' Visual Basic
    Public Readonly Property Timestamp As Date
```

```
// C#
    public DateTime Timestamp {get;}
```

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。

プロトタイプ

```
' Visual Basic
    Public Sub ClearProperties()

// C#
    public void ClearProperties();
```

Dispose メソッド

使用されているリソースをクリーンアップします。

プロトタイプ

```
' Visual Basic
    Overloads Overrides Protected Sub Dispose( _
    ByVal disposing As Boolean _
    )

// C#
    protected override void Dispose(
    bool disposing
    );
```

パラメータ

- **disposing** マネージ・リソースとアンマネージ・リソースの両方を解放する場合は true、アンマネージ・リソースだけを解放する場合は false。

GetBooleanProperty メソッド

boolean 型のメッセージ・プロパティを取得します。

プロトタイプ

```
' 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。

例外

- QAException クラス - プロパティ値の取得時に変換エラーが発生した場合、または該当するプロパティが存在しない場合。

GetDoubleProperty メソッド

double 型のメッセージ・プロパティを取得します。

プロトタイプ

```
' 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。

例外

- QAException クラス - プロパティ値の取得時に変換エラーが発生した場合、または該当するプロパティが存在しない場合。

GetFloatProperty メソッド

float 型のメッセージ・プロパティを取得します。

プロトタイプ

```
' 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。

例外

- QAException クラス - プロパティ値の取得時に変換エラーが発生した場合、または該当するプロパティが存在しない場合。

GetIntProperty メソッド

int 型のメッセージ・プロパティを取得します。

プロトタイプ

```
' 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。

例外

- QAException クラス - プロパティ値の取得時に変換エラーが発生した場合、または該当するプロパティが存在しない場合。

GetLongProperty メソッド

long 型のメッセージ・プロパティを取得します。

プロトタイプ

```
' 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。

例外

- QAException クラス - プロパティ値の取得時に変換エラーが発生した場合、または該当するプロパティが存在しない場合。

GetProperty メソッド

メッセージのプロパティを取得します。

プロトタイプ

```
' 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。

例外

- QAException クラス - 該当するプロパティが存在しない場合。

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。

プロトタイプ

```
' 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。

プロトタイプ

```
' Visual Basic
    Public Function GetPropertyType( _
    ByVal propName As String _
    ) As QAPropertyType

// C#
    public QAPropertyType GetPropertyType(
    string propName
    );
```

パラメータ

- **propName** プロパティ名。

戻り値

プロパティ型。

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。

プロトタイプ

```
' 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。

例外

- QAException クラス - プロパティ値の取得時に変換エラーが発生した場合、または該当するプロパティが存在しない場合。

GetShortProperty メソッド

short 型のメッセージ・プロパティを取得します。

プロトタイプ

```
' 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。

例外

- QAException クラス - プロパティ値の取得時に変換エラーが発生した場合、または該当するプロパティが存在しない場合。

GetStringProperty メソッド

文字列型のメッセージ・プロパティを取得します。

プロトタイプ

```
' 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。

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。

プロトタイプ

```
' 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かどうかを示す値。

SetBooleanProperty メソッド

boolean 型のプロパティを設定します。

プロトタイプ

```
' 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。

SetDoubleProperty メソッド

double 型のプロパティを設定します。

プロトタイプ

```
' 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。

SetFloatProperty メソッド

float 型のプロパティを設定します。

プロトタイプ

```
' 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。

SetIntProperty メソッド

int 型のプロパティを設定します。

プロトタイプ

```
' 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。

SetLongProperty メソッド

long 型のプロパティを設定します。

プロトタイプ

```
' 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。

SetProperty メソッド

プロパティを設定します。プロパティ型は、使用可能ないずれかのプリミティブ型、または文字列型でなければなりません。

プロトタイプ

```
' 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。

SetSbyteProperty メソッド

byte 型のプロパティを設定します。

プロトタイプ

```
' 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。

SetShortProperty メソッド

short 型のプロパティを設定します。

プロトタイプ

```
' 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。

SetStringProperty メソッド

文字列型のプロパティを設定します。

プロトタイプ

```
' 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。

QAPropertyType 列挙

QAMessage プロパティ型列挙子。C# の型に自然に対応します。

プロトタイプ

' Visual Basic
Public Enum **QAPropertyType**

// C#
public enum **QAPropertyType**

メンバ

メンバ	説明
BOOLEAN	プロパティ型が boolean であることを示します。
BYTE	プロパティ型が signed byte であることを示します。
DOUBLE	プロパティ型が double であることを示します。
FLOAT	プロパティ型が float であることを示します。
INT	プロパティ型が int であることを示します。
LONG	プロパティ型が long であることを示します。
SHORT	プロパティ型が short であることを示します。
STRING	プロパティ型が文字列型であることを示します。
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。

QATextMessage クラス

テキスト・メッセージのカプセル化。

プロトタイプ

