QAnywhere

バージョン 12.0.1

2012 年 1 月

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

バージョン 12.0.1
2012 年 1 月

Copyright © 2012 iAnywhere Solutions, Inc. Portions copyright © 2012 Sybase, Inc. All rights reserved.

iAnywhere との間に書面による合意がないかぎり、このマニュアルは現状のまま提供されるものであり、その使用または記載内容の誤りに対して一切の責任を負いません。

次の条件に従うかぎり、このマニュアルの一部または全体を使用、印刷、複製、配布することができます。1) マニュアルの一部または全体にかかわらず、ここに示したものとそれ以外のすべての版権と商標の表示をすべてのコピーに含めること。2) マニュアルに変更を加えないこと。3) iAnywhere 以外の人間がマニュアルの著者または情報源であるかのように示す一切の行為をしないこと。

iAnywhere®、Sybase®、http://www.sybase.com/detail?id=1011207 に示す商標は Sybase, Inc. またはその関連会社の商標です。® は米国での登録商標を示します。

このマニュアルに記載されているその他の会社名と製品名は各社の商標である場合があります。

目次

はじめに vii

QAnywhere テクノロジ 1

QAnywhere の機能 2
QAnywhere アーキテクチャー 3
QAnywhere メッセージ配信 10
SQL Anywhere と Ultra Light のいずれを使用するかの決定 10
QAnywhere 12 プラグイン 11
QAnywhere のクイックスタート 12

QAnywhere メッセージ 15

メッセージヘッダー 15
メッセージプロパティ 16
送信先の概要 17

QAnywhere メッセージストア 19

ローカルメッセージストア 19
サーバーメッセージストア 23
クライアントメッセージストア 25

QAnywhere メッセージングの設定 31

サーバーサイドコンポーネントの設定 31
クライアントサイドコンポーネントの設定 34
Push 通知 34
フェールオーバーメカニズムの設定 38

QAnywhere Agent 41

メッセージ転送ポリシー 41
転送ルール 46

削除ルール 46
QAnywhere Agent の起動 47
QAnywhere Agent の配備 49
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 49
信頼性の低いネットワーク 49

QAnywhere クライアントアプリケーションの作成 51

QAnywhere のインターフェイス 51
クライアントアプリケーション作成のクイックスタート 53
QAnywhere API の初期化 54
QAnywhere メッセージアドレス 60
QAnywhere メッセージの送信 63
QAnywhere メッセージのキャンセル 70
QAnywhere メッセージの受信 71
サイズの大きなメッセージ 75
QAnywhere のメッセージの参照 76
QAnywhere での例外の処理 80
QAnywhere の停止 84
マルチスレッドでの考慮事項 84
QAnywhere Manager の設定プロパティ 85

QAnywhere スタンドアロンクライアント 91

スタンドアロンクライアントのメッセージストア 91
スタンドアロンクライアントの配備 92
スタンドアロンクライアント API 92

モバイル Web サービス 95

モバイル Web サービスの設定 95
iAnywhere WSDL コンパイラー 97
モバイル Web サービスアプリケーション 98
モバイル Web サービスアプリケーションのコンパイルと実行 104
Web サービス要求 105
モバイル Web サービスの例 107

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

QAnywhere アプリケーションの配備 117

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 121

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 121
通信ストリームの暗号化 123
Mobile Link に対するパスワード認証 123
サーバー管理要求のセキュリティ保護 124
Mobile Link ユーザー認証ユーティリティを使用したユーザーの追加 124
Relay Server を使用する場合のセキュリティ 125

サーバーメッセージストアの管理 127

転送ルール 127
メッセージのアーカイブの管理 128
サーバー管理要求を使用したサーバーメッセージストアの管理 129

クライアントメッセージストアの管理 131

QAnywhere クライアントのモニター 131
クライアントプロパティのモニター 131
クライアントメッセージストアプロパティの管理 131

送信先エイリアス 133

送信先エイリアスの作成 133

コネクター 135

JMS コネクター 135
JMS コネクターの設定 135
JMS コネクターへの QAnywhere メッセージの送信 139
JMS コネクターから QAnywhere クライアントへのメッセージの送信 140
Web サービスコネクター 144
チュートリアル : JMS コネクターの使用 148

サーバー管理要求 153

サーバー管理要求の記述 154
サーバー管理要求を使用したサーバーメッセージストアの管理 156
サーバー管理要求によるコネクターの管理 159
サーバー管理要求でのサーバープロパティの設定 168
サーバー管理要求での転送ルールの指定 169
サーバー管理要求を使用した送信先エイリアスの作成 170
QAnywhere のモニター 173

チュートリアル：TestMessage の解説 177

レッスン 1：メッセージング対応 Mobile Link の起動 177
レッスン 2：TestMessage アプリケーションの実行 180
レッスン 3：メッセージの送信 182
レッスン 4：TestMessage クライアントのソースコードの概要 183
クリーンアップ 187

QAnywhere リファレンス 189

クライアント用 QAnywhere .NET API リファレンス 189
Web サービス用 QAnywhere .NET API リファレンス 325
クライアント用 QAnywhere C++ API リファレンス 375
クライアント用 QAnywhere Java API リファレンス 495
Web サービス用 QAnywhere Java API リファレンス 620
QAnywhere SQL API リファレンス 655
メッセージヘッダーとメッセージプロパティ 695
サーバー管理要求のリファレンス 704
QAnywhere Agent ユーティリティのリファレンス 712
QAnywhere のプロパティ 761
QAnywhere 転送ルールと削除ルール 774

索引 789

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

はじめに

このマニュアルでは、モバイルクライアント、ワイヤレスクライアント、デスクトップクライアント、およびラップトップクライアント用のメッセージングプラットフォームである、QAnywhere について説明します。

QAnywhere テクノロジ

QAnywhere は、モバイルデバイス用のアプリケーション間メッセージングの開発を支援します。アプリケーション間メッセージングを使用すると、モバイルデバイスや無線デバイスで動作しているカスタムプログラムと、集中管理されているサーバーアプリケーションとの間で通信できます。

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

QAnywhere によって、随時接続環境での通信が実現します。QAnywhere は、接続速度が遅い、場所によっては接続できないことがある、接続が切断されるなどの無線ネットワーク特有の問題に対応しています。QAnywhere メッセージングは蓄積転送機能を備えています。つまり、送信先アプリケーションにネットワークを介して接続できない場合でも、メッセージを作成しておくと、後でネットワークが使用可能になったときにそのメッセージが配信されます。

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

QAnywhere には、次のような追加機能があります。

- パブリックネットワークを介して送信されるすべてのメッセージを暗号化し、機密情報を保護します。

- 転送ルールを使用してメッセージ配信をカスタマイズすれば、優先度の低い大きなメッセージがピーク時以外の時間に配信されるようにすることなども可能です。

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

- QAnywhere は、モバイルアプリケーションとエンタープライズサーバーの間で送信されるデータを圧縮します。

- プログラミングインターフェイスとして C++、Java、.NET および SQL の API が用意されているので、開発者は自分のスキルに応じてソリューションを選択できます。

- QAnywhere を使用すると、JMS インターフェイ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、Java EE アプリケーションとの統合が可能になります。

●QAnywhere にはモバイル Web サービスインターフェイスが含まれており、エンタープライズ Web サービスを基礎とする信頼性の高いモバイルアプリケーションの作成を支援します。

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

QAnywhere の機能

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

- **QAnywhere API**　オブジェクト指向の QAnywhere API は、Windows デスクトップ向けや Windows Mobile デバイス向けのメッセージングアプリケーションを構築するためのインフラストラクチャとして使用できます。QAnywhere API は Java、C++、.NET、SQL で使用できます。

- **蓄積転送**　QAnywhere アプリケーションは、クライアントとサーバーの間の接続が確立されてデータ転送が可能になるまで、ローカルにメッセージを格納しておきます。

- **データ同期の補完**　QAnywhere アプリケーションは、リレーショナルデータベースを一時的なメッセージストアとして使用します。リレーショナルデータベースを使用することで、メッセージストアのセキュリティ保護、トランザクションベースのコンピューティング、リレーショナルデータベースのその他の利点が得られます。

  SQL Anywhere リレーショナルデータベースをメッセージストアとして使用すると、QAnywhere とデータ同期ソリューションを容易に組み合わせることができます。これは、どちらも Mobile Link 同期を使用して、クライアント／サーバー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 インターフェイスをサポートする外部メッセージングシステムに QAnywhere クライアントを統合することもできます。

- **暗号化**　メッセージは、トランスポートレイヤーセキュリティを使用し、暗号化して送信できます。また、単純暗号化または FIPS 認定の任意の AES アルゴリズムを使用して暗号化できます。

- **圧縮**　メッセージの内容は、一般的な ZLIB 圧縮ライブラリを使用して圧縮してから格納できます。

- **認証**　QAnywhere クライアントの認証は、組み込みの機能を使用したり、組織で使用している認証サーバーなどのカスタム認証スクリプトを使用したりして実行できます。

- **複数ネットワーク対応**　QAnywhere は、TCP/IP または HTTP をサポートしている有線ネットワークまたは無線ネットワークのすべてで動作します。

- **フェールオーバー**　複数の Mobile Link サーバーを実行し、いずれかのサーバーで障害が発生したときに代替サーバーに処理が引き継がれるようにすることができます。

- **管理**　QAnywhere アプリケーションは、クライアント側とサーバー側でメッセージの参照と操作を実行できます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- **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。

- **サーバーによって開始されるメッセージの送受信**　QAnywhere を使用すると、クライアントデバイスにメッセージをプッシュできるので、クライアントアプリケーションにメッセージ駆動型のロジックを実装できます。

- **転送ルール**　転送ルールを作成すると、メッセージの転送を実行するタイミングを指定できます。

- **再開可能なダウンロード**　サイズの大きなメッセージやまとまった量のメッセージがある場合、それらのメッセージは少しずつ QAnywhere クライアントに送信されます。これによって、ネットワーク障害が発生した場合のデータの再転送が最小限に抑えられます。

- **配信の保証**　QAnywhere を使用すると、メッセージが確実に配信されます。

- **モバイル Web サービス**　モバイル Web サービスは、QAnywhere 経由の Web サービス要求と応答の転送を容易にします。

QAnywhere アーキテクチャー

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ーについて説明します。まず、簡単なメッセージングシナリオを取り上げ、次に、より高度なシナリオについて解説します。

クライアントアプリケーションは、QAnywhere API を使用してメッセージの送受信を行います。メッセージは、クライアントメッセージストア内にキューイングされます。メッセージ転送とは、クライアントメッセージストアと、中央の QAnywhere サーバー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。

- **ローカルのアプリケーション間メッセージング**　ローカルメッセージストアとしての SQL Anywhere データベースを使用して、アプリケーション間でメッセージを交換する場合です。メッセージは、アプリケーションからデータベースへの接続を介してアプリケーション間で転送されます。

- **簡単なクライアント／サーバーメッセージング**　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ーメッセージストアの間でメッセージを転送するタイミングを制御します。

  「簡単なメッセージングシナリオ」5 ページを参照してください。

- **Push 通知によるクライアント／サーバーメッセージング**　QAnywhere クライアント間でメッセージを交換する場合です。このシナリオでは、Mobile Link サーバーがクライアント間の

メッセージ転送を開始できます。これは、クライアントとサーバーのメッセージストア間でメッセージを交換することによって行われます。

- **外部メッセージングシステムによるクライアント／サーバーメッセージング**　JMS プロバイダーを提供する外部システム (BEA WebLogic や Sybase EAServer など) を介して QAnywhere クライアント間でメッセージ交換を行う場合です。

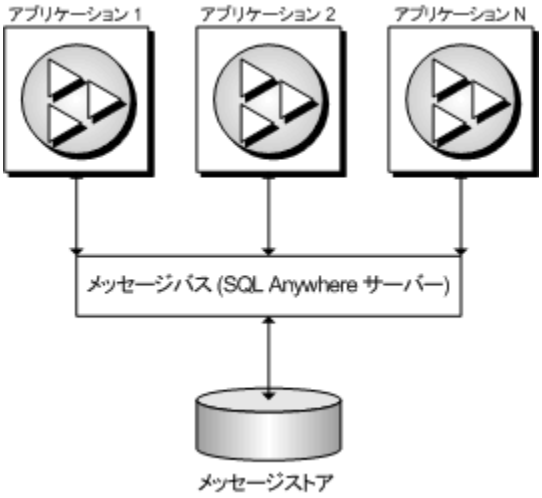
Push 通知と外部メッセージングシステムを組み合わせて使用すると、最も汎用的なソリューションを実現できます。

参照

- 「ローカルメッセージストアの設定」19 ページ
- 「簡単なメッセージングシナリオ」5 ページ
- 「Push 通知によるメッセージングのシナリオ」7 ページ
- 「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」9 ページ

アプリケーション間メッセージングのシナリオ

アプリケーション間メッセージングは、ローカルメッセージストアとして動作する SQL Anywhere データベースを介してメッセージを転送するアプリケーションで構成されます。メッセージは、アプリケーションからデータベースへの接続を介して、アプリケーション間で転送されます。次の図は、2 つのアプリケーションがローカルメッセージストアを使用してアプリケーション間でメッセージを転送する一般的なシナリオを示しています。



このアーキテクチャーは、以下のコンポーネントで構成されています。

- **ローカルメッセージストア**　メッセージは、SQL Anywhere データベースに格納されます。データベースは、ローカルメッセージストアとして設定する必要があります。

- **QAnywhere Agent**　QAnywhere Agent は、メッセージ転送を管理します。この処理は QAnywhere アプリケーションから独立しています。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- **QAnywhere アプリケーション**　QAnywhere C++、Java、または .NET API を使用して作成されたアプリケーションから、メッセージを送受信するためのメソッドが呼び出されます。クライアントアプリケーションで使用される基本オブジェクトは QAManager です。

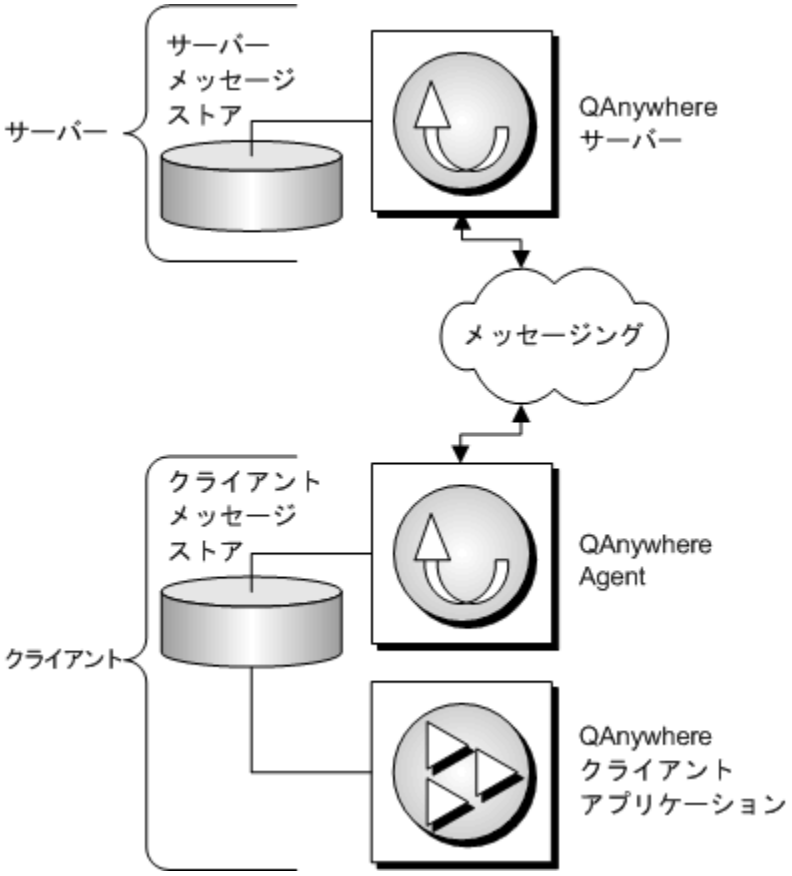
メッセージの送信と受信は、SQL Anywhere データベースに接続されている QAnywhere アプリケーションによって行われます。メッセージは、アプリケーションがメッセージ転送を開始するまで取り出されません。QAnywhere アプリケーションは、「ポリシー」に基づいて、メッセージ転送を実行するタイミングを決定します。ポリシーには、オンデマンド、自動、スケジュール済み、カスタムがあります。オンデマンドポリシーでは、ユーザーがメッセージ転送を制御できます。自動ポリシーでは、クライアントとの間でメッセージの配信準備が行われるたびにメッセージの転送が開始されます。カスタムポリシーでは、転送ルールを使用してメッセージ転送に詳細な制御を追加します。

参照

- 「QAnywhere Agent の起動」47 ページ
- 「QAnywhere クライアントアプリケーションの作成」51 ページ
- 「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」49 ページ

簡単なメッセージングシナリオ

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



このアーキテクチャーは、以下のコンポーネントで構成されています。

- **サーバーメッセージストア**　サーバー側ではメッセージがリレーショナルデータベースに格納されます。データベースは Mobile Link 統合データベースとして設定する必要があります。サポートされている統合データベースであればいずれのデータベースでもかまいません。

- **クライアントメッセージストア**　各クライアントのメッセージはリレーショナルデータベースに格納されます。QAnywhere では、SQL Anywhere データベースと Ultra Light データベースをサポートしています。データ同期アプリケーションには、SQL Anywhere データベースを推奨します。メッセージの格納と転送専用のアプリケーションには、Ultra Light データベースを推奨します。

- **QAnywhere サーバー**　QAnywhere サーバーは、メッセージングが有効になっている Mobile Link サーバーです。Mobile Link 同期は、QAnywhere クライアントと QAnywhere サーバーの間で、メッセージの転送や追跡のための手段として使用されます。Mobile Link は、セキュリティ、認証、暗号化、柔軟性を実現します。また、Mobile Link が使用されている場合、メッセージングとデータ同期を組み合わせることができます。

  QAnywhere サーバーを起動するには、-m オプションを指定して Mobile Link サーバーを起動します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- **QAnywhere Agent**　QAnywhere Agent は、クライアント側のメッセージ送信を管理します。この処理は QAnywhere クライアントアプリケーションから独立しています。

- **QAnywhere クライアントアプリケーション**　QAnywhere C++、Java、または .NET API を使用して作成されたアプリケーションから、メッセージを送受信するためのメソッドが呼び出されます。クライアントアプリケーションで使用される基本オブジェクトは QAManager です。

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。サーバー側のメッセージは、クライアントがメッセージ転送を開始するまで取り出されません。QAnywhere クライアントは、「ポリシー」に基づいて、メッセージ転送を実行するタイミングを決定します。ポリシーには、オンデマンド、自動、スケジュール設定、カスタムがあります。オンデマンドポリシーでは、ユーザーがメッセージ転送を制御できます。自動ポリシーでは、クライアントとの間でメッセージの配信準備が行われるたびにメッセージの転送が開始されます。カスタムポリシーでは、転送ルールを使用してメッセージ転送に詳細な制御を追加します。

参照

- 「有効な Mobile Link での QAnywhere の起動」31 ページ
- 「QAnywhere クライアントアプリケーションの作成」51 ページ
- 「QAnywhere Agent の起動」47 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」49 ページ

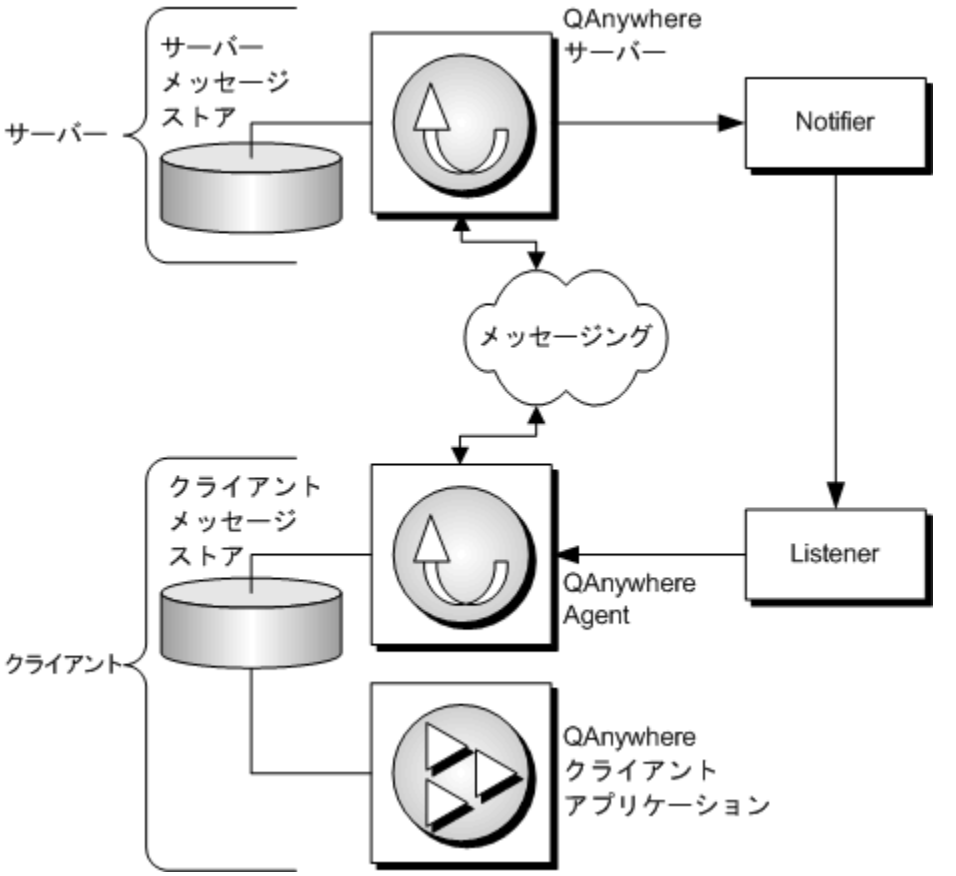
Push 通知によるメッセージングのシナリオ

Push 通知は、サーバーから 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ーメッセージストアに着信したときに送信されます。メッセージングサーバーは、受信側クライアントの Listener に Push 要求を自動的に通知します。クライアントは通知を受信するとメッセージ転送を開始し、サーバーで待機しているメッセージを受信したり、カスタムアクションを実行したりします。

Push 通知を使用できるようにするため、QAnywhere アーキテクチャーには 2 つのコンポーネントが追加されています。サーバー側では、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ーメッセージストアからクライアントメッセージストアに送信されます。ただし、その場合は、スケジュール転送ポリシーなどを使用してクライアント側からメッセージ転送を開始する必要があります。

Push 通知によるメッセージングのアーキテクチャーは、「簡単なメッセージングシナリオ」5 ページで説明したアーキテクチャーを拡張したものです。次の図は、このアーキテクチャーを示しています。



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

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

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

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

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」49 ページ
- 「Push 通知」34 ページ
- 「非同期的なメッセージ受信」73 ページ
- 「サーバー起動同期」『Mobile Link サーバー起動同期』

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

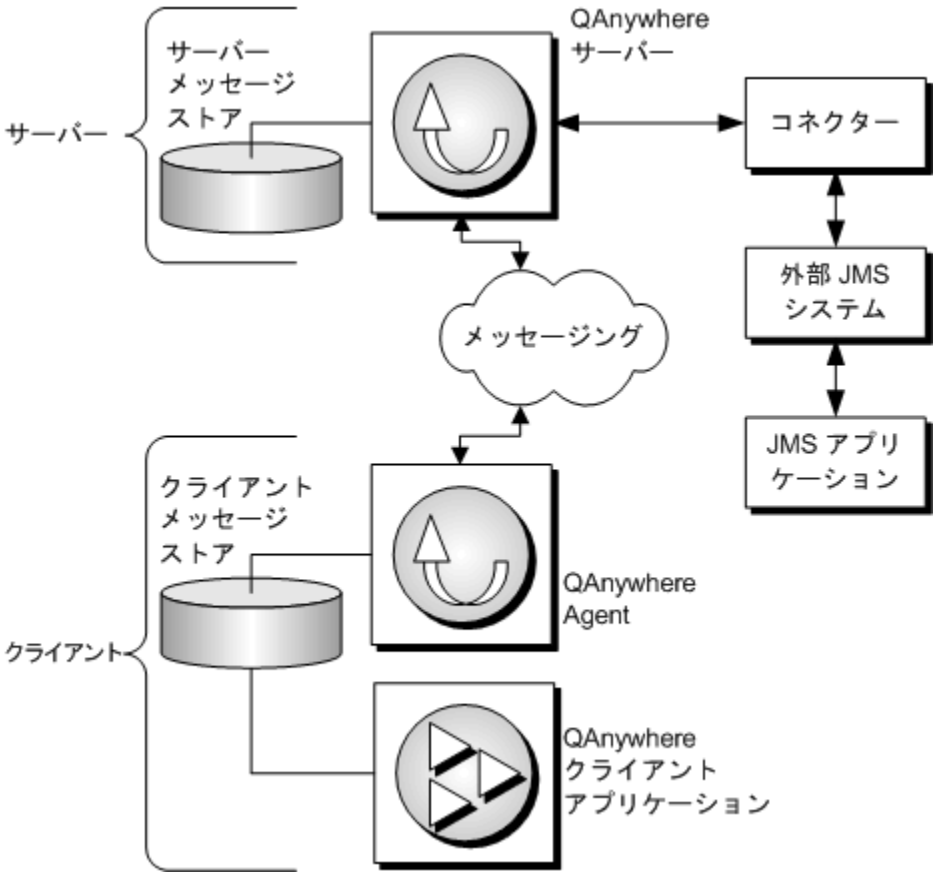
外部メッセージングシステムとのメッセージングのシナリオ

コネクター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 インターフェイスを備えたシステムとのメッセージ交換も可能になります。JMS とは、Java アプリケーションにメッセージング機能を追加する役割を果たす、Java Message Service API のことです。

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

外部メッセージングシステムを含むメッセージングアーキテクチャーは、「簡単なメッセージングシナリオ」5 ページで説明したアーキテクチャーを拡張したものです。

次の図は、このアーキテクチャーを示しています。



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

- **QAnywhere JMS コネクター**　JMS コネクターは、QAnywhere と外部メッセージングシステムの間のインターフェイスを提供します。

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

参照

- 「コネクター」135 ページ
- 「チュートリアル：JMS コネクターの使用」148 ページ

QAnywhere メッセージ配信

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

一度送信されたメッセージは、次のいずれかの場合を除き、確実に配信されます。

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

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

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

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

Ultra Light の主な利点には、次のようなものがあります。

- アプリケーションの占有容量が小さく、SQL Anywhere のフルインストールが不要。
- プロセスの占有容量が小さい。QAnywhere Agent では、3 つのプロセス (uleng12、dblsn、qauagent) だけが必要で、SQL Anywhere のように 4 つのプロセス (dbeng12、dbmlsync、dblsn、qaagent) は不要です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

Ultra Light の制限事項

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

- 転送ルールの条件構文で、"ESCAPE" キーワードはサポートされていません。
- プロパティ属性のサポートが制限されています。プロパティ属性の機能は、事前に定義された ias_Network プロパティでのみ使用可能です。

推奨事項

SQL Anywhere がまだインストールされていない場合はいつも、SQL Anywhere の代わりに Ultra Light を使用する必要があります。SQL Anywhere は、すでに実装されている SQL Anywhere データ同期ソリューションと並行してメッセージング機能を追加したいという状況のときに使用可能です。しかし、純粋なメッセージング環境では、すべて Ultra Light を推奨します。

参照

- 「カスタムのクライアントメッセージストアプロパティ」762 ページ

QAnywhere 12 プラグイン

Sybase Central QAnywhere 12 プラグインを使用して、QAnywhere アプリケーションの作成や管理が行えます。このプラグインでは、次の作業を実行できます。

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

♦ QAnywhere 12 プラグインの起動

1. Sybase Central を起動します。

   [スタート] » **[プログラム]** » **[SQL Anywhere 12]** » **[管理ツール]** » **[Sybase Central]** をクリックします。

2. **[接続]** » **[QAnywhere 12 に接続]** をクリックします。

3. **[ODBC データソース名]** または **[ODBC データソースファイル]** を指定し、必要に応じて **[ユーザー ID]** と **[パスワード]** を指定します。

4. **[接続]** をクリックします。

QAnywhere のクイックスタート

♦ QAnywhere メッセージングの設定と起動

1. ローカルのアプリケーション間メッセージストアを設定します。

   「ローカルメッセージストアの設定」19 ページを参照してください。

   または

   サーバー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。

   「サーバーメッセージストアの設定」24 ページを参照してください。

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

   「有効な Mobile Link での QAnywhere の起動」31 ページを参照してください。

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

   「クライアントメッセージストアの設定」25 ページを参照してください。

4. クライアントごとに、メッセージングアプリケーションを作成します。

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

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

   「コネクター」135 ページを参照してください。

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

   「QAnywhere Agent の起動」47 ページを参照してください。

モバイル Web サービスの設定の詳細については、「モバイル Web サービス」95 ページを参照してください。

クイックスタートのためのその他の資料

- 「チュートリアル：TestMessage の解説」177 ページ
- 「チュートリアル：JMS コネクターの使用」148 ページ
- *%SQLANYSAMP12%¥QAnywhere* にサンプルアプリケーションがあります。
- 質問は、次の SQL Anywhere フォーラムに投稿できます。http://sqlanywhere-forum.sybase.com

QAnywhere メッセージ

QAnywhere メッセージは、次の要素から構成されています。

- ヘッダー
- プロパティ
- 内容

メッセージプロパティは、転送ルールや削除ルールで参照したり、アプリケーションで参照したりできます。

次の項ではメッセージヘッダーとプロパティについて説明します。また、ヘッダーとプロパティを QAnywhere メッセージに設定する方法についても説明します。

注意

- メッセージヘッダー、メッセージプロパティ、メッセージの内容を、メッセージの送信後に変更することはできません。

- メッセージヘッダー、メッセージプロパティ、メッセージの内容は、メッセージの受信後に読み込むことができます。QAnywhere SQL API を使用している場合は、コミットまたはロールバックが発生すると、メッセージヘッダー、メッセージプロパティ、メッセージの内容が読み込み不可能になります。

- 内容は、すべての API で確認またはコミットが行われると読み込み不可能になります。

メッセージヘッダー

QAnywhere のすべてのメッセージで、同じヘッダーフィールドセットがサポートされます。ヘッダーフィールドには、メッセージを識別してルートを指定するために、クライアントとプロバイダーの両方で使用される値が設定されます。メッセージヘッダーの使い方は、使用するクライアントアプリケーションの種類によって異なります。

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

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

参照

- 「メッセージヘッダー」695 ページ

メッセージプロパティ

各メッセージには、アプリケーションで定義されたプロパティ値をサポートする組み込みの機能があります。これらのメッセージプロパティを使用すると、アプリケーションで定義されたメッセージフィルタリングを実行できます。

メッセージプロパティは名前と値の組み合わせです。オプションでメッセージに挿入して、構造を示すことができます。たとえば、.NET API では、定数 MessageProperties.ORIGINATOR で識別される、事前に定義されたメッセージプロパティ ias_Originator は、メッセージを送信したメッセージストアの ID を指定します。メッセージプロパティを転送ルールで使用して、メッセージを転送することが妥当であるかどうかを判別することができます。

メッセージプロパティには、次の 2 種類があります。

- **事前に定義されたメッセージプロパティ**　このメッセージプロパティには、必ず ias_ または IAS_ のプレフィクスが付けられています。
- **カスタムのメッセージプロパティ**　ユーザーが定義したメッセージプロパティです。このプロパティに ias_ または IAS_ のプレフィクスを付けることはできません。

事前定義またはカスタムのプロパティのどちらの場合も、get メソッドと set メソッドを使用してメッセージストアプロパティにアクセスし、プロパティの名前を最初のパラメーターとして渡します。「メッセージプロパティの管理」701 ページを参照してください。

事前に定義されたメッセージプロパティ

利便性のためにいくつかのメッセージプロパティは事前に定義されています。事前に定義されたプロパティは読み込み可能ですが、設定はできません。事前に定義されたメッセージプロパティは次のとおりです。

- ias_Adapters
- ias_DeliveryCount
- ias_MessageType
- ias_RASNames
- ias_NetworkStatus
- ias_Originator
- ias_Status
- ias_StatusTime

メッセージプロパティの詳細については、「メッセージプロパティ」698 ページを参照してください。

参照

- 「事前に定義されたメッセージプロパティ」698 ページ
- 「カスタムのメッセージプロパティ」700 ページ

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

送信先の概要

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

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

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

最初の例は、JMS アプリケーション宛てにメッセージを送信する場合を示しています。この場合、識別子は Mobile Link サーバーで実行されている JMS コネクターの ID になります。

2 つ目の例は、モバイルアプリケーション宛てにメッセージを送信する場合を示しています。この場合、識別子は QAnywhere メッセージストアのメッセージストア ID になります。

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

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

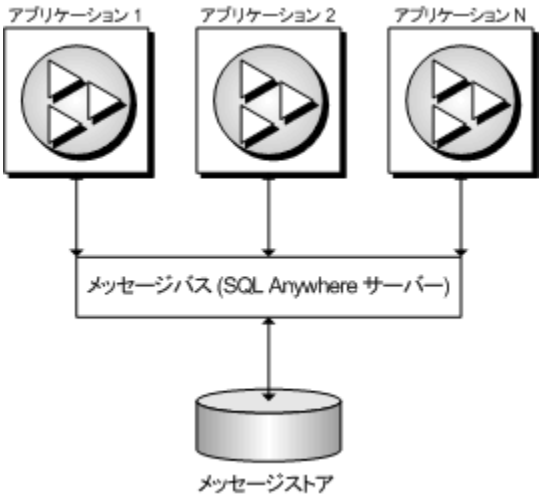
注意
QAnywhere の送信先は、必ず英字で指定する必要があります。

参照

- 「JMS コネクター」135 ページ
- 「クライアントメッセージストアの設定」25 ページ
- 「-id qaagent オプション」719 ページ
- 「送信先エイリアス」133 ページ
- 「QAnywhere メッセージの送信」63 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」49 ページ

QAnywhere メッセージストア

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



参照

- 「サーバーメッセージストア」23 ページ
- 「クライアントメッセージストア」25 ページ

ローカルメッセージストア

ローカルメッセージストア設定では、ローカルメッセージバスとして動作する SQL Anywhere データベースにアプリケーションが接続され、メッセージは、アプリケーションからデータベースへの接続を介してアプリケーション間で転送されます。

ローカルメッセージストアの設定

ローカルメッセージストアには、QAnywhere のインストールが必要です。QAnywhere は、SQL Anywhere インストールプログラムの同期とメッセージング機能にあります。SQL Anywhere データベースでメッセージングを有効にするには、QAnywhere スキーマを SQL Anywhere データベースにインストールしてローカルメッセージストアとして使用できるようにする必要があります。これを行うには、SQL Anywhere 用 QAnywhere Agent の **-sil** オプションを使用します。**-sil** オプションは、データベースをローカルメッセージストアとして初期化するよう Agent に指示します。エージェントはローカルメッセージストアの初期化にのみ使用され、他の操作には必要ありません。データベースで作成されるすべての QAnywhere オブジェクトは、

ml_qa_message_group の所有者に属します。SQL Anywhere データベースがローカルメッセージストアとして初期化されると、アプリケーションで QAnywhere クライアント API を使用してメッセージを交換できます。

♦ ローカルメッセージストアの作成 (.NET の例)

1. 新しい SQL Anywhere データベースを作成します。既存の SQL Anywhere データベースを使用する場合は、この手順を省略できます。

```
dbinit localmsgstore.db
```

2. QAnywhere スキーマを SQL Anywhere データベースにインストールして、ローカルメッセージストアとして使用できるようにします。

```
qaagent -sil -c "dbf=localmsgstore.db;uid=dba;pwd=sql"
```

3. メッセージストアにメッセージを入れる送信側アプリケーションを作成します。QAnywhere クライアント API の .NET バージョンの使用の詳細については、「クライアント用 QAnywhere .NET API リファレンス」189 ページを参照してください。

```
using System;
using System.IO;
using iAnywhere.QAnywhere.Client;
namespace sender
{
class sender
  {
    public static void Main() {
      try {
        // QAnywhere initialization
        QAManager mgr = QAManagerFactory.Instance.CreateQAManager();
        // Be sure to set the DATABASE_TYPE property
        mgr.SetProperty( "DATABASE_TYPE", "sqlanywhere" );
        mgr.SetProperty( "CONNECT_PARAMS", "dbf=localmsgstore.db;uid=dba;pwd=sql" );
        mgr.Open( AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT );
        mgr.Start();
        // Create a text message
        QATextMessage msg = mgr.CreateTextMessage();
        msg.Text = "Sample text";
        // Queue the message
        mgr.PutMessage( "dbqueue", msg );
        // QAnywhere finalization
        mgr.Stop();
        mgr.Close();
        } catch( Exception exc ) {
        Console.WriteLine( exc.Message );
      }
    }
  }
}
```

4. 次のコマンドラインを使用して、プログラムをコンパイルします。コンピューターに Visual Studio がインストールされている必要があります。

```
csc /reference:"%SQLANY12%¥Assembly¥v2¥iAnywhere.QAnywhere.Client.dll" sender.cs
```

5. メッセージストアからメッセージを取り出す受信側アプリケーションを作成します。QAnywhere クライアント API の .NET バージョンの使用の詳細については、「クライアント用 QAnywhere .NET API リファレンス」189 ページを参照してください。

```
using System;
using System.IO;
using iAnywhere.QAnywhere.Client;
namespace receiver
{
class receiver
{
public static void Main() {
try {
// QAnywhere initialization
QAManager mgr = QAManagerFactory.Instance.CreateQAManager();
// Be sure to set the DATABASE_TYPE property
mgr.SetProperty( "DATABASE_TYPE", "sqlanywhere" );
mgr.SetProperty( "CONNECT_PARAMS", "dbf=localmsgstore.db;uid=dba;pwd=sql" );
mgr.Open( AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT );
mgr.Start();
// Get the message
QATextMessage msg = (QATextMessage)mgr.GetMessage( "dbqueue" );
// Display the text
Console.WriteLine( msg.Text );
// QAnywhere finalization
mgr.Stop();
mgr.Close();
} catch( Exception exc ) {
Console.WriteLine( exc.Message );
}
}
}
}
```

6. 次のコマンドラインを使用して、プログラムをコンパイルします。コンピューターに Visual Studio がインストールされている必要があります。

   ```
   csc /reference:"%SQLANY12%¥Assembly¥v2¥iAnywhere.QAnywhere.Client.dll" receiver.cs
   ```

7. SQL Anywhere データベースを起動します。

   ```
   dbsrv12 localmsgstore.db
   ```

8. 送信側アプリケーションを実行します。

   ```
   sender
   ```

9. 受信側アプリケーションを実行します。

   ```
   receiver
   ```

   文字列 *"Sample Text"* が表示されます。

♦ ローカルメッセージストアの作成 (Java の例)

1. 新しい SQL Anywhere データベースを作成します。既存の SQL Anywhere データベースを使用する場合は、この手順を省略できます。

   ```
   dbinit localmsgstore.db
   ```

2. QAnywhere スキーマを SQL Anywhere データベースにインストールして、メッセージバスとして使用できるようにします。

   ```
   qaagent -sil -c "dbf=localmsgstore.db;uid=dba;pwd=sql"
   ```

3. メッセージストアにメッセージを入れる送信側アプリケーションを作成します。
   QAnywhere クライアント API の .NET バージョンの使用の詳細については、「クライアント用 QAnywhere Java API リファレンス」495 ページを参照してください。

```
import java.util.*;
import ianywhere.qanywhere.client.*;
public class sender
{
public static void main( String [] args ) {
try {
// QAnywhere initialization
QAManager mgr = QAManagerFactory.getInstance().createQAManager();
// Be sure to specify the DATABASE_TYPE
mgr.setProperty( "DATABASE_TYPE", "sqlanywhere" );
mgr.setProperty( "CONNECT_PARAMS", "dbf=localmsgstore.db;uid=dba;pwd=sql" );
mgr.open( AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT );
mgr.start();
// Create a text message
QATextMessage msg = mgr.createTextMessage();
msg.setText( "Sample text" );
// Queue the message
mgr.putMessage( "dbqueue", msg );
// QAnywhere finalization
mgr.stop();
mgr.close();
} catch( Exception exc ) {
System.out.println( exc.getMessage() );
}
}
}
```

4. 次のコマンドラインを使用して、プログラムをコンパイルします。コンピューターに Java JDK がインストールされている必要があります。

```
javac  -cp "%SQLANY12%¥java¥qaclient.jar" sender.java
```

5. メッセージストアからメッセージを取り出す受信側アプリケーションを作成します。
   QAnywhere クライアント API の Java バージョンの使用の詳細については、「クライアント用 QAnywhere Java API リファレンス」495 ページを参照してください。

```
import java.util.*;
import ianywhere.qanywhere.client.*;
public class receiver
{
public static void main( Strings [] args ) {
try {
// QAnywhere initialization
QAManager mgr = QAManagerFactory.getInstance().createQAManager();
// Be sure to set the DATABASE_TYPE property.
mgr.setProperty( "DATABASE_TYPE", "sqlanywhere" );
mgr.setProperty( "CONNECT_PARAMS", "dbf=localmsgstore.db;uid=dba;pwd=sql" );
mgr.open( AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT );
mgr.start();
// Get the message
QATextMessage msg = (QATextMessage)mgr.getMessage( "dbqueue" );
// Display the text
System.out.println( msg.getText() );
// QAnywhere finalization
mgr.stop();
mgr.close();
} catch( Exception exc ) {
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
System.out.println( exc.getMessage() );
}
}
}
```

6. 次のコマンドラインを使用して、プログラムをコンパイルします。コンピューターに Java JDK がインストールされている必要があります。

```
javac -cp "%SQLANY12%¥java¥qaclient.jar" receiver.java
```

7. SQL Anywhere データベースを起動します。

```
dbsrv12 localmsgstore.db
```

8. 送信側アプリケーションを実行します。

```
java -cp ".¥;%SQLANY12%¥java¥qaclient.jar;%SQLANY12%¥java¥jodbc.jar" sender
```

9. 受信側アプリケーションを実行します。

```
java -cp ".¥;%SQLANY12%¥java¥qaclient.jar;%SQLANY12%¥java¥jodbc.jar" receiver
```

文字列 *"Sample Text"* が表示されます。

メッセージストアのクリーニング

デフォルトでは、最終ステータスに到達したメッセージはメッセージストアに残ります。しばらくの間使用するとメッセージが累積し、データベースが増大します。メッセージバスとして使用するデータベースを初期化した時点で、**ml_qa_clearreceivedmessages** というストアドプロシージャーが作成されています。最終ステータスに到達したメッセージは、このストアドプロシージャーによってメッセージストアから削除されます。データベース内のメッセージの増大を管理するには、トリガーされたときに **ml_qa_clearreceivedmesssages** ストアドプロシージャーを起動するイベントを作成すると便利です。たとえば、次のデータベースイベントは、最終ステータスに到達したメッセージを毎日夜中の 12 時に削除します。

```
CREATE EVENT message_cleanup
SCHEDULE
START TIME '12:00AM' EVERY 24 HOURS
HANDLER begin call ml_qa_message_group.ml_qa_clearreceivedmessages() end
```

メッセージの表示

QAnywhere プラグインを使用してクライアントストアに接続することによって、Sybase Central を使用して SQL Anywhere データベースにあるメッセージを表示します。

サーバーメッセージストア

サーバーメッセージストアはサーバー上のリレーショナルデータベースです。このデータベースは、メッセージを、クライアントメッセージストア、Web サービス、または JMS システムに転送されるまで一時的に格納します。メッセージは、サーバー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。

サーバー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、サーバーメッセージストアとして使用できるのは、Mobile Link がサポートしている RDBMS (MySQL を除く)

です。新しくデータベースを作成して使用することも、既存のデータベースを使用することもできます。

サーバーメッセージストアの設定

サーバーメッセージストアを設定すると、アーカイブメッセージストアが作成されます。アーカイブメッセージストアはサーバーメッセージストアと共存するテーブルのセットで、削除待機中のすべてのメッセージがここに格納されます。定期的に実行されるシステムプロセスでは、サーバーメッセージストアにあるメッセージのうち最終ステータスに到達したものすべてを削除し、アーカイブメッセージストアに挿入することで、メッセージストア間でのメッセージ転送を行います。メッセージは、サーバー削除ルールによって削除されるまで、アーカイブメッセージストアに残ります。アーカイブメッセージストアを使用することにより、同期中にフィルターする必要のあるメッセージの量が最小限に抑えられるため、サーバーメッセージストアのパフォーマンスが向上します。「アーカイブメッセージストア要求」155 ページを参照してください。

サーバーメッセージストアとして使用できるようにデータベースを設定するには、設定スクリプトを実行します。大文字と小文字を区別しない比較や文字列操作をするように、統合データベースを設定する必要があります。**同期モデル作成ウィザード**を使用して統合データベースを作成した場合は、設定は自動的に行われます。

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

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

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

データベースのアップグレードの詳細については、「SQL Anywhere 12 へのアップグレード」『SQL Anywhere 12 変更点とアップグレード』を参照してください。

> **注意**
> サーバーメッセージストアを作成および管理するには、Sybase Central を使用する方法が最も簡単です。QAnywhere 12 プラグインのタスクのウィンドウ枠から、**[サーバーメッセージストア]** をクリックします。

例

qanytest.db という名前の SQL Anywhere データベースを作成するには、次のコマンドを実行します。

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

データベース上で Mobile Link 設定スクリプトを実行します。

```
%SQLANY12%¥MobiLink¥setup¥syncsa.sql
```

このデータベースは、サーバーメッセージストアとして使用できます。

サーバー管理要求

QAnywhere クライアントアプリケーションは、「サーバー管理要求」という特殊なメッセージをサーバーに送信できます。サーバー管理要求は、XML フォーマットの内容を含み、QAnywhere システムキューに送信されます。特殊な認証文字列が必要です。サーバー管理要求は、アクティブなクライアントのクエリやメッセージストアプロパティのクエリ、メッセージのクエリなど、さまざまな機能を実行できます。

サーバー管理要求で実行できる機能や、その使用方法については、「サーバー管理要求の記述」154 ページを参照してください。

クライアントメッセージストア

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

クライアントのメッセージストアは、QAnywhere アプリケーション専用にする必要があります。QAnywhere API を使用する QAnywhere アプリケーション以外のアプリケーションから QAnywhere メッセージストアデータベースにアクセスしないでください。ただし、データベースサーバー内で別のデータベースを実行することはできます。これは、QAnywhere のクライアントメッセージストアと Mobile Link 同期クライアントを同じデバイスで実行する場合に便利です。

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

「セキュリティ保護されたクライアントメッセージストアの作成」121 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の作成

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。

> **注意**
> クライアントメッセージストアを作成および管理するには、Sybase Central を使用する方法が最も簡単です。QAnywhere 12 プラグインのタスクのウィンドウ枠から、**[クライアントメッセージストア]** をクリックします。

1. 新しい SQL Anywhere または Ultra Light データベースを作成します。

   「SQL Anywhere データベースの作成」『SQL Anywhere サーバー データベース管理』を参照してください。

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

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

     「-c qaagent オプション」715 ページを参照してください。

   - **-si オプション**　データベースを初期化します。-si オプションを指定すると、デフォルトのデータベースユーザーとパスワードが作成されます。エージェントは、データベースを初期化した後、停止します。

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

     「-si qaagent オプション」731 ページを参照してください。

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

     「クライアントメッセージストア ID の作成」26 ページと 「-id qaagent オプション」719 ページを参照してください。

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

3. -mu オプションを指定してユーザー名を作成した場合は、その名前をサーバーメッセージストアに追加する必要があります。-zu+ オプションを指定して mlsrv12 を実行すると、この処理が自動的に行われるようになります。この処理は、それ以外の方法でも実行できます。

   「QAnywhere クライアントユーザー名の登録」32 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」121 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 を指定しなかった場合には、-si オプションを指定して qaagent を実行した後、最初に qaagent を実行したときに、デバイス名がクライアントメッセージストア ID として割り当てられます。ID は QAnywhere Agent ウィンドウに表示されます。

ID は手動で指定するほうが便利です。ID は次の方法で指定できます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

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

「QAnywhere Agent ユーティリティのリファレンス」712 ページを参照してください。

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

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

トランザクションログ

トランザクションログは使用することをおすすめします。SQL Anywhere データベースはトランザクションログを使用したときに実行効率が最もよく、またトランザクションログはデータベース障害が発生したときにデータベースを保護する手段になるからです。しかし、トランザクションログは非常に大きくなる可能性があります。このため、QAnywhere Agent では、トランザクションログの内容がチェックポイントで削除されるように、デフォルトで dbsrv12 -m オプションが設定されます。この設定をおすすめします。qaagent -c オプションで StartLine パラメーターを指定した場合は、-m を指定してください。

クライアントメッセージストアの保護

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

クライアントメッセージストアの作成例

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

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

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

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

「初期化ユーティリティ (dbinit)」『SQL Anywhere サーバー データベース管理』と「QAnywhere Agent ユーティリティのリファレンス」712 ページを参照してください。

SQL Anywhere と Ultra Light クライアントの相違点

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

Ultra Light の主な利点には、次のようなものがあります。

- アプリケーションの占有容量が小さく、SQL Anywhere のフルインストールが不要。
- プロセスの占有容量が小さい。QAnywhere Agent では、3 つのプロセス (uleng12、dblsn、qauagent) だけが必要で、SQL Anywhere のように 4 つのプロセス (dbeng12、dbmlsync、dblsn、qaagent) は不要です。

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

- 転送ルールの条件構文で、"ESCAPE" キーワードはサポートされていません。
- プロパティ属性のサポートが制限されています。プロパティ属性の機能は、事前に定義された ias_Network プロパティでのみ使用可能です。 「カスタムのクライアントメッセージストアプロパティ」762 ページを参照してください。

SQL Anywhere がまだインストールされていない場合は、SQL Anywhere の代わりに Ultra Light を常に使用する必要があります。SQL Anywhere は、すでに実装されている SQL Anywhere データ同期ソリューションと並行してメッセージング機能を追加したいという状況のときに使用可能です。しかし、純粋なメッセージング環境では、すべて Ultra Light を推奨します。

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

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

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

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

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

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

参照

- 「カスタムのクライアントメッセージストアプロパティ」762 ページ
- 「qauagent ユーティリティ」738 ページ

クライアントメッセージストアプロパティ

クライアントメッセージストアプロパティには、次の 2 種類があります。

- **事前に定義されたメッセージストアプロパティ**　このメッセージストアプロパティには、必ず ias_ または IAS_ のプレフィクスが付けられています。
- **カスタムのメッセージストアプロパティ**　ユーザーが定義したメッセージストアプロパティです。このプロパティに ias_ または IAS_ のプレフィクスを付けることはできません。

適切なクラスで定義された get メソッドと set メソッドを使用してクライアントメッセージストアプロパティにアクセスし、事前定義またはカスタムのプロパティの名前を最初のパラメーターとして渡すことができます。

「クライアントメッセージストアプロパティの管理」131 ページを参照してください。

メッセージストアプロパティは、転送ルール、削除ルール、メッセージセレクターでも使用できます。次の項を参照してください。

- 「QAnywhere 転送ルールと削除ルール」774 ページ

事前に定義されたクライアントメッセージストアプロパティ

利便性のために、いくつかのクライアントメッセージストアプロパティは事前に定義されています。事前に定義されたメッセージストアプロパティは次のとおりです。

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

クライアントメッセージストアの事前に定義されたプロパティの詳細については、次の項を参照してください。

- 「事前に定義されたクライアントメッセージストアプロパティ」761 ページ

カスタムのクライアントメッセージストアプロパティ

QAnywhere では、QAnywhere C++、Java、SQL、または .NET API を使用して、ユーザー独自のクライアントメッセージストアプロパティを定義できます。これらのプロパティは、同じメッセージストアに接続されたアプリケーション間で共有されます。また、サーバーメッセージストアと同期されるため、このクライアント用のサーバー側の転送ルールでも使用できます。

クライアントメッセージストアプロパティ名では大文字と小文字を区別しません。文字、数字、アンダースコアを使用できますが、先頭は文字でなければなりません。次の名前は予約語なので、メッセージストアプロパティ名として使用することはできません。

- NULL
- TRUE
- FALSE
- NOT
- AND
- OR
- BETWEEN
- LIKE
- IN
- IS
- ESCAPE (SQL Anywhere メッセージストアのみ)
- **ias_** で始まるすべての名前

クライアントメッセージストアプロパティには、ユーザーが属性を定義できます。属性を定義するには、プロパティ名の後にドットを付け、その後に属性名を追加します。この機能の主な目的は、ユーザーのネットワーク情報を転送ルールで使用できるようにすることです。

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

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

- 「カスタムのクライアントメッセージストアプロパティ属性の使用」763 ページ
- 「事前に定義されたクライアントメッセージストアプロパティ」761 ページ

QAnywhere メッセージングの設定

以降の項では、クライアントとサーバーでの QAnywhere メッセージングの設定方法、Push 通知の使用方法、フェールオーバーメカニズムの設定方法について説明します。

サーバーサイドコンポーネントの設定

♦ QAnywhere サーバーサイドコンポーネントの設定の概要

> **注意**
> サーバーメッセージストアを作成および管理するには、Sybase Central を使用する方法が最も簡単です。QAnywhere 12 プラグインのタスクのウィンドウ枠から、**[サーバーメッセージストア]** を選択します。

1. サーバー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。

   「サーバーメッセージストアの設定」24 ページを参照してください。

2. -m オプションを指定して mlsrv12 を起動し、サーバーメッセージストアへの接続を開始します。

   「有効な Mobile Link での QAnywhere の起動」31 ページを参照してください。

3. クライアントユーザー名をサーバーメッセージストアに追加します。

   「QAnywhere クライアントユーザー名の登録」32 ページを参照してください。

有効な Mobile Link での QAnywhere の起動

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

QAnywhere サーバーを実行するには、次のオプションを指定して Mobile Link サーバー (mlsrv12) を起動します。

- **-c *connection-string*** サーバーメッセージストアに接続するための接続文字列を指定します。これは mlsrv12 に必須のオプションです。

  「-c mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。

- **-m** QAnywhere のメッセージング機能を有効にします。

上記以外にも、Mobile Link サーバー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。

> **注意**
> JMS メッセージングシステムと統合する場合は、Mobile Link サーバー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。

例

サンプルのサーバーメッセージストア (*%SQLANYSAMP12%¥QAnywhere¥server¥qanyserv.db*) を使用して QAnywhere のメッセージングを開始するには、次のコマンドを実行します。

```
mlsrv12 -m -c "dsn=QAnywhere 12 Demo"
```

QAnywhere のサンプルのサーバーメッセージストアでは、**[QAnywhere 12 Demo]** という ODBC データソースを使用します。

参照

- 「統合データベースの設定」『Mobile Link サーバー管理』
- 「-m mlsrv12 オプション」『Mobile Link サーバー管理』
- 「mlsrv12 構文」『Mobile Link サーバー管理』
- 「JMS 統合用 Mobile Link サーバーの起動」138 ページ

QAnywhere クライアントユーザー名の登録

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

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

- mluser ユーティリティを使用する。
- Sybase Central の Mobile Link 12 プラグインを使用する。
- mlsrv12 で -zu+ オプションを指定する。この場合、最初に同期するときに、統合データベースに追加されていない既存の Mobile Link ユーザーが追加されます。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。

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

クライアントメッセージストア ID の詳細については、「-id qaagent オプション」719 ページを参照してください。

参照

- 「Mobile Link ユーザー認証ユーティリティ (mluser)」『Mobile Link サーバー管理』
- 「-zu mlsrv12 オプション」『Mobile Link サーバー管理』

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

QAnywhere サーバー上でのクライアントのプロパティの設定

QAnywhere 12 プラグインを使用して、QAnywhere サーバー上で QAnywhere クライアントのプロパティを設定できます。この操作を行う場合、サーバーにクライアントを追加する必要があります。クライアントに対して初めて同期を実行すると、プロパティがダウンロードされます。

♦ Sybase Central を使用したクライアントユーザー名の追加

1. Sybase Central を起動します。
   a. **[スタート]** » **[プログラム]** » **[SQL Anywhere 12]** » **[管理ツール]** » **[Sybase Central]** をクリックします。
   b. **[接続]** » **[QAnywhere 12 に接続]** をクリックします。
   c. **[ODBC データソース名]** または **[ODBC データソースファイル]** を指定し、必要に応じて **[ユーザー ID]** と **[パスワード]** を指定します。**[OK]** をクリックします。
2. **[ファイル]** » **[新規]** » **[クライアント]** をクリックします。
3. クライアント名を入力します。
4. **[OK]** をクリックします。

参照

- 「QAnywhere クライアントユーザー名の登録」32 ページ

QAnywhere サーバーのログ

QAnywhere サーバーは、メッセージングが有効になっている Mobile Link サーバーです。QAnywhere サーバーのログファイルは、Mobile Link のログファイルです。

Mobile Link サーバーログファイルビューアー

QAnywhere サーバーのログファイルを表示するには、Sybase Central を開き、**[ツール]** » **[QAnywhere 12]** » **[Mobile Link サーバーログファイルビューアー]** をクリックします。表示するログファイルの選択が要求されます。

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

ログビューアーでは、表示する情報をフィルターできます。また、ログ内の情報に基づく統計値も表示できます。

参照

- 「Mobile Link サーバーの動作のロギング」『Mobile Link サーバー管理』

Relay Server の使用

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

次に例を示します。

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

参照

- 「Relay Server の概要」『Relay Server』
- 「Mobile Link サーバーのサーバーファームでの実行」『Mobile Link サーバー管理』

クライアントサイドコンポーネントの設定

♦ クライアントサイドコンポーネントの設定の概要

> **注意**
> クライアントメッセージストアを作成および管理するには、Sybase Central を使用する方法が最も簡単です。QAnywhere 12 プラグインのタスクのウィンドウ枠から、**[クライアントメッセージストア]** をクリックします。

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

   「クライアントメッセージストアの設定」25 ページを参照してください。

2. クライアントアプリケーションを作成します。

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

3. QAnywhere Agent を起動します。

   「QAnywhere Agent の起動」47 ページを参照してください。

Push 通知

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、サーバーメッセージストアから QAnywhere クライアントに配信される特殊なメッセージです。Push 通知はデフォルトでオンになっていますが、必須ではありません。Push 通知が行われるとき、QAnywhere アーキテクチャーでは次の追加コンポーネントが動作します。

●サーバー側では、QAnywhere Notifier が Push 通知を送信します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

●クライアント側では、QAnywhere Listener が Push 通知を受け取り、QAnywhere Agent に渡します。

●クライアント側では、Push 通知ごとに通知がシステムキューに送信されます。

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

Push 通知の手動処理の詳細については、「Push 通知の通知」63 ページを参照してください。

QAnywhere Agent ポリシーの詳細については、「クライアントにメッセージを転送するタイミングの決定」49 ページを参照してください。

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

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

参照

- 「Push 通知によるメッセージングのシナリオ」7 ページ
- 「Push 通知の通知」63 ページ
- 「-push qaagent オプション」729 ページ

Push 通知の設定

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

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

Push 通知を無効にし、Notifier または Listener が使用されないようにすることもできます。「-push qaagent オプション」729 ページを参照してください。

Push 通知に対するクライアントの応答については、「クライアントにメッセージを転送するタイミングの決定」49 ページを参照してください。

参照

- 「QAnywhere Notifier の設定」36 ページ
- 「Mobile Link Listener の設定」37 ページ
- 「QAnywhere ゲートウェイの設定」38 ページ

QAnywhere Notifier の設定

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

QAnyNotifier_client では、「サーバー起動同期の Mobile Link サーバー設定」『Mobile Link サーバー起動同期』に示すデフォルト値が使用されます。ただし、次の例外があります。

- gui プロパティは off に設定されるので、Notifier が実行されるコンピューターで Notifier のウィンドウが表示されない。
- enable プロパティは no に設定されるので、Notifier を起動するには -m オプションを指定して mlsrv12 を実行する必要がある。
- poll_every プロパティは 5 に設定されるので、Notifier では 5 秒ごとにポーリングが実行され、Push 通知を送信する必要があるかどうかが確認される。

Notifier の次のプロパティを変更できます。

- poll_every プロパティ
- request_cursor プロパティの再送間隔 (resend interval)
- request_cursor プロパティの存続期間 (time to live)

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

poll_every プロパティ

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

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

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

再送間隔と存続期間

QAnywhere Notifier には、デフォルトの request_cursor が含まれます。request_cursor は、どの情報を Push 要求で送信するか、誰が、いつ、どこで情報を受信するかを決定します。再送間隔と存続期間以外のデフォルト値は変更しないでください。再送間隔は、未受信の Push 通知を再送す

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

る間隔を指定します。この間隔はデフォルトでは 5 分です。存続期間は、未受信の Push 通知の再送を続ける時間を指定します。この時間はデフォルトでは 3 時間です。通常、これらのデフォルト値が最適です。QAnyNotifier_client のデフォルトの request_cursor は次のとおりです。

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

request_cursor のカラムの詳細については、「Push 要求の要件」『Mobile Link サーバー起動同期』を参照してください。

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

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

詳細については、「request_cursor イベント」『Mobile Link サーバー起動同期』を参照してください。

参照

- 「Notifier イベントとプロパテの設定」『Mobile Link サーバー起動同期』
- 「サーバー起動同期の Mobile Link サーバー設定」『Mobile Link サーバー起動同期』
- 「Notifier」『Mobile Link サーバー起動同期』
- 「ml_add_property システムプロシージャー」『Mobile Link サーバー管理』
- 「Push 要求」『Mobile Link サーバー起動同期』

Mobile Link Listener の設定

Mobile Link Listener は、クライアントメッセージストアと同じデバイスで実行されます。Mobile Link Listener は、サーバーから Push 通知を受信して、QAnywhere Agent に渡します。

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

たとえば、カスタムオプションを指定して Mobile Link Listener を手動で起動する必要がある場合は、次のコマンドを使用できます。

```
dblsn.exe -u ias_mystore_lsn -e mystore -t+ mystore
        ... other options ...
```

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

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

5001 は、dblsn の UDP リスナーライブラリの受信側ポートです。

参照

- 「Listener」『Mobile Link サーバー起動同期』
- 「Windows デバイス用の Mobile Link Listener ユーティリティ (dblsn)」『Mobile Link サーバー起動同期』
- 「QAnywhere ゲートウェイの設定」38 ページ

QAnywhere ゲートウェイの設定

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

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

SMS ゲートウェイを使用するには、新しいオプションを指定して Mobile Link Listener を起動する必要があります。 「Mobile Link Listener の設定」37 ページを参照してください。

UDP ゲートウェイを使用するには、qaagent の -push disconnected オプションを設定する必要があります。 「-push qaagent オプション」729 ページを参照してください。

参照

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

フェールオーバーメカニズムの設定

QAnywhere アプリケーションは、フェールオーバーメカニズムを使用するように設定できます。フェールオーバーメカニズムを使用すると、Mobile Link サーバーで障害が発生しても代替サー

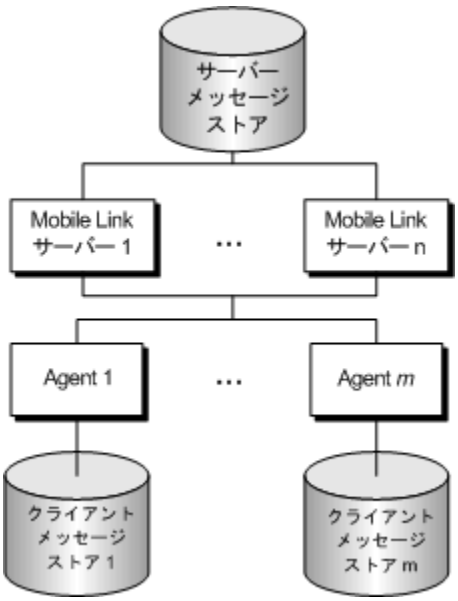
バーを使用できます。フェールオーバーをサポートするには、各 QAnywhere Agent を起動するときに、Mobile Link サーバーのリストを指定する必要があります。リストの先頭に指定された Mobile Link サーバーがプライマリサーバーになります。残りのサーバーは代替サーバーになります。

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ーと 1 台の代替サーバーが存在する 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ーを指定できます。

次の図に、複数の Mobile Link サーバーと複数の QAnywhere Agent を使用したフェールオーバー構成を示します。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 サーバーはいずれも同一のサーバー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ーに関係なく、サーバー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ー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。サーバーメッセージストアが存在するコンピューターが使用できなくなると、メッセージを転送できない。

フェールオーバー Mobile Link サーバーが設定されている場合、デフォルトでは、プライマリサーバーへの接続で障害が発生するとすぐに QAnywhere Agent は代替サーバーへの接続を試行しま

す。このデフォルト動作を変更するには、QAnywhere Agent の -fr オプションを使用します。-fr オプションを指定すると、QAnywhere Agent は代替サーバーに接続する前に、プライマリサーバーへの接続を再試行します。再試行の回数も指定できます。-fd オプションを併せて使用することで、プライマリサーバーへの接続試行の間隔も指定できます。

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

参照

- 「-x qaagent オプション」736 ページ
- 「-fd qaagent オプション」717 ページ
- 「-fr qaagent オプション」718 ページ
- 「QAnywhere Agent の起動」47 ページ

QAnywhere Agent

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

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

メッセージ転送ポリシー

QAnywhere クライアントは、ポリシーに基づいて、メッセージ転送を実行するタイミングを決定します。ポリシーには、次のようなものがあります。

- **「スケジュール設定ポリシー」41 ページ**　スケジュール設定ポリシーは、指定された時間間隔でメッセージの転送を開始します。
- **「自動ポリシー」42 ページ**　自動ポリシーでは、クライアントとの間でメッセージの配信準備が行われるたびにメッセージの転送が開始されます。
- **「オンデマンドポリシー」43 ページ**　オンデマンドポリシーでは、ユーザーがメッセージ転送を制御できます。
- **「カスタムポリシー」43 ページ**　カスタムポリシーでは、転送ルールを使用してメッセージ転送に詳細な制御を追加します。

スケジュール設定ポリシー

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

スケジュールを呼び出すには、**コマンドファイルの [プロパティ]** ウィンドウで **scheduled** をクリックするか、QAnywhere Agent を起動するときに、次のようにキーワードを指定します。

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

interval に間隔を秒単位で指定します。

デフォルト値は 900 秒 (15 分) です。

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

- 前回の時間間隔が経過した後、クライアントメッセージストアに新しいメッセージを受信した。

●前回の時間間隔が経過した後、メッセージステータスが変化した。この現象は、通常、アプリケーションがメッセージの受信を確認したときに起こります。

受信確認の詳細については、次の項を参照してください。

○ AcknowledgementMode 列挙体 [QAnywhere .NET]317 ページ
○ AcknowledgementMode クラス [QAnywhere C++]375 ページ
○ AcknowledgementMode インターフェイス [QAnywhere Java]495 ページ

●前回の時間間隔が経過した後、Push 通知を受信した。

●前回の時間間隔が経過した後、ネットワークステータス変更通知を受信した。

●Push 通知が無効にされた。

時間間隔を無視するには、トリガーの送受信メソッドを呼び出します。このメソッドを使用すると、時間間隔が経過する前にメッセージを転送できます。

次の項を参照してください。

● QAManagerBase.TriggerSendReceive メソッド [QAnywhere .NET]278 ページ
● QAManagerBase.triggerSendReceive メソッド [QAnywhere C++]450 ページ
● QAManagerBase.triggerSendReceive メソッド [QAnywhere Java]577 ページ
● SQL：「ml_qa_triggersendreceive」694 ページ

自動ポリシー

自動ポリシーは、メッセージが送受信されるたびに同期を行い、クライアントメッセージストアとサーバーメッセージストアをできるかぎり最新の状態に維持します。このポリシーは、頻繁にメッセージを送受信するアプリケーションにはおすすめしません。