```
' Visual Basic
    Public Class QATextMessage
    Inherits QAMessage

// C#
    public class QATextMessage :
    QAMessage
```

メンバ QATextMessage

パブリック・インスタンス・プロパティ

メンバ	説明
Address プロパティ (継承 QAMessage)	メッセージのアドレス。null の場合もありますが、getMessage またはメッセージ・リスナによって受信したメッセージの場合は null になりません。
Expiration プロパティ (継承 QAMessage)	メッセージの有効期限を示します。この期限を過ぎても受信されなかったメッセージは、メッセージ・システムから削除されます。
InReplyToID プロパティ (継承 QAMessage)	どのメッセージへの返信であるかを示すメッセージ ID。null の場合もあります。
MessageID プロパティ (継承 QAMessage)	メッセージのグローバルにユニークな ID。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。
Priority プロパティ (継承 QAMessage)	メッセージの優先度 (0 ～ 9)。
Redelivered プロパティ (継承 QAMessage)	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。

メンバ	説明
ReplyToAddress プロパティ (継承 QAMessage)	メッセージの replyTo アドレス。null の場合もあります。
Text プロパティ	メッセージ・テキスト。メッセージがメッセージの最大チャンク・サイズを超えている場合、この Text の値は null になります。この場合、readText メソッドを使用してテキストを受信する必要があります。
TextLength プロパティ	メッセージの文字数。
Timestamp プロパティ (継承 QAMessage)	メッセージのタイムスタンプ。

パブリック・インスタンス・メソッド

メンバ	説明
ClearProperties メソッド (継承 QAMessage)	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 (継承 QAMessage)	boolean 型のメッセージ・プロパティを取得します。
GetDoubleProperty メソッド (継承 QAMessage)	double 型のメッセージ・プロパティを取得します。
GetFloatProperty メソッド (継承 QAMessage)	float 型のメッセージ・プロパティを取得します。
GetIntProperty メソッド (継承 QAMessage)	int 型のメッセージ・プロパティを取得します。

メンバ	説明
GetLongProperty メソッド (継承 QAMessage)	long 型のメッセージ・プロパティを取得します。
GetProperty メソッド (継承 QAMessage)	メッセージのプロパティを取得します。
GetPropertyNames メソッド (継承 QAMessage)	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 (継承 QAMessage)	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 (継承 QAMessage)	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 (継承 QAMessage)	short 型のメッセージ・プロパティを取得します。
GetStringProperty メソッド (継承 QAMessage)	文字列型のメッセージ・プロパティを取得します。
PropertyExists メソッド (継承 QAMessage)	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。
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。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。
SetBooleanProperty メソッド (継承 QAMessage)	boolean 型のプロパティを設定します。

メンバ	説明
SetDoubleProperty メソッド (継承 QAMessage)	double 型のプロパティを設定します。
SetFloatProperty メソッド (継承 QAMessage)	float 型のプロパティを設定します。
SetIntProperty メソッド (継承 QAMessage)	int 型のプロパティを設定します。
SetLongProperty メソッド (継承 QAMessage)	long 型のプロパティを設定します。
SetProperty メソッド (継承 QAMessage)	プロパティを設定します。プロパティ型は、使用可能ないずれかのプリミティブ型、または文字列型でなければなりません。
SetSbyteProperty メソッド (継承 QAMessage)	byte 型のプロパティを設定します。
SetShortProperty メソッド (継承 QAMessage)	short 型のプロパティを設定します。
SetStringProperty メソッド (継承 QAMessage)	文字列型のプロパティを設定します。
WriteText メソッド	メッセージ・テキストの末尾にテキストを追加します。

プロテクト・インスタンス・メソッド

メンバ	説明
Dispose メソッド (継承 QAMessage)	使用されているリソースをクリーンアップします。

Text プロパティ

メッセージ・テキスト。メッセージがメッセージの最大チャンク・サイズを超えている場合、この Text の値は null になります。この場合、readText メソッドを使用してテキストを受信する必要があります。

プロトタイプ

```
' Visual Basic
    Public Property Text As String

// C#
    public string Text {get;set;}
```

TextLength プロパティ

メッセージの文字数。

プロトタイプ

```
' 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。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。

プロトタイプ

```
' 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。

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。

プロトタイプ

```
' 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 クラス

トランザクション志向 QA Manager。この Manager では、メッセージのキューへの登録とキューからの取り出しのすべてをトランザクション志向で実行します。つまり、キューへの登録とキューからの取り出しはいずれもトランザクション内で実行され、まとめてコミットまたはロールバックされます。トランザクションは常に 1 つだけ存在します。トランザクションがコミットまたはロールバックされると、新しいトランザクションが暗黙的に開始されます。

プロトタイプ

```
' Visual Basic
    Public Class QATransactionalManager
    Inherits QAManagerBase

// C#
    public class QATransactionalManager :
    QAManagerBase
```

メンバ QATransactionalManager

パブリック・インスタンス・プロパティ

メンバ	説明
LastError プロパティ (継承 QAManagerBase)	最後に実行されたメソッドのエラー・コード。0 は正常終了を意味します。
LastErrorMessage プロパティ (継承 QAManagerBase)	最後に実行されたメソッドに対応するエラー・テキスト。最後に実行されたメソッドの LastError が 0 の場合は null になります。
Mode プロパティ (継承 QAManagerBase)	この Manager を介してすべてのメッセージを受信する場合の受信確認モード。

パブリック・インスタンス・メソッド

メンバ	説明
BrowseMessages メソッド (継承 QAManagerBase)	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。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。同じ Manager から返された各列挙子の間で、メソッドを交互に呼び出すことはできません。交互に呼び出しを行うと、ある呼び出しで参照されるべきメッセージが、別の呼び出しで参照される可能性があります。
Close メソッド (継承 QAManagerBase)	QA メッセージ・システムへの接続をクローズして、すべてのリソースを解放します。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 以外のメソッドを呼び出すと、例外が発生します。
Commit メソッド	メッセージのキューへの登録またはキューからの取り出しのうち、コミットされていないものをすべてコミットします。トランザクション志向 Manager を使用してキューに登録したメッセージは、コミットが実行されないかぎり、受信されません。同様に、トランザクション志向 Manager を使用してキューから取得したメッセージは、コミットが実行されないかぎり、未取得とみなされます。
CreateBinaryMessage メソッド (継承 QAManagerBase)	送信用の BinaryMessage インスタンスを作成します。
CreateTextMessage メソッド (継承 QAManagerBase)	送信用の TextMessage インスタンスを作成します。
GetBooleanStoreProperty メソッド (継承 QAManagerBase)	boolean 型のメッセージ・ストア・プロパティを取得します。
GetDoubleStoreProperty メソッド (継承 QAManagerBase)	double 型のメッセージ・ストア・プロパティを取得します。

メンバ	説明
GetFloatStoreProperty メソッド (継承 QAManagerBase)	float 型のメッセージ・ストア・プロパティを取得します。
GetIntStoreProperty メソッド (継承 QAManagerBase)	int 型のメッセージ・ストア・プロパティを取得します。
GetLongStoreProperty メソッド (継承 QAManagerBase)	long 型のメッセージ・ストア・プロパティを取得します。
GetMessage メソッド (継承 QAManagerBase)	指定されたアドレスに送信された受信待機中のメッセージのうち、次に取得可能なものを取得します。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。
GetMessageNoWait メソッド (継承 QAManagerBase)	指定されたアドレスに送信された受信待機中のメッセージのうち、次に取得可能なものを取得します。該当するメッセージが存在しない場合は、ブロックされることなくただちに null を返します。
GetMessageTimeout メソッド (継承 QAManagerBase)	指定されたアドレスに送信された受信待機中のメッセージのうち、次に取得可能なものを取得します。該当するメッセージが存在しない場合、タイムアウト時間になるまで新しいメッセージの着信を待機します。
GetSbyteStoreProperty メソッド (継承 QAManagerBase)	signed byte 型のメッセージ・ストア・プロパティを取得します。
GetShortStoreProperty メソッド (継承 QAManagerBase)	short 型のメッセージ・ストア・プロパティを取得します。
GetStoreProperty メソッド (継承 QAManagerBase)	メッセージ・ストアのプロパティを取得します。