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

●PutMessage() が呼び出された。

次の項を参照してください。

○ QAManagerBase.PutMessage メソッド [QAnywhere .NET]263 ページ
○ QAManagerBase.putMessage メソッド [QAnywhere C++]441 ページ
○ QAManagerBase.putMessage メソッド [QAnywhere Java]565 ページ
○ SQL：「ml_qa_putmessage」693 ページ

●メッセージステータスが変化した。この現象は、通常、アプリケーションがメッセージの受信を確認したときに起こります。

次の項を参照してください。

- ○ AcknowledgementMode 列挙体 [QAnywhere .NET]317 ページ
- ○ AcknowledgementMode クラス [QAnywhere C++]375 ページ
- ○ AcknowledgementMode インターフェイス [QAnywhere Java]495 ページ
- ○ SQL：SQL API を使用するメッセージングはすべてトランザクション指向です。

●Push 通知を受信した。

「Push 通知」34 ページを参照してください。

●ネットワークステータス変更通知を受信した。

「Push 通知の通知」63 ページを参照してください。

●TriggerSendReceive() が呼び出された。

次の項を参照してください。

- ○ QAManagerBase.TriggerSendReceive メソッド [QAnywhere .NET]278 ページ
- ○ QAManagerBase.triggerSendReceive メソッド [QAnywhere C++]450 ページ
- ○ QAManagerBase.triggerSendReceive メソッド [QAnywhere Java]577 ページ
- ○ SQL：「ml_qa_triggersendreceive」694 ページ

オンデマンドポリシー

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

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

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

次の項を参照してください。

- ● QAManagerBase.TriggerSendReceive メソッド [QAnywhere .NET]278 ページ
- ● QAManagerBase.triggerSendReceive メソッド [QAnywhere C++]450 ページ
- ● QAManagerBase.triggerSendReceive メソッド [QAnywhere Java]577 ページ
- ● SQL：「ml_qa_triggersendreceive」694 ページ

Push 通知とネットワークステータス変更通知の処理の詳細については、「システムキュー」61 ページを参照してください。

カスタムポリシー

カスタムポリシーを使用すると、メッセージ転送のタイミングと、そのメッセージ転送で送信するメッセージを定義できます。

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

```
auto=DATEADD( day, 1, ias_StatusTime ) < ias_CurrentTimestamp
```

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

●メッセージサイズ

●同期の頻度

●帯域幅とネットワークの信頼性

●優先度が高いメッセージ

●データ転送コスト

カスタムポリシーは、転送ルールセットを使って定義します。

各ルールは次の形式で定義します。

schedule = *condition*

condition には条件を指定し、*schedule* にはその条件を評価するタイミングを指定します。 詳細については、「ルールの構文」774 ページを参照してください。

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

●PutMessage() が呼び出された。

次の項を参照してください。

○ QAManagerBase.PutMessage メソッド [QAnywhere .NET]263 ページ
○ QAManagerBase.putMessage メソッド [QAnywhere C++]441 ページ
○ QAManagerBase.putMessage メソッド [QAnywhere Java]565 ページ
○ SQL：「ml_qa_putmessage」693 ページ

●メッセージステータスが変化した。この現象は、通常、アプリケーションがメッセージの受信を確認したときに起こります。

次の項を参照してください。

○ AcknowledgementMode 列挙体 [QAnywhere .NET]317 ページ
○ AcknowledgementMode クラス [QAnywhere C++]375 ページ
○ AcknowledgementMode インターフェイス [QAnywhere Java]495 ページ
○ SQL：SQL API を使用するメッセージングはすべてトランザクション指向です。

●Push 通知を受信した。

「Push 通知」34 ページを参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

●ネットワークステータス変更通知を受信した。

●TriggerSendReceive() が呼び出された。

次の項を参照してください。

○ QAManagerBase.TriggerSendReceive メソッド [QAnywhere .NET]278 ページ
○ QAManagerBase.triggerSendReceive メソッド [QAnywhere C++]450 ページ
○ QAManagerBase.triggerSendReceive メソッド [QAnywhere Java]577 ページ
○ SQL：「ml_qa_triggersendreceive」694 ページ

転送ステータス

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

- **転送済み**　メッセージは送信されました。
- **転送中**　メッセージは送信中です。
- **未転送**　メッセージは送信されていません。
- **転送しない**　メッセージは送信されません。

メッセージのステータス

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

- **保留**　メッセージは送信されましたが、まだ受信されていません。
- **受信中**　メッセージは受信中であるか、または受信されましたがまだ確認されていません。
- **最終**　メッセージは最終ステータスになりました。
- **失効**　有効期限が切れる前にメッセージが受信されませんでした。
- **キャンセル済み**　メッセージはキャンセルされました。
- **受信不可**　メッセージに間違った形式があるか、失敗した試行回数が多すぎて配信されませんでした。
- **受信済み**　メッセージは受信され、確認されました。

転送ルール

メッセージ転送とは、クライアントメッセージストアとサーバーメッセージストアの間でメッセージを移動するアクションを指します。

メッセージ転送は、QAnywhere Agent と Mobile Link サーバーによって次のように処理されます。

- QAnywhere Agent は、QAnywhere クライアントメッセージストアと接続されており、Mobile Link サーバーとの間でメッセージを転送します。

- Mobile Link サーバーは、サーバー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ーメッセージストアの間だけです。メッセージ転送は、QAnywhere Agent が Mobile Link サーバーに接続されているときしか実行されません。

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

ルールには、スケジュール部分と条件部分があります。スケジュールはイベントが実行されるタイミングを定義し、条件はイベントに含めるメッセージを定義します。たとえば、イベントがメッセージ転送である場合は、転送を実行するタイミングをスケジュールで指定し、転送に含めるメッセージを条件で定義します。イベントがメッセージの削除である場合は、削除を行うタイミングをスケジュールで指定し、削除対象のメッセージを条件で指定します。

転送ルールでは、メッセージ転送のタイミングと転送するメッセージを指定できます。転送ルールは、サーバーとクライアントの両方に対して指定できます。

参照

- 「クライアント側の転送ルール」783 ページ
- 「サーバー側の転送ルール」784 ページ
- 「ルールの構文」774 ページ
- 「ルール変数」780 ページ

削除ルール

メッセージストアからメッセージを削除するタイミングについてデフォルトの動作を使わない場合は、削除ルールで指定できます。削除ルールは、サーバーとクライアントの両方に対して指定できます。

参照

- 「メッセージの削除ルール」786 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

QAnywhere Agent の起動

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

- **接続パラメーター**　クライアントメッセージストアに接続します。

  これは、**エージェント設定ファイルの [プロパティ]** ウィンドウで **[メッセージストア]** タブに表示される情報です。

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

  「-c qaagent オプション」715 ページを参照してください。

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

  これは、**エージェント設定ファイルの [プロパティ]** ウィンドウの **[一般]** タブで指定されます。

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

  「-id qaagent オプション」719 ページを参照してください。

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

  これは、**エージェント設定ファイルの [プロパティ]** ウィンドウで **[サーバー]** タブに表示されるサーバー情報です。

  qaagent コマンドラインでは、-x オプションです。

  「-x qaagent オプション」736 ページを参照してください。

QAnywhere Agent のオプションの完全なリストについては、「qaagent ユーティリティ」712 ページを参照してください。

Windows Mobile 上での qaagent の起動

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

「-qi qaagent オプション」731 ページを参照してください。

複数の QAnywhere Agent インスタンスの実行

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

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

「-id qaagent オプション」719 ページを参照してください。

QAnywhere Agent の停止

QAnywhere Agent を停止するには、QAnywhere Agent のメッセージウィンドウで **[シャットダウン]** をクリックします。

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

参照

- 「qastop ユーティリティ」760 ページ
- 「-qi qaagent オプション」731 ページ

QAnywhere Agent によって開始されるプロセス

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

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

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

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

- **データベースサーバー**　クライアントメッセージストアは SQL Anywhere または Ultra Light データベースです。QAnywhere Agent がデータベースを実行するためには、データベースサーバーが必要です。Windows Mobile では、データベースサーバーは *dbsrv12.exe* です。Windows では、データベースサーバーはパーソナルデータベースサーバー *dbeng12.exe* です。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

QAnywhere Agent は、データベースサーバーを生成するか、動作中のサーバーに接続します。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ーによって決まります。

参照

- 「-push qaagent オプション」729 ページ
- 「-c qaagent オプション」715 ページ

QAnywhere Agent の配備

配備の詳細については、「QAnywhere アプリケーションの配備」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ーメッセージストアにメッセージを転送するタイミングを QAnywhere Agent に指示します。ポリシーを指定しない場合は、サーバーへの配信対象メッセージがキューに登録されると、自動的に転送が行われます。これがデフォルトの動作です。ポリシーには、カスタムポリシーと、スケジュール設定、自動、オンデマンドの 3 つの事前定義済みポリシーがあります。

ポリシーは、次の方法で指定できます。

- Sybase Central の QAnywhere 12 プラグインを使用し、**[ツール] » [QAnywhere 12] » [SQL Anywhere のエージェント設定ファイルの新規作成]** をクリックします。ポリシーは、**エージェント設定ファイルの [プロパティ]** ウィンドウの **[一般]** タブで指定します。このタスクにより、Sybase Central の規則に従って拡張子 *.qaa* の付いたファイルが作成されます。

  Sybase Central の QAnywhere 12 プラグインを使用してカスタムのプロパティを指定するには、**[ツール] » [QAnywhere 12] » [新しいエージェントルールファイル]** をクリックします。このタスクにより、Sybase Central の規則に従って拡張子 *.qar* の付いたファイルが作成されます。

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

参照

- 「メッセージ転送ポリシー」41 ページ
- 「転送ルール」46 ページ
- 「-policy qaagent オプション」727 ページ

信頼性の低いネットワーク

インクリメンタルアップロードとインクリメンタルダウンロードを使用しないと、メッセージは単体で送信されるため、メッセージのアップロード中またはダウンロード中にネットワーク接続

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

インクリメンタルアップロードを実装する方法の詳細については、「-iu qaagent オプション」720 ページを参照してください。

インクリメンタルダウンロードを実装する方法の詳細については、「-idl qaagent オプション」720 ページを参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

QAnywhere クライアントアプリケーションの作成

以降の項では、QAnywhere で使用できる各種インターフェイスと関連する手順の概要を含め、QAnywhere クライアントアプリケーションの作成方法について説明します。

QAnywhere のインターフェイス

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

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

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

QAnywhere .NET API

QAnywhere .NET API は、Microsoft .NET Framework を使用している Windows コンピューターや、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ーフェイスです。QAnywhere .NET API は、iAnywhere.QAnywhere.Client ネームスペースとして提供されています。

QAnywhere は、Microsoft Visual Studio をサポートしています。

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

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

.NET バージョンの TestMessage サンプルアプリケーションの詳細については、「レッスン 4：TestMessage クライアントのソースコードの概要」183 ページを参照してください。

「クライアント用 QAnywhere .NET API リファレンス」189 ページを参照してください。

QAnywhere C++ API

QAnywhere C++ API は、Microsoft Visual Studio をサポートしています。

QAnywhere C++ API の構成ファイルは次のとおりです。

- *%SQLANY12%¥sdk¥include* にある一連のヘッダーファイル (メインのヘッダーファイルは *qa.hpp*)
- *%SQLANY12%¥sdk¥lib¥x86*、および *%SQLANY12%¥sdk¥lib¥ce¥arm.50* にあるインポートライブラリ (*qany12.lib*)
- *%SQLANY12%¥bin32*、および *%SQLANY12%¥ce¥arm.50* にあるランタイム DLL (*qany12.dll*)

API にアクセスするには、ソースコードファイルに上記のヘッダー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、*%SQLANYSAMP12%¥QAnywhere ¥Windows¥MFC* にあります。

「クライアント用 QAnywhere C++ API リファレンス」375 ページを参照してください。

QAnywhere Java API

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

QAnywhere Java API の構成ファイルは次のとおりです。

- API リファレンス。このマニュアルから情報を入手することも、SQL Anywhere 12 インストール環境のサブフォルダー *documentation¥ja¥javadocs¥QAnywhere* から Javadoc フォーマットで入手することもできます。
- SQL Anywhere 12 インストール環境のサブフォルダー *bin32* にある Ultra Light メッセージストア用のランタイム DLL (*qadbiuljni12.dll*)
- SQL Anywhere 12 インストール環境のサブフォルダー *java* にあるクラスファイルのアーカイブ (*qaclient.jar*)

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

Java バージョンの TestMessage サンプルアプリケーションは、*%SQLANYSAMP12%¥QAnywhere ¥Java¥* にあります。

「クライアント用 QAnywhere Java API リファレンス」495 ページを参照してください。

QAnywhere SQL API

QAnywhere SQL API は、SQL を使用してメッセージング API を実装するストアドプロシージャーの集まりです。QAnywhere SQL API を使用して、メッセージの作成、メッセージのプロ

パティと内容の取得や設定、メッセージの送受信、メッセージ同期のトリガー、メッセージストアプロパティの取得と設定を実行できます。

「QAnywhere SQL API リファレンス」655 ページを参照してください。

JMS コネクター

QAnywhere には、QAnywhere と JMS アプリケーション間を接続する JMS コネクターが含まれています。

次の項を参照してください。

- 「外部メッセージングシステムとのメッセージングのシナリオ」9 ページ
- 「JMS コネクター」135 ページ
- 「チュートリアル : JMS コネクターの使用」148 ページ

モバイル Web サービスコネクター

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

「モバイル Web サービス」95 ページを参照してください。

クライアントアプリケーション作成のクイックスタート

♦ クライアントアプリケーション設定の概要

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

   - 「.NET アプリケーションの設定」54 ページ
   - 「C++ アプリケーションの設定」56 ページ
   - 「Java アプリケーションの設定」58 ページ
   - 「SQL アプリケーションの設定」59 ページ

2. QAnywhere Manager の設定プロパティを設定します。「QAnywhere Manager の設定プロパティ」85 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」15 ページ
   - 「クライアントメッセージストアプロパティ」29 ページ
   - 「QAnywhere メッセージの送信」63 ページ
   - 「QAnywhere メッセージの受信」71 ページ
   - 「サイズの大きなメッセージ」75 ページ
   - 「トランザクション指向メッセージングの実装」66 ページ
   - 「QAnywhere の停止」84 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」117 ページを参照してください。

クイックスタートのためのその他の資料

- 「チュートリアル：TestMessage の解説」177 ページ
- *%SQLANYSAMP12%¥QAnywhere* にサンプルアプリケーションがあります。

QAnywhere API の初期化

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

.NET アプリケーションの設定

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

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

- QAnywhere .NET DLL への参照を追加する。参照を追加すると、QAnywhere .NET API のコードを検索するためにインクルードする必要がある DLL を Visual Studio .NET に認識させることができます。

- QAnywhere .NET API のクラスを参照する行をソースコードに追加する。QAnywhere .NET API を使用するには、データプロバイダーを参照する行もソースコードに追加する必要があります。C# では、Visual Basic .NET 用とは異なる行を追加します。

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

♦ Visual Studio のプロジェクトへの QAnywhere .NET API の参照の追加

1. Visual Studio を起動し、プロジェクトを開きます。

2. **[ソリューション エクスプローラー]** ウィンドウで、**[参照設定]** フォルダーを右クリックし、**[参照の追加]** をクリックします。

3. **[.NET]** タブで、**[参照]** をクリックして *iAnywhere.QAnywhere.Client.dll* を検索します。デフォルトロケーションは次のとおりです。

   - .NET Framework 2.0: *%SQLANY12%¥Assembly¥V2*
   - .NET Compact Framework 2.0: *%SQLANY12%¥ce¥Assembly¥V2*

   DLL を選択して **[開く]** をクリックします。

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

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ソースコードのデータプロバイダークラスを参照する

♦ コード内での QAnywhere .NET API のクラスの参照

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

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

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

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

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

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

一方、追加した場合には、次のようにクラス名を指定できます。

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

♦ QAnywhere .NET API の初期化

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

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

2. QAManager オブジェクトを作成します。

たとえば、デフォルトの QAManager オブジェクトを作成するには、CreateQAManager を呼び出します。

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

> **ヒント**
> 同時実行性の利点を最大限に生かすために、マルチスレッドアプリケーションでは、スレッドごとに QAManager を作成する必要があります。 「マルチスレッドでの考慮事項」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して、カスタマイズされた QAManager オブジェクトを作成することもできます。次のように、CreateQAManager メソッドの引数としてプロパティファイルを指定します。

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

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

3. QAManager オブジェクトを初期化します。次に例を示します。

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

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

これで、メッセージを送信する準備ができました。

> **注意**
> QAManager を作成する代わりに、QATransactionalManager を作成できます。

参照

- 「クライアント用 QAnywhere .NET API リファレンス」189 ページ
- QAManagerFactory クラス [QAnywhere .NET]279 ページ
- 「トランザクション指向メッセージングの実装 (.NET クライアントの場合)」66 ページ
- AcknowledgementMode 列挙体 [QAnywhere .NET]317 ページ

C++ アプリケーションの設定

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

♦ QAnywhere C++ API の初期化

> **注意**
> QAManager を作成する代わりに、QATransactionalManager を作成できます。 「トランザクション指向メッセージングの実装 (C++ クライアントの場合)」67 ページを参照してください。

1. QAnywhere ヘッダーファイルをインクルードします。

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

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

2. QAnywhere を初期化します。

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

```
QAManagerFactory * factory;

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

QAManagerFactory の詳細については、QAManagerFactory クラス [QAnywhere C++]450 ページを参照してください。

3. QAManager インスタンスを作成します。

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

```
QAManager *  mgr;

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

QAManager クラス [QAnywhere C++]411 ページを参照してください。

> **ヒント**
> 同時実行性の利点を最大限に生かすために、マルチスレッドアプリケーションでは、スレッドごとに QAManager を作成する必要があります。「マルチスレッドでの考慮事項」84 ページを参照してください。

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

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

  「プログラムによる QAnywhere Manager の設定プロパティの設定」88 ページを参照してください。

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

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

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

  「ファイルでの QAnywhere Manager 設定プロパティの設定」86 ページを参照してください。

4. QAManager オブジェクトを初期化します。

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

open メソッドの引数には、メッセージの受信確認方法を示す受信確認モードを指定します。これは、**IMPLICIT_ACKNOWLEDGEMENT** または **EXPLICIT_ACKNOWLEDGEMENT** のどちらかにする必要があります。前者は暗黙的な受信確認モードを意味します。このモー

ドの場合、メッセージはクライアントで受信されると受信確認されます。後者は明示的な受信確認モードを意味します。このモードの場合、QAManager のいずれかの acknowledgement メソッドを呼び出してメッセージを受信確認する必要があります。

受信確認モードの詳細については、AcknowledgementMode クラス [QAnywhere C++] 375 ページを参照してください。

これで、メッセージを送信する準備ができました。

参照

- 「クライアント用 QAnywhere C++ API リファレンス」375 ページ

Java アプリケーションの設定

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

♦ QAnywhere Java API の初期化

> **注意**
> QAManager を作成する代わりに、QATransactionalManager を作成できます。

1. *qaclient.jar* のロケーションをクラスパスに追加します。デフォルトで、このファイルは *%SQLANY12%¥Java* にあります。
2. ianywhere.qanywhere.client パッケージをインポートします。

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

3. QAManager オブジェクトを作成します。

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

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

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

   > **ヒント**
   > 同時実行性の利点を最大限に生かすために、マルチスレッドアプリケーションでは、スレッドごとに QAManager を作成する必要があります。 「マルチスレッドでの考慮事項」84 ページを参照してください。

4. QAManager オブジェクトを初期化します。

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

   open メソッドの引数には、メッセージの受信確認方法を示す受信確認モードを指定します。これは、IMPLICIT_ACKNOWLEDGEMENT または EXPLICIT_ACKNOWLEDGEMENT のど

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ちらかにする必要があります。前者は暗黙的な受信確認モードを意味します。このモードの場合、メッセージはクライアントで受信されると受信確認されます。後者は明示的な受信確認モードを意味します。このモードの場合、QAManager のいずれかの acknowledgement メソッドを呼び出してメッセージを受信確認する必要があります。

これで、メッセージを送信する準備ができました。

参照

- 「クライアント用 QAnywhere Java API リファレンス」495 ページ
- 「トランザクション指向メッセージングの実装 (Java クライアントの場合)」69 ページ
- AcknowledgementMode インターフェイス [QAnywhere Java]495 ページ

SQL アプリケーションの設定

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

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

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

パーミッション

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

受信確認モード

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

例

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

```
CREATE TRIGGER inventory_trigger AFTER UPDATE ON inventory
REFERENCING old AS oldinv new AS newinv
FOR EACH ROW
```

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

参照

- 「QAnywhere SQL API リファレンス」655 ページ
- 「QAnywhere クライアントアプリケーションの作成」51 ページ
- 「ml_qa_grant_messaging_permissions」691 ページ

QAnywhere メッセージアドレス

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

```
id¥queue-name
```

キュー名は、アプリケーション内で指定します。他のデバイスの送信アプリケーションのインスタンスも、このキュー名を認識していなければなりません。

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

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

アドレスの長さは 255 文字以内です。

システムキュー

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

「システムキュー」61 ページを参照してください。

JMS コネクター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。

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

●JMS キュー名。JMS 管理ツールを使用して作成したキューの名前です。

送信先アドレスの形式は次のとおりです。

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

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

- 「クライアントメッセージストアの設定」25 ページ
- 「JMS コネクターへの QAnywhere メッセージの送信」139 ページ
- 「JMS コネクタープロパティの設定」138 ページ
- 「JMS から QAnywhere に送信されるメッセージのアドレス指定」142 ページ
- 「コネクター」135 ページ

システムキュー

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

- 「ネットワークステータス通知」
- 「Push 通知の通知」

例

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

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

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

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

システムメッセージハンドラーは、メッセージプロパティを参照して、プロパティに設定されている情報を確認します。メッセージタイププロパティの値によって、メッセージがネットワーク

ステータス通知を保持しているかどうかを判定できます。たとえば、メッセージ msg に対して次の処理を実行できます。

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

ネットワークステータス通知

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

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

- **ias_Adapters**　文字列。Mobile Link サーバーへの接続で使用できるネットワークアダプターのリスト。各文字列はパイプ記号で区切られます。このプロパティは読み込み可能ですが、設定はできません。次の項を参照してください。

  - MessageProperties.ADAPTER フィールド [QAnywhere .NET]191 ページ
  - MessageProperties.ADAPTER 変数 [QAnywhere C++]379 ページ
  - MessageProperties.ADAPTER 変数 [QAnywhere Java]499 ページ

- **ias_RASNames**　文字列。Mobile Link サーバーへの接続で使用できるネットワーク名のリスト。各文字列はパイプ記号で区切られます。 次の項を参照してください。

  - MessageProperties.RASNAMES フィールド [QAnywhere .NET]196 ページ
  - MessageProperties.RASNAMES 変数 [QAnywhere C++]383 ページ
  - MessageProperties.RASNAMES 変数 [QAnywhere Java]503 ページ

- **ias_NetworkStatus**　整数。ネットワーク接続のステータス。値 1 は接続済み、0 はそれ以外を意味します。. 次の項を参照してください。

  - MessageProperties.NETWORK_STATUS フィールド [QAnywhere .NET]195 ページ
  - MessageProperties.NETWORK_STATUS 変数 [QAnywhere C++]382 ページ
  - MessageProperties.NETWORK_STATUS 変数 [QAnywhere Java]501 ページ

ネットワークの可用性のモニタリング

ネットワークステータス通知を使用すると、ネットワークの可用性をモニタリングして、デバイスがネットワークカバレッジの範囲内に入ったときに何らかのアクションを実行できます。たとえば、タイプ NETWORK_STATUS_NOTIFICATION のシステムキューメッセージを

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ias_NetworkStatus=1 のステータスで受信した場合、オンデマンドポリシーを使用して QAManagerBase triggerSendReceive を呼び出します。

参照

- 「事前に定義されたメッセージプロパティ」698 ページの ias_MessageType
- 「システムキュー」60 ページ

Push 通知の通知

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

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

参照

- 「Push 通知によるメッセージングのシナリオ」7 ページ
- 「Push 通知」34 ページ
- 「システムキュー」60 ページ
- 「非同期的なメッセージ受信」73 ページ
- 「事前に定義されたメッセージプロパティ」698 ページの ias_MessageType
- MessageProperties クラス [QAnywhere .NET]189 ページ
- MessageProperties クラス [QAnywhere C++]377 ページ
- MessageProperties インターフェイス [QAnywhere Java]497 ページ

QAnywhere メッセージの送信

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

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

メッセージ転送が行われるタイミングの詳細については、「クライアントにメッセージを転送するタイミングの決定」49 ページを参照してください。

♦ メッセージの送信 (.NET の場合)

1. 新しいメッセージを作成します。

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

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

2. メッセージのプロパティを設定します。

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

   「QAnywhere メッセージ」15 ページを参照してください。

3. メッセージをキューに登録します。これで送信準備が完了します。

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

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

   QAManagerBase.PutMessage メソッド [QAnywhere .NET]263 ページと 「クライアントにメッセージを転送するタイミングの決定」49 ページを参照してください。

♦ メッセージの送信 (C++ の場合)

1. 新しいメッセージを作成します。

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

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

2. メッセージのプロパティを設定します。

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

   「QAnywhere メッセージ」15 ページを参照してください。

3. メッセージをキューに登録します。これで送信準備が完了します。

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

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

   QAManagerBase.putMessage メソッド [QAnywhere C++]441 ページと 「クライアントにメッセージを転送するタイミングの決定」49 ページを参照してください。

♦ メッセージの送信 (Java の場合)

1. 新しいメッセージを作成します。

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

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

2. メッセージのプロパティを設定します。

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

   「QAnywhere メッセージ」15 ページを参照してください。

3. メッセージをキューに登録します。

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

QAManagerBase.putMessage メソッド [QAnywhere Java]565 ページと 「クライアントにメッセージを転送するタイミングの決定」49 ページを参照してください。

♦ メッセージの送信 (SQL の場合)

1. メッセージ ID を格納する変数を宣言します。

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

2. 新しいメッセージを作成します。

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

3. メッセージのプロパティを設定します。

   詳細については、「メッセージプロパティ」665 ページを参照してください。

4. メッセージをキューに登録します。

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

「ml_qa_putmessage」693 ページと「クライアントにメッセージを転送するタイミングの決定」49 ページを参照してください。

トランザクション指向メッセージングの実装

トランザクション指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。

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

参照

- QATransactionalManager インターフェイス [QAnywhere .NET]309 ページ
- QATransactionalManager クラス [QAnywhere C++]484 ページ
- QATransactionalManager インターフェイス [QAnywhere Java]609 ページ
- SQL クライアント : SQL クライアントでは、すべてのメッセージングがトランザクション指向であるため、トランザクション指向マネージャーは不要です。

トランザクション指向メッセージングの実装 (.NET クライアントの場合)

♦ トランザクション指向マネージャーの作成

1. QAnywhere を初期化します。

   この手順は、非トランザクション指向メッセージングの場合と同じです。

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

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

   たとえば、デフォルトの QATransactionalManager オブジェクトを作成するには、CreateQATransactionalManager を呼び出します。

   ```
   QATransactionalManager mgr = fact.CreateQATransactionalManager();  // no argument
   ```

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

   ```
   QAManagerFactory fact = QAManagerFactory.Instance( "qa_mgr.props" );
   ```

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

3. QAManager オブジェクトを初期化します。

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

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

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

♦ 1 つのトランザクションでの複数のメッセージの送信

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();
msg_2 = mgr.CreateTextMessage();
mgr.PutMessage( "jms_1¥¥queue_name", msg_1 );
mgr.PutMessage( "jms_1¥¥queue_name", msg_2 );
mgr.Commit();
```

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

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

参照

- QATransactionalManager インターフェイス [QAnywhere .NET]309 ページ
- QAManagerFactory クラス [QAnywhere .NET]279 ページ

トランザクション指向メッセージングの実装 (C++ クライアントの場合)

♦ トランザクション指向マネージャーの作成

1. QAnywhere を初期化します。

   この手順は、非トランザクション指向メッセージングの場合と同じです。

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

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

2. トランザクション指向マネージャーを作成します。

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

非トランザクション指向マネージャーの場合と同様に、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. マネージャーを初期化します。

```
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。

♦ 1 つのトランザクションでの複数のメッセージの送信

1. メッセージオブジェクトを初期化します。

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

2. メッセージを送信します。

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

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

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

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

参照

- QATransactionalManager クラス [QAnywhere C++]484 ページ
- QATransactionalManager インターフェイス [QAnywhere Java]609 ページ

トランザクション指向メッセージングの実装 (Java クライアントの場合)

♦ トランザクション指向マネージャーの作成

1. QAnywhere を初期化します。

   この手順は、非トランザクション指向メッセージングの場合と同じです。

   ```
   import ianywhere.qanywhere.client.*;
   QAManagerFactory fact = QAManagerFactory.getInstance();
   ```

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

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

   ```
   QATransactionalManager mgr = fact.createQATransactionalManager();  // no argument
   ```

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

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

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

3. QAManager オブジェクトを初期化します。

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

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

♦ 1 つのトランザクションでの複数のメッセージの送信

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();
   msg_2 = mgr.createTextMessage();
   mgr.putMessage( "jms_1¥¥queue_name", msg_1 );
   mgr.putMessage( "jms_1¥¥queue_name", msg_2 );
   mgr.commit();
   ```

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

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

参照

- QAManagerFactory クラス [QAnywhere Java]578 ページ

QAnywhere メッセージのキャンセル

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

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

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

♦ メッセージのキャンセル (.NET の場合)

1. キャンセルするメッセージの ID を取得します。

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

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

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

♦ メッセージのキャンセル (C++ の場合)

1. キャンセルするメッセージの ID を取得します。

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

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

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

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

♦ **メッセージのキャンセル (Java の場合)**

1. キャンセルするメッセージの ID を取得します。

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

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

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

参照

- QAManagerBase.CancelMessage メソッド [QAnywhere .NET]247 ページ
- QAManagerBase.cancelMessage メソッド [QAnywhere C++]427 ページ
- QAManagerBase.cancelMessage メソッド [QAnywhere Java]550 ページ

QAnywhere メッセージの受信

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

同期的なメッセージ受信

メッセージを同期的に受信するには、アプリケーションから明示的にキューをポーリングしてメッセージの有無を確認します。キューのポーリングは、定期的に、またはユーザー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。

♦ **メッセージの同期的受信 (.NET の場合)**

1. 着信メッセージを格納するメッセージオブジェクトを宣言します。

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

2. メッセージキューをポーリングして、メッセージを収集します。

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

♦ **メッセージの同期的受信 (C++ の場合)**

1. 着信メッセージを格納するメッセージオブジェクトを宣言します。

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

2. メッセージキューをポーリングして、メッセージを収集します。

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

♦ メッセージの同期的受信 (Java の場合)

1. 着信メッセージを格納するメッセージオブジェクトを宣言します。

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

2. メッセージキューをポーリングして、メッセージを収集します。

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

♦ メッセージの同期的受信 (SQL の場合)

1. メッセージ ID を格納するオブジェクトを宣言します。

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

2. メッセージキューをポーリングして、メッセージを収集します。

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

次の項を参照してください。

- QAManagerBase.GetMessageNoWait メソッド [QAnywhere .NET]256 ページ
- QAManagerBase.getMessageNoWait メソッド [QAnywhere C++]437 ページ
- QAManagerBase.getMessageNoWait メソッド [QAnywhere Java]559 ページ
- 「ml_qa_getmessagenowait」689 ページ
- 「ml_qa_getmessagetimeout」690 ページ
- 「ml_qa_getmessage」687 ページ

非同期的なメッセージ受信

.NET、C++、Java の API を使用して、非同期的にメッセージを受信するには、メッセージリスナー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ー関数が QAnywhere によって呼び出されます。メッセージリスナーは、着信メッセージを引数とします。メッセージリスナー内で実行する処理は、アプリケーションによって異なります。たとえば、TestMessage サンプルアプリケーションのメッセージリスナーは、TestMessage のメインウィンドウのメッセージリストに着信メッセージを追加します。

.NET、C++、Java での開発のヒント

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

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

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

♦ メッセージの非同期的受信 (.NET の場合)

1. メッセージハンドラーメソッドを実装します。

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

2. メッセージハンドラーを登録します。

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

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

♦ メッセージの非同期的受信 (C++ の場合)

1. QAMessageListener インターフェイスを実装するクラスを作成します。

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

2. onMessage メソッドを実装します。

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

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

3. メッセージリスナーを登録します。

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

♦ メッセージの非同期的受信 (Java の場合)

1. メッセージハンドラーメソッドと例外ハンドラーメソッドを実装します。

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

2. メッセージハンドラーを登録します。

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

♦ メッセージの非同期的受信 (SQL の場合)

- **ml_qa_listener_*queue*** という名前のストアドプロシージャーを作成します。*queue* はメッセージキューの名前です。

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

参照

- MessageListener デリゲート [QAnywhere .NET]316 ページ
- QAManagerBase.SetMessageListener メソッド [QAnywhere .NET]270 ページ
- QAMessageListener クラス [QAnywhere C++]477 ページ
- QAManagerBase.setMessageListener メソッド [QAnywhere C++]445 ページ
- QAMessageListener インターフェイス [QAnywhere Java]600 ページ
- QAManagerBase.setMessageListener メソッド [QAnywhere Java]571 ページ
- 「ml_qa_listener_queue」692 ページ

セレクターを使用したメッセージの受信

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

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

「条件構文」776 ページを参照してください。

例

注意
SQL API では、セレクターを使用したメッセージ受信はサポートしていません。

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

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

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

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

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

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

参照

- QAManagerBase.GetMessageBySelector メソッド [QAnywhere .NET]254 ページ
- QAManagerBase.GetMessageBySelectorNoWait メソッド [QAnywhere .NET]255 ページ
- QAManagerBase.getMessageBySelector メソッド [QAnywhere C++]435 ページ
- QAManagerBase.getMessageBySelectorNoWait メソッド [QAnywhere C++]436 ページ
- QAManagerBase.getMessageBySelector メソッド [QAnywhere Java]557 ページ
- QAManagerBase.getMessageBySelectorNoWait メソッド [QAnywhere Java]557 ページ

サイズの大きなメッセージ

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

接読み込むことはできます。それには、QATextMessage.readText() メソッドまたは QABinaryMessage.readBinary() メソッドを使用してループ処理します。

注意
SQL API では、サイズの大きなメッセージを受信できません。

詳細については、次の項を参照してください。

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。

参照

- QABinaryMessage.ReadBinary メソッド [QAnywhere .NET]204 ページ
- QATextMessage.ReadText メソッド [QAnywhere .NET]307 ページ
- QABinaryMessage.readBinary メソッド [QAnywhere C++]394 ページ
- QATextMessage.readText メソッド [QAnywhere C++]482 ページ
- QABinaryMessage.readBinary メソッド [QAnywhere Java]514 ページ
- QATextMessage.readText メソッド [QAnywhere Java]606 ページ
- 「受信確認モード」59 ページ

QAnywhere のメッセージの参照

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

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

すべてのメッセージの参照

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

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

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

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

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

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

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

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

参照

- 「事前に定義されたメッセージプロパティ」698 ページ
- QAManagerBase.BrowseMessages メソッド [QAnywhere .NET]243 ページ
- QAManagerBase.browseMessages メソッド [QAnywhere C++]423 ページ
- QAManagerBase.browseMessages メソッド [QAnywhere Java]548 ページ
- SQL：SQL API では、メッセージを参照できません。

特定のキューに登録されているメッセージの参照

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

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

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

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

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

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

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

参照

- QAManagerBase.BrowseMessagesByQueue メソッド [QAnywhere .NET]246 ページ
- QAManagerBase.browseMessagesByQueue メソッド [QAnywhere C++]425 ページ
- QAManagerBase.browseMessagesByQueue メソッド [QAnywhere Java]549 ページ
- SQL：SQL API では、メッセージを参照できません。

ID の指定によるメッセージの参照

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

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

```
QAMessage msg;
IEnumerator msgs = mgr.BrowseMessagesByID( "ID:123" );
if( msgs.MoveNext() ) {
  msg = (QAMessage)msgs.Current;
  // Process message.
}
```

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

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

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

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

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- QAManagerBase.BrowseMessagesByID メソッド [QAnywhere .NET]245 ページ
- QAManagerBase.browseMessagesByID メソッド [QAnywhere C++]424 ページ
- QAManagerBase.browseMessagesByID メソッド [QAnywhere Java]548 ページ
- SQL：SQL API では、メッセージを参照できません。

セレクターを使用したメッセージの参照

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

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

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

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

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

```
QAMessage *msg;
qa_browse_handle bh = mgr->browseMessagesBySelector( _T("intprop = 1") );
for (;;) {
  msg = mgr->browseNextMessage( bh );
  if( msg == qa_null ) {
    break;
  }
  // Process message.
}
mgr->browseClose( bh );
```

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

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

参照

- 「条件構文」776 ページ
- QAManagerBase.BrowseMessagesBySelector メソッド [QAnywhere .NET]246 ページ
- QAManagerBase.browseMessagesBySelector メソッド [QAnywhere C++]425 ページ
- QAManagerBase.browseMessagesBySelector メソッド [QAnywhere Java]550 ページ
- SQL：SQL API では、メッセージを参照できません。

QAnywhere での例外の処理

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

.NET 例外

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

QAException がメッセージリスナーデリゲートでスローされ、メッセージリスナーでキャッチされなかった場合は、その内容が QAManager ログファイルに記録されることに注意してください。キャッチされなかった QAException はログに記録されるだけなので、すべての例外がメッセージリスナーデリゲート内または例外リスナーデリゲートによって処理されるようにすることをおすすめします。こうすることで、例外は適切に処理されます。

メッセージリスナーデリゲートと例外リスナーデリゲートの詳細については、次の項を参照してください。

- MessageListener デリゲート [QAnywhere .NET]316 ページ
- MessageListener2 デリゲート [QAnywhere .NET]316 ページ
- ExceptionListener デリゲート [QAnywhere .NET]315 ページ
- ExceptionListener2 デリゲート [QAnywhere .NET]315 ページ

ログファイルの詳細については、「QAnywhere Manager の設定プロパティ」85 ページを参照してください。

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

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

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

詳細については、次の項を参照してください。

- QAException クラス [QAnywhere .NET]218 ページ
- QAException.ErrorCode プロパティ [QAnywhere .NET]222 ページ
- 「事前に定義されたクライアントメッセージストアプロパティ」761 ページ

C++ 例外

C++ の場合は、QAError クラスが QAnywhere クライアントアプリケーションの例外をカプセル化します。QAManagerBase::getLastError() メソッドまたは QAManagerFactory::getLastError() メソッドを使用すると、最後に実行されたメソッドに関連するエラーコードを特定できます。また、対応する getLastErrorMessage() メソッドを使用すると、エラーテキストを取得できます。

エラーコードのリストと詳細については、QAError クラス [QAnywhere C++]403 ページを参照してください。

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

- QAManagerBase.getLastError メソッド [QAnywhere C++]433 ページ
- QAManagerBase.getLastErrorMsg メソッド [QAnywhere C++]433 ページ
- QAManagerFactory.getLastError メソッド [QAnywhere C++]453 ページ
- QAManagerFactory.getLastErrorMsg メソッド [QAnywhere C++]453 ページ

Java 例外

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

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

メッセージリスナーと例外リスナーの詳細については、次の項を参照してください。

- QAMessageListener インターフェイス [QAnywhere Java]600 ページ
- QAMessageListener2 インターフェイス [QAnywhere Java]601 ページ
- QAException クラス [QAnywhere Java]527 ページ

ログファイルの詳細については、「QAnywhere Manager の設定プロパティ」85 ページを参照してください。

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

プロパティ ias_MaxDeliveryAttempts が、mgr.SetIntStoreProperty("ias_MaxDeliveryAttempts", 5) のように QAnywhere アプリケーションによって正の整数 *n* に設定されると、QAnywhere クラ

イアントは、メッセージのステータスを受信不可にする前に、受信確認されていないメッセージを最大 *n* 回まで受信するように試みます。プロパティ ias_MaxDeliveryAttempts が設定されていないか、負の値に設定されると、QAnywhere クライアントは、メッセージの受信を無制限に試みます。

詳細については、次の項を参照してください。

- QAException.ErrorCode プロパティ [QAnywhere .NET]222 ページ
- 「事前に定義されたクライアントメッセージストアプロパティ」761 ページ

エラーコード

QAnywhere のエラーコードを次の表に示します。

エラー値	説明
0	エラーはありません。
1000	初期化エラーです。
1001	終了エラーです。
1002	クライアントのプロパティファイルにアクセスできません。
1003	宛先が指定されていません。
1004	この関数は実装されていません。
1005	読み込み専用モードであるため、メッセージに書き込みできません。
1006	クライアントメッセージストアにメッセージを格納するときにエラーが発生しました。
1007	クライアントメッセージストアからメッセージを取得するときにエラーが発生しました。
1008	バックグラウンドスレッドを初期化中にエラーが発生しました。
1009	メッセージストアへの接続を開くときにエラーが発生しました。
1010	クライアントのプロパティファイルに無効なプロパティが存在します。
1011	ログファイルを開くときにエラーが発生しました。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

エラー値	説明
1012	予期しないメッセージの終わりに到達しました。
1013	メッセージストアがデバイスのディスク空き容量と比べて大きすぎます。
1014	メッセージストアがメッセージを送るために初期化されていません。
1015	キューの長さの取得中にエラーが発生しました。
1016	メッセージストア ID が設定されていない場合は QAManagerBase.getQueueDepth を使用できません。
1017	フィルターが ALL の場合、指定された宛先には QAManagerBase.getQueueDepth を使用できません。
1018	メッセージのキャンセル中にエラーが発生しました。
1019	メッセージのキャンセル中にエラーが発生しました。すでに送信されたメッセージをキャンセルすることはできません。
1020	メッセージの確認中にエラーが発生しました。
1021	QAManager が開いていません。
1022	QAManager はすでに開いています。
1023	指定されたセレクターに構文エラーがあります。
1024	タイムスタンプが許容範囲外です。
1025	サーバーの最大同時要求数が少なすぎるため QAManager を開くことができません。 「-gn dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』を参照してください。
1026	メッセージストアからプロパティを取得するときにエラーが発生しました。

エラー値	説明
1027	メッセージストアにプロパティを格納するときにエラーが発生しました。

QAnywhere の停止

メッセージの送信と受信を完了したら、次のいずれかの手順に従って QAnywhere メッセージングシステムを停止します。

♦ QAnywhere の停止 (.NET の場合)

● QAnywhere Manager を停止し、クローズします。

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

♦ QAnywhere の停止 (C++ の場合)

1. QAnywhere Manager をクローズします。

```
mgr->stop();
mgr->close();
```

2. ファクトリを終了します。

```
QAnywhereFactory_term();
```

これで、アプリケーションのメッセージング部分が停止します。

♦ QAnywhere の停止 (Java の場合)

● QAnywhere Manager を停止し、クローズします。

```
mgr.stop();
mgr.close();
```

参照

● QAManagerBase.Stop メソッド [QAnywhere .NET]277 ページ
● QAManagerBase.stop メソッド [QAnywhere C++]449 ページ
● QAManagerBase.stop メソッド [QAnywhere Java]577 ページ

● SQL：SQL API では、QAnywhere を停止できません。

マルチスレッドでの考慮事項

QAManager へのアクセスは直列化されます。1 つの QAManager へのアクセスにマルチスレッドを使用すると、あるスレッドが QAManager に対してメソッド呼び出しを実行している間、他の

スレッドはブロックされます。同時実行性を最大化するためには、スレッドごとに異なる QAManager を使用します。QAManager のインスタンスに同時にアクセスできるスレッドは、1 つだけです。その他のスレッドは、最初のスレッドによって呼び出された QAManager メソッドが戻るまで、ブロックされます。

QAnywhere Manager の設定プロパティ

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

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

  「ファイルでの QAnywhere Manager 設定プロパティの設定」86 ページを参照してください。

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

  「プログラムによる QAnywhere Manager の設定プロパティの設定」88 ページを参照してください。

次に、QAnywhere Manager の設定プロパティを示します。

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

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

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

  このプロパティはスタンドアロンクライアントではサポートされていません。

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

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

  データベースユーザーとパスワードの管理の詳細については、「セキュリティ保護されたメッセージングアプリケーション」121 ページを参照してください。

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

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

- **MAX_IN_MEMORY_MESSAGE_SIZE=n**　メッセージの読み込み時、バッファーに割り当てられるメッセージの最大値を *n* バイトにします。サイズが *n* バイトを超えるメッセージは、

ストリーム操作を使用して読み込む必要があります。デフォルト値は、Windows では 1 MB、Windows Mobile では 64 KB です。

次に、スタンドアロンクライアント専用のプロパティを示します。

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

ファイルでの QAnywhere Manager 設定プロパティの設定

注意
QAnywhere Manager の設定ファイルは Sybase Central で作成したり、開いたりできます。QAnywhere 12 プラグインのタスクのウィンドウ枠で、**[エージェント設定ファイルの新規作成]** をクリックします。ファイル名とロケーションを選択すると、設定ファイルの **[プロパティ]** ウィンドウが開きます。このウィンドウでプロパティを設定できます。

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

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

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

プロパティファイル名の指定の詳細については、次の項を参照してください。

- QAManagerFactory.CreateQAManager メソッド [QAnywhere .NET]280 ページ
- QAManagerFactory.createQAManager メソッド [QAnywhere C++]451 ページ
- QAManagerFactory.createQAManager メソッド [QAnywhere Java]578 ページ
- SQL API：QAnywhere SQL API を使用して、プロパティをファイルに設定することはできません。「プログラムによる QAnywhere Manager の設定プロパティの設定」88 ページを参照してください。

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

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

QAnywhere Manager の設定プロパティファイルのエントリは、*name*=*value* の形式を取ります。*value* にスペースが含まれる場合は、二重引用符で囲みます。コメント行の先頭には # を付けます。次に例を示します。

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

プロパティ名のリストについては、「QAnywhere Manager の設定プロパティ」85 ページを参照してください。

設定ファイルの参照

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

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

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

次は、C# のコード例です。

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

.NET API については、QAManager インターフェイス [QAnywhere .NET]231 ページと QAManagerFactory クラス [QAnywhere .NET]279 ページを参照してください。

次は、C++ のコード例です。

```
QAManagerFactory *  qa_factory;
QAManager *    mgr;
qa_factory = QAnywhereFactory_init();
```

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

C++ API については、QAManager クラス [QAnywhere C++]411 ページとQAManagerFactory クラス [QAnywhere C++]450 ページを参照してください。

次は、Java のコード例です。

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

Java API については、QAManagerFactory クラス [QAnywhere Java]578 ページとQAManager インターフェイス [QAnywhere Java]536 ページを参照してください。

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

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

例

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

```
QAManager     mgr;
mgr = QAManagerFactory.Instance.CreateQAManager( "null" );
mgr.SetProperty( "COMPRESSION_LEVEL", "9" );
mgr.SetProperty( "CONNECT_PARAMS", "DBF=mystore.db" );
mgr.SetProperty( "DATABASE_TYPE", "sqlanywhere" );
mgr.Open( 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。

```
QAManagerFactory *  qa_factory;
QAManager *    mgr;
qa_factory = QAnywhereFactory_init();
mgr = qa_factory->createQAManager( NULL );
mgr->setProperty( "COMPRESSION_LEVEL", "9" );
mgr->setProperty( "CONNECT_PARAMS", "DBF=mystore.db" );
mgr->setProperty( "DATABASE_TYPE", "sqlanywhere" );
mgr->open( AcknowledgementMode::EXPLICIT_ACKNOWLEDGEMENT );
```

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

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

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

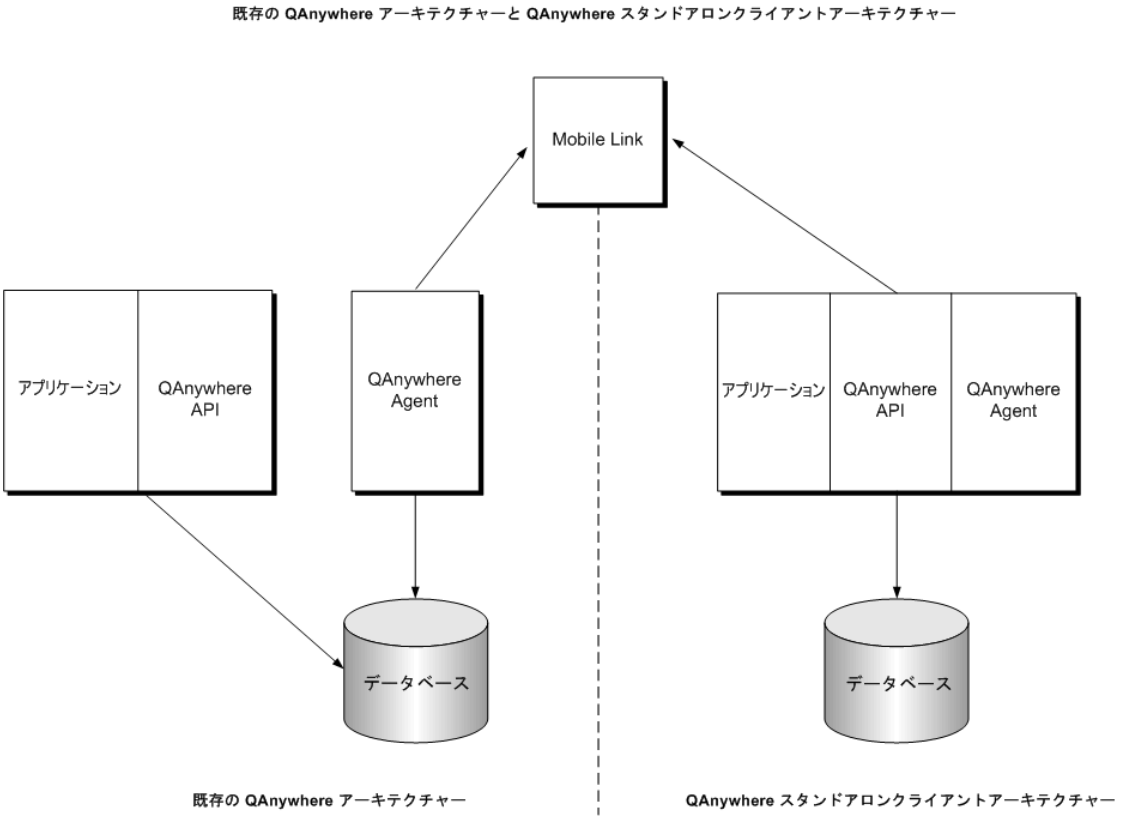
参照

- 「QAnywhere Manager の設定プロパティ」85 ページ
- QAManager インターフェイス [QAnywhere .NET]231 ページ
- QAManagerFactory クラス [QAnywhere .NET]279 ページ
- QAManager クラス [QAnywhere C++]411 ページ
- QAManagerFactory クラス [QAnywhere C++]450 ページ
- QAManagerFactory クラス [QAnywhere Java]578 ページ
- QAManager インターフェイス [QAnywhere Java]536 ページ

QAnywhere スタンドアロンクライアント

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

次の図に、スタンドアロンクライアントアーキテクチャーと既存の QAnywhere アーキテクチャーを示します。



スタンドアロンクライアントのメッセージストア

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

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

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

参照

- 「QAnywhere Manager の設定プロパティ」85 ページ

スタンドアロンクライアントの配備

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

- .NET 実装は、*iAnywhere.QAnywhere.StandAloneClient.dll* という DLL で配布され、ネームスペース iAnywhere.QAnywhere.StandAloneClient を使用します
- Java 実装は、*qastandaloneclient.jar* という JAR ファイルで配布され、パッケージ名 ianywhere.qanywhere.standaloneclient を使用します

参照

- 「クライアント用 QAnywhere .NET API リファレンス」189 ページ
- 「クライアント用 QAnywhere Java API リファレンス」495 ページ

スタンドアロンクライアント API

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

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

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

- **ML_PROTOCOL_PARAMS**　Mobile Link 接続パラメーターを指定します。

- **ML_PROTOCOL_USENAME**　Mobile Link ユーザー名を指定します。QAnywhere Agent の -mu オプションと同じです。「-mu qaagent オプション」723 ページを参照してください。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- **ML_PROTOCOL_PASSWORD**　Mobile Link ユーザーの新しいパスワードを指定します。QAnywhere Agent の -mn オプションと同じです。「-mn qaagent オプション」722 ページを参照してください。

- **INC_UPLOAD**　インクリメンタルアップロードのサイズを指定します。QAnywhere Agent の -iu オプションと同じです。「-iu qaagent オプション」720 ページを参照してください。

- **INC_DOWNLOAD**　インクリメンタルダウンロードのサイズを指定します。QAnywhere Agent の -idl オプションと同じです。「-idl qaagent オプション」720 ページを参照してください。

- **STORE_ID**　スタンドアロンクライアントが接続するクライアントメッセージストアの ID を指定します。QAnywhere Agent の -id オプションと同じです。「-id qaagent オプション」719 ページを参照してください。

- **STORE_ENCRYPTION_KEY**　メッセージストアの暗号化に使用する暗号化キーを指定します。

- **POLICY**　メッセージ転送のタイミングを決定するポリシーを指定します。QAnywhere Agent の -policy オプションと同じです。「-policy qaagent オプション」727 ページを参照してください。

- **DELETE_PERIOD**　最終ステータスに達したメッセージを削除する間隔を秒数で指定します。

- **PUSH**　Push 通知の配信方法を指定します。スタンドアロンクライアントのデフォルトが -push lwpoll であることを除き、QAnywhere Agent の -push オプションと同じです。「-push qaagent オプション」729 ページを参照してください。

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

- CONNECT_PARAMS
- DATABASE_TYPE

参照

- 「QAnywhere Manager の設定プロパティ」85 ページ
- 「qaagent ユーティリティ」712 ページ

モバイル Web サービス

Web サービスは、アプリケーションの機能を公開するための一般的な方法になっています。Web サービスを使用すると、さまざまな企業のリソース間で、より良い相互運用性を確保できます。これらのサービスは、モバイルアプリケーションの機能を広げ、開発プロセスを簡素化します。

モバイル環境に Web サービスを実装することは、取り組みがいのある仕事になるでしょう。なぜなら、接続を利用できなかったり (または中断されたり)、無線通信環境やデバイスに起因する制限事項があったりするためです。たとえば、モバイルアプリケーションを使用するユーザーは、オフラインの状態で Web サービスに対する要求を作成したり、オフラインに変更するときに応答を取得したりできます。また、IT 管理者は、モバイルアプリケーションが使用しているネットワーク接続のタイプ (GPRS、802.11、クレードルなど) に基づいて Web サービス応答のサイズを制限するルールを指定できます。

QAnywhere は、QAnywhere の蓄積転送メッセージングアーキテクチャーを活用する、モバイルに最適化された非同期 Web サービスを使用して、これらの課題に取り組みます。QAnywhere のモバイル Web サービスを使用すると、モバイルアプリケーションでは、オフラインの状態でも Web サービス要求を作成し、後で転送できるように要求をキューイングできます。要求は、QAnywhere メッセージとして配信されます。サーバー側の Web サービスコネクターが要求を作成し、Web サービスから要求を受け取って、クライアントに対してメッセージとして応答が返されます。QAnywhere 転送ルールは、さまざまなパラメーター (使用されるネットワーク、要求／応答のサイズ) に基づいて、送信される要求や応答を制御できます。転送ルールで制御することにより、モバイルアプリケーションが実証済みのテクノロジや単純なプログラミングモデルを使用してさまざまな Web サービスを利用できるような、洗練された柔軟なアーキテクチャーが構築されます。

開発の観点から見ると、接続された環境での場合と同じように Web サービスのプロキシクラスを扱うことができます。QAnywhere は、転送、認証、直列化などの処理を行います。WSDL ドキュメントを取得し、モバイルアプリケーションが Web サービスの起動に使用できる特定のクラスプロキシ (.NET または Java) を生成することを目的に、WSDL コンパイラーが提供されます。これらのクラスは、基本の QAnywhere インフラストラクチャを使用して、要求を送信し、応答を受信します。オブジェクトのメソッド呼び出しが行われると、SOAP 要求が自動的に生成され、メッセージとしてサーバーに配信されます。サーバー側では、コネクターが Web サービス要求を作成し、メッセージとして結果を返します。

参照

- 「モバイル Web サービス」『SQL Anywhere 12 紹介』

モバイル Web サービスの設定

♦ モバイル Web サービス設定の概要

1. サーバーメッセージストアがない場合は設定します。

   「サーバーメッセージストアの設定」24 ページを参照してください。

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

   「有効な Mobile Link での QAnywhere の起動」31 ページを参照してください。

3. クライアントメッセージストアがない場合は設定します。このメッセージストアは、メッセージを一時的に格納するための SQL Anywhere データベースです。

   「クライアントメッセージストアの設定」25 ページを参照してください。

4. iAnywhere WSDL コンパイラーを実行し、アプリケーションで使用するクラスを作成します。

   「iAnywhere WSDL コンパイラー」97 ページを参照してください。

5. 各クライアントで、WSDL コンパイラーによって生成されたクラスを使用する Web サービスクライアントアプリケーションを記述します。

   「モバイル Web サービスアプリケーション」98 ページを参照してください。

6. Web サービスコネクターを作成します。

   「Web サービスコネクター」144 ページを参照してください。

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

   「QAnywhere Agent の起動」47 ページを参照してください。

クイックスタートのためのその他の資料

- Java で天気予報 Web サービスを設定する方法を示す例は、「モバイル Web サービスの例」で述べられています。
- 通貨換算 Web サービスを使用したモバイル Web サービスのサンプルアプリケーションは、*%SQLANYSAMP12%¥QAnywhere¥MobileWebServices* にあります。このサンプルは、Java と C# の両方で提供されています。
- 質問は、次の SQL Anywhere フォーラムに投稿できます。http://sqlanywhere-forum.sybase.com

モバイル Web サービス開発のヒント

- モバイル Web サービスの .NET アプリケーションについては、アプリケーションの実行プログラムに、WSDL コンパイラーで生成されたプロキシクラスをコンパイルする必要があります。これらのプロキシクラスをそれ自体のアセンブリにコンパイルすることはできません。

- モバイル Web サービスの Java アプリケーションについては、JDK 1.5.x を使用する必要があります。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

iAnywhere WSDL コンパイラー

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

WSDL コンパイラーで生成される Java クラスまたは C# クラスは、QAnywhere で使用します。これらのクラスは、メソッド呼び出しとして Web サービス操作を公開します。生成されるクラスは以下のとおりです。

- メインのサービスバインディングクラス (このクラスは、モバイル Web サービスランタイムの ianywhere.qanywhere.ws.WSBase を継承します)
- プロキシクラス (WSDL ファイルに指定された複合型ごとに作成されます)

作成されるプロキシクラスの詳細については、次の項を参照してください。

- 「Web サービス用 QAnywhere .NET API リファレンス」325 ページ
- 「Web サービス用 QAnywhere Java API リファレンス」620 ページ

WSDL コンパイラーは、HTTP と HTTPS を介して WSDL 1.1 と SOAP 1.1 をサポートします。

構文

wsdlc [*options*] *wsdl-uri*

wsdl-uri :

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

オプション

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

Java 固有のオプション

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

C# 固有のオプション

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

SQL 固有のオプション

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

モバイル Web サービスアプリケーション

アプリケーションは、Web サービス要求を QAnywhere に送信します。QAnywhere は、受信した要求を、Mobile Link サーバーのモバイル Web サービスコネクターに送信します。コネクターは、受信した要求を Web サービスに送信します。または、Web サービスが利用可能になるまで、その要求をキューに格納します。QAnywhere は、要求を受け取ると、アプリケーションに通知します。または、アプリケーションが利用可能になるまで、その要求をキューに格納します。

.NET モバイル Web サービスアプリケーションの設定

QAnywhere で .NET を使用する前に、Visual Studio プロジェクトに対して以下の変更を加える必要があります。

- QAnywhere .NET DLL とモバイル Web サービス .NET DLL への参照を追加します。これにより、Visual Studio に対して、QAnywhere .NET API とモバイル Web サービス .NET API のコードを見つけるためにインクルードする DLL を指定します。
- ソースコードに、QAnywhere .NET API クラスとモバイル Web サービス .NET API クラスを参照する行を追加します。QAnywhere .NET API を使用するには、データプロバイダーを参照する行もソースコードに追加する必要があります。C# では、Visual Basic 用とは異なる行を追加します。

♦ Visual Studio のプロジェクトへの QAnywhere .NET API とモバイル Web サービス API の参照の追加

1. Visual Studio を起動し、プロジェクトを開きます。
2. [ソリューション エクスプローラー] ウィンドウで、**[参照設定]** フォルダーを右クリックし、**[参照の追加]** をクリックします。
3. **[参照]** タブで、次のディレクトリにある *iAnywhere.QAnywhere.Client.dll* と *iAnywhere.QAnywhere.WS.dll* を探します。
   - .NET Framework 2.0: *%SQLANY12%¥Assembly¥V2*
   - .NET Compact Framework 2.0: *%SQLANY12%¥ce¥Assembly¥V2*

   使用環境に応じた適切なディレクトリから、それぞれの DLL を選択し、**[開く]** をクリックします。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

4. DLL がプロジェクトに追加されたことを確認するには、ソリューションエクスプローラーで **[参照]** ツリーを展開します。リストに *iAnywhere.QAnywhere.Client.dll* と *iAnywhere.QAnywhere.WS.dll* が表示されています。

ソースコードのデータプロバイダークラスを参照する

♦ コード内での QAnywhere .NET API クラスとモバイル Web サービス API クラスの参照

1. Visual Studio を起動し、プロジェクトを開きます。

2. C# を使用している場合は、ファイルの先頭にある using ディレクティブのリストに次の行を追加します。

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

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

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

Imports 行は必須ではありません。しかし、この行を追加すると、QAnywhere とモバイル Web サービスのクラスの省略形を使用できるようになります。この行を追加しなくても、完全に修飾されたクラス名をコードで使用できます。たとえば、次のコードは長形式を使用しています。

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

次のコードは省略形を使用しています。

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

♦ .NET 用の QAnywhere とモバイル Web サービスの初期化

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

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

2. QAManager オブジェクトを作成します。

たとえば、デフォルトの QAManager オブジェクトを作成するには、CreateQAManager を呼び出します。

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

ヒント
同時実行性の利点を最大限に生かすために、マルチスレッドアプリケーションでは、スレッドごとに QAManager を作成する必要があります。「マルチスレッドでの考慮事項」84 ページを参照してください。

QAManagerFactory の詳細については、QAManagerFactory クラス [QAnywhere .NET] 279 ページを参照してください。

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

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

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

3. QAManager オブジェクトを初期化します。次に例を示します。

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

open メソッドの引数には、メッセージの受信確認方法を示す受信確認モードを指定します。これは、IMPLICIT_ACKNOWLEDGEMENT または EXPLICIT_ACKNOWLEDGEMENT のどちらかにする必要があります。

モバイル Web サービスによって使用される QAnywhere メッセージは、モバイル Web サービスアプリケーションにアクセスできません。EXPLICIT_ACKNOWLEDGEMENT モードで QAManager を使用する場合は、WSResult の Acknowledge メソッドを使用して、Web サービス要求の結果を含む QAnywhere メッセージの受信確認を行います。このメソッドは、アプリケーションが応答を正常に処理したことを示します。

受信確認モードの詳細については、次の項を参照してください。

- WSBase.SetQAManager メソッド [QAnywhere .NET]330 ページ
- WSResult.Acknowledge メソッド [QAnywhere .NET]344 ページ

注意
QAManager を作成する代わりに、QATransactionalManager を作成できます。「トランザクション指向メッセージングの実装 (.NET クライアントの場合)」66 ページを参照してください。

4. サービスバインディングクラスのインスタンスを作成します。

モバイル Web サービスの WSDL コンパイラーは、Web サービスを定義する WSDL ドキュメントからサービスバインディングクラスを生成します。

QAManager は、Web サービスバインディングクラスのインスタンスによって、Web サービス要求を作成する処理でメッセージ操作を行うために使用されます。QAnywhere を介して Web サービス要求を送信するためのコネクターアドレスを設定します。これは、サービスバインディングクラスのプロパティ WS_CONNECTOR_ADDRESS を設定することによって行うことができます。接続先 Web サービスの URL を使用して、各 QAnywhere Web サービスコ

ネクターを設定します。複数の URL に配置される Web サービスを必要とするアプリケーションの場合は、各 URL についてコネクターを設定します。

次に例を示します。

```
CurrencyConverterSoap service = new CurrencyConverterSoap( )
service.SetQAManager(mgr);
service.setProperty(
  "WS_CONNECTOR_ADDRESS",
  "ianywhere.connector.currencyconvertor¥¥");
```

アドレス末尾の ¥¥ は、必ず含めてください。

参照

- 「Web サービス用 QAnywhere .NET API リファレンス」325 ページ
- 「クライアント用 QAnywhere .NET API リファレンス」189 ページ

例

モバイル Web サービスを初期化するには、QAManager を作成し、サービスバインディングクラスのインスタンスを作成する必要があります。次に例を示します。

```
// QAnywhere initialization
  QAManager mgr = QAManagerFactory.Instance.CreateQAManager();
  mgr.SetProperty( "CONNECT_PARAMS",
"server=qanywhere;dbf=qanywhere.db;uid=ml_qa_user;pwd=qanywhere" );
  mgr.Open( AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT );
  mgr.Start();

  // Instantiate the web service proxy
  CurrencyConvertorSoap service = new CurrencyConvertorSoap();
  service.SetQAManager( mgr );
  service.SetProperty( "WS_CONNECTOR_ADDRESS", "ianywhere.connector.currencyconvertor¥¥" );
```

CurrencyConvertor サンプルの応答時間は、Web サービスの可用性によって異なります。非同期の Web サービス要求は、モバイル Web サービスアプリケーションが利用不可能になるときがある場合に便利です。この方法では、サービスバインディングクラスでメソッドの呼び出しを行ったときに Web サービス要求が作成され、作成された要求は出力キューに置かれます。メソッドは WSResult を返します。この WSResult は、応答のステータスについて後から問い合わせるときに使用できます。アプリケーションを再起動した後でも使用できます。「非同期の Web サービス要求」105 ページを参照してください。

Java モバイル Web サービスアプリケーションの設定

Java でモバイル Web サービスアプリケーションを作成するには、次の初期化処理を実行する必要があります。

♦ Java 用の QAnywhere とモバイル Web サービスの初期化

1. 次のファイルのロケーションをクラスパスに追加します。デフォルトでは、これらのファイルは *%SQLANY12%¥Java* に配置されています。

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

2. ianywhere.qanywhere.client と ianywhere.qanywhere.ws パッケージをインポートします。

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

3. QAManager オブジェクトを作成します。

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

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

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

> **ヒント**
> 同時実行性の利点を最大限に生かすために、マルチスレッドアプリケーションでは、スレッドごとに QAManager を作成する必要があります。「マルチスレッドでの考慮事項」84 ページを参照してください。

4. QAManager オブジェクトを初期化します。

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

   open メソッドの引数には、メッセージの受信確認方法を示す受信確認モードを指定します。これは、IMPLICIT_ACKNOWLEDGEMENT または EXPLICIT_ACKNOWLEDGEMENT のどちらかにする必要があります。

   モバイル Web サービスによって使用される QAnywhere メッセージは、モバイル Web サービスアプリケーションにアクセスできません。EXPLICIT_ACKNOWLEDGEMENT モードで QAManager を使用する場合は、WSResult の Acknowledge メソッドを使用して、Web サービス要求の結果を含む QAnywhere メッセージの受信確認を行います。このメソッドは、アプリケーションが応答を正常に処理したことを示します。

   受信確認モードの詳細については、次の項を参照してください。

   - WSBase.setQAManager メソッド [QAnywhere Java]625 ページ
   - WSResult.acknowledge メソッド [QAnywhere Java]635 ページ