メンバ	説明
GetStringStoreProperty メソッド (継承 QAManagerBase)	文字列型のメッセージ・ストア・プロパティを取得します。
Open メソッド	トランザクション志向 Manager 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。
PutMessage メソッド (継承 QAManagerBase)	指定されたアドレスのメッセージ・システムにメッセージを登録します。
PutMessageTimeToLive メソッド (継承 QAManagerBase)	指定されたアドレスのメッセージ・システムに、有効期限付きでメッセージを登録します。
Rollback メソッド	メッセージのキューへの登録またはキューからの取り出しのうち、コミットされていないものをすべてロールバックします。トランザクション志向 Manager を使用してキューに登録したメッセージは、送信されなくなります。同様に、トランザクション志向 Manager を使用してキューから取得したメッセージは、取得されていないことになります。
SetBooleanStoreProperty メソッド (継承 QAManagerBase)	boolean 型のメッセージ・ストア・プロパティを設定します。
SetDoubleStoreProperty メソッド (継承 QAManagerBase)	double 型のメッセージ・ストア・プロパティを設定します。
SetFloatStoreProperty メソッド (継承 QAManagerBase)	float 型のメッセージ・ストア・プロパティを設定します。
SetIntStoreProperty メソッド (継承 QAManagerBase)	int 型のメッセージ・ストア・プロパティを設定します。

メンバ	説明
SetLongStoreProperty メソッド (継承 QAManagerBase)	long 型のメッセージ・ストア・プロパティを設定します。
SetMessageListener メソッド (継承 QAManagerBase)	指定されたアドレスで使用可能なメッセージ・リスナを設定します。指定されたアドレスで設定できるリスナは 1 つだけです。指定されたアドレスに NULL リスナを割り当てると、そのアドレスへのリスナの割り当てが解除されます。
SetProperty メソッド (継承 QAManagerBase)	指定されたプロパティに、指定された値を設定します。QAManagerBase の作成時にプロパティ・ファイルを指定する代わりに、このメソッドを使用して QAManagerBase の各プロパティを設定できます。各プロパティは、派生クラスの open() メソッドを呼び出す前に設定しておく必要があります。
SetSbyteStoreProperty メソッド (継承 QAManagerBase)	byte 型のメッセージ・ストア・プロパティを設定します。
SetShortStoreProperty メソッド (継承 QAManagerBase)	short 型のメッセージ・ストア・プロパティを設定します。
SetStoreProperty メソッド (継承 QAManagerBase)	メッセージ・ストアのプロパティを設定します。プロパティ型は、使用可能ないずれかのプリミティブ型、または文字列型でなければなりません。
SetStringStoreProperty メソッド (継承 QAManagerBase)	文字列型のメッセージ・ストア・プロパティを設定します。
Start メソッド (継承 QAManagerBase)	Manager は、開始されると、すべての着信メッセージを受信します。この Start メソッドを繰り返し呼び出そうとしても、間に Stop メソッドを挟まないかぎり 2 回目以降の呼び出しは無視されます。

メンバ	説明
Stop メソッド (継承 QAManagerBase)	Manager は、停止されると、着信メッセージを一切受信しなくなります。ただし、メッセージが失われることはありません。Manager が開始されるまで受信されないだけです。この Stop メソッドを繰り返し呼び出そうとしても、間に Start メソッドを挟まないかぎり 2 回目以降の呼び出しは無視されます。
TriggerSendReceive メソッド (継承 QAManagerBase)	QA メッセージ・サーバとの同期処理を発生させて、このクライアント宛ではないメッセージをアップロードし、このクライアント宛のメッセージをダウンロードします。

プロテクト・インスタンス・フィールド

メンバ	説明
isOpen フィールド (継承 QAManagerBase)	インスタンスがオープンされているかどうかを示します。
mgrBase フィールド (継承 QAManagerBase)	基になっている C++ の QA Manager のハンドル。

プロテクト・インスタンス・メソッド

メンバ	説明
Dispose メソッド	使用されているリソースをクリーンアップします。

Commit メソッド

メッセージのキューへの登録またはキューからの取り出しのうち、コミットされていないものをすべてコミットします。トランザクション志向 Manager を使用してキューに登録したメッセージは、コミットが実行されないかぎり、受信されません。同様に、トランザクション志向 Manager を使用してキューから取得したメッセージは、コミットが実行されないかぎり、未取得とみなされます。

プロトタイプ

```
' Visual Basic
    Public Sub Commit()

// C#
    public void Commit();
```

例外

- QAException クラス - コミット時に問題が発生した場合。

Dispose メソッド

使用されているリソースをクリーンアップします。

プロトタイプ

```
' Visual Basic
    Overloads Overrides Protected Sub Dispose( _
    ByVal disposing As Boolean _
    )

// C#
    protected override void Dispose(
    bool disposing
    );
```

パラメータ

- **disposing** マネージ・リソースとアンマネージ・リソースの両方を解放する場合は true、アンマネージ・リソースだけを解放する場合は false。

Open メソッド

トランザクション志向 Manager 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。

プロトタイプ

```
' Visual Basic
    Public Sub Open()

// C#
    public void Open();
```

例外

- QAException クラス - Manager のオープン時に問題が発生した場合。

Rollback メソッド

メッセージのキューへの登録またはキューからの取り出しのうち、コミットされていないものをすべてロールバックします。トランザクション志向 Manager を使用してキューに登録したメッセージは、送信されなくなります。同様に、トランザクション志向 Manager を使用してキューから取得したメッセージは、取得されていないことになります。

プロトタイプ

```
' Visual Basic
    Public Sub Rollback()

// C#
    public void Rollback();
```

例外

- QAException クラス - ロールバック時に問題が発生した場合。

索引

記号

@data オプション
QAnywhere Agent [qaagent] 105
~QABinaryMessage 関数
QAnywhere C++ API 172
~QAError 関数
QAnywhere C++ API 177
~QAManagerBase 関数
QAnywhere C++ API 197
~QAManagerFactory 関数
QAnywhere C++ API 200
~QAManager 関数
QAnywhere C++ API 181
~QAMessageListener 関数
QAnywhere C++ API 217
~QAMessage 関数
QAnywhere C++ API 216
~QATextMessage 関数
QAnywhere C++ API 221
~QATransactionalManager 関数
QAnywhere C++ API 224

A

ABS_RETRY_TIMEOUT フィールド
ネームスペース iAnywhere.QAnywhere.Client 229
ABS_RETRY_TIMEOUT 変数
QAnywhere C++ API 158
acknowledgeAll 関数
QAnywhere C++ API 179
AcknowledgeAll メソッド
ネームスペース iAnywhere.QAnywhere.Client 257
AcknowledgementMode クラス
QAnywhere C++ API 156
AcknowledgementMode 列挙
ネームスペース iAnywhere.QAnywhere.Client 226
acknowledgeUntil 関数
QAnywhere C++ API 180
AcknowledgeUntil メソッド
ネームスペース iAnywhere.QAnywhere.Client 257
acknowledge 関数
QAnywhere C++ API 179
Acknowledge メソッド
ネームスペース iAnywhere.QAnywhere.Client 256
ADAPTER フィールド
ネームスペース iAnywhere.QAnywhere.Client 229
ADAPTER 変数
QAnywhere C++ API 158
Address プロパティ
ネームスペース iAnywhere.QAnywhere.Client 292
automatic ポリシー
QAnywhere Agent 115