> **注意**
> QAManager を作成する代わりに、QATransactionalManager を作成できます。「トランザクション指向メッセージングの実装 (Java クライアントの場合)」69 ページを参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

5. サービスバインディングクラスのインスタンスを作成します。

   モバイル Web サービスの WSDL コンパイラーは、Web サービスを定義する WSDL ドキュメントからサービスバインディングクラスを生成します。

   Web サービス要求を作成する処理において、QAManager は、Web サービスバインディングクラスのインスタンスによって、メッセージ操作を行うために使用されます。QAnywhere を介して Web サービス要求を送信するためのコネクターアドレスを設定します。これは、サービスバインディングクラスの WS_CONNECTOR_ADDRESS を設定することによって行うことができます。接続先 Web サービスの URL を使用して、各 QAnywhere Web サービスコネクターを設定します。複数の URL に配置される Web サービスを必要とするアプリケーションの場合は、QAnywhere コネクターはサービスの URL ごとに設定する必要があります。

   次に例を示します。

```
CurrencyConverterSoap service = new CurrencyConverterSoap( );
service.setQAManager(mgr);
service.setProperty("WS_CONNECTOR_ADDRESS", "ianywhere.connector.currencyconvertor¥¥");
```

   アドレス末尾の ¥¥ は、必ず含めてください。

参照

- 「Web サービス用 QAnywhere Java API リファレンス」620 ページ
- 「クライアント用 QAnywhere Java API リファレンス」495 ページ

例

モバイル Web サービスを初期化するには、QAManager を作成し、サービスバインディングクラスのインスタンスを作成する必要があります。次に例を示します。

```
// QAnywhere initialization
  Properties props = new Properties();
  props.put( "CONNECT_PARAMS",
"server=qanywhere;dbf=qanywhere.db;uid=ml_qa_user;pwd=qanywhere" );
  QAManager mgr = QAManagerFactory.getInstance().createQAManager( props );
  mgr.open( AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT );
  mgr.start();

  // Instantiate the web service proxy
  CurrencyConvertorSoap service = new CurrencyConvertorSoap();
  service.setQAManager( mgr );
  service.setProperty( "WS_CONNECTOR_ADDRESS", "ianywhere.connector.currencyconvertor¥¥" );
```

サービスバインディングクラスの複数のインスタンス

サービスバインディングクラスのインスタンスは、QAManager ごとに作成する必要があります。サービスバインディングクラスの複数のインスタンスを使用するモバイル Web サービスアプリケーションである場合は、SetServiceID メソッドを使用してサービス ID を設定することが重要です。次に例を示します。

```
service1.SetServiceID("1")
service2.SetServiceID("2")
```

サービス ID は、サービス名を組み合わせて、Web サービス応答の受信に使用するキュー名を形成します。指定するサービスの各インスタンスには、ユニークなサービス ID を設定することが重要です。これにより、特定のインスタンスが、サービスの別のインスタンスによって作成された要求に対する応答を受け取ることはありません。サービス ID が設定されていないと、デフォルトの "" になります。キュー名では複数のアプリケーションにわたって一時的にメッセージストア内のメッセージを保持するため、サービス ID は、同じサービスを使用する複数のアプリケーションが互いに競合しないようにするためにも重要です。

モバイル Web サービスアプリケーションのコンパイルと実行

ランタイムライブラリ

Java のランタイムライブラリは、*%SQLANY12%¥Java* にある *iawsrt.jar* です。

C# のランタイムライブラリは、*iAnywhere.QAnywhere.WS.dll* で、次のフォルダーにあります。

- .NET Framework 2.0: *%SQLANY12%¥Assembly¥V2*
- .NET Compact Framework 2.0: *%SQLANY12%¥ce¥Assembly¥V2*

次の項では、モバイル Web サービスアプリケーションをコンパイルして実行するために必要なファイルについて説明します。

必要なランタイムライブラリ (Java)

クラスパスに次のファイルを含めます。これらのファイルは *%SQLANY12%¥Java* にあります。

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

必要なランタイムライブラリ (.NET)

SQL Anywhere 12 をインストールすると、次のファイルは Global Assembly Cache に自動的に登録されます。

- *iAnywhere.QAnywhere.Client.dll*
- *iAnywhere.QAnywhere.WS.dll*

モバイル Web サービスの停止

モバイル Web サービスアプリケーションは、QAManager を閉じることによって、順次停止されます。次に例を示します。

```
// QAnywhere finalization in C#:
mgr.Stop();
mgr.Close();
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
// QAnywhere finalization in Java:
mgr.stop();
mgr.close();
```

Web サービス要求

モバイル Web サービスアプリケーションで Web サービス要求を作成するには、2 つの基本的な方法があります。

- **同期**　「同期の Web サービス要求」を参照してください。
- **非同期**　「非同期の Web サービス要求」を参照してください。

同期の Web サービス要求

同期の Web サービス要求は、アプリケーションがネットワークに接続されている場合に使用します。この方法では、サービスバインディングクラスでメソッドの呼び出しを行ったときに、Web サービス要求が作成されます。そして、サーバーからの Web サービス応答を受信した場合にのみ、結果が返されます。

例

次の例では、USD から CAD への為替レートを取得する要求を作成します。

```
//C#
double r = service.ConversionRate( Currency.USD, Currency.CAD );

// Java
double r = service.conversionRate( NET.webserviceX.Currency.USD,
NET.webserviceX.Currency.CAD );
```

非同期の Web サービス要求

非同期の Web サービス要求は、モバイル Web サービスアプリケーションがまれにしかネットワークに接続しない場合に有用です。この方法では、サービスバインディングクラスでメソッドの呼び出しを行ったときに Web サービス要求が作成され、作成された要求は出力キューに置かれます。メソッドは WSResult を返します。この WSResult は、応答のステータスについて後から問い合わせるときに使用できます。アプリケーションを再起動した後でも使用できます。

次の例では、USD から CAD への為替レートを取得する非同期の要求を作成します。

```
// C#
WSResult r = service.AsyncConversionRate( Currency.USD, Currency.CAD );

// Get the request ID.  Save it for later use if necessary.
string reqID = r.GetRequestID();

// Later: get the response for the specified request ID
WSResult r = service.GetResult( reqID );
```

```
if( r.GetStatus() == WSStatus.STATUS_RESULT_AVAILABLE ) {
  Console.WriteLine( "The conversion rate is " + r.GetDoubleValue( "ConversionRateResult" ) );
} else {
  Console.WriteLine( "Response not available" );
}
// Java
WSResult r = service.asyncConversionRate( NET.webserviceX.Currency.USD,
NET.webserviceX.Currency.CAD );

// Get the request ID.  Save it for later use if necessary.
String reqID = r.getRequestID();

// Later: get the response for the specified request ID
WSResult r = service.getResult( reqID );
if( r.getStatus() == WSStatus.STATUS_RESULT_AVAILABLE ) {
    System.out.println( "The conversion rate is " + r.getDoubleValue( "ConversionRateResult" ) );
} else {
    System.out.println( "Response not available" );
}
```

Web サービス要求に対する応答を利用できるときは、WSListener を使用して、非同期のコールバックを取得することもできます。次に例を示します。

```
// C#
// Make a request to get the USD to CAD exchange rate
WSResult r = service.AsyncConversionRate( Currency.USD, Currency.CAD );

// Register a listener for the result
service.SetListener( r.GetRequestID(), new CurrencyConvertorListener() );

// Java
// Make a request to get the USD to CAD exchange rate
WSResult r = service.asyncConversionRate( NET.webserviceX.Currency.USD,
NET.webserviceX.Currency.CAD );

// Register a listener for the result
service.setListener( r.getRequestID(), new CurrencyConvertorListener() );
```

WSListener インターフェイスは、非同期イベントを扱うための 2 つのメソッドを定義します。

- **OnResult**　OnResult メソッドは、Web サービス要求に対する応答を処理するために実装されます。Web サービス要求の結果を表す WSResult オブジェクトが渡されます。

- **OnException**　OnException メソッドは、Web サービス要求への応答を処理しているときに発生したエラーを扱うために実装されます。WSException オブジェクトと WSResult オブジェクトが渡されます。WSException オブジェクトには、発生したエラーに関する情報が含まれます。WSResult オブジェクトは、応答に対応する要求 ID を取得するために使用されます。

```
// C#
class CurrencyConvertorListener : WSListener
{
  public CurrencyConvertorListener() {
  }

  public void OnResult( WSResult r ) {
   try {
       USDToCAD._statusMessage = "USD to CAD currency exchange rate: " +
r.GetDoubleValue( "ConversionRateResult" );
   } catch( Exception exc ) {
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
        USDToCAD._statusMessage = "Request " + r.GetRequestID() + " failed: " + exc.Message;
    }
   }

   public void OnException( WSException exc, WSResult r ) {
    USDToCAD._statusMessage = "Request " + r.GetRequestID() + " failed: " + exc.Message;
   }
}
// Java
private class CurrencyConvertorListener implements WSListener
{
  public CurrencyConvertorListener() {
  }

  public void onResult( WSResult r ) {
   try {
        USDToCAD._statusMessage = "USD to CAD currency exchange rate: " +
r.getDoubleValue( "ConversionRateResult" );
   } catch( Exception exc ) {
        USDToCAD._statusMessage = "Request " + r.getRequestID() + " failed: " + exc.getMessage();
    }
    }

   public void onException( WSException exc, WSResult r ) {
        USDToCAD._statusMessage = "Request " + r.getRequestID() + " failed: " + exc.getMessage();
    }
}
```

モバイル Web サービスの例

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

Global Weather Web サービス

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

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:soap="http://
schemas.xmlsoap.org/wsdl/soap/" xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns="http://www.webserviceX.NET"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:mime="http://schemas.xmlsoap.org/
wsdl/mime/" targetNamespace="http://www.webserviceX.NET" xmlns:wsdl="http://
schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://www.webserviceX.NET">
      <s:element name="GetWeather">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="CityName" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="CountryName" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
```

```
    <s:element name="GetWeatherResponse">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="GetWeatherResult" type="s:string" />
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="GetCitiesByCountry">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="CountryName" type="s:string" />
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="GetCitiesByCountryResponse">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="GetCitiesByCountryResult" type="s:string" />
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="string" nillable="true" type="s:string" />
  </s:schema>
</wsdl:types>
<wsdl:message name="GetWeatherSoapIn">
  <wsdl:part name="parameters" element="tns:GetWeather" />
</wsdl:message>
<wsdl:message name="GetWeatherSoapOut">
  <wsdl:part name="parameters" element="tns:GetWeatherResponse" />
</wsdl:message>
<wsdl:message name="GetCitiesByCountrySoapIn">
  <wsdl:part name="parameters" element="tns:GetCitiesByCountry" />
</wsdl:message>
<wsdl:message name="GetCitiesByCountrySoapOut">
  <wsdl:part name="parameters" element="tns:GetCitiesByCountryResponse" />
</wsdl:message>
<wsdl:message name="GetWeatherHttpGetIn">
  <wsdl:part name="CityName" type="s:string" />
  <wsdl:part name="CountryName" type="s:string" />
</wsdl:message>
<wsdl:message name="GetWeatherHttpGetOut">
  <wsdl:part name="Body" element="tns:string" />
</wsdl:message>
<wsdl:message name="GetCitiesByCountryHttpGetIn">
  <wsdl:part name="CountryName" type="s:string" />
</wsdl:message>
<wsdl:message name="GetCitiesByCountryHttpGetOut">
  <wsdl:part name="Body" element="tns:string" />
</wsdl:message>
<wsdl:message name="GetWeatherHttpPostIn">
  <wsdl:part name="CityName" type="s:string" />
  <wsdl:part name="CountryName" type="s:string" />
</wsdl:message>
<wsdl:message name="GetWeatherHttpPostOut">
  <wsdl:part name="Body" element="tns:string" />
</wsdl:message>
<wsdl:message name="GetCitiesByCountryHttpPostIn">
  <wsdl:part name="CountryName" type="s:string" />
</wsdl:message>
<wsdl:message name="GetCitiesByCountryHttpPostOut">
  <wsdl:part name="Body" element="tns:string" />
</wsdl:message>
<wsdl:portType name="GlobalWeatherSoap">
  <wsdl:operation name="GetWeather">
```

```
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get weather report for all major cities
around the world.</documentation>
      <wsdl:input message="tns:GetWeatherSoapIn" />
      <wsdl:output message="tns:GetWeatherSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="GetCitiesByCountry">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get all major cities by country
name(full / part).</documentation>
      <wsdl:input message="tns:GetCitiesByCountrySoapIn" />
      <wsdl:output message="tns:GetCitiesByCountrySoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="GlobalWeatherHttpGet">
    <wsdl:operation name="GetWeather">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get weather report for all major cities
around the world.</documentation>
      <wsdl:input message="tns:GetWeatherHttpGetIn" />
      <wsdl:output message="tns:GetWeatherHttpGetOut" />
    </wsdl:operation>
    <wsdl:operation name="GetCitiesByCountry">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get all major cities by country
name(full / part).</documentation>
      <wsdl:input message="tns:GetCitiesByCountryHttpGetIn" />
      <wsdl:output message="tns:GetCitiesByCountryHttpGetOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="GlobalWeatherHttpPost">
    <wsdl:operation name="GetWeather">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get weather report for all major cities
around the world.</documentation>
      <wsdl:input message="tns:GetWeatherHttpPostIn" />
      <wsdl:output message="tns:GetWeatherHttpPostOut" />
    </wsdl:operation>
    <wsdl:operation name="GetCitiesByCountry">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get all major cities by country
name(full / part).</documentation>
      <wsdl:input message="tns:GetCitiesByCountryHttpPostIn" />
      <wsdl:output message="tns:GetCitiesByCountryHttpPostOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="GlobalWeatherSoap" type="tns:GlobalWeatherSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
    <wsdl:operation name="GetWeather">
      <soap:operation soapAction="http://www.webserviceX.NET/GetWeather" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetCitiesByCountry">
      <soap:operation soapAction="http://www.webserviceX.NET/GetCitiesByCountry"
style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="GlobalWeatherHttpGet" type="tns:GlobalWeatherHttpGet">
    <http:binding verb="GET" />
```

```
    <wsdl:operation name="GetWeather">
      <http:operation location="/GetWeather" />
      <wsdl:input>
        <http:urlEncoded />
      </wsdl:input>
      <wsdl:output>
        <mime:mimeXml part="Body" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetCitiesByCountry">
      <http:operation location="/GetCitiesByCountry" />
      <wsdl:input>
        <http:urlEncoded />
      </wsdl:input>
      <wsdl:output>
        <mime:mimeXml part="Body" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="GlobalWeatherHttpPost" type="tns:GlobalWeatherHttpPost">
    <http:binding verb="POST" />
    <wsdl:operation name="GetWeather">
      <http:operation location="/GetWeather" />
      <wsdl:input>
        <mime:content type="application/x-www-form-urlencoded" />
      </wsdl:input>
      <wsdl:output>
        <mime:mimeXml part="Body" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetCitiesByCountry">
      <http:operation location="/GetCitiesByCountry" />
      <wsdl:input>
        <mime:content type="application/x-www-form-urlencoded" />
      </wsdl:input>
      <wsdl:output>
        <mime:mimeXml part="Body" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="GlobalWeather">
    <wsdl:port name="GlobalWeatherSoap" binding="tns:GlobalWeatherSoap">
      <soap:address location="http://www.webservicex.net/globalweather.asmx" />
    </wsdl:port>
    <wsdl:port name="GlobalWeatherHttpGet" binding="tns:GlobalWeatherHttpGet">
      <http:address location="http://www.webservicex.net/globalweather.asmx" />
    </wsdl:port>
    <wsdl:port name="GlobalWeatherHttpPost" binding="tns:GlobalWeatherHttpPost">
      <http:address location="http://www.webservicex.net/globalweather.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

プロキシクラスの生成

Global Weather Web サービスにアクセスするモバイル アプリケーションを作成するには、まず、QAnywhere WSDL コンパイラーを実行します。コンパイラーにより、Global Weather Web サービスの要求を作成するためにアプリケーションで使用できるプロキシクラスが生成されます。この例では、アプリケーションは Java で記述されています。

```
wsdlc -l java globalweather.wsdl
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

このコマンドは、プロキシクラス *GlobalWeatherSoap.java* を生成します。このプロキシクラスは、現在のディレクトリの *NET¥webserviceX* サブフォルダーに格納されます。このプロキシクラスは、アプリケーション用のサービスバインディングクラスです。次に、*GlobalWeatherSoap.java* の内容を示します。

```
/*
 * GlobalWeatherSoap.java
 *
 * Generated by the iAnywhere WSDL Compiler Version 10.0.1.3415
 * Do not edit this file.
 */

package NET.webserviceX;

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

import java.io.*;

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

public class GlobalWeatherSoap extends ianywhere.qanywhere.ws.WSBase
{

    public GlobalWeatherSoap(String iniFile) throws WSException
    {
        super(iniFile);
        init();
    }

    public GlobalWeatherSoap() throws WSException
    {
        init();
    }

    public void init()
    {

        setServiceName("GlobalWeather");
    }


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

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

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

```
2001/XMLSchema",true,true);
            writeSOAPFooter( hd, "GetWeather" );

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

            WSResult wsResult = invokeWait( qaRequestMsg );

            returnValue = wsResult.getStringValue(responsePartName);

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

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

            hd.setResult( new StreamResult( sw ) );

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

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

            WSResult wsResult = invoke( qaRequestMsg );

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


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

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

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

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

            WSResult wsResult = invokeWait( qaRequestMsg );

            returnValue = wsResult.getStringValue(responsePartName);

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

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

            hd.setResult( new StreamResult( sw ) );

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

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

            WSResult wsResult = invoke( qaRequestMsg );

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

モバイル Web サービスアプリケーションの記述

次に、Web サービス要求の作成や結果の処理にサービスバインディングクラスを使用するアプリケーションを記述します。次に、2 つのアプリケーションを示します。いずれも、オフラインの状態で Web サービス要求を作成し、接続が有効なときに結果を処理します。

最初のアプリケーションである RequestWeather は、Global Weather Web サービスの要求を作成し、その ID を表示します。*RequestWeather.java* ファイルに次のコードをコピーします。

```
import ianywhere.qanywhere.client.*;
import ianywhere.qanywhere.ws.*;
import NET.webserviceX.GlobalWeatherSoap;

class RequestWeather
{
    public static void main( String [] args ) {
    try {
        // QAnywhere initialization
        QAManager mgr = QAManagerFactory.getInstance().createQAManager();
        mgr.open( AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT );
        mgr.start();
```

```
        // Instantiate the web service proxy
        GlobalWeatherSoap service = new GlobalWeatherSoap();
        service.setQAManager( mgr );
        service.setProperty( "WS_CONNECTOR_ADDRESS", "ianywhere.connector.globalweather¥¥" );

        // Make a request to get weather for Beijing
        WSResult r = service.asyncGetWeather( "Beijing", "China" );

        // Display the request ID so that it can be used by ShowWeather
        System.out.println( "Request ID: " + r.getRequestID() );

        // QAnywhere finalization
        mgr.stop();
        mgr.close();

    } catch( Exception exc ) {
        System.out.println( exc.getMessage() );
    }
    }
}
```

2 番目のアプリケーションである ShowWeather は、指定された要求 ID の気象情報を表示します。*ShowWeather.java* ファイルに次のコードをコピーします。

```
import ianywhere.qanywhere.client.*;
import ianywhere.qanywhere.ws.*;
import NET.webserviceX.GlobalWeatherSoap;

class ShowWeather
{
  public static void main( String [] args ) {
  try {
    // QAnywhere initialization
    QAManager mgr = QAManagerFactory.getInstance().createQAManager();
    mgr.open( AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT );
    mgr.start();

    // Instantiate the web service proxy
    GlobalWeatherSoap service = new GlobalWeatherSoap();
    service.setQAManager( mgr );

    // Get the response for the specified request ID
    WSResult r = service.getResult( args[0] );
    if( r.getStatus() == WSStatus.STATUS_RESULT_AVAILABLE ) {
  System.out.println( "The weather is " + r.getStringValue( "GetWeatherResult" ) );
  r.acknowledge();
    } else {
  System.out.println( "Response not available" );
    }

    // QAnywhere finalization
    mgr.stop();
    mgr.close();

 } catch( Exception exc ) {
    System.out.println( exc.getMessage() );
 }
   }
}
```

アプリケーションとサービスバインディングクラスをコンパイルします。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
javac -classpath ".;%sqlany12%¥java¥iawsrt.jar;%sqlany12%¥java¥qaclient.jar"
NET.webserviceX.GlobalWeatherSoap.java RequestWeather.java
javac -classpath ".;%sqlany12%¥java¥iawsrt.jar;%sqlany12%¥java¥qaclient.jar"
NET.webserviceX.GlobalWeatherSoap.java ShowWeather.java
```

QAnywhere メッセージストアの作成と QAnywhere Agent の起動

作成したモバイル Web サービスアプリケーションは、各モバイルデバイス上にクライアントメッセージストアを要求します。サーバーメッセージストアも必要ですが、この例では、QAnywhere のサンプルのサーバーメッセージストアを使用します。

クライアントメッセージストアを作成するには、dbinit ユーティリティを使用して SQL Anywhere データベースを作成し、QAnywhere Agent を実行してクライアントメッセージストアとして設定します。

```
dbinit -i qanywhere.db
qaagent -q -si -c "dbf=qanywhere.db"
```

QAnywhere Agent を起動し、クライアントメッセージストアに接続します。

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

QAnywhere サーバーを起動します。

```
mlsrv12 -m -zu+ -c "dsn=QAnywhere 12 Demo;uid=ml_server;pwd=sql;start=dbsrv12" -v+ -ot
qanyserv.mls
```

♦ Web サービスコネクターの作成

GetWeather Web サービスに送信される QAnywhere メッセージを受信し、メッセージが到着したときに Web サービス呼び出しを行い、応答を送信元のクライアントに送り返すための、Web サービスコネクターを作成します。

1. Sybase Central を開き、**[接続]** » **[QAnywhere 12 に接続]** を選択します。
2. **[ユーザー ID]** フィールドに **ml_server** と入力します。
3. **[パスワード]** フィールドに **sql** と入力します。
4. **[ODBC データソース名]** をクリックし、**QAnywhere 12 Demo** と入力します。
5. **[接続]** をクリックします。
6. **[ファイル]** » **[新規]** » **[コネクター]** をクリックします。
7. **[Web サービス]** をクリックします。**[次へ]** をクリックします。
8. **[コネクター名]** フィールドに **ianywhere.connector.globalweather** と入力します。**[次へ]** をクリックします。
9. **[URL]** フィールドに **http://www.webservicex.net/globalweather.asmx** と入力します。**[完了]** をクリックします。

Web サービスの使用

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

```
java -classpath ".;%sqlany12%¥java¥iawsrt.jar;%sqlany12%¥java¥qaclient.jar" RequestWeather
```

要求 ID が返されます。

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

```
java -classpath ".;%sqlany12%¥java¥iawsrt.jar;%sqlany12%¥java¥qaclient.jar" ShowWeather
REQ123123123
```

詳細な天気予報が返されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

QAnywhere アプリケーションの配備

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

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

- 「SQL Anywhere Mobile Link クライアントの配備」『Mobile Link サーバー管理』の Mobile Link 同期クライアント、リスナー、セキュリティ (任意) に関する各項に記載されたすべてのファイル。

  Mobile Link Listener ファイルは、Push 通知を使用している場合 (デフォルト設定) のみ必要です。

- dbeng12 または dbsrv12 ファイル (「データベースサーバーの配備」『SQL Anywhere サーバー プログラミング』より)。

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

Windows アプリケーション

すべてのディレクトリは、*%SQLANY12%* を基準とした相対ディレクトリです。

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

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

クライアント API	Windows ファイル
C++	● *bin32¥qany12.dll* ● *bin32¥qaagent.exe* ● *bin32¥qastop.exe*
Java	● *bin32¥qaagent.exe* ● *bin32¥qastop.exe* ● *java¥qaclient.jar* ● *java¥jodbc.jar* モバイル Web サービスアプリケーションについては、次のファイルも必要です。 ● *java¥iawsrt.jar* ● *java¥jaxrpc.jar*

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

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

クライアント API	Windows ファイル
Java	● *bin32¥qauagent.exe* ● *bin32¥qastop.exe* ● *bin32¥qadbiuljni12.dll* ● *java¥qaclient.jar* モバイル Web サービスアプリケーションについては、次のファイルも必要です。 ● *java¥iawsrt.jar* ● *java¥jaxrpc.jar*
.NET	● *bin32¥qazlib.dll* ● *bin32¥qauagent.exe* ● *bin32¥qastop.exe* ● *assembly¥v2¥iAnywhere.QAnywhere.Client.dll* ● *assembly¥v2¥iAnywhere.QAnywhere.Resources.dll* ● *ultralite¥ultralite.NET¥assembly¥v2¥iAnywhere.Data.UltraLite.dll* モバイル Web サービスアプリケーションについては、次のファイルも必要です。 ● *Assembly¥v2¥iAnywhere.QAnywhere.WS.dll*

Ultra Light メッセージストアを作成するときは、Ultra Light データベース作成ユーティリティを使用して udb データベースファイルを作成し、それから QAnywhere Ultra Light Agent の -si オプションを使用してデータベースを初期化する必要があります。「Ultra Light データベース初期化ユーティリティ (ulinit)」『Ultra Light データベース管理とリファレンス』と 「qauagent ユーティリティ」738 ページを参照してください。

Windows Mobile アプリケーション

すべてのディレクトリは、*%SQLANY12%* を基準とした相対ディレクトリです。

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

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

クライアント API	Windows Mobile ファイル
C++	● *ce¥arm.50¥qany12.dll* ● *ce¥arm.50¥qaagent.exe* ● *ce¥arm.50¥qastop.exe*
Java	● *ce¥arm.50¥qaagent.exe* ● *ce¥arm.50¥qastop.exe* ● *java¥qaclient.jar* ● *java¥jodbc.jar* モバイル Web サービスアプリケーションについては、次のファイルも必要です。 ● *java¥iawsrt.jar* ● *java¥jaxrpc.jar*
.NET	● *ce¥arm.50¥qazlib.dll* ● *ce¥arm.50¥qaagent.exe* ● *ce¥arm.50¥qastop.exe* ● *ce¥assembly¥v2¥iAnywhere.QAnywhere.Client.dll* ● *ce¥assembly¥v2¥iAnywhere.QAnywhere.Resources.dll* ● *ce¥assembly¥v2¥iAnywhere.Data.SQLAnywhere.dll* モバイル Web サービスアプリケーションについては、次のファイルも必要です。 ● *ce¥Assembly¥v2¥iAnywhere.QAnywhere.WS.dll*

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

クライアント API	Windows Mobile ファイル
Java	● *ce¥arm.50¥qauagent.exe* ● *ce¥arm.50¥qastop.exe* ● *ce¥arm.50¥qadbiuljni12.dll* ● *java¥qaclient.jar* モバイル Web サービスアプリケーションについては、次のファイルも必要です。 ● *java¥iawsrt.jar* ● *java¥jaxrpc.jar*
.NET	● *ce¥arm.50¥qazlib.dll* ● *ce¥arm.50¥qauagent.exe* ● *ce¥arm.50¥qastop.exe* ● *ce¥assembly¥v2¥iAnywhere.QAnywhere.Client.dll* ● *ce¥assembly¥v2¥iAnywhere.QAnywhere.Resources.dll* ● *ultralite¥ultralite.NET¥ce¥assembly¥v2¥iAnywhere.Data.UltraLite.dll* モバイル Web サービスアプリケーションについては、次のファイルも必要です。 ● *ce¥Assembly¥v2¥iAnywhere.QAnywhere.WS.dll*

Ultra Light メッセージストアを作成するときは、Ultra Light データベース作成ユーティリティを使用してデータベースファイルを作成し、それから QAnywhere Ultra Light Agent を使用してデータベースを初期化する必要があります。 「Ultra Light データベース初期化ユーティリティ (ulinit)」『Ultra Light データベース管理とリファレンス』と「qauagent ユーティリティ」738 ページを参照してください。

QAnywhere .NET API DLL の登録

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

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

セキュリティ保護されたメッセージングアプリケーション

以降の項では、メッセージングアプリケーションがセキュリティ保護されていることを確認する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

クライアントメッセージストアをセキュリティで保護するには、次の方法があります。

●デフォルトのパスワードを変更する。

●メッセージストアの内容を暗号化する。

例

まず、暗号化キーを使用して SQL Anywhere データベースを作成します。

```
dbinit mystore.db -i -s -ek some_phrase
```

小型デバイスの場合は、-i オプションと -s オプションが最適です。強力な暗号化を使用する場合は、-ek オプションで暗号化キーを指定します。「初期化ユーティリティ (dbinit)」『SQL Anywhere サーバー データベース管理』を参照してください。

次に、データベースをクライアントメッセージストアとして初期化します。

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

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

```
CREATE USER secure_user IDENTIFIED BY secure_password
GRANT MEMBERSHIP IN GROUP ml_qa_user_group TO secure_user
GRANT REMOTE DBA TO secure_user
REVOKE CONNECT FROM ml_qa_user
ALTER USER DBA IDENTIFIED BY new_dba_password
COMMIT
```

注意
すべての QAnywhere ユーザーが ml_qa_user_group に属しており、REMOTE DBA 権限を持っていることが必要です。

次に、セキュリティ保護された DBA ユーザーとして、QAnywhere Agent を起動します。

```
qaagent -id mystore -c "dbf=mystore.db;dbkey=some_phrase;uid=secure_user;pwd=secure_password"
```

参照

- 「クライアントメッセージストアのパスワード管理」
- 「クライアントメッセージストアの暗号化」

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ー ID のパスワードは変更しなければなりません。すべての SQL Anywhere データベースには、デフォルトのユーザー 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ー ID ml_qa_user とデフォルトのパスワード qanywhere が作成されます。これらのパスワードを変更するには、GRANT 文を使用します。

「パスワードの変更」『SQL Anywhere サーバー データベース管理』を参照してください。

クライアントメッセージストアの暗号化

次のコマンドを使用すると、クライアントメッセージストアを作成時に暗号化できます。

```
dbinit -i -s -ek encryption-key database-file
```

小型デバイスにデータベースを作成する場合は、-i オプションと -s オプションが最適です。暗号化キーを使用してメッセージストアを初期化した場合は、暗号化されたメッセージストアに対してデータベースサーバーを起動するために、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ーを起動します。

```
qaagent -si -c "DBF=database-file;DBKEY=encryption-key"
```

これで、アプリケーションは、QAnywhere API を介して、暗号化されたメッセージストアにアクセスできるようになります。メッセージストアを管理するためのデータベースサーバーはすでに実行されているため、アプリケーションが暗号化キーを指定する必要はありません。

QAnywhere Agent が実行されていないときに、アプリケーションが暗号化されたメッセージストアにアクセスする必要がある場合、QAnywhere API は、QAnywhere Manager 初期化ファイルに指定された接続パラメーターを使用して自動的にデータベースサーバーを起動します。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ーを起動するには、次のように、データベース接続パラメーター内に暗号化キーを指定する必要があります。

```
CONNECT_PARAMS=DBF=database-file;DBKEY=encryption-key
```

参照

- 「データベースの暗号化と復号化」『SQL Anywhere サーバー データベース管理』
- 「初期化ユーティリティ (dbinit)」『SQL Anywhere サーバー データベース管理』
- QAnywhere Agent：「-c qaagent オプション」 715 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

通信ストリームの暗号化

qaagent -x オプションを使用すると、QAnywhere Agent が Mobile Link サーバーと通信するときに使用するセキュリティ保護された通信ストリームを指定できます。また、サーバー側証明書を使用したサーバー認証を実装し、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。

Mobile Link サーバーのトランスポートレイヤーセキュリティの設定も行う必要があります。

注意
別途ライセンスが必要な必須コンポーネント

ECC 暗号化と FIPS 認定の暗号化には、別途ライセンスが必要です。強力な暗号化テクノロジはすべて、輸出規制対象品目です。

「別途ライセンスが必要なコンポーネント」『SQL Anywhere 12 紹介』を参照してください。

例

次の例では、QAnywhere Agent と Mobile Link サーバーとの間でセキュリティ保護された通信ストリームを確立する方法を示します。この例では、SQL Anywhere セキュリティオプションと一緒にインストールされるサンプルの ID ファイルを使用しています。

RSA を使用してセキュリティ保護された TCP/IP：

```
mlsrv12 -x tls(tls_type=rsa;identity=rsaserver.id;identity_password=test)
qaagent -x tls(tls_type=rsa;trusted_certificate=rsaroot.crt)
```

ECC を使用してセキュリティ保護された TCP/IP：

```
mlsrv12 -x tls(tls_type=ecc;identity=eccserver.id;identity_password=test)
qaagent -x tls(tls_type=ecc;trusted_certificate=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)：