B

BodyLength プロパティ
ネームスペース iAnywhere.QAnywhere.Client 238
BrowseMessages メソッド
ネームスペース iAnywhere.QAnywhere.Client 266

C

castToBinaryMessage 関数

QAnywhere C++ API 202
castToTextMessage 関数
QAnywhere C++ API 203
clearProperties 関数
QAnywhere C++ API 203
ClearProperties メソッド
ネームスペース iAnywhere.QAnywhere.Client 295
close 関数
QAnywhere C++ API 183
Close メソッド
ネームスペース iAnywhere.QAnywhere.Client 266
commit 関数
QAnywhere C++ API 223
Commit メソッド
ネームスペース iAnywhere.QAnywhere.Client 319
COMMON_GET_INIT_FILE_ERROR 変数
QAnywhere C++ API 174
COMMON_INIT_ERROR 変数
QAnywhere C++ API 174
COMMON_INIT_THREAD_ERROR 変数
QAnywhere C++ API 174
COMMON_INVALID_PROPERTY 変数
QAnywhere C++ API 174
COMMON_MSG_NOT_WRITEABLE_ERROR 変数
QAnywhere C++ API 174
COMMON_MSG_RETRIEVE_ERROR 変数
QAnywhere C++ API 175
COMMON_MSG_STORE_ERROR 変数
QAnywhere C++ API 175
COMMON_MSG_STORE_NOT_INITIALIZED 変数
QAnywhere C++ API 175
COMMON_MSG_STORE_TOO_LARGE 変数
QAnywhere C++ API 175
COMMON_NO_DEST_ERROR 変数
QAnywhere C++ API 175
COMMON_NO_IMPLEMENTATION 変数
QAnywhere C++ API 176
COMMON_OPEN_ERROR 変数
QAnywhere C++ API 176
COMMON_OPEN_LOG_FILE_ERROR 変数
QAnywhere C++ API 176
COMMON_TERMINATE_ERROR 変数
QAnywhere C++ API 176
COMMON_UNEXPECTED_EOM_ERROR 変数
QAnywhere C++ API 176
COMPRESSED フィールド
ネームスペース iAnywhere.QAnywhere.Client 229
COMPRESSION_LEVEL プロパティ
QAManager のプロパティ 85
createBinaryMessage 関数
QAnywhere C++ API 183
CreateBinaryMessage メソッド
ネームスペース iAnywhere.QAnywhere.Client 267
createQAManager 関数
QAnywhere C++ API 198
CreateQAManager メソッド
ネームスペース iAnywhere.QAnywhere.Client 288
createQATransactionalManager 関数
QAnywhere C++ API 199
CreateQATransactionalManager メソッド
ネームスペース iAnywhere.QAnywhere.Client 288
createTextMessage 関数
QAnywhere C++ API 184
CreateTextMessage メソッド
ネームスペース iAnywhere.QAnywhere.Client 267
QAManager クラス 88
createTextMessage メソッド
QAManager クラス 87
-c オプション
QAnywhere Agent [qaagent] 105

D

dbeng9
QAnywhere Agent 102
dblsn ユーティリティ

QAnywhere Agent 102
dbmlsync ユーティリティ
QAnywhere Agent 102
DEFAULT_PRIORITY 変数
QAnywhere C++ API 202
DEFAULT_TIME_TO_LIVE 変数
QAnywhere C++ API 202
deleteMessage 関数
QAnywhere C++ API 184
deleteQAManager 関数
QAnywhere C++ API 199
deleteQATransactionalManager 関数
QAnywhere C++ API 199
Dispose メソッド
ネームスペース iAnywhere.QAnywhere.Client 258, 267, 289, 295, 320

E

EAServer
QAnywhere 11
ErrorCode プロパティ
ネームスペース iAnywhere.QAnywhere.Client 250
Expiration プロパティ
ネームスペース iAnywhere.QAnywhere.Client 293
EXPLICIT_ACKNOWLEDGEMENT 変数
QAnywhere C++ API 156

F

Finalize メソッド
ネームスペース iAnywhere.QAnywhere.Client 289
FROM_ADDR フィールド
ネームスペース iAnywhere.QAnywhere.Client 230
FROM_ADDR 変数
QAnywhere C++ API 159

G

getAddress 関数
QAnywhere C++ API 203
getBodyLength 関数
QAnywhere C++ API 164
getBooleanProperty 関数
QAnywhere C++ API 203
GetBooleanProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 295
getBooleanStoreProperty 関数
QAnywhere C++ API 184
GetBooleanStoreProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 268
getByteProperty 関数
QAnywhere C++ API 204
getByteStoreProperty 関数
QAnywhere C++ API 185
getDoubleProperty 関数
QAnywhere C++ API 204
GetDoubleProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 296
getDoubleStoreProperty 関数
QAnywhere C++ API 185
GetDoubleStoreProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 268
getExpiration 関数
QAnywhere C++ API 205
getFloatProperty 関数
QAnywhere C++ API 205
GetFloatProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 296
getFloatStoreProperty 関数
QAnywhere C++ API 186
GetFloatStoreProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 269
getInReplyToID 関数
QAnywhere C++ API 205

getIntProperty 関数
QAnywhere C++ API 206
GetIntProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 297
getIntStoreProperty 関数
QAnywhere C++ API 186
GetIntStoreProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 269
getLastErrorMsg 関数
QAnywhere C++ API 187, 200
getLastError 関数
QAnywhere C++ API 186, 200
getLongProperty 関数
QAnywhere C++ API 206
GetLongProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 298
getLongStoreProperty 関数
QAnywhere C++ API 187
GetLongStoreProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 270
getMessageID 関数
QAnywhere C++ API 206
getMessageNoWait 関数
QAnywhere C++ API 188
GetMessageNoWait メソッド
ネームスペース iAnywhere.QAnywhere.Client 271
getMessageTimeout 関数
QAnywhere C++ API 188
GetMessageTimeout メソッド
ネームスペース iAnywhere.QAnywhere.Client 272
getMessage 関数
QAnywhere C++ API 187
GetMessage メソッド
ネームスペース iAnywhere.QAnywhere.Client 271
getMode 関数
QAnywhere C++ API 188
getPriority 関数
QAnywhere C++ API 207
getPropertyNames 関数
QAnywhere C++ API 207
GetPropertyNames メソッド
ネームスペース iAnywhere.QAnywhere.Client 299
getPropertyType 関数
QAnywhere C++ API 207
GetPropertyType メソッド
ネームスペース iAnywhere.QAnywhere.Client 299
GetProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 298
getRedelivered 関数
QAnywhere C++ API 208
getReplyToAddress 関数
QAnywhere C++ API 208
GetSbyteProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 299
GetSbyteStoreProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 272
getShortProperty 関数
QAnywhere C++ API 208
GetShortProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 300
getShortStoreProperty 関数
QAnywhere C++ API 189
GetShortStoreProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 273
GetStoreProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 274
getStringProperty 関数
QAnywhere C++ API 209
GetStringProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 300
getStringStoreProperty 関数
QAnywhere C++ API 189

GetStringStoreProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 274
getTextLength 関数
QAnywhere C++ API 220
getText 関数
QAnywhere C++ API 219
getTimestampAsString 関数
QAnywhere C++ API 210
getTimestamp 関数
QAnywhere C++ API 210

I

ianywhere.connector.address プロパティ
QAnywhere 58
ianywhere.connector.compressionLevel プロパティ
QAnywhere 59
ianywhere.connector.id プロパティ
QAnywhere 58
ianywhere.connector.logLevel プロパティ
QAnywhere 58
ianywhere.connector.NativeConnection プロパティ
QAnywhere 58
ianywhere.connector.outgoing.deadMessageAddress プロパティ
QAnywhere 58
iAnywhere.QAnywhere.Client ネームスペース
ネームスペース iAnywhere.QAnywhere.Client 225
ID
QAnywhere アドレスについて 75
QAnywhere のサーバ・メッセージ・ストアへの追加 45
-id オプション
QAnywhere Agent [qaagent] 107
IMPLICIT_ACKNOWLEDGEMENT 変数
QAnywhere C++ API 156
InReplyToID プロパティ
ネームスペース iAnywhere.QAnywhere.Client 293
InstanceCount プロパティ
ネームスペース iAnywhere.QAnywhere.Client 287
InstanceID フィールド
ネームスペース iAnywhere.QAnywhere.Client 286
Instance プロパティ
ネームスペース iAnywhere.QAnywhere.Client 286
isOpen フィールド
ネームスペース iAnywhere.QAnywhere.Client 264
-iu オプション
QAnywhere Agent [qaagent] 108

J

JMS
メッセージング対応 Mobile Link の実行と JMS コネクタ 55
JMS から QAnywhere に送信されるメッセージのアドレス指定
説明 64
JMS コネクタ
QAnywhere アーキテクチャ 12
QAnywhere アドレス 75
プロパティ・ファイル 57
JMS コネクタの使用方法
QAnywhere 55
JMS コネクタ・プロパティ・ファイルの設定
QAnywhere 57
JMS のプロパティ
JMS メッセージの QAnywhere メッセージへのマッピング 66
JMS プロバイダ
QAnywhere アーキテクチャ 11
JMS メッセージの QAnywhere メッセージへのマッピング
説明 64

L

LastErrorMessage プロパティ
ネームスペース iAnywhere.QAnywhere.Client 265, 287
LastError プロパティ
ネームスペース iAnywhere.QAnywhere.Client 265, 287
-la オプション
QAnywhere Agent [qaagent] 109
Listener ユーティリティ
QAnywhere Agent 10, 102
LOG_FILE プロパティ
QAManager のプロパティ 85

M

MAX_IN_MEMORY_MESSAGE_SIZE プロパティ
QAManager のプロパティ 85
MessageID プロパティ
ネームスペース iAnywhere.QAnywhere.Client 293
MessageListener クラス
QAnywhere (C++) 92
QAnywhere システム・メッセージ 94
MessageProperties クラス
ネームスペース iAnywhere.QAnywhere.Client 227
QAnywhere C++ API 158
MessageProperties コンストラクタ
ネームスペース iAnywhere.QAnywhere.Client 228
MessageType クラス
QAnywhere C++ API 161
MessageType 列挙
ネームスペース iAnywhere.QAnywhere.Client 233
mgrBase フィールド
ネームスペース iAnywhere.QAnywhere.Client 264
Mobile Link
メッセージングのデータ同期アプリケーションへの統合 67
Mobile Link 同期サーバ
QAnywhere 44
Mobile Link に対するパスワード認証の使用
QAnywhere アプリケーション 129
Mode プロパティ
ネームスペース iAnywhere.QAnywhere.Client 265
-mp オプション
QAnywhere Agent [qaagent] 110
MSG_TYPE フィールド
ネームスペース iAnywhere.QAnywhere.Client 230
MSG_TYPE 変数
QAnywhere C++ API 159

N

NETWORK_STATUS_NOTIFICATION 変数
QAnywhere C++ API 161
NETWORK_STATUS フィールド
ネームスペース iAnywhere.QAnywhere.Client 230
NETWORK_STATUS 変数
QAnywhere C++ API 159
NETWORK フィールド
ネームスペース iAnywhere.QAnywhere.Client 230
NETWORK 変数
QAnywhere C++ API 159

O

ondemand ポリシー
QAnywhere Agent 113
onMessage 関数
QAnywhere C++ API 217
onMessage メソッド
QAManager クラス (C++) 91
-on オプション
QAnywhere Agent [qaagent] 111
open 関数

QAnywhere C++ API 180, 223
Open メソッド
ネームスペース iAnywhere.QAnywhere.Client 258, 320
-os オプション
QAnywhere Agent [qaagent] 112
-ot オプション
QAnywhere Agent [qaagent] 112
-o オプション
QAnywhere Agent [qaagent] 110

P

peekFirstMessage 関数
QAnywhere C++ API 190
peekNextMessage 関数
QAnywhere C++ API 190
-policy オプション
QAnywhere Agent [qaagent] 113
-port オプション
QAnywhere Agent [qaagent] 116
Priority プロパティ
ネームスペース iAnywhere.QAnywhere.Client 294
propertyExists 関数
QAnywhere C++ API 211
PropertyExists メソッド
ネームスペース iAnywhere.QAnywhere.Client 301
publishMessage 関数
QAnywhere C++ API 191
PUSH_NOTIFICATION 変数
QAnywhere C++ API 161
-push オプション
QAnywhere Agent [qaagent] 116
Push 通知
QAnywhere の概要 53
Push 通知とネットワーク・ステータスの変化の処理
QAnywhere 94
Push 通知によるメッセージング
QAnywhere アーキテクチャ 9
Push 通知によるメッセージングのシナリオ
QAnywhere 9
Push 通知の使用方法
QAnywhere 53
putMessageTimeToLive 関数
QAnywhere C++ API 191
PutMessageTimeToLive メソッド
ネームスペース iAnywhere.QAnywhere.Client 275
putMessage 関数
QAnywhere C++ API 191
PutMessage メソッド
ネームスペース iAnywhere.QAnywhere.Client 275

Q

qa.hpp
QAnywhere ヘッダ・ファイル 77
QA_NO_ERROR 変数
QAnywhere C++ API 176
qaagent ユーティリティ
構文 102
説明 49
QABinaryMessage クラス
ネームスペース iAnywhere.QAnywhere.Client 234
QAnywhere C++ API 162
インスタンス化 (.NET) 88
インスタンス化 (C++) 87
QAError クラス
QAnywhere C++ API 173
QAException クラス
ネームスペース iAnywhere.QAnywhere.Client 248
QAException コンストラクタ
ネームスペース iAnywhere.QAnywhere.Client 249, 250
QAManagerBase.MessageListener 委譲
ネームスペース iAnywhere.QAnywhere.Client 284
QAManagerBase クラス
ネームスペース iAnywhere.QAnywhere.Client 260

QAnywhere C++ API 182
QAManagerFactory クラス
ネームスペース iAnywhere.QAnywhere.Client 285
QAnywhere C++ API 198
インスタンス化 (C++) 77
初期化 (.NET) 81
QAManager クラス
ネームスペース iAnywhere.QAnywhere.Client 251
QAnywhere C++ API 178
インスタンス化 (.NET) 81
インスタンス化 (C++) 77
受信確認モード (.NET) 81
受信確認モード (C++) 78
初期化 (.NET) 81
初期化 (C++) 78
QAManager のプロパティ
COMPRESSION_LEVEL 85
LOG_FILE 85
MAX_IN_MEMORY_MESSAGE_SIZE 85
RECEIVER_INTERVAL 85
説明 83
プロパティ・ファイル 77, 81
リスト 85
QAManager のプロパティの設定
説明 83
QAMessageListener クラス
QAnywhere C++ API 217
QAMessage クラス
ネームスペース iAnywhere.QAnywhere.Client 290
QAnywhere C++ API 201
QAnywhere .NET API
概要 72
QAnywhere
JMS コネクタの使用方法 55
アーキテクチャ 6
アドレス 75
アプリケーションの配備 100
機能 4
クイック・スタート 13
クライアント側コンポーネントの設定 46
クライアント・メッセージ・ストアへの接続 105
サーバ側コンポーネントの設定 42
削除ルール 153
セキュリティ 123
説明 1
チュートリアル 15
通知の受信 91
転送ルール 131
転送ルール変数 143
フェールオーバ 69
プログラミング・インタフェース 72
メッセージングのデータ同期アプリケーションへの統合 67
QAnywhere Agent
簡単なメッセージング・アーキテクチャ 8
構文 102
説明 49
QAnywhere Agent の実行
説明 49
QAnywhere C++ API
概要 72
QAnywhere Manager のプロパティ vii
QAManager のプロパティを参照
QAnywhere Notifier
アーキテクチャ 10
QAnywhere アーキテクチャ
説明 6
QAnywhere から JMS に送信されるメッセージのアドレス指定
説明 61
QAnywhere クライアント
概要 8
停止 99
QAnywhere クライアント API
初期化 77
QAnywhere クライアント API の初期化
説明 77
QAnywhere クライアント・アプリケーション
作成 71
QAnywhere クライアント・アプリケーショ

ンの作成
説明　71
QAnywhere 転送ルール
説明　131
QAnywhere ネームスペース
インクルード　81
QAnywhere の概要　1
QAnywhere の機能　4
QAnywhere の停止
説明　99
QAnywhere の配備
説明　100
QAnywhere のプロパティ　vii
QAManager のプロパティを参照
QAnywhere メッセージの JMS メッセージへのマッピング　63
QAnywhere ヘッダ・ファイル
qa.hpp　77
QAnywhere メッセージ・アドレスについて
説明　75
QAnywhere メッセージの JMS メッセージへのマッピング
説明　62
QAnywhere メッセージの送信
説明　87
QAnywhere メッセージングと Mobile Link データ同期の統合
説明　67
QAnywhere メッセージングの設定
説明　41
QAnywhere メッセージング用 Mobile Link 同期サーバの起動
説明　44
QAPropertyType 列挙
ネームスペース iAnywhere.QAnywhere.Client　307
qastop ユーティリティ
qaagent -qi (クワイエット・モード) での使用　117
qaagent -q (クワイエット・モード) での使用　117
QATextMessage クラス
ネームスペース iAnywhere.QAnywhere.Client　308
QAnywhere C++ API　218
インスタンス化 (.NET)　88
インスタンス化 (C++)　87
QATransactionalManager クラス
ネームスペース iAnywhere.QAnywhere.Client　314
QAnywhere (C++)　96
QAnywhere C++ API　222
-qi オプション
QAnywhere Agent [qaagent]　117
-q オプション
QAnywhere Agent [qaagent]　117

R

readBinary 関数
QAnywhere C++ API　164
ReadBinary メソッド
ネームスペース iAnywhere.QAnywhere.Client　239
readBoolean 関数
QAnywhere C++ API　165
ReadBoolean メソッド
ネームスペース iAnywhere.QAnywhere.Client　239
readByte 関数
QAnywhere C++ API　165
readChar 関数
QAnywhere C++ API　166
ReadChar メソッド
ネームスペース iAnywhere.QAnywhere.Client　240
readDouble 関数
QAnywhere C++ API　166
ReadDouble メソッド
ネームスペース iAnywhere.QAnywhere.Client　240
readFloat 関数
QAnywhere C++ API　166
ReadFloat メソッド
ネームスペース iAnywhere.QAnywhere.Client　241

readInt 関数
QAnywhere C++ API 167
ReadInt メソッド
ネームスペース iAnywhere.QAnywhere.Client 241
readLong 関数
QAnywhere C++ API 167
ReadLong メソッド
ネームスペース iAnywhere.QAnywhere.Client 241
ReadSbyte メソッド
ネームスペース iAnywhere.QAnywhere.Client 242
readShort 関数
QAnywhere C++ API 168
ReadShort メソッド
ネームスペース iAnywhere.QAnywhere.Client 242
readString 関数
QAnywhere C++ API 168
ReadString メソッド
ネームスペース iAnywhere.QAnywhere.Client 242
readText 関数
QAnywhere C++ API 220
ReadText メソッド
ネームスペース iAnywhere.QAnywhere.Client 312
recover 関数
QAnywhere C++ API 180
Recover メソッド
ネームスペース iAnywhere.QAnywhere.Client 259
Redelivered プロパティ
ネームスペース iAnywhere.QAnywhere.Client 294
REGULAR 変数
QAnywhere C++ API 161
ReplyToAddress プロパティ
ネームスペース iAnywhere.QAnywhere.Client 294
reset 関数
QAnywhere C++ API 168
Reset メソッド
ネームスペース iAnywhere.QAnywhere.Client 243
RETRY_FAILED_ADDR フィールド
ネームスペース iAnywhere.QAnywhere.Client 231
RETRY_FAILED_ADDR 変数
QAnywhere C++ API 160
RETRY_FAILED_PRIORITY フィールド
ネームスペース iAnywhere.QAnywhere.Client 231
RETRY_FAILED_PRIORITY 変数
QAnywhere C++ API 160
RETRY_FAILED フィールド
ネームスペース iAnywhere.QAnywhere.Client 231
RETRY_FAILED 変数
QAnywhere C++ API 159
RETRY_MAX フィールド
ネームスペース iAnywhere.QAnywhere.Client 232
RETRY_MAX 変数
QAnywhere C++ API 160
RETRY_TIMEOUT フィールド
ネームスペース iAnywhere.QAnywhere.Client 232
RETRY_TIMEOUT 変数
QAnywhere C++ API 160
rollback 関数
QAnywhere C++ API 224
Rollback メソッド
ネームスペース iAnywhere.QAnywhere.Client 321