```
mlsrv12 -x https(identity=rsaserver.id;identity_password=test)
qaagent -x https(trusted_certificate=rsaroot.crt)
```

参照

- 「-x qaagent オプション」736 ページ
- 「Mobile Link サーバー／クライアント通信の暗号化」『SQL Anywhere サーバー データベース管理』

Mobile Link に対するパスワード認証

リモートデバイスとサーバー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、デバイスのユーザーを認証してサーバーと通信できるようにするのが普通です。

これを行うには、クライアントメッセージストア用の Mobile Link ユーザー名を作成し、サーバーメッセージストアに登録します。

参照

- 「-mu qaagent オプション」723 ページ
- 「-mp qaagent オプション」722 ページ
- 「Mobile Link ユーザー」『Mobile Link クライアント管理』

サーバー管理要求のセキュリティ保護

サーバー管理要求は、パスワードを使用して保護することができます。メッセージ文字列プロパティの ias_ServerPassword は、サーバーのパスワードを指定します。サーバーのパスワードは、ianywhere.qa.server.password.e プロパティを使用して設定されます。このプロパティが設定されていない場合、パスワードは QAnywhere です。

サーバーのパスワードはテキストとして転送されます。サーバーのパスワードを必要とするサーバー管理要求を送信する場合は、暗号化された通信ストリームを使用してください。

ianywhere.qa.server.password.e プロパティの詳細については、「サーバープロパティ」768 ページを参照してください。

Mobile Link ユーザー認証ユーティリティを使用したユーザーの追加

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

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

-zu+ オプション付きで mlsrv12 を実行して新しいユーザーを追加することもできますが、セキュリティ上問題がある場合はこのオプションを使用しないでください。-zu+ オプション付きで mlsrv12 を使用すると、すべての新しいユーザーが、最初の同期時に統合データベースに追加されます。これは、認識されていないユーザーが認証を行わずに追加されることを意味します。

参照

- 「Mobile Link ユーザー認証ユーティリティ (mluser)」『Mobile Link サーバー管理』
- 「-zu mlsrv12 オプション」『Mobile Link サーバー管理』

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

Relay Server を使用する場合のセキュリティ

Relay Server を使用する場合は、Relay Server の設定ファイルのオプションを設定して、必要なセキュリティレベルを制御します。

Web サーバーとのセキュリティ保護された通信ストリーム (信頼できる証明書付きの HTTPS など) を設定し、QAnywhere Agent と Web サーバーの間の通信のセキュリティを確保します。

Relay Server と Mobile Link の間でセキュリティ保護された通信ストリームを設定する方法については、Relay Server のマニュアルを参照してください。

参照

- 「Relay Server 設定ファイル」『Relay Server』
- 「-x qaagent オプション」736 ページ
- 「Relay Server の概要」『Relay Server』

サーバーメッセージストアの管理

以降の項では、サーバーメッセージストアの管理方法について説明します。

転送ルール

転送ルールでは、メッセージ転送のタイミングと転送するメッセージを指定できます。転送ルールは、サーバーとクライアントの両方に対して指定できます。

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

サーバー側の転送ルールは、サーバー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。サーバー側の転送ルールは Mobile Link サーバー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。

サーバー側の転送ルールは、次の 3 つの方法で設定できます。

- サーバー管理要求を作成して転送ルールを設定する。

  「サーバー管理要求での転送ルールの指定」169 ページを参照してください。

- Sybase Central を使用して転送ルールを設定する。

  「Sybase Central での転送ルールの指定」128 ページを参照してください。

- サーバー側の転送ルールファイルを作成して、Mobile Link サーバーの起動時に指定する。この方法は推奨されなくなりました。

  「転送ルールファイルによるサーバー側の転送ルールの指定 (旧式)」784 ページを参照してください。

サーバー側の転送ルール

サーバー側の転送ルールは、サーバー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。サーバー側の転送ルールは Mobile Link サーバー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。

サーバー側の転送ルールは、次の 3 つの方法で設定できます。

- サーバー管理要求を作成して転送ルールを設定する。

  「サーバー管理要求での転送ルールの指定」169 ページを参照してください。

- Sybase Central を使用して転送ルールを設定する。

  「Sybase Central での転送ルールの指定」128 ページを参照してください。

- サーバー側の転送ルールファイルを作成して、Mobile Link サーバーの起動時に指定する。このメソッドは推奨されなくなりました。

「転送ルールファイルによるサーバー側の転送ルールの指定 (旧式)」784 ページを参照してください。

Sybase Central での転送ルールの指定

♦ **サーバー側のデフォルトの転送ルールの指定**

1. Sybase Central を起動します。
   - **[スタート]** » **[プログラム]** » **[SQL Anywhere 12]** » **[管理ツール]** » **[Sybase Central]** をクリックします。
   - **[接続]** » **[QAnywhere 12 に接続]** をクリックします。
   - **[ODBC データソース名]** または **[ODBC データソースファイル]** を指定し、必要に応じて **[ユーザー ID]** と **[パスワード]** を指定します。**[接続]** をクリックします。
2. **[サーバーメッセージストア]** の下で、データソース名をクリックします。
3. **[ファイル]** » **[プロパティ]** をクリックします。
4. **[転送ルール]** タブを開き、**[デフォルトの転送ルールをカスタマイズする]** をクリックします。
5. **[新規]** をクリックして、ルールを追加します。
6. 条件をテキストフィールドに入力するか、ドロップダウンリストから **[メッセージ変数]** または **[ステータス定数]** を選択して、条件を追加します。
7. **[OK]** をクリックして終了します。

サーバー管理要求でのサーバー側の転送ルールの指定

サーバー管理要求を使用して、すべてのユーザーに適用されるサーバー側のデフォルトの転送ルールを指定したり、クライアントごとに転送ルールを指定したりできます。

サーバーのデフォルトの転送ルールを指定するには、クライアント ianywhere.server.defaultClient の ianywhere.qa.server.rules プロパティを設定します。クライアントの場合は、ianywhere.qa.server.rules プロパティを使用してサーバーの転送ルールを指定します。

参照

- 「サーバー管理要求での転送ルールの指定」169 ページ

メッセージのアーカイブの管理

アーカイブメッセージストアはサーバーメッセージストアと共存するテーブルのセットで、削除待機中のすべてのメッセージがここに格納されます。定期的に実行されるシステムプロセスでは、サーバーメッセージストアにあるメッセージのうち最終ステータスに到達したものすべてを

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

削除し、アーカイブメッセージストアに挿入することで、メッセージストア間でのメッセージ転送を行います。

メッセージは、サーバー削除ルールによって削除されるまで、アーカイブメッセージストアに残ります。アーカイブメッセージストアを使用することにより、同期中にフィルターする必要のあるメッセージの量が最小限に抑えられるため、サーバーメッセージストアのパフォーマンスが向上します。

参照

- 「削除ルール」46 ページ

サーバー管理要求を使用したサーバーメッセージストアの管理

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

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

- 「クライアントの転送ルールの更新」156 ページ
- 「メッセージのキャンセル」157 ページ
- 「メッセージの削除」158 ページ

サーバー管理要求の使用の詳細については、次を参照してください。

- 「サーバー管理要求を使用したサーバーメッセージストアの管理」156 ページ
- 「サーバー管理要求の記述」154 ページ
- 「サーバー管理要求のリファレンス」704 ページ

クライアントメッセージストアの管理

以降の項では、クライアントメッセージストアの管理方法について説明します。

QAnywhere クライアントのモニター

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

サーバー管理要求を使用して、現在サーバー上にあるクライアントのリストを取得できます。このリストには、サーバーに登録されているクライアントが含まれます。リモートクライアント、開いているコネクター、送信先エイリアスも含まれます。

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

参照

- 「QAnywhere クライアントのモニター」175 ページ

クライアントプロパティのモニター

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

サーバー管理要求を使用して、クライアントに設定されているプロパティの内容を確認できます。要求の応答によってリストされるプロパティは、クライアントに設定されているプロパティだけです。デフォルトのプロパティは含まれません。

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

参照

- 「プロパティのモニター」176 ページ

クライアントメッセージストアプロパティの管理

クライアントメッセージストアのプロパティは、クライアントメッセージストアごとにクライアントアプリケーションで設定できます。

クライアントメッセージストアのプロパティは、クライアントに送信されるメッセージをフィルターするために転送ルールで使用したり、追加するメッセージを決定するために削除ルールで使用することができます。

クライアントメッセージストアのプロパティをサーバー管理メッセージで指定し、サーバーメッセージストアに格納することもできます。

参照

- 「アプリケーションでのクライアントメッセージストアプロパティの管理」766 ページ
- 「QAnywhere 転送ルールと削除ルール」774 ページ
- 「サーバー管理要求」153 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

送信先エイリアス

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

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

例

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

client1 に次の配信条件を定義します。

```
ias_Priority=1
```

client2 に次の配信条件を定義します。

```
ias_Priority=9
```

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

送信先エイリアスの作成

送信先エイリアスは、次のいずれかの方法で作成および管理できます。

- サーバー管理要求
- Sybase Central

Sybase Central の使用

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

♦ Sybase Central を使用した送信先エイリアスの作成

1. Sybase Central を起動します。
   - **[スタート]** » **[プログラム]** » **[SQL Anywhere 12]** » **[管理ツール]** » **[Sybase Central]** をクリックします。
   - **[接続]** » **[QAnywhere 12 に接続]** をクリックします。
   - **[ODBC データソース名]** または **[ODBC データソースファイル]** を指定し、必要に応じて **[ユーザー ID]** と **[パスワード]** を指定します。
   - **[OK]** をクリックします。
2. **[ファイル]** » **[新規]** » **[送信先エイリアス]** をクリックします。

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

参照

- 「サーバー管理要求を使用した送信先エイリアスの作成」170 ページ
- 「Sybase Central の使用」133 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

コネクター

以降の項では、QAnywhere で JMS コネクターと Web サービスコネクターを使用する場合について説明します。

JMS コネクター

JMS (Java Message Service) API は、Java アプリケーションにメッセージング機能を追加する役割を果たします。メッセージの交換は、QAnywhere クライアントアプリケーション間だけでなく、JMS インターフェイ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、コネクターと呼ばれる特殊なクライアントを使用します。QAnywhere 配備では、外部メッセージングシステムを、QAnywhere クライアントと同様に動作するように設定します。これには、固有のアドレスと構成を使用します。

サーバーファーム環境で QAnywhere メッセージング対応 Mobile Link を実行すると、プライマリサーバーの QAnywhere コネクターのみが起動します。プライマリサーバーが失敗した場合、新しいプライマリサーバーで QAnywhere コネクターが自動的に起動するため、JMS などの外部メッセージングシステムでデータを交換する間もメッセージの順序は保持されます。

参照

- 「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」9 ページ
- 「Mobile Link サーバーのサーバーファームでの実行」『Mobile Link サーバー管理』

JMS コネクターの設定

以下の手順では、QAnywhere に JMS コネクターを統合するのに必要なタスクの概要について説明します。この手順では、QAnywhere がすでに設定されているものとします。

♦ QAnywhere アプリケーションと外部 JMS システムの統合の概要

1. JMS システムの JMS 管理ツールを使用して、JMS キューを作成します。QAnywhere コネクターは、1 つの JMS キューを使用して JMS メッセージを受信します。このキューが存在しない場合、作成する必要があります。

   キューの作成方法の詳細については、ご使用の JMS 製品のマニュアルを参照してください。

2. Sybase Central を開きます。

3. QAnywhere プラグインを使用して、サーバーメッセージングストアに接続します。

   [接続] » **[QAnywhere 12 に接続]** をクリックします。

   [汎用 ODBC データベースに接続] ウィンドウが表示されます。

4. **[ユーザー ID]** フィールドに **ml_server** と入力します。

5. **[パスワード]** フィールドに **sql** と入力します。

6. **[OK]** をクリックします。

7. **[ファイル]** » **[新規]** » **[コネクター]** をクリックします。

8. **[JMS]** をクリックし、**[使用している JMS システムを指定してください。]** リストで使用している Web サーバーの種類をクリックします。**[次へ]** をクリックします。

9. **[コネクター名]** ページで、次の操作を行います。

   - **[コネクター名]** フィールドに、使用している JMS システムを入力します。「JMS コネクターへの QAnywhere メッセージの送信」139 ページを参照してください。
   - **[受信側]** フィールドに、JMS からの QAnywhere クライアント宛てのメッセージを受信するコネクターで使用されるキュー名 **qanywhere_receive** を入力します。
   - **[次へ]** をクリックします。

10. **[JNDI の設定]** ページで、次の操作を行います。

    - **[JNDI ファクトリ]** フィールドに、外部 JMS JNDI ネームサービスへのアクセスに使用されるファクトリ名を入力します。
    - **[ネームサービス URL]** フィールドに、JMS JNDI ネームサービスにアクセスするための URL を入力します。
    - **[ユーザー名]** フィールドに、外部 JMS JNDI ネームサービスに接続するための認証名を入力します。
    - **[パスワード]** フィールドに、外部 JMS JNDI ネームサービスに接続するための認証パスワードを入力します。
    - **[次へ]** をクリックします。

11. **[JMS キューの設定]** ページで、次の操作を行います。

    - **[キューファクトリ]** フィールドに、外部 JMS プロバイダーのキューファクトリ名を入力します。
    - **[ユーザー名]** フィールドに、外部 JMS キュー接続に接続するためのユーザー ID を入力します。
    - **[パスワード]** フィールドに、外部 JMS キュー接続に接続するためのパスワードを入力します。
    - **[次へ]** をクリックします。

12. **[JMS トピックの設定]** ページで、次の操作を行います。

    - **[トピックファクトリ]** フィールドに、外部 JMS プロバイダーのトピックファクトリ名を入力します。
    - **[ユーザー名]** フィールドに、外部 JMS トピック接続に接続するためのユーザー ID を入力します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- **[パスワード]** フィールドに、外部 JMS トピック接続に接続するためのパスワードを入力します。
- **[完了]** をクリックします。

13. **[OK]** をクリックします。

14. サーバーメッセージストアに接続するための接続文字列と -sl java オプションを指定して、Mobile Link サーバーを起動します。 「JMS 統合用 Mobile Link サーバーの起動」138 ページを参照してください。

> **Mobile Link サーバーの起動**
> この方法で JMS コネクターを定義した場合、-sl java オプションで JMS クライアントクラスを定義する説明したとおりの方法で常に Mobile Link サーバーを起動する必要があります。JMS コネクターが不要になり -sl java オプションを使用せず Mobile Link を起動するには、Sybase Central を使用して最初に JMS コネクターを削除します。

15. JMS コネクターにその他のオプションを設定するには、作成したコネクターを右クリックし、プロパティをクリックします。または、サーバー管理要求を使用することもできます。

    使用可能なプロパティのリストについては、「JMS コネクタープロパティの設定」138 ページを参照してください。

    サーバー管理要求を使用してコネクターのプロパティを設定する方法については、「サーバー管理要求によるコネクターの管理」159 ページを参照してください。

♦ メッセージの送信

1. QAnywhere システム内のアプリケーションから外部メッセージングシステムにメッセージを送信するには、QAnywhere メッセージを作成し、***connector-address¥JMS-queue-name*** に送信します。

   「JMS コネクターへの QAnywhere メッセージの送信」139 ページを参照してください。

2. 外部メッセージングシステムから QAnywhere システム内のアプリケーションにメッセージを送信するには、次の手順に従います。

   - JMS メッセージを作成します。
   - ias_ToAddress プロパティを QAnywhere の ***id¥queue*** (*id* はクライアントメッセージストアの ID、*queue* はアプリケーションキュー名) に設定します。
   - そのメッセージを JMS キューに登録します。

   「JMS から QAnywhere に送信されるメッセージのアドレス指定」142 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

- *%SQLANYSAMP12%¥QAnywhere¥jms* に、QAnywhere JMS のサンプルがあります。
- 質問は、次の SQL Anywhere フォーラムに投稿できます。http://sqlanywhere-forum.sybase.com
- サーバーファーム環境でメッセージングを設定する方法の詳細については、「Mobile Link サーバーのサーバーファームでの実行」『Mobile Link サーバー管理』を参照してください。

JMS 統合用 Mobile Link サーバーの起動

JMS インターフェイスをサポートする外部メッセージングシステムとメッセージを交換するには、Mobile Link サーバー (mlsrv12) 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*** サーバーメッセージストアに接続します。

  「-c mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。

- **-m** QAnywhere のメッセージングに対応します。

- **-sl java (-cp "*jarfile.jar*")** 外部 JMS プロバイダーを使用するために必要なクライアント JAR ファイルを追加します。

  「-sl java mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。

例

次の例では、JMS クライアントライブラリ *jmsclient.jar* (現在の作業ディレクトリにあります) と QAnywhere サンプルデータベース (メッセージストア) を指定して、Mobile Link サーバーを起動しています。このコマンドは、1 行で入力する必要があります。

```
mlsrv12 -sl java(-cp
"jmsclient.jar") -m -c "dsn=QAnywhere 12 Demo" ...
```

JMS コネクタープロパティの設定

JMS コネクタープロパティを使用して、JMS システムとの接続情報を指定します。このプロパティにより、BEA WebLogic などのサードパーティ製 JMS メッセージングシステムや Sybase EAServer に接続するコネクターを設定します。

プロパティの設定や参照を行うには、いくつかの方法があります。

- Sybase Central の**コネクターウィザード**

  「JMS コネクターの設定」135 ページを参照してください。

- Sybase Central の **[コネクタープロパティ]** ウィンドウ

- サーバー管理要求

  「コネクターの作成と設定」159 ページを参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

●ml_qa_global_props Mobile Link システムテーブル

JMS コネクタープロパティのリストについては、「JMS コネクタープロパティ」770 ページを参照してください。

複数のコネクターの設定

JMS メッセージシステムごとに JMS コネクターを定義すると、QAnywhere から複数の JMS メッセージシステムに接続できます。コネクターのプロパティのうち、ianywhere.connector.address だけは、設定するコネクターごとにユニークにする必要があります。

ianywhere.connector.address プロパティは、QAnywhere クライアントが JMS システムへのメッセージの宛先アドレスを指定するときに使用する必要があるアドレスプレフィクスです。

参照

- 「JMS コネクターへの QAnywhere メッセージの送信」139 ページ
- 「JMS コネクタープロパティの設定」138 ページ
- 「コネクターの作成と設定」159 ページ

JMS コネクターへの QAnywhere メッセージの送信

QAnywhere クライアントは、アドレスに次の値を設定することで、JMS システムにメッセージを送信できます。

connector-address¥*JMS-queue-name*

connector-address は、コネクタープロパティ ianywhere.connector.address の値です。*JMS-queue-name* は、JNDI (Java Naming and Directory Interface) を使用して JMS のキューまたはトピックを検索するときに使用される名前です。

JMS-queue-name に円記号が含まれている場合は、円記号をもう 1 つ追加して、この円記号をエスケープする必要があります。たとえば、ss というコンテキストを持つ qq というキューは、ss¥¥qq と指定します。

```
// C# example
QAMessage msg;
QAManager mgr;
...
mgr.PutMessage( @"ianywhere.connector.wsmqfs¥ss¥¥qq",msg);

// C++ example
QAManagerBase *mgr;
QATextMessage *msg;
...
mgr->putMessage("ianywhere.connector.easerver¥¥ss¥¥¥qq", msg);
```

例

たとえば、ianywhere.connector.address プロパティが ianywhere.connector.easerver に設定されており、JMS キュー名が myqueue である場合には、アドレスを設定するコードは次のようになります。

```
// C# example
QAManagerBase mgr;
QAMessage msg;

// Initialize the manager.
...
msg = mgr.CreateTextMessage()

 // Set the message content.
...
mgr.PutMessage(@"ianywhere.connector.easerver¥myqueue", msg);

// C++ example
QAManagerBase *mgr;
QATextMessage *msg;

// Initialize the manager.
...
msg = mgr.createTextMessage();

// Set the message content.
...
mgr->putMessage("ianywhere.connector.easerver¥¥myqueue", msg);
```

参照

- 「QAnywhere メッセージアドレス」60 ページ
- 「JMS コネクタープロパティの設定」138 ページ

JMS コネクターから QAnywhere クライアントへのメッセージの送信

QAnywhere メッセージは、JMS メッセージに自然にマッピングされます。

QAnywhere メッセージの内容

QAnywhere	JMS	備考
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 の組み込みヘッダー

次の表は、組み込みヘッダーのマッピングを示します。C++ と JMS では、これらのヘッダー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

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

setAddress() に対応し、JMS では getJMSDestination() または setJMSDestination() に対応します。.NET では、これらのヘッダー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	備考
Address	JMSDestination と JMS プロパティ ias_ToAddress	送信先アドレスに円記号が含まれている場合は、円記号をもう 1 つ追加して、この円記号をエスケープする必要がある。 Destination にはアドレスの JMS 情報部分のみがマッピングされる。まれに QAnywhere にメッセージがループバックされることがあるが、そのようなメッセージには QAnywhere アドレスがサフィックスとして追加される。これは ias_ToAddress に格納される。
Expiration	JMSExpiration	
InReplyToID	なし	マッピングされない
MessageID	なし	マッピングされない
Priority	JMSPriority	
Redelivered	なし	マッピングされない
ReplyToAddress	JMS プロパティ ias_ReplyToAddress	JMS プロパティにマッピングされる
コネクターの xjms.receiveDestination プロパティの値	JMSReplyTo	ReplyTo には、コネクターが JMS メッセージを受信するときに使用する Destination が設定される。
Timestamp	なし	マッピングされない。
なし	JMSTimestamp	JMS メッセージが QAnywhere メッセージにマッピングされるとき、QAnywhere メッセージの JMSTimestamp プロパティが、JMS メッセージの JMSTimestamp に設定される。

QAnywhere	JMS	備考
Timestamp	なし	QAnywhere メッセージが JMS メッセージにマッピングされるとき、JMS メッセージの JMSTimestamp が、JMS メッセージの作成時刻に設定される。

QAnywhere のプロパティ

QAnywhere のプロパティは、JMS のプロパティに自然にマッピングされ、型も維持されます。ただし、1 つだけ例外があります。それは、QAnywhere メッセージの JMSType プロパティは、JMS のヘッダープロパティ JMSType にマッピングされることです。

JMS から QAnywhere に送信されるメッセージのアドレス指定

JMS クライアントから QAnywhere クライアントにメッセージを送信するには、JMS メッセージプロパティ ias_ToAddress に送信先の QAnywhere アドレスを設定して、コネクタープロパティ xjms.receiveDestination に対応する JMS Destination にそのメッセージを送信します。

「QAnywhere メッセージアドレス」60 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 のコネクター設定は "qanywhere_receive" であるとします)。

```
import javax.jms.*;
try {
    QueueSession session;
    QueueSender sender;
    TextMessage mgr;
    Queue connectorQueue;

    // Initialize the session.
    connectorQueue = session.createQueue("qanywhere_receive");
    sender = session.createSender( connectorQueue );
    msg = session.createTextMessage();
    msg.setStringProperty("ias_ToAddress", "qaddr");

    // Set the message content.
    sender.send(msg);
}
catch( JMSException e ) {
    // Handle the exception.
}
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

JMS メッセージの QAnywhere メッセージへのマッピング

JMS メッセージは、QAnywhere メッセージに自然にマッピングされます。

JMS メッセージの内容

JMS	QAnywhere	備考
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 の組み込みヘッダー

次の表は、組み込みヘッダーのマッピングを示します。C++ と JMS では、これらのヘッダー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、これらのヘッダー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	備考
JMS Destination	なし	JMS Destination には、コネクタープロパティ xjms.receiveDestination に指定されているキューを設定する必要がある。
JMS Expiration	Expiration	
JMS CorrelationID	InReplyToID	
JMS MessageID	なし	マッピングされない。
JMS Priority	Priority	
JMS Redelivered	なし	マッピングされない

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

JMS のプロパティ

JMS のプロパティは、QAnywhere のプロパティに自然にマッピングされます。型も維持されます。ただし、いくつか例外があります。QAnywhere の Address プロパティは、JMS のメッセージプロパティ ias_ToAddress の値を基に設定されます。JMS のメッセージプロパティ ias_ReplyToAddress が設定されている場合、その値が、QAnywhere の ReplyToAddress の値に、間に文字 '¥' を挟んでサフィックスとして付加されます。

Web サービスコネクター

Web サービスコネクターは、特定のアドレスに送信された QAnywhere メッセージを受信し、メッセージが到着したときに Web サービス呼び出しを行います。Web サービス応答は、QAnywhere メッセージとして送信元のクライアントに送り返されます。Web サービスコネクターに送信されるすべてのメッセージは、QAnywhere WSDL コンパイラーによって生成されたプロキシクラスを使用して作成される必要があります。

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

Web サービスコネクターの設定

♦ Web サービスコネクターの作成

1. Sybase Central を開き、サーバーメッセージストアに接続します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

2. **[ファイル]** » **[新規]** » **[コネクター]** をクリックします。

3. **[Web サービス]** をクリックします。**[次へ]** をクリックします。

4. **[コネクター名]** フィールドに、QAnywhere クライアントがコネクターを指定するために使用するコネクターのアドレスを入力します。**[次へ]** をクリックします。

5. **[URL]** フィールドに、Web サービスの URL (*http://localhost:8080/qanyserv/F2C* など) を入力します。**[次へ]** をクリックします。

   オプションで、タイムアウト時間 (ミリ秒単位) を設定することもできます。タイムアウトを設定すると、指定した時刻に Web サービスが応答しない場合に要求はキャンセルされます。値は、プロパティ webservice.socket.timeout に設定されます。

6. **[HTTP パラメーター]** ページで、**[Web サービスに HTTP 認証を使用する]** をクリックして、次のフィールドに入力します。

   - **[HTTP ユーザー名]** フィールドにユーザー名を入力します。値は、プロパティ webservice.http.authName に設定されます。
   - **[HTTP パスワード]** フィールドにパスワードを入力します。値は、プロパティ webservice.http.password.e に設定されます。

7. **[プロキシ経由で Web サービスにアクセスする]** をクリックして、次のフィールドに入力します。

   - **[プロキシホスト名]** フィールドにホスト名を入力します。このプロパティを指定する場合は、webservice.http.proxy.port プロパティも設定する必要があります。
   - **[プロキシポート]** フィールドに、プロキシサーバー上の接続先ポートを入力します。このプロパティを指定する場合は、webservice.http.proxy.host プロパティも設定する必要があります。
   - **[プロキシユーザー名]** フィールドに、プロキシで認証が必要な場合に使用されるプロキシユーザー名を入力します。このプロパティを指定する場合は、webservice.http.proxy.password.e プロパティも設定する必要があります。
   - **[完了]** をクリックします。

   Web サービスコネクターにその他のオプションを設定するには、作成したコネクターを右クリックし、**[プロパティ]** をクリックします。または、サーバー管理要求を使用することもできます。

   使用可能なプロパティのリストについては、「Web サービスコネクターのプロパティ」146 ページを参照してください。

   サーバー管理要求の使用の詳細については、「サーバー管理要求によるコネクターの管理」159 ページを参照してください。

Web サービスコネクターのプロパティ

Web サービスコネクターのプロパティを使用して、Web サービスとの接続情報を指定します。これらのプロパティは、Sybase Central のコネクターウィザードで設定できます。

「Web サービスコネクター」144 ページを参照してください。

Web サービスコネクターのプロパティは、Sybase Central の **[コネクタープロパティ]** ウィンドウまたは ml_qa_global_props Mobile Link システムテーブルで参照できます。

[コネクタープロパティ] ウィンドウを開くには、Sybase Central でコネクターを右クリックし、**[プロパティ]** をクリックします。

♦ Web サービスのプロパティの表示

1. Sybase Central を開き、サーバーメッセージストアに接続します。
2. 左ウィンドウ枠の **[サーバーメッセージストア]** の下で、データソースの名前をクリックします。
3. 右ウィンドウ枠で **[コネクター]** タブをクリックし、Web サービスコネクターの名前をクリックします。
4. **[ファイル]** - **[プロパティ]** をクリックします。

Web サービスコネクターのプロパティ

- **ianywhere.connector.nativeConnection**　JMS コネクターを実装する Java クラスを指定します。QAnywhere の内部使用のためのものである場合は、削除や変更はできません。
- **ianywhere.connector.id (旧式)**　JMS コネクターをユニークに識別する識別子を指定します。デフォルトは ianywhere.connector.address です。
- **ianywhere.connector.address**　コネクター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、コネクターを指定するために使用されます。指定したアドレスは、コネクターの Mobile Link メッセージウィンドウに表示されるすべてのログ出力エラー、警告、情報メッセージの先頭に付けられます。

  Sybase Central では、このプロパティは**コネクターウィザード**の **[コネクター名]** ページにある **[コネクター名]** フィールドで設定します。
- **ianywhere.connector.compressionLevel**　Web サービスから受信したメッセージのデフォルトのメッセージ圧縮率です。圧縮率は、0 ～ 9 の整数値で指定します。0 は圧縮なし、9 は最大の圧縮率を表します。

  Sybase Central では、このプロパティは **[コネクタープロパティ]** ウィンドウの **[一般]** タブにある **[圧縮レベル]** セクションで設定します。
- **ianywhere.connector.logLevel**　Mobile Link メッセージウィンドウと Mobile Link サーバーメッセージログファイルに出力されるコネクター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。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

○ **1** エラーメッセージを出力します。

○ **2** エラーメッセージと警告メッセージを出力します。

○ **3** エラー、警告、情報の各メッセージを出力します。

○ **4** エラー、警告、情報、デバッグのメッセージを出力します。

Sybase Central では、このプロパティは **[コネクタープロパティ]** ウィンドウの **[一般]** タブにある **[ロギングレベル]** セクションで設定します。

- **ianywhere.connector.outgoing.retry.max** QAnywhere から外部メッセージングシステムへの出力メッセージについての、デフォルトのリトライ回数です。デフォルト値は 5 です。コネクターのリトライ回数を無制限にするには 0 を指定します。

  Sybase Central では、このプロパティは **[コネクタープロパティ]** ウィンドウの **[プロパティ]** タブで、**[新規]** をクリックして設定します。

- **ianywhere.connector.startupType** 起動のタイプには、automatic、manual、disabled のいずれかを指定できます。

- **webservice.http.authName** Web サービスが HTTP 認証を要求する場合に、このプロパティを使用してユーザー名を指定します。

- **webservice.http.password.e** Web サービスが HTTP 認証を要求する場合に、このプロパティを使用してパスワードを指定します。

- **webservice.http.proxy.authName** プロキシが認証を要求する場合は、このプロパティを使用してプロキシユーザー名を設定します。このプロパティを指定する場合は、webservice.http.proxy.password.e プロパティも設定する必要があります。

- **webservice.http.proxy.host** HTTP プロキシ経由で Web サービスにアクセスすることが必要である場合に、このプロパティを使用してホスト名を指定します。このプロパティを指定する場合は、webservice.http.proxy.port プロパティも設定する必要があります。

- **webservice.http.proxy.password.e** プロキシが認証を要求する場合は、このプロパティを使用してプロキシパスワードを設定します。このプロパティを指定する場合は、webservice.http.proxy.authName プロパティも設定する必要があります。

- **webservice.http.proxy.port** プロキシサーバーに接続するポートです。このプロパティを指定する場合は、webservice.http.proxy.host プロパティも設定する必要があります。

Web サービスコネクターへのメッセージの送信

メッセージは、モバイル Web サービス API を介して Web サービスコネクターに送信されます。ianywhere.qanywhere.ws.WSBase クラスの setProperty メソッドを使用して、WS_CONNECTOR_ADDRESS プロパティに Web サービスコネクターの ID を設定します。WSBase クラス [QAnywhere Java]621 ページを参照してください。

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

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

チュートリアル：JMS コネクターの使用

JMS コネクターを使用すると、JMS メッセージシステムと QAnywhere を接続できます。このチュートリアルでは、Windows JMS クライアントアプリケーションと QAnywhere クライアントアプリケーションの間でメッセージを送信します。

必要なソフトウェア

- SQL Anywhere 12
- Java ソフトウェア開発キット
- JMS システム

前提知識と経験

次の知識と経験が必要です。

- Java の知識
- JMS コネクターの設定に関する基礎知識

目的

次の項目について、知識と経験を得ることができます。

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

主要な概念

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

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

関連項目

JMS コネクターの使用の詳細については、「コネクター」135 ページを参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

レッスン 1：クライアントおよびサーバーのコンポーネントの設定

♦ JMS プロバイダーの準備

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

2. JMS サーバーで次のキューを作成します。

   - **testmessage** TestMessage サンプルでは、このキュー名を使用してメッセージを受信します。
   - **qanywhere_receive** QAnywhere の JMS コネクターでは、このキュー名を使用してメッセージを受信します。

   キューを作成した後、サーバーの再起動が必要になる場合があります。詳細については、JMS サーバーのマニュアルを参照してください。

♦ QAnywhere クライアントとサーバーコンポーネントの起動

1. Sybase Central を使用して、JMS システムの QAnywhere JMS コネクターを作成します。「JMS コネクターの設定」135 ページを参照してください。

2. コマンドプロンプトで次のコマンドを実行します。

   ```
   mlsrv12 -m -c "dsn=QAnywhere 12 Demo" -sl
   java(-cp JMS-client-jar-files) -vcrs
   -zu+
   ```

   ここで、*JMS-client-jar-files* は、JMS サーバーにアクセスするために必要な jar ファイルのリストです。各ファイル名の間はセミコロンで区切ります。詳細については、JMS サーバーのマニュアルを参照してください。

   Mobile Link サーバーがメッセージングに対応した状態で起動します。

3. **[スタート] » [プログラム] » [SQL Anywhere 12] » [QAnywhere] » [SQL Anywhere を使用するチュートリアル] » [SQL Anywhere 用 QAnywhere Agent -- saclient1]** をクリックします。

   QAnywhere Agent がロードされます。

4. **[スタート] » [プログラム] » [SQL Anywhere 12] » [QAnywhere] » [SQL Anywhere を使用するチュートリアル] » [TestMessage -- saclient1]** をクリックします。

   QAnywhere サンプルアプリケーションがロードされます。

♦ TestMessage クライアントの JMS バージョンの起動

1. コマンドプロンプトで次のコマンドを実行します。

   ```
   edit "%SQLANYSAMP12%/QAnywhere/JMS/TestMessage/build.bat"
   ```

2. *build.bat* ファイルのコードを調べて、JMS サーバーのファイルパスが正しいことを確認します。

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

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

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

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

完了したら、変更内容を保存してエディターを終了します。

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

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

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

4. コマンドプロンプトで次のコマンドを実行します。

```
edit "%SQLANYSAMP12%/QAnywhere/JMS/TestMessage/run.bat"
```

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

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

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

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

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

完了したら、変更内容を保存してエディターを終了します。

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

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

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

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

7. JMS 用の TestMessage のウィンドウを、画面右側にすでに置かれている **[saclient1 - TestMessage]** ウィンドウの下に移動します。

8. 「レッスン 2：JMS クライアントから QAnywhere クライアントへのメッセージの送信」に進みます。

レッスン 2：JMS クライアントから QAnywhere クライアントへのメッセージの送信

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。「レッスン 1：クライアントおよびサーバーのコンポーネントの設定」を参照してください。

♦ JMS クライアントからの QAnywhere クライアントへのメッセージの送信

1. JMS TestMessage の **[Message]** メニューから **[New]** をクリックします。

2. **[Destination ID]** フィールドに **saclient1** と入力します。

3. **[Subject]** フィールドと **[Message]** フィールドにサンプルテキストを入力します。

4. **[Send]** をクリックします。

   メッセージが受信されたことを示すウィンドウが表示されます。

5. 「レッスン 3：QAnywhere クライアントから JMS クライアントへのメッセージの送信」に進みます。

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

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。「レッスン 1：クライアントおよびサーバーのコンポーネントの設定」を参照してください。

♦ QAnywhere の JMS コネクターの名前の検索

1. **[スタート]** - **[プログラム]** - **[SQL Anywhere]** - **[Sybase Central]** をクリックします。

2. **[接続]** » **[QAnywhere 12 に接続]** をクリックします。

3. **[ODBC データソース名]** をクリックします。

4. **[参照]** をクリックし、**[QAnywhere 12 Demo]** をクリックします。

5. **[OK]** をクリックします。

6. **[OK]** をクリックします。

7. **[コネクター]** タブをクリックします。

   右ウィンドウ枠には、すべてのアクティブな JMS コネクターのリストが表示されます。

8. 名前フィールドを確認します。

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

♦ QAnywhere クライアントからの JMS クライアントへのメッセージの送信

1. **[saclient1 - TestMessage]** ウィンドウで、**[Message]** » **[New]** をクリックします。

2. **[Destination ID]** フィールドに JMS システムの名前を入力します。

3. **[Subject]** フィールドと **[Message]** フィールドにサンプルテキストを入力します。

4. **[Send]** をクリックします。

   メッセージが受信されたことを示すウィンドウが表示されます。

5. 「クリーンアップ」に進みます。

クリーンアップ

このレッスンは、受講者がこれまでのすべてのレッスンを終了していることを前提としています。「レッスン 1：クライアントおよびサーバーのコンポーネントの設定」を参照してください。

TestMessage クライアント、QAnywhere Agent、Mobile Link サーバーをそれぞれ停止します。

♦ すべてのアプリケーションの停止

1. JMS TestMessage クライアントアプリケーションを閉じるには、**[File]** » **[Exit]** をクリックします。

2. **[saclient1 - TestMessage Client]** ウィンドウを閉じるには、**[File]** » **[Exit]** をクリックします。

3. **[saclient1 - QAnywhere Agent]** ウィンドウと Mobile Link サーバーを閉じるには、それぞれのウィンドウで **[シャットダウン]** をクリックします。

4. JMS コネクターから切断する方法については、JMS サーバーのマニュアルを参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

サーバー管理要求

QAnywhere クライアントアプリケーションは、「サーバー管理要求」という特殊なメッセージをサーバーに送信できます。サーバー管理要求は、XML フォーマットの内容を含み、QAnywhere システムキューに送信されます。特殊な認証文字列が必要です。サーバー管理要求は、次のようなさまざまな機能を実行できます。

●コネクターと Web サービスの開始と停止

「コネクターを開く」161 ページと「コネクターを閉じる」162 ページを参照してください。

●サーバー管理要求のスケジュール設定

「Schedule タグ」706 ページを参照してください。

●コネクターのステータスのモニタリング

「コネクターのモニター」162 ページを参照してください。

●クライアント転送ルールの設定と更新

「サーバー管理要求での転送ルールの指定」169 ページを参照してください。

●メッセージステータスのモニタリング

「QAnywhere のモニター」173 ページを参照してください。

●サーバー上に置かれているクライアントメッセージストアプロパティの設定、更新、削除、クエリ

「サーバー管理要求でのサーバープロパティの設定」168 ページを参照してください。

●メッセージのキャンセル

「メッセージのキャンセル」157 ページを参照してください。

●アクティブなクライアント、メッセージストアプロパティ、メッセージのクエリ

サーバー管理要求のアドレス指定

デフォルトでは、サーバー管理要求を **ianywhere.server¥system** に送信する必要があります。このアドレスのクライアント ID の部分を変更するには、ianywhere.qa.server.id プロパティを設定してからサーバーを再起動します。たとえば、ianywhere.qa.server.id プロパティが myServer に設定されていると、サーバー管理要求は myServer¥system に送信されます。

ianywhere.qa.server.id プロパティの設定の詳細については、「サーバープロパティ」768 ページを参照してください。

QAnywhere メッセージのアドレス指定の詳細については、「QAnywhere メッセージの送信」63 ページを参照してください。

システムキューの詳細については、「システムキュー」61 ページを参照してください。

例

次に、メッセージの詳細要求のサンプルを示します。現在サーバー上に置かれている優先度 9 のすべてのメッセージについて、メッセージ ID、ステータス、送信先アドレスを表示するレポートを 1 つ生成します。

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

次の例は、C# での記述です。これは、サーバー側のクライアント用転送ルールを設定します。優先度が 4 より大きい場合に、サーバーからのメッセージを someClient というクライアントにのみ転送するというルールです。

```
QAManager mgr = ...; // Initialize the QAManager
QAMessage msg = mgr.CreateTextMessage();
msg.SetStringProperty( "ias_ServerPassword", "QAnywhere" );

// Indenting and newlines are just for readability
msg.Text = "<?xml version="1.0" encoding="UTF-8"?>¥n"
+ "<actions>¥n"
+ " <SetProperty>¥n"
+ "  <prop>¥n"
+ "   <client>someClient</client>¥n"
+ "   <name>ianywhere.qa.server.rules</name>¥n"
+ "   <value>ias_Priority &gt; 4</value>¥n"
+ "  </prop>¥n"
+ " </SetProperty>¥n"
+ " <RestartRules>¥n"
+ "  <client>someClient</client>¥n"
+ " </RestartRules>¥n"
+ "</actions>¥n";

mgr.PutMessage( @"ianywhere.server¥system", msg );
```

サーバー管理要求の認証

「ianywhere.qa.server.password.e」 サーバープロパティは、サーバー管理要求を認証する際のパスワードを指定するために使用されます。このプロパティが設定されていない場合、パスワードは QAnywhere です。「サーバープロパティ」 768 ページを参照してください。

サーバー管理要求の記述

サーバー管理要求は XML フォーマットの内容を含んでいます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

> **注意**
> > や < などの記号は、サーバー管理要求内で使用できません。代わりに、> と < を使用してください。

サーバー管理要求には、その種類ごとに独自の XML タグが含まれます。たとえば、コネクターを閉じるには <CloseConnector> タグを使用します。

各サーバー管理要求は <actions> タグで開始します。

actionsResponseId タグ

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

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

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

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

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

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

アーカイブメッセージストア要求

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

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

例

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

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

送信先エイリアスの作成

サーバー管理要求を使用して送信先エイリアスを作成したり変更したりできます。

参照

- 「サーバー管理要求を使用した送信先エイリアスの作成」170 ページ
- 「送信先エイリアス」133 ページ.
- 「サーバー管理要求」153 ページ

サーバー管理要求を使用したサーバーメッセージストアの管理

サーバー管理要求を使用して、サーバーメッセージストアを管理できます。

参照

- 「サーバー管理要求」153 ページ

クライアントの転送ルールの更新

サーバー側のクライアント転送ルールが変更されたら、対応するクライアントのルールも変更する必要があります。クライアント転送ルールを変更するには、サーバー管理要求で iianywhere.qa.server.rules プロパティを設定します。

<RestartRules> タグには、変更するクライアント名を指定する <client> タグが 1 つ含まれます。

<RestartRules> タグのサブタグ	**説明**
<client>	転送ルールを更新するクライアントの名前です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

サーバー XML は新しい転送ルールプロパティを指定した後で、<RestartRules> タグを使用してルール処理を再起動します。たとえば、次の XML では、クライアント cyclient に対応するサーバー側の転送ルールを auto = ias_Priority > 4 に変更します。この XML で、">" が適切にエンコードされていることに注意してください。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <SetProperty>
    <prop>
      <client>myclient</client>
      <name>ianywhere.qa.server.rules</name>
      <value>auto = ias_Priority &gt; 4</value>
    </prop>
  </SetProperty>
  <RestartRules>
      <client>myclient</client>
  </RestartRules>
</actions>
```

メッセージのキャンセル

サーバーメッセージストア内にあるメッセージをキャンセルするサーバー管理要求を作成できます。ワンタイムキャンセル要求を作成することも、キャンセル要求のスケジュールを設定して自動的にキャンセルすることもできます。また、オプションで、キャンセルされたメッセージの詳細を示すレポートを生成することもできます。

メッセージは、要求がアクティブになった時点で最終ステータスに至っておらず、受信者に送信されていない場合にのみキャンセルできます。

<CancelMessageRequest> タグのサブタグ	**説明**
<request>	特定の要求に関する情報をグループ化します。複数の <request> タグを指定することは、複数のメッセージ管理要求を別々に送信することと同じです。

<Request> タグのサブタグ	**説明**
<condition>	キャンセルするメッセージを含めるための条件をグループ化します。「Condition タグ」704 ページを参照してください。
<persistent>	要求をサーバーデータベース内で永続的にすることを指定します。これにより、サーバーが再起動された場合でもメッセージをキャンセルできます。<schedule> タグでのみ使用します。

<Request> タグのサブタグ	説明
<requestId>	この要求により生成される各レポートに含まれる要求に対して、ユニークな識別子を指定します。このフィールドに複数の値を指定すると、一度に複数の要求をアクティブにできます。同じ要求 ID を指定すると、クライアントはアクティブな要求を上書きまたは削除します。
<replyAddr>	この要求により生成される各レポートの返信アドレスを指定します。このタグが指定されていない場合は、レポートのデフォルトの返信アドレスが送信メッセージの返信アドレスになります。
<report>	要求がアクティブになるたびにレポートを送信します。要求がアクティブになるたびにレポートを送信するには、<request> タグ内に空の <report> タグを指定します。
<schedule>	スケジュールに従ってレポートを生成することを指定します。 「サーバー管理要求の親タグ」 704 ページを参照してください。

例

次の要求は、ianywhere.connector.myConnector¥deadqueue というアドレスが指定された、サーバー上にあるメッセージをキャンセルします。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <CancelMessageRequest>
  <request>
   <requestId>cancelRequest</requestId>
   <condition>
    <customRule>ias_Address='ianywhere.connector.myConnector¥deadqueue'</customRule>
   </condition>
  </request>
 </CancelMessageRequest>
</actions>
```

メッセージの削除

サーバーのクリーンアップポリシーを指定するには、サーバーから削除できるメッセージを制御するルールを使用して、特別なクライアント ianywhere.server.deleteRules に ianywhere.qa.server.deleteRules プロパティを設定します。

次の例では、有効期限の切れたキャンセル済みメッセージを削除するように、メッセージのクリーンアップポリシーを変更します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <SetProperty>
 <prop>
    <client>ianywhere.server.deleteRules</client>
    <name>ianywhere.qa.server.deleteRules</name>
    <value>auto = ias_Status in ( ias_ExpiredStatus, ias_CancelledStatus ) and ias_TransmissionStatus
= IAS_TRANSMITTED</value>
 </prop>
   </SetProperty>
   <RestartRules>
 <client>ianywhere.server.deleteRules</client>
   </RestartRules>
</actions>
```

サーバー管理要求によるコネクターの管理

サーバー管理要求を使用して、コネクターの作成、設定、削除、起動、停止、モニターを行うことができます。

参照

- 「サーバー管理要求」153 ページ
- 「コネクター」135 ページ
- 「Web サービスコネクター」144 ページ

コネクターの作成と設定

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

例

次の例では、サーバー管理要求は最初に関係する複数のプロパティを設定して、クライアントの ianywhere.connector.jboss にこれらのプロパティを関連付けます。ianywhere.connector.jboss は新しいコネクターのクライアント ID です。JMS 固有のプロパティは、ローカル JBOSS JMS サーバーに接続されるコネクターが示される方法で設定されます。この後で、コネクターは <OpenConnector> タグを使用して起動されます。JMS クライアントの関連 jar ファイルを指定して Mobile Link サーバーを起動していない場合は、コネクターが起動しません。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <SetProperty>
      <prop>
     <client>ianywhere.connector.jboss</client>
     <name>ianywhere.connector.nativeConnection</name>
  <value>ianywhere.message.connector.jms.NativeConnectionJMS</value>
   </prop>
      <prop>
     <client>ianywhere.connector.jboss</client>
     <name>ianywhere.connector.address</name>
  <value>ianywhere.connector.jboss</value>
   </prop>
```

```
      <prop>
     <client>ianywhere.connector.jboss</client>
     <name>xjms.jndi.factory</name>
<value>org.jnp.interfaces.NamingContextFactory</value>
 </prop>
      <prop>
     <client>ianywhere.connector.jboss</client>
     <name>xjms.jndi.url</name>
<value>jnp://0.0.0.0:1099</value>
 </prop>
      <prop>
     <client>ianywhere.connector.jboss</client>
     <name>xjms.topicFactory</name>
<value>ConnectionFactory</value>
 </prop>
      <prop>
     <client>ianywhere.connector.jboss</client>
     <name>xjms.queueFactory</name>
<value>ConnectionFactory</value>
 </prop>
      <prop>
     <client>ianywhere.connector.jboss</client>
     <name>xjms.receiveDestination</name>
<value>qanywhere_receive</value>
 </prop>
      <prop>
     <client>ianywhere.connector.jboss</client>
     <name>xjms.deadMessageDestination</name>
<value>qanywhere_deadMessage</value>
 </prop>
  </SetProperty>
  <OpenConnector>
      <client>ianywhere.connector.jboss</client>
  </OpenConnector>
</actions>
```

コネクターの変更

コネクターを変更するには、コネクターを閉じてから <SetProperty> タグを使用してプロパティを変更し、コネクターを開きます。

例

次の例では、コネクターのログレベルを 4 に変更します。ianywhere.connector.jboss という ID を持つコネクターを閉じ、このコネクターのプロパティ logLevel を 4 に変更してから、新しいログレベルでコネクターをもう一度開きます。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <CloseConnector>
      <client>ianywhere.connector.jboss</client>
  </CloseConnector>
  <SetProperty>
      <prop>
     <client>ianywhere.connector.jboss</client>
<name>ianywhere.connector.logLevel</name>
<value>4</value>
 </prop>
  </SetProperty>
```

```
    <OpenConnector>
      <client>ianywhere.connector.jboss</client>
    </OpenConnector>
  </actions>
```

コネクターの削除

コネクターを削除するには、<SetProperty> を使用してクライアントのすべてのプロパティを削除します。

例

次の例では、ianywhere.connector.jboss という ID を持つコネクターを閉じます。このコネクターのすべてのプロパティは、<name> タグと <value> タグを指定せずに <SetProperty> タグを使用することで削除されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <SetProperty>
    <prop>
   <client>ianywhere.connector.jboss</client>
   <name>ianywhere.connector.nativeConnection</name>
  </prop>
  </SetProperty>
</actions>
```

コネクターを開く

コネクターを開くには、<OpenConnector> を使用します。

<OpenConnector> タグには、開くコネクターの名前を指定する <client> タグが 1 つ含まれます。

<OpenConnector> タグのサブタグ	**説明**
<client>	開くコネクターの名前

参照

- 「コネクター」135 ページ
- 「Web サービスコネクター」144 ページ

例

次の例では、simpleGroup というコネクターを開きます。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <OpenConnector>
   <client>simpleGroup</client>
  </OpenConnector>
</actions>
```

コネクターを閉じる

コネクターを閉じるには、<CloseConnector> タグを使用します。<CloseConnector> には、閉じるコネクターの名前を指定する <client> タグが 1 つ含まれます。

<CloseConnector> タグのサブタグ	説明
<client>	閉じるコネクターの名前

参照

- 「コネクター」135 ページ
- 「Web サービスコネクター」144 ページ

例

次の例では、simpleGroup というコネクターを閉じます。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <CloseConnector>
    <client>simpleGroup</client>
   </CloseConnector>
</actions>
```

コネクターのモニター

コネクターの情報を取得するには、クライアントステータス要求という特別な種類のサーバー管理要求を作成します。これには <ClientStatusRequest> タグが含まれています。このタグでは、要求の登録に必要な情報を含む 1 つまたは複数の <request> タグを使用します。

クライアントステータス要求では、次のような複数の方法でコネクターに関するレポートを取得します。

- ワンタイム要求を作成する。
- コネクターのステータスが変更するたびにレポートが送信される状態変更リスナー (State Change Listener) を登録する。
- コネクターでエラーが発生するたびにレポートが送信されるエラーリスナーを登録する。

レポートを特定の時刻や一定の間隔で送信するようにスケジュール設定することもできます。

ClientStatusRequest タグ

コネクターの情報を取得するには、<ClientStatusRequest> タグを使用します。

クライアントステータス要求は、要求の登録に必要なすべての情報を含む 1 つまたは複数の <request> タグから構成されています。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

<ClientStatusRequest> タグのサブタグ	説明
<request>	要求内の情報をグループ化します。

クライアントステータス要求の <request> タグ

この要求により生成されるレポートごとに返信アドレスを指定するには、<request> タグ内で、オプションの <replyAddr> タグを使用します。このタグが指定されていない場合は、レポートのデフォルトの返信アドレスが送信メッセージの返信アドレスになります。

各レポートに含まれる要求のラベルを追加するには、オプションの <requestId> タグを使用します。複数の要求を登録する場合や、要求を削除または変更する場合は、特定の要求により生成されたレポートがどれであるかを、この ID で特定できます。

要求のコネクターのリストを指定するには、1 つまたは複数の <client> タグが含まれるようにします。タグごとにコネクターアドレスを 1 つ指定してください。ワンタイム要求の場合は、すべてのコネクターがレポートに含まれます。イベントリスナー要求の場合は、サーバーは各コネクターを受信します。

サーバーのダウンタイムを通してイベントの詳細を永久的にするように指定するには、<persistent> タグを指定します。このタグにはデータを指定する必要はありません。<persistent/> または <persistent></persistent> のどちらのフォームを使用することもできます。

オプションで 1 つまたは複数の <onEvent> タグを含めて、イベントのリストを指定できます。<onEvent> タグごとにイベントタイプを 1 つ指定してください。このタグの指定がない場合は、クライアントステータス要求でワンタイム要求が生成されます。それ以外の場合は、指定されたイベントのイベントリスナーが、クライアントステータス要求で登録されます。

クライアントステータス要求の <request> タグのサブタグ	説明
<client>	1 つまたは複数の <client> タグを含めることができます。タグごとにコネクターアドレスを 1 つ指定してください。ワンタイム要求の場合は、リストされているすべてのコネクターがレポートに含まれます。イベントリスナー要求の場合は、サーバーは各コネクターの受信を開始します。

クライアントステータス要求の <request> タグのサブタグ	説明
<onEvent>	発生時にサーバーにレポートを生成させるイベントを指定します。1 つまたは複数の <onEvent> タグを含めることができます。タグごとにイベントタイプを 1 つ指定してください。<onEvent> タグが指定されていない場合は、クライアントステータス要求でワンタイム要求が生成されます。それ以外の場合は、指定されたイベントのイベントリスナーが、クライアントステータス要求で登録されます。
<persistent>	このクライアントステータス要求の詳細情報をサーバーデータベース内で永続的にすることを指定します。
<replyAddr>	この要求により生成される各レポートの返信アドレスを指定します。このタグが指定されていない場合は、レポートのデフォルトの返信アドレスが送信メッセージの返信アドレスになります。
<requestId>	レポートのラベルです。この値は要求のラベルとして使用され、この要求により生成される各レポートに含まれます。複数の要求が登録されており、未処理の要求を削除または変更する場合には、特定の要求により生成されたレポートがどれであるかを、このラベルで特定できます。
<schedule>	「サーバー管理要求の親タグ」を参照してください。

ワンタイムクライアントステータス要求

ワンタイム要求を作成するには、クライアントステータス要求から <onEvent> タグと <schedule> タグを除外します。このようにすると、クライアントステータス要求で指定された各コネクターについて、現在のステータス情報を含むレポートが 1 つ生成されます。

次の XML メッセージでは <onEvent> タグと <schedule> タグが除外されているので、これはワンタイム要求の例です。<ClientStatusRequest> タグで指定された各コネクターについて、現在のステータス情報を含むレポートが 1 つ生成されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <ClientStatusRequest>
  <request>
   <replyAddr>ianywhere.connector.beajms¥q11</replyAddr>
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
    <requestId>myOneTimeRequest</requestId>
    <client>ianywhere.server</client>
    <client>ianywhere.connector.beajms</client>
   </request>
  </ClientStatusRequest>
 </actions>
```

イベント発生時のクライアントステータス要求

発生時に QAnywhere サーバーにステータスレポートを生成させるイベントを指定するには、クライアントステータス要求に 1 つまたは複数の <onEvent> タグが含まれるようにします。ワンタイム要求とは異なり、サーバーは要求にすぐに応答しません。代わりにコネクターの受信を開始して、指定されたイベントが発生するまで待機します。いずれかのイベントがトリガーされるたびに、イベントを発生させたコネクターの情報を含むレポートが送信されます。

次のイベントは、イベント発生時の要求で使用できます。

イベント	発生するタイミング
open	閉じられているコネクターが開かれたとき
close	開いていたコネクター、または一時停止状態のコネクターが閉じられたとき
statusChange	コネクターのステータスに変更があったとき。該当するステータスとして、オープンとクローズがあります。
error	予期しないエラーがコネクターからスローされたとき
fatalError	処理できない致命的なエラーがコネクターからスローされたとき
none	このイベントが発生することはありません。これまでのイベントウォッチが、コネクターからすべて削除されます。

次の例では、サーバーコネクターのステータスに変更があるたびに、またはサーバーコネクターがエラーを生成するたびに、アドレスが ianywhere.connector.beajms¥q11 のコネクターがステータスレポートに送信されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <ClientStatusRequest>
  <request>
   <replyAddr>ianywhere.connector.beajms¥q11</replyAddr>
   <requestId>myEventRequest</requestId>
   <client>ianywhere.server</client>
   <onEvent>statusChange</onEvent>
   <onEvent>error</onEvent>
  </request>
 </ClientStatusRequest>
</actions>
```

複数の同時要求

サーバーコネクターを含む任意の数のコネクターについて、返信アドレスごとに独自のイベントリスナーセットを指定できます。コネクターにイベントリスナーを追加しても、サーバーの他のイベントリスナーが妨害されることはありません (ただし、置換中のイベントリスナーは妨害される可能性があります)。

要求の置換

イベントリスナーがすでに登録されているコネクターに対して、同じ返信アドレスでイベントリスナーを追加すると、古いリスナーが新しいリスナーで置換されます。たとえば、コネクター abc の statusChange リスナーがアドレス x/y に登録されている場合に、abc のエラーリスナーをアドレス x/y に登録すると、abc は statusChange イベントに応答しなくなります。

同じアドレスに複数のイベントを登録するには、複数の <onEvent> タグを使用して 1 つの要求を作成します。

要求の削除

コネクターのイベントリスナーがアドレスに登録されている場合は、"none" イベントを指定して同じアドレスから別のクライアントステータス要求を指定することで、イベントリスナーを削除できます。

次の例では、アドレス ianywhere.connector.beajms¥q11 に登録されているサーバーコネクターから、すべてのイベントリスナーが削除されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <ClientStatusRequest>
  <request>
   <replyAddr>ianywhere.connector.beajms¥q11</replyAddr>
   <client>ianywhere.server</client>
   <onEvent>none</onEvent>
  </request>
 </ClientStatusRequest>
</actions>
```

永続的なクライアントステータス要求

要求の詳細をメッセージストアのグローバルプロパティに保存するように指定し、サーバーの再起動後に要求の詳細を自動的に回復できるようにするには、クライアントステータス要求に <persistent> タグが含まれるようにします。永続性の機能は、スケジュール設定されたイベントとイベントリスナーで使用できますが、ワンタイム要求では使用できません。永続的な要求を追加および削除するルールは、通常の要求のルールに類似しています。ただし、スケジュール設定されたイベントとイベントリスナーを別々に追加することはできません。代わりに、クライアントは永続的な要求を追加するときに、特定のコネクター／返信アドレスペアのすべてのイベントリスナーとスケジュールを同じ要求に指定する必要があります。

次の例では、イベントリスナーとスケジュールを ianywhere.connector.myConnector に追加して、これらを永続的にします。また、このコネクター／返信アドレスペアに永続的な要求が既存していた場合は、それらを上書きします。レポートは 30 分ごとに送信されます。コネクターのステータスに変化があった場合にも送信されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
```

```
<ClientStatusRequest>
 <request>
  <replyAddr>ianywhere.connector.beajms¥q11</replyAddr>
  <client>ianywhere.connector.myConnector</client>
  <onEvent>statusChange</onEvent>
  <schedule>
   <everyminute>30</everyminute>
  </schedule>
  <persistent/>
 </request>
</ClientStatusRequest>
</actions>
```

イベントリスナーの永続性

コネクターが自身のアドレスに登録したイベントリスナーは、コネクターが閉じられた場合でも、サーバーが停止するまでサーバー上に存在します。コネクターが再び開かれた場合は、格納されているイベントリスナーがもう一度アクティブになります。

コネクターのステータス

コネクターのステータスは、次の 2 つのうちのいずれかになります。

- **実行中**　コネクターは着信と送信メッセージを受け入れて処理しています。このステータスの場合は、コネクタープロパティ ianywhere.connector.state が 1 になります。
- **非実行中**　コネクターは、着信と送信メッセージの受け入れと処理を行っていません。このステータスの場合は、コネクタープロパティ ianywhere.connector.state が 2 になります。コネクターのステータスが「実行中」に変わると、コネクターは初期化されます。

参照

- 「コネクターの変更」160 ページ

クライアントステータスレポート

サーバーは、コネクターがレポートを要求するたび、または登録されているイベントが発生するたびに、クライアントステータスレポートを生成します。クライアントステータスレポートは、メッセージプロパティを含まない簡単なテキストメッセージで生成されます。

イベントの発生時に取得できる情報の種類に応じて、各レポートには次のいずれかの値が含まれます。

- クライアント (常に表示される)
- UTCDatetime (常に表示される)
- vendorStatusDescription (常に表示される)
- statusCode (常に表示される)
- vendorStatusCode
- statusSubCode
- statusDescription

次に例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<ClientStatusReport>
 <requestId>myRequest</requestId>
 <componentReport>
  <client>ianywhere.server</client>
  <UTCDatetime>Tue May 31 13:53:02 EDT 2005</UTCDatetime>
  <statusCode>Running</statusCode>
  <vendorStatusDescription></vendorStatusDescription>
 </componentReport>
 <componentReport>
  <client>ianywhere.connector.beajms</client>
  <UTCDatetime>Tue May 31 13:53:02 EDT 2005</UTCDatetime>
  <statusCode>Not running</statusCode>
  <vendorStatusDescription></vendorStatusDescription>
 </componentReport>
</ClientStatusReport>
```

サーバー管理要求でのサーバープロパティの設定

<SetProperty> タグには 1 つまたは複数の <prop> タグが含まれています。各タグは、設定するプロパティを指定します。各 prop タグは、<client> タグ、<name> タグ、<value> タグから構成されています。プロパティを削除するには、<value> タグを除外します。

<prop> タグのサブタグ	**説明**
<client>	サーバープロパティを設定するクライアントの名前です。
<name>	設定するプロパティの名前です。
<value>	設定されるプロパティの値です。このタグが含まれていない場合は、プロパティが削除されます。

例

次のサーバー管理要求は、simpleGroup という送信先エイリアスの ianywhere.qa.member.client3 プロパティを Y に設定して、client3 を simpleGroup に追加します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <SetProperty>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.qa.member.client3</name>
      <value>Y</value>
    </prop>
   </SetProperty>
</actions>
```

次の例では、以下のことを実行します。

●client1 プロパティの myProp1 を作成または変更して値を 3 に設定する。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

●client1 プロパティの myProp2 を削除する。

●client2 プロパティの myProp3 の値を "some value" に変更する。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <SetProperty>
 <prop>
    <client>client1</client>
    <name>myProp1</name>
    <value>3</value>
 </prop>
 <prop>
    <client>client1</client>
    <name>myProp2</name>
 </prop>
 <prop>
    <client>client2</client>
    <name>myProp3</name>
    <value>some value</value>
 </prop>
   </SetProperty>
</actions>
```

参照

● 「サーバー管理要求」153 ページ

サーバー管理要求での転送ルールの指定

サーバー管理要求を使用して、すべてのユーザーに適用されるサーバー側のデフォルトの転送ルールを指定したり、クライアントごとに転送ルールを指定したりできます。

サーバーのデフォルトの転送ルールを指定するには、クライアント ianywhere.server.defaultClient の ianywhere.qa.server.rules プロパティを設定します。クライアントの場合は、ianywhere.qa.server.rules プロパティを使用してサーバーの転送ルールを指定します。

サーバー管理要求の使い方の概要、サーバー管理要求の認証方法とスケジュールの設定方法については、「サーバー管理要求」153 ページを参照してください。

例

次の例では、優先度の高い (優先度が 7 以上) メッセージだけを送信するデフォルトルールを作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <SetProperty>
    <prop>
      <client>ianywhere.server.defaultClient</client>
      <name>ianywhere.qa.server.rules</name>
      <value>auto = ias_Priority &gt; 6</value>
    </prop>
  </SetProperty>
  <RestartRules>
    <client>ianywhere.server.defaultClient</client>
```

```
  </RestartRules>
</actions>
```

次の例では、内容のサイズが 100 未満で、午前 8 時から午後 6 時の営業時間内に送信する必要のあるメッセージのみを送信するルールを作成し、myClient というクライアントに適用します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <SetProperty>
    <prop>
      <client>myClient</client>
      <name>ianywhere.qa.server.rules</name>
      <value>auto = ias_ContentSize &lt; 100
        and ias_CurrentTime &gt; '8:00:00'
        and ias_CurrentTime &lt; '18:00:00'</value>
    </prop>
  </SetProperty>
  <RestartRules>
    <client>myClient</client>
  </RestartRules>
</actions>
```

サーバー管理要求を使用した送信先エイリアスの作成

サーバー管理要求を使用して送信先エイリアスを作成したり変更したりできます。

送信先エイリアスの詳細については、「送信先エイリアス」133 ページを参照してください。

送信先エイリアスを作成するには、その送信先エイリアスと同じ名前のクライアントにサーバー管理要求を送信します。次のプロパティを指定します。グループは、group、address、nativeConnection の各プロパティで識別されます。グループのメンバーは、member プロパティで指定します。

```
<prop>
  <client>simpleGroup</client>
  <name>ianywhere.connector.nativeConnection</name>
  <value>ianywhere.message.connector.group.GroupConnector
  </value>
</prop>
```

プロパティ	説明

ianywhere.qa.group	送信先エイリアスを設定していることを示すには、このプロパティを Y に設定します。次に例を示します。
	
	`<prop>`
	`  <client>simpleGroup</client>`
	`  <name>ianywhere.qa.group</name>`
	`  <value>Y</value>`
	`</prop>`

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

プロパティ	説明

ianywhere.connector.address	送信先エイリアスのクライアント ID を指定します。次に例を示します。
	
	`<prop>`
	`<client>simpleGroup</client>`
	`<name>ianywhere.connector.address</name>`
	`<value>simpleGroup</value>`
	`</prop>`

ianywhere.connector.nativeConnection	ianywhere.message.connector.group.GroupConnector に設定します。次に例を示します。
	
	`<prop>`
	`<client>simpleGroup</client>`
	`<name>ianywhere.connector.nativeConnection</name>`
	
	`<value>ianywhere.message.connector.group.GroupConnector`
	`</value>`
	`</prop>`

ianywhere.qa.member.*client-name¥queue-name*	メンバーを追加する場合は Y、メンバーを削除する場合は N を指定します。オプションで配信条件を指定することもできます。 「条件構文」776 ページを参照してください。
	
	たとえば、client1 を送信先エイリアス simpleGroup に追加するには、プロパティを次のように設定します。キュー名はオプションです。追加するクライアントごとに、このプロパティを繰り返し設定します。
	
	`<prop>`
	`<client>simpleGroup</client>`
	`<name>ianywhere.qa.member.client1¥queue1</name>`
	`<value>Y</value>`
	`</prop>`

参照

- 「サーバー管理要求」153 ページ
- 「QAnywhere 転送ルールと削除ルール」774 ページ

例

次のサーバー管理要求では、client1 と client2¥q11 というメンバーを持つ送信先エイリアス simpleGroup を作成します。この例では送信先エイリアスを起動して、メッセージの処理をすぐに開始します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <SetProperty>
```

```
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.qa.group</name>
      <value>Y</value>
    </prop>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.connector.address</name>
      <value>simpleGroup</value>
    </prop>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.connector.nativeConnection</name>
      <value>ianywhere.message.connector.group.GroupConnector</value>
    </prop>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.connector.logLevel</name>
      <value>4</value>
    </prop>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.qa.member.client1</name>
      <value>Y</value>
    </prop>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.qa.member.client2¥q11</name>
      <value>Y</value>
    </prop>
  </SetProperty>
  <OpenConnector>
    <client>simpleGroup</client>
  </OpenConnector>
</actions>
```

送信先エイリアスのメンバーの追加と削除

送信先エイリアスにメンバーを追加するには、プロパティでメンバーを指定するサーバー管理要求を作成します。グループを再起動してメンバー設定を有効にしてください。

次の例では、メンバー client3 を追加して、グループ simpleGroup を再起動します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <SetProperty>
  <prop>
      <client>simpleGroup</client>
      <name>ianywhere.qa.member.client3</name>
      <value>Y</value>
  </prop>
    </SetProperty>
    <CloseConnector>
  <client>simpleGroup</client>
    </CloseConnector>
    <OpenConnector>
  <client>simpleGroup</client>
    </OpenConnector>
</actions>
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

送信先エイリアスからメンバーを削除するには、メンバーを削除する必要があることを示すプロパティを持つサーバー管理要求を作成します。グループを再起動してメンバーの削除設定を有効にする必要があります。