S

scheduled ポリシー
QAnywhere Agent 114
setAddress 関数
QAnywhere C++ API 211
setBooleanProperty 関数
QAnywhere C++ API 212
SetBooleanProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client

301
setBooleanStoreProperty 関数
QAnywhere C++ API 192
SetBooleanStoreProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 276
setByteProperty 関数
QAnywhere C++ API 212
setByteStoreProperty 関数
QAnywhere C++ API 192
setDoubleProperty 関数
QAnywhere C++ API 212
SetDoubleProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 302
setDoubleStoreProperty 関数
QAnywhere C++ API 193
SetDoubleStoreProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 276
setFloatProperty 関数
QAnywhere C++ API 213
SetFloatProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 302
setFloatStoreProperty 関数
QAnywhere C++ API 193
SetFloatStoreProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 277
setInReplyToID 関数
QAnywhere C++ API 213
setIntProperty 関数
QAnywhere C++ API 213
SetIntProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 303
setIntStoreProperty 関数
QAnywhere C++ API 193
SetIntStoreProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 277
setLongProperty 関数
QAnywhere C++ API 214
SetLongProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 303
setLongStoreProperty 関数
QAnywhere C++ API 194
SetLongStoreProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 278
setMessageID 関数
QAnywhere C++ API 214
setMessageListener 関数
QAnywhere C++ API 194
SetMessageListener メソッド
ネームスペース iAnywhere.QAnywhere.Client 278
setPriority 関数
QAnywhere C++ API 214
setProperty 関数
QAnywhere C++ API 195
SetProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 279, 304
setRedelivered 関数
QAnywhere C++ API 215
setReplyToAddress 関数
QAnywhere C++ API 215
SetSbyteProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 304
SetSbyteStoreProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 280
setShortProperty 関数
QAnywhere C++ API 215
SetShortProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 305
setShortStoreProperty 関数
QAnywhere C++ API 195
SetShortStoreProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 280
SetStoreProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client

281
setStringProperty 関数
QAnywhere C++ API 216
SetStringProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 305
setStringStoreProperty 関数
QAnywhere C++ API 196
SetStringStoreProperty メソッド
ネームスペース iAnywhere.QAnywhere.Client 281
setText 関数
QAnywhere C++ API 220
setTimestamp 関数
QAnywhere C++ API 216
-si オプション
QAnywhere Agent [qaagent] 118
SMS
QAnywhere 通知の使用方法 54
SMS を介した Push 通知の使用方法
QAnywhere 54
SQL Anywhere Studio
マニュアル viii
start 関数
QAnywhere C++ API 196
Start メソッド
ネームスペース iAnywhere.QAnywhere.Client 282
stop 関数
QAnywhere C++ API 196
Stop メソッド
ネームスペース iAnywhere.QAnywhere.Client 282
-su オプション
QAnywhere Agent [qaagent] 119

T

TestMessage アプリケーション
QAnywhere チュートリアル 15, 23
ソース・コード 29
TextLength プロパティ
ネームスペース iAnywhere.QAnywhere.Client 312
Text プロパティ
ネームスペース iAnywhere.QAnywhere.Client 312
Timestamp プロパティ
ネームスペース iAnywhere.QAnywhere.Client 294
TRANSACTIONAL 変数
QAnywhere C++ API 156
triggerSendReceive 関数
QAnywhere C++ API 196
TriggerSendReceive メソッド
ネームスペース iAnywhere.QAnywhere.Client 282

V

-v オプション
QAnywhere Agent [qaagent] 120

W

WebLogic
QAnywhere 11
writeBinary 関数
QAnywhere C++ API 169
WriteBinary メソッド
ネームスペース iAnywhere.QAnywhere.Client 243
writeBoolean 関数
QAnywhere C++ API 169
WriteBoolean メソッド
ネームスペース iAnywhere.QAnywhere.Client 244
writeByte 関数
QAnywhere C++ API 169
writeChar 関数
QAnywhere C++ API 170
WriteChar メソッド
ネームスペース iAnywhere.QAnywhere.Client 244
writeDouble 関数

QAnywhere C++ API 170
WriteDouble メソッド
ネームスペース iAnywhere.QAnywhere.Client 245
writeFloat 関数
QAnywhere C++ API 170
WriteFloat メソッド
ネームスペース iAnywhere.QAnywhere.Client 245
writeInt 関数
QAnywhere C++ API 171
WriteInt メソッド
ネームスペース iAnywhere.QAnywhere.Client 245
writeLong 関数
QAnywhere C++ API 171
WriteLong メソッド
ネームスペース iAnywhere.QAnywhere.Client 246
WriteSbyte メソッド
ネームスペース iAnywhere.QAnywhere.Client 246
writeShort 関数
QAnywhere C++ API 171
WriteShort メソッド
ネームスペース iAnywhere.QAnywhere.Client 247
writeString 関数
QAnywhere C++ API 172
WriteString メソッド
ネームスペース iAnywhere.QAnywhere.Client 247
writeText 関数
QAnywhere C++ API 221
WriteText メソッド
ネームスペース iAnywhere.QAnywhere.Client 313

X

xjms.jndi.authName プロパティ
QAnywhere 59
xjms.jndi.factory プロパティ
QAnywhere 59
xjms.jndi.password.e プロパティ
QAnywhere 59
xjms.jndi.url プロパティ
QAnywhere 59
xjms.password.e プロパティ
QAnywhere 59
xjms.queueConnectionAuthName プロパティ
QAnywhere 59
xjms.queueConnectionPassword.e プロパティ
QAnywhere 59
xjms.queueFactory プロパティ
QAnywhere 59
xjms.receiveDestination プロパティ
QAnywhere 60
xjms.topicConnectionAuthName プロパティ
QAnywhere 59
xjms.topicConnectionPassword.e プロパティ
QAnywhere 60
xjms.topicFactory プロパティ
QAnywhere 59
-x オプション
QAnywhere Agent [qaagent] 121

あ

アーキテクチャ
QAnywhere 6
メッセージングのデータ同期アプリケーションへの統合 67
アイコン
マニュアルで使用 xiii
圧縮
QAnywhere JMS コネクタ 59
アドレス
QAnywhere 75
QAnywhere JMS コネクタ 75
QAnywhere メッセージの設定 (.NET) 88
QAnywhere メッセージの設定 (C++) 87
アプリケーション間メッセージング vii
メッセージングを参照
QAnywhere 2
暗号化

QAnywhere クライアント・メッセージ・ストア 125
QAnywhere 通信ストリーム 127

か

外部メッセージング・システムとのメッセージング
QAnywhere アーキテクチャ 11
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
QAnywhere 11
カスタムのクライアント・ストア・プロパティ
QAnywhere 148
簡単なメッセージング
QAnywhere アーキテクチャ 7
簡単なメッセージング・シナリオ
QAnywhere 7
簡単なメッセージングを行う Mobile Link の実行
QAnywhere 44

き

規則
表記 xi
キュー
QAnywhere アドレスについて 75

く

クイック・スタート
QAnywhere 13
クライアント側コンポーネントの設定
QAnywhere 46
クライアント・ストア・プロパティ
QAnywhere 146
QAnywhere カスタム 148
QAnywhere 事前定義 146
クライアント・メッセージ・ストア
ID の作成 47
QAnywhere アーキテクチャ 8
QAnywhere チュートリアル 21
QAnywhere での設定 46
QAnywhere の暗号化 125
QAnywhere のセキュリティ 124
-si オプションによる初期化 118
カスタムのクライアント・ストア・プロパティ 148
クライアント・ストア・プロパティ 146
事前定義のクライアント・ストア・プロパティ 146
クライアント・メッセージ・ストア ID
QAnywhere のサーバ・メッセージ・ストアへの追加 45
クライアント・メッセージ・ストア ID の作成
QAnywhere 47
クライアント・メッセージ・ストア ID の追加
QAnywhere 45
クライアント・メッセージ・ストアの暗号化
QAnywhere 125
クライアント・メッセージ・ストアの設定
QAnywhere 46
クライアント・メッセージ・ストアのパスワード管理
QAnywhere 125
クワイエット・モード
QAnywhere Agent [qaagent] -q 117
QAnywhere Agent [qaagent] -qi 117

こ

コネクタ
QAnywhere JMS コネクタ・プロパティ・ファイル 57
QAnywhere での複数設定 60

さ

サーバ側コンポーネントの設定
QAnywhere 42
サーバ・メッセージ・ストア
QAnywhere アーキテクチャ 7
QAnywhere での設定 42
サーバ・メッセージ・ストアの設定
QAnywhere 42
サイズの大きなメッセージの読み込み
説明 93
削除ルール
QAnywhere 153
作成
QAnywhere サーバ・メッセージ・ストア 42
サポート
ニュースグループ xv

し

システム・メッセージ
QAnywhere 94
事前定義のクライアント・ストア・プロパティ
QAnywhere 146
持続性
QAnywhere メッセージ 153
受信確認モード
QAManager クラス (.NET) 81
QAManager クラス (C++) 78
条件
QAnywhere スケジュール構文 138
条件構文
QAnywhere 138

す

スケジュール
QAnywhere 転送ルール 136
スケジュール構文
QAnywhere 転送ルール 136

せ

セキュリティ
QAnywhere 123
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
QAnywhere 124
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
QAnywhere 123
設定
QAnywhere 13

ち

チュートリアル
QAnywhere 15

つ

通信ストリーム
QAnywhere での暗号化 127
通信ストリームの暗号化
QAnywhere 127
通知
QAnywhere での処理 94
QAnywhere と SMS 54
QAnywhere の概要 53
QAnywhere の通知の無効化 116

て

テクニカル・サポート
ニュースグループ xv
転送ルール
QAnywhere 131
QAnywhere 構文 136
QAnywhere の例 133
削除ルール 153
説明 132
変数 143

転送ルール変数
QAnywhere 143

と

同期的なメッセージ受信
QAnywhere 89
トランザクション
QAnywhere メッセージ 96
トランザクション志向メッセージング
QAnywhere 96
トランザクション志向メッセージングの実装
説明 96

に

ニュースグループ
テクニカル・サポート xv
認証
QAnywhere 129

ね

ネットワーク・ステータス
QAnywhere でのステータス変化の処理 94

は

はじめに
QAnywhere 13

ひ

非同期的なメッセージ受信
QAnywhere 91
表記
規則 xi

ふ

ファイルによるプロパティの設定
QAManager 83
フィードバック
提供 xv
マニュアル xv
フェールオーバ
QAnywhere 69
フェールオーバの設定
QAnywhere 69
複数のコネクタの設定
QAnywhere 60
プログラムによるプロパティの設定
QAManager 84
プロパティ
QAManager 85

へ

変数
QAnywhere 転送ルール 143

ほ

ポリシー
QAnywhere 50
QAnywhere アーキテクチャ 8
QAnywhere チュートリアル 24

ま

マニュアル
SQL Anywhere Studio viii

め

メッセージ・アドレス
QAnywhere 75
メッセージ・ストア
QAnywhere クライアント・アーキテクチャ

8
QAnywhere クライアント・メッセージ・ストアの暗号化 125
QAnywhere クライアント・メッセージ・ストアの作成 46
QAnywhere サーバ・アーキテクチャ 7
QAnywhere サーバ・メッセージ・ストアの作成 42
メッセージ・ストア ID
QAnywhere のサーバ・メッセージ・ストアへの追加 45
メッセージ転送
QAnywhere 131
メッセージの受信
QAnywhere (.NET) 90
QAnywhere (C++) 89
メッセージの送信
QAnywhere 87
QAnywhere 内のトランザクション 97
メッセージの同期的受信
QAnywhere 89
メッセージの非同期的受信
QAnywhere 91
メッセージ・プロパティ
QAnywhere 144
メッセージ・プロパティ・ファイル
QAnywhere 56
メッセージ・ヘッダ
QAnywhere 転送ルール 143
メッセージ・リスナ
QAnywhere 91
メッセージング vii
QAnywhere を参照
QAnywhere アドレス 75
QAnywhere クイック・スタート 13
メッセ - ジング
QAnywhere の機能 4
メッセージング
アプリケーション間 2
メッセージング・システム
JMS と QAnywhere の統合 55
メッセージング対応 Mobile Link
QAnywhere チュートリアル 17
QAnywhere の設定 41
簡単なメッセージング・アーキテクチャ 8
起動 44
メッセージング対応 Mobile Link の実行と JMS コネクタ
QAnywhere 56

よ

読み込み
サイズの大きな QAnywhere メッセージ 93

る

ルール vii
転送ルールを参照
ルール・ファイル
QAnywhere Agent -policy オプション 115
QAnywhere の説明 132
ルール変数
QAnywhere 転送ルール 143