次の例では、メンバー client3 を削除して、グループ simpleGroup を再起動します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <SetProperty>
 <prop>
    <client>simpleGroup</client>
    <name>ianywhere.qa.member.client3</name>
 </prop>
   </SetProperty>
   <CloseConnector>
 <client>simpleGroup</client>
   </CloseConnector>
   <OpenConnector>
 <client>simpleGroup</client>
   </OpenConnector>
</actions>
```

QAnywhere のモニター

サーバー管理要求を使用して、メッセージセットの情報を取得できます。サーバーは情報をコンパイルし、メッセージとしてクライアントに返信します。ワンタイムのメッセージの詳細要求を作成することも、スケジュールを設定してメッセージの詳細要求を自動的に実行することもできます。また、要求を永続的にすることを指定し、サーバーが再起動された場合でもメッセージが送信されるようにすることもできます。

メッセージの詳細要求

メッセージの詳細要求に関するサーバー管理要求を作成するには、<MessageDetailsRequest> タグを使用します。

メッセージの詳細要求には、要求の登録に必要なすべての情報を含む 1 つまたは複数の <request> タグが含まれます。複数の <request> タグを指定することは、複数のメッセージ詳細要求を別々に送信することと同じです。

この要求により生成される各レポートの返信アドレスを指定するには、オプションの <replyAddr> タグを使用します。このタグが指定されていない場合は、レポートのデフォルトの返信アドレスが送信メッセージの返信アドレスになります。

この要求により生成される各レポートに含まれる要求に対して、ユニークな識別子を指定するには、<requestId> タグを使用します。このフィールドに複数の値を指定すると、一度に複数の要求をアクティブにできます。同じ要求 ID を指定すると、クライアントはアクティブな要求を上書きまたは削除します。

レポートに含めるメッセージを特定するには、<condition> タグを指定します。

詳細リストを指定して、各メッセージのどの詳細情報をレポートに含めるかを特定することもできます。これを行うには、一連の空の詳細要素タグを要求に含めます。

サーバーのダウンタイムを通してイベントの詳細を永続的にすることを指定するには、<persistent> タグを使用します。このタグにはデータを指定する必要はありません。<persistent/> または <persistent></persistent> のどちらのフォームを使用することもできます。

スケジュール設定されたレポートの登録に必要な詳細情報をすべて含めるには、<schedule> を使用します。

<MessageDetailsRequest> タグのサブタグ	**説明**
<request>	特定の要求に関する情報をグループ化します。複数の <request> タグを指定することは、メッセージ情報に関する複数のサーバー管理要求を別々に送信することと同じです。以下を参照してください。

Request タグ

<Request> タグのサブタグ	**説明**
<address>	各メッセージのアドレスを要求します。
<archived>	メッセージがアーカイブストア内に存在するかどうかを要求します。
<condition>	メッセージをレポートに含めるための条件をグループ化します。
<contentSize>	各メッセージの内容のサイズを要求します。
<expires>	各メッセージの有効期限を要求します。
<kind>	メッセージがテキストであるかバイナリであるかを要求します。
<messageId>	各メッセージのメッセージ ID を要求します。
<originator>	各メッセージの発信者を要求します。
<persistent>	このタグを含めることで、要求の結果をサーバーデータベース内で永続的にすることを示す。これにより、サーバーが再起動された場合であってもレポートが送信されます。
<priority>	各メッセージの優先度を要求します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

<Request> タグのサブタグ	説明
<property>	すべてのメッセージプロパティと各メッセージの値のリストを要求します。
<statusTime>	各メッセージのステータス時間を要求します。
<replyAddr>	この要求により生成される各レポートの返信アドレスを指定します。このタグが指定されていない場合は、レポートのデフォルトの返信アドレスが送信メッセージの返信アドレスになります。
<requestId>	要求のユニークな識別子です。この要求により生成される各レポートに含まれます。このフィールドに複数の値を指定すると、一度に複数の要求をアクティブにできます。同じ要求 ID を指定すると、クライアントはアクティブな要求を上書きまたは削除します。
<schedule>	スケジュールに従ってレポートを生成することを指定します。<schedule> タグのサブタグは、レポートを実行するスケジュールを特定します。
<status>	各メッセージのステータスを要求します。
<transmissionStatus>	各メッセージの転送ステータスを要求します。

参照

- 「サーバー管理要求」153 ページ
- 「Condition タグ」704 ページ
- 「サーバー管理要求の親タグ」704 ページ
- 「サーバー管理要求の親タグ」704 ページ

QAnywhere クライアントのモニター

サーバー管理要求を使用して、現在サーバー上にあるクライアントのリストを取得できます。このリストには、サーバーに登録されているクライアントが含まれます。リモートクライアント、開いているコネクター、送信先エイリアスも含まれます。

クライアントのリストを取得するには、サーバー管理要求で <GetClientList> タグを使用します。次に例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
```

```
 <GetClientList/>   (or <GetClientList></GetClientList> )
</actions>
```

生成された応答は、要求が含まれているメッセージの返信先アドレスに送信されます。応答には <client> タグのリストが含まれています。タグごとに、サーバーに接続されているクライアント名が 1 つ示されています。次に例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<GetClientListResponse>
 <client>ianywhere.server</client>
 <client>ianywhere.connector.myConnector</client>
 <client>myClient</client>
</GetClientListResponse>
```

参照

- 「サーバー管理要求」153 ページ

プロパティのモニター

サーバー管理要求を使用して、クライアントに設定されているプロパティの内容を確認できます。要求の応答によってリストされるプロパティは、クライアントに設定されているプロパティだけです。デフォルトのプロパティは含まれません。

クライアントのプロパティのリストを取得するには、サーバー管理要求で <GetProperties> タグを使用します。次に例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <GetProperties>
  <client>ianywhere.connector.myConnector</client>
 </GetProperties>
</actions>
```

生成された応答は、要求が含まれているメッセージの返信先アドレスに送信されます。応答にはクライアント名と <prop> タグのリストが含まれます。タグごとに 1 つのプロパティの詳細が含まれています。次に例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<GetPropertiesResponse>
 <client>ianywhere.connector.myConnector</client>
 <prop>
  <name>ianywhere.connector.logLevel</name>
  <value>4</value>
 </prop>
 <prop>
  <name>ianywhere.connector.state</name>
  <value>2</value>
 </prop>
</GetPropertiesResponse>
```

参照

- 「サーバー管理要求」153 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

チュートリアル：TestMessage の解説

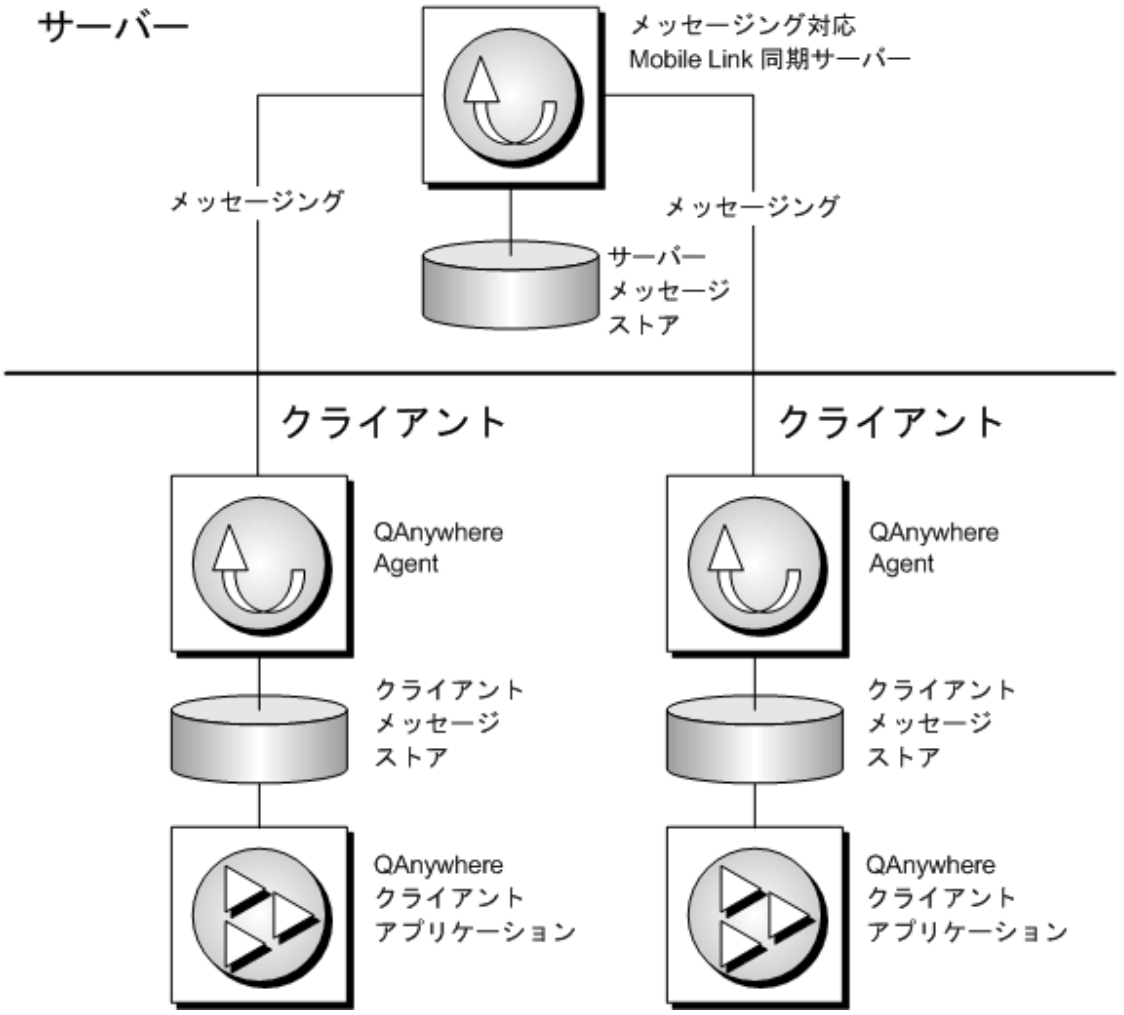
TestMessage は QAnywhere クライアントアプリケーションのサンプルです。このアプリケーションは、QAnywhere を使用してユーザー独自のメッセージクライアントアプリケーションを作成する方法を示しています。TestMessage では、単一のクライアント間インターフェイスを使用して、メッセージを送信、受信、表示します。このサンプルでは、QAnywhere メッセージングの機能をわかりやすく示すため、人間が読めるテキストメッセージを使用しています。ただし、QAnywhere はテキスト以外のメッセージも扱うことができます。QAnywhere は、多数のクライアントの間でメッ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

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

レッスン 1：メッセージング対応 Mobile Link の起動

背景

QAnywhere は、Mobile Link 同期を使用してメッセージを送受信します。クライアント間でやり取りされるメッセージは、いずれも中央の Mobile Link サーバー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ーを次の図に示します。このアーキテクチャーでは、QAnywhere クライアントが 2 つだけ使用されています。



サーバー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、サーバーメッセージストアとして SQL Anywhere 統合データベースを使用します。

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

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

通常、QAnywhere メッセージングは、複数の異なるコンピューターにまたがって実行されます。ただし、このチュートリアルでは、すべてのコンポーネントが 1 つのコンピューターで実行されます。サーバー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。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

このレッスンでは、サーバー側でアクションを実行します。

アクティビティ

Mobile Link サーバーは、-m オプションと、サーバー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、サーバーメッセージストアとして QAnywhere サンプルデータベースを使用します。
TestMessage サンプルアプリケーションを使用するときには、コマンドラインオプションを指定するか、SQL Anywhere インストール環境にあるサンプルショートカットをクリックするか、Sybase Central 用の QAnywhere 12 プラグインを使用すると、Mobile Link サーバー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ーを起動する

1. **[スタート]** » **[プログラム]** » **[SQL Anywhere 12]** » **[QAnywhere]** » **[Mobile Link QAnywhere のサンプル]** をクリックします。

   または、コマンドプロンプトで次のコマンドを実行します。

   ```
   mlsrv12 -m -c "dsn=QAnywhere 12 Demo" -vcrs -zu+
   ```

   このコマンドラインでは、以下に示す mlsrv12 のオプションを使用します。

オプション	説明
-m	メッセージングを有効にします。「-m mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。
-c	サーバメッセージストアに接続するための接続文字列を指定します。この例では、ODBC データソース QAnywhere 12 Demo が接続文字列に含まれています。「-c mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。
-vcrs	サーバーのアクティビティに関する冗長ロギングを有効にします。開発時に指定すると便利です。「-v mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。
-zu+	ユーザー名を自動的にシステムに追加します。このオプションは、チュートリアルや開発用に使用すると便利ですが、運用環境では通常使用しません。「-zu mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。

2. Mobile Link サーバーメッセージウィンドウを画面中央に移動します。これは、このチュートリアルのサーバーになります。

3. 「レッスン 2：TestMessage アプリケーションの実行」に進みます。

参照

- 「有効な Mobile Link での QAnywhere の起動」31 ページ
- 「-m mlsrv12 オプション」『Mobile Link サーバー管理』
- 「QAnywhere のクイックスタート」12 ページ
- 「簡単なメッセージングシナリオ」5 ページ

レッスン 2：TestMessage アプリケーションの実行

背景

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

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

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

アクティビティ

♦ TestMessage のクライアントメッセージストアを使用した QAnywhere Agent の起動

1. **[スタート] » [プログラム] » [SQL Anywhere 12] » [QAnywhere] » [SQL Anywhere を使用するチュートリアル] » [SQL Anywhere 用 QAnywhere Agent - saclient1]** をクリックします。

   QAnywhere Agent は TestMessage の最初のサンプルクライアントメッセージストアに接続し、そのメッセージストア間とのメッセージ転送を管理します。

2. 最初の **QAnywhere Agent** ウィンドウを画面右側に移動します。

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

3. **[スタート] » [プログラム] » [SQL Anywhere 12] » [QAnywhere] » [SQL Anywhere を使用するチュートリアル] » [SQL Anywhere 用 QAnywhere Agent - saclient2]** をクリックします。

   2 番目の **QAnywhere Agent** が起動し、TestMessage の 2 番目のサンプルクライアントメッセージストアに接続し、そのメッセージストア間とのメッセージ転送を管理します。

4. 2 番目の **QAnywhere Agent** ウィンドウを画面左側に移動します。

♦ TestMessage の起動

1. **[スタート] » [プログラム] » [SQL Anywhere 12] » [QAnywhere] » [SQL Anywhere を使用するチュートリアル] » [TestMessage -- saclient1]** をクリックします。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

2. **[saclient1 - TestMessage]** ウィンドウを画面右側に移動します。
3. **[saclient1 - TestMessage]** ウィンドウで、**[Tools]** » **[Options]** をクリックします。
4. **testmessage** が **[Queue name used to listen for incoming messages]** フィールドに表示されていることを確認します。**[Cancel]** をクリックします。
5. **[スタート]** » **[プログラム]** » **[SQL Anywhere 12]** » **[QAnywhere]** » **[SQL Anywhere を使用するチュートリアル]** » **[TestMessage -- saclient2]** をクリックします。
6. **[saclient2 - TestMessage]** ウィンドウを画面左側に移動します。
7. **[saclient2 - TestMessage]** ウィンドウで、**[Tools]** » **[Options]** をクリックします。
8. **testmessage** が **[Queue name used to listen for incoming messages]** フィールドに表示されていることを確認します。**[Cancel]** をクリックします。

説明

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

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

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

「レッスン 3：メッセージの送信」に進みます。

参照

- 「QAnywhere Agent の起動」47 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」49 ページ
- 「qaagent ユーティリティ」712 ページ
- 「QAnywhere 転送ルールと削除ルール」774 ページ
- 「QAnywhere クライアントアプリケーションの作成」51 ページ

- *%SQLANYSAMP12%¥QAnywhere* に QAnywhere のサンプルがあります。

レッスン 3：メッセージの送信

背景

このレッスンでは、TestMessage の saclient1 アプリケーションから saclient2 アプリケーションにメッセージを送信します。

アクティビティ

♦ TestMessage からのメッセージの送信

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

1. **[saclient1 - TestMessage]** ウィンドウで、**[Message]** » **[New]** をクリックします。
2. **[Destination ID]** フィールドに **saclient2** と入力します。
3. **[Subject]** フィールドに、現在時刻を入力します。現在時刻を使用すると、個々のメッセージを追跡しやすくなります。
4. **[Message]** フィールドに **sample** と入力します。
5. **[Send]** をクリックします。
6. **[saclient2 - TestMessage]** ウィンドウで、メッセージをクリックします。メッセージの内容が下ウィンドウ枠に表示されます。
7. 「レッスン 4：TestMessage クライアントのソースコードの概要」に進みます。

参照

- 「QAnywhere メッセージアドレス」60 ページ
- 「QAnywhere メッセージの送信」63 ページ
- 「メッセージの削除ルール」786 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

レッスン 4：TestMessage クライアントのソースコードの概要

背景

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

コードの多くは、メッセージを送信、受信、表示するための Windows インターフェイスの実装です。しかし、ここでは、QAnywhere の機能を実装しているコードを中心に解説していきます。

TestMessage のソースコードは、*Samples¥QAnywhere* にあります。

TestMessage のソースコードにはいくつかのバージョンがあります。以下のバージョンが Windows プラットフォーム用に提供されています。

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

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

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

必要なソフトウェア

ソリューションファイルを開いたり、.NET Framework プロジェクトや .NET Compact Framework プロジェクトをビルドしたりする場合は、Visual Studio 2005 以降が必要です。

C# ソースの確認

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

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

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

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

2. ソリューションエクスプローラーが開いていることを確認します。

   ソリューションエクスプローラーは、**[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]** フォルダーの内容を確認します。

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

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

5. インクルードされているネームスペースを調べます。

   QAnywhere アプリケーションは、いずれも iAnywhere.QAnywhere.Client ネームスペースをインクルードしている必要があります。このネームスペースを定義するアセンブリは、*iAnywhere.QAnywhere.Client.dll* という DLL として提供されています。ファイルは、次の場所にあります。

   - .NET Framework 2.0: *%SQLANY12%¥Assembly¥V2*
   - .NET Compact Framework 2.0: *%SQLANY12%¥ce¥Assembly¥V2*

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

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

6. startReceiver メソッドを調べます。

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

   - QAManager オブジェクトを作成する。

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

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

     QAnywhere には、QAManager と QATransactionalManager の 2 種類の Manager が用意されています。QATransactionalManager を使用すると、送信と受信はすべてトランザクション内で処理されます。したがって、すべてのメッセージが送信 (受信) されるか、何も送信 (受信) されないかのどちらかになります。

   - メッセージを処理するメソッドを記述する。

     システム関連メッセージではない通常のメッセージを処理するための onMessage() メソッドが、QAnywhere から呼び出されます。このメソッドが受信するメッセージは、QAMessage オブジェクトとしてエンコードされています。この QAMessage クラスとその

子クラス (QATextMessage と QABinaryMessage) のプロパティとメソッドに、QAnywhere アプリケーションが必要とするメッセージ関連情報が格納されています。

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

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

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

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

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

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

メッセージリスナー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ーに渡すようになります。1 つのキューに複数のリスナーを割り当てることはできません。キューに NULL リスナーを割り当てると、そのキューへのリスナーの割り当てが解除されます。

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

その他の初期化タスクには、次のものがあります。

●QAManager オブジェクトをオープンして開始する。

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

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

●キュー内で待ち状態にあるメッセージをすべてロードする。

```
loadMessages();
```

●以降のメッセージが着信するキューにリスナーを割り当てる。

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

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

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

7. 同じファイルにある addMessage() メソッドを調べます。

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

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

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

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

9. sendMessage() メソッドを調べます。

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

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

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

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

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

10. 「クリーンアップ」に進みます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「クライアント用 QAnywhere C++ API リファレンス」375 ページ
- 「クライアント用 QAnywhere .NET API リファレンス」189 ページ
- 「QAnywhere クライアントアプリケーションの作成」51 ページ
- TestMessage のサンプルは、*%SQLANYSAMP12%¥QAnywhere* にあります。

クリーンアップ

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

QAnywhere リファレンス

この項では、QAnywhere API の参照マニュアルを示します。

クライアント用 QAnywhere .NET API リファレンス

ネームスペース (通常のクライアントの場合)

```
iAnywhere.QAnywhere.Client
```

ネームスペース (スタンドアロンクライアントの場合)

```
iAnywhere.QAnywhere.StandAloneClient
```

MessageProperties クラス

標準のメッセージプロパティ名を格納するフィールドを提供します。

Visual Basic 構文

```
Public Class MessageProperties
```

C# 構文

```
public class MessageProperties
```

メンバー

継承されたメンバーを含む MessageProperties クラスのすべてのメンバー。

名前	説明
ADAPTER フィールド	システムキューメッセージの場合は、QAnywhere サーバーへの接続で使用されているネットワークアダプターです。
ADAPTERS フィールド	このプロパティ名は、QAnywhere サーバーへの接続で使用できるネットワークアダプターのデリミター付きリストを示します。
DELIVERY_COUNT フィールド	このプロパティ名は、メッセージを配信するために、これまでに実行された試行回数を示します。

名前	説明
IP フィールド	システムキューメッセージの場合は、QAnywhere サーバーへの接続で使用されているネットワークアダプターの IP アドレスです。
MAC フィールド	システムキューメッセージの場合は、QAnywhere サーバーへの接続で使用されているネットワークアダプターの MAC アドレスです。
MSG_TYPE フィールド	このプロパティ名は、QAnywhere メッセージに関連付けられている MessageType 値を示します。
NETWORK_STATUS フィールド	このプロパティ名は、ネットワーク接続のステータスを示します。
ORIGINATOR フィールド	このプロパティ名は、メッセージの発信者のメッセージストア ID を示します。
RAS フィールド	システムキューメッセージの場合は、QAnywhere サーバーへの接続で使用されている RAS エントリ名です。
RASNAMES フィールド	システムキューメッセージの場合は、QAnywhere サーバーに接続するときに使用できる RAS エントリ名のデリミター付きリストです。
STATUS フィールド	このプロパティ名は、メッセージの現在のステータスを示します。
STATUS_TIME フィールド	このプロパティ名は、メッセージが現在のステータスになった時刻を示します。
TRANSMISSION_STATUS フィールド	このプロパティ名は、メッセージの現在の転送ステータスを示します。

備考

MessageProperties クラスは、標準のメッセージプロパティ名を提供します。MessageProperties フィールドは、メッセージプロパティの取得と設定で使用する QAMessage メソッドに渡すことができます。

詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- QAMessage インターフェイス [QAnywhere .NET]285 ページ
- QAMessage.GetIntProperty メソッド [QAnywhere .NET]291 ページ
- QAMessage.GetStringProperty メソッド [QAnywhere .NET]294 ページ

例

次の QAMessage インスタンスがあるとします。

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

次の例では、QAMessage.GetIntProperty メソッドを使用して MessageProperties.MSG_TYPE に対応する値を取得します。MessageType 列挙体は、int 値の結果を適切なメッセージタイプにマッピングします。

```
msg_type = (MessageType)t_msg.GetIntProperty(
   MessageProperties.MSG_TYPE
);
```

次の例は、QAnywhere システムメッセージの処理で使用される onSystemMessage メソッドを示します。メッセージタイプは、MessageProperties.MSG_TYPE 変数と QAMessage.GetIntProperty メソッドを使用して評価されます。MessageProperties.RASNAMES と QAMessage.GetStringProperty メソッドを使用して、RAS エントリ名のリストが取得されます。

```
private void onSystemMessage(QAMessage msg) {
   QATextMessage    t_msg;
   MessageType      msg_type;
   String           network_adapters;
   String           network_names;
   String           network_info;

   t_msg = (QATextMessage)msg;
   if( t_msg != null ) {
        // Evaluate message type.
        msg_type = (MessageType)t_msg.GetIntProperty( MessageProperties.MSG_TYPE );

       if( msg_type == MessageType.NETWORK_STATUS_NOTIFICATION ) {
          // Handle network status notification.
          network_info = "";
          network_adapters = t_msg.GetStringProperty( MessageProperties.ADAPTERS );
          if( network_adapters != null && network_adapters.Length > 0) {
             network_info += String.Format( _resources.GetString( "NetworkAdapter" ),
network_adapters );
          }

          network_names = t_msg.GetStringProperty( MessageProperties.RASNAMES );
          //...
        }
   }
}
```

ADAPTER フィールド

システムキューメッセージの場合は、QAnywhere サーバーへの接続で使用されているネットワークアダプターです。

Visual Basic 構文

```
Public Const ADAPTER As String
```

C# 構文

```
public const string ADAPTER;
```

備考

このフィールドの値は ias_Network.Adapter です。

詳細については、「事前に定義されたクライアントメッセージストアプロパティ」761 ページを参照してください。

MessageProperties.ADAPTER を QAMessage.GetStringProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ
- QAMessage.GetStringProperty メソッド [QAnywhere .NET]294 ページ

ADAPTERS フィールド

このプロパティ名は、QAnywhere サーバーへの接続で使用できるネットワークアダプターのデリミター付きリストを示します。

Visual Basic 構文

```
Public Const ADAPTERS As String
```

C# 構文

```
public const string ADAPTERS;
```

備考

これは、システムキューメッセージで使用されます。

MessageProperties.ADAPTERS を QAMessage.GetStringProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

詳細については、「事前に定義されたクライアントメッセージストアプロパティ」761 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ
- QAMessage.GetStringProperty メソッド [QAnywhere .NET]294 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

DELIVERY_COUNT フィールド

このプロパティ名は、メッセージを配信するために、これまでに実行された試行回数を示します。

Visual Basic 構文

```
Public Const DELIVERY_COUNT As String
```

C# 構文

```
public const string DELIVERY_COUNT;
```

備考

このフィールドの値は ias_DeliveryCount です。

MessageProperties.DELIVERY_COUNT を QAMessage.GetIntProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ
- QAMessage.GetIntProperty メソッド [QAnywhere .NET]291 ページ

IP フィールド

システムキューメッセージの場合は、QAnywhere サーバーへの接続で使用されているネットワークアダプターの IP アドレスです。

Visual Basic 構文

```
Public Const IP As String
```

C# 構文

```
public const string IP;
```

備考

このフィールドの値は ias_Network.IP です。

詳細については、「事前に定義されたクライアントメッセージストアプロパティ」761 ページを参照してください。

MessageProperties.IP を QAMessage.GetStringProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ
- QAMessage.GetStringProperty メソッド [QAnywhere .NET]294 ページ

MAC フィールド

システムキューメッセージの場合は、QAnywhere サーバーへの接続で使用されているネットワークアダプターの MAC アドレスです。

Visual Basic 構文

```
Public Const MAC As String
```

C# 構文

```
public const string MAC;
```

備考

このフィールドの値は ias_Network.MAC です。

詳細については、「事前に定義されたクライアントメッセージストアプロパティ」761 ページを参照してください。

MessageProperties.MAC を QAMessage.GetStringProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ
- QAMessage.GetStringProperty メソッド [QAnywhere .NET]294 ページ

MSG_TYPE フィールド

このプロパティ名は、QAnywhere メッセージに関連付けられている MessageType 値を示します。

Visual Basic 構文

```
Public Const MSG_TYPE As String
```

C# 構文

```
public const string MSG_TYPE;
```

備考

このフィールドの値は ias_MessageType です。

MessageProperties.MSG_TYPE を QAMessage.GetIntProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ
- MessageType 列挙体 [QAnywhere .NET]318 ページ
- QAMessage.GetIntProperty メソッド [QAnywhere .NET]291 ページ
- QAMessage.GetStringProperty メソッド [QAnywhere .NET]294 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

NETWORK_STATUS フィールド

このプロパティ名は、ネットワーク接続のステータスを示します。

Visual Basic 構文

```
Public Const NETWORK_STATUS As String
```

C# 構文

```
public const string NETWORK_STATUS;
```

備考

ネットワークがアクセス可能な場合は 1、それ以外の場合は 0 です。

ネットワークステータスは、システムキューメッセージ (ネットワークステータスの変更など) で使用されます。

詳細については、「事前に定義されたクライアントメッセージストアプロパティ」761 ページを参照してください。

MessageProperties.NETWORK_STATUS を QAMessage.GetIntProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ
- QAMessage.GetIntProperty メソッド [QAnywhere .NET]291 ページ

ORIGINATOR フィールド

このプロパティ名は、メッセージの発信者のメッセージストア ID を示します。

Visual Basic 構文

```
Public Const ORIGINATOR As String
```

C# 構文

```
public const string ORIGINATOR;
```

備考

このフィールドの値は ias_Originator です。

MessageProperties.ORIGINATOR を QAMessage.GetStringProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ
- QAMessage.GetStringProperty メソッド [QAnywhere .NET]294 ページ

RAS フィールド

システムキューメッセージの場合は、QAnywhere サーバーへの接続で使用されている RAS エントリ名です。

Visual Basic 構文

```
Public Const RAS As String
```

C# 構文

```
public const string RAS;
```

備考

このフィールドの値は ias_Network.RAS です。

詳細については、「事前に定義されたクライアントメッセージストアプロパティ」761 ページを参照してください。

MessageProperties.RAS を QAMessage.GetStringProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ
- QAMessage.GetStringProperty メソッド [QAnywhere .NET]294 ページ

RASNAMES フィールド

システムキューメッセージの場合は、QAnywhere サーバーに接続するときに使用できる RAS エントリ名のデリミター付きリストです。

Visual Basic 構文

```
Public Const RASNAMES As String
```

C# 構文

```
public const string RASNAMES;
```

備考

このフィールドの値は ias_RASNames です。

詳細については、「事前に定義されたクライアントメッセージストアプロパティ」761 ページを参照してください。

MessageProperties.RASNAMES を QAMessage.GetStringProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ
- QAMessage.GetStringProperty メソッド [QAnywhere .NET]294 ページ

STATUS フィールド

このプロパティ名は、メッセージの現在のステータスを示します。

Visual Basic 構文

```
Public Const STATUS As String
```

C# 構文

```
public const string STATUS;
```

備考

プロパティ値のリストについては、StatusCodes を参照してください。

このフィールドの値は ias_Status です。

MessageProperties.STATUS を QAMessage.GetIntProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ
- QAMessage.GetIntProperty メソッド [QAnywhere .NET]291 ページ

STATUS_TIME フィールド

このプロパティ名は、メッセージが現在のステータスになった時刻を示します。

Visual Basic 構文

```
Public Const STATUS_TIME As String
```

C# 構文

```
public const string STATUS_TIME;
```

備考

これはローカル時間です。STATUS_TIME は、QAMessage.GetProperty に渡されると、DateTime オブジェクトを返します。このフィールドの値は ias_StatusTime です。

参照

- QAMessage.GetProperty メソッド [QAnywhere .NET]292 ページ
- MessageProperties クラス [QAnywhere .NET]189 ページ
- QAMessage.GetProperty メソッド [QAnywhere .NET]292 ページ

TRANSMISSION_STATUS フィールド

このプロパティ名は、メッセージの現在の転送ステータスを示します。

Visual Basic 構文

```
Public Const TRANSMISSION_STATUS As String
```

C# 構文

```
public const string TRANSMISSION_STATUS;
```

備考

プロパティ値のリストについては、StatusCodes を参照してください。

このフィールドの値は ias_TransmissionStatus です。

MessageProperties.TRANSMISSION_STATUS を QAMessage.GetIntProperty メソッドに渡して、関連付けられているプロパティにアクセスできます。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ
- QAMessage.GetIntProperty メソッド [QAnywhere .NET]291 ページ

MessageStoreProperties クラス

このクラスは、有用なメッセージストアプロパティ名の定数値を定義します。

Visual Basic 構文

```
Public Class MessageStoreProperties
```

C# 構文

```
public class MessageStoreProperties
```

メンバー

継承されたメンバーを含む MessageStoreProperties クラスのすべてのメンバー。

名前	説明
MAX_DELIVERY_ATTEMPTS フィールド	このプロパティ名は、メッセージのステータスが StatusCodes.UNRECEIVABLE に設定されるまでに、明示的な受信確認がないままメッセージを受信できる最大回数を示します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

MessageStoreProperties クラスは、標準のメッセージプロパティ名を提供します。MessageProperties フィールドは、事前定義済みまたはカスタムのメッセージストアプロパティの取得と設定で使用する QAManagerBase メソッドに渡すことができます。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

MAX_DELIVERY_ATTEMPTS フィールド

このプロパティ名は、メッセージのステータスが StatusCodes.UNRECEIVABLE に設定されるまでに、明示的な受信確認がないままメッセージを受信できる最大回数を示します。

Visual Basic 構文

```
Public Const MAX_DELIVERY_ATTEMPTS As String
```

C# 構文

```
public const string MAX_DELIVERY_ATTEMPTS;
```

備考

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

iAnywhere.QAnywhere.Client.StatusCodes MessageStoreProperties

QABinaryMessage インターフェイス

QABinaryMessage オブジェクトは、未解釈のバイトストリームが含まれるメッセージの送信に使用します。

Visual Basic 構文

```
Public Interface QABinaryMessage Inherits QAMessage
```

C# 構文

```
public interface QABinaryMessage : QAMessage
```

基本クラス

- QAMessage インターフェイス [QAnywhere .NET]285 ページ

メンバー

継承されたメンバーを含む QABinaryMessage インターフェイスのすべてのメンバー。

名前	説明
ClearBody メソッド	メッセージの本文をクリアします。

名前	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。
GetByteProperty メソッド	byte 型のメッセージプロパティを取得します。
GetDoubleProperty メソッド	double 型のメッセージプロパティを取得します。
GetFloatProperty メソッド	float 型のメッセージプロパティを取得します。
GetIntProperty メソッド	int 型のメッセージプロパティを取得します。
GetLongProperty メソッド	long 型のメッセージプロパティを取得します。
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	string 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。
ReadBinary メソッド	QABinaryMessage インスタンスの本文未読部分の先頭から length で指定されたバイト数まで読み込んで、**dest** 配列に格納します。
ReadBoolean メソッド	QABinaryMessage インスタンスのメッセージ本文未読部分の先頭から boolean 値を読み込みます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
ReadChar メソッド	QABinaryMessage メッセージ本文未読部分の先頭から char 値を読み込みます。
ReadDouble メソッド	QABinaryMessage メッセージ本文未読部分の先頭から double 値を読み込みます。
ReadFloat メソッド	QABinaryMessage メッセージ本文未読部分の先頭から float 値を読み込みます。
ReadInt メソッド	QABinaryMessage メッセージ本文未読部分の先頭から int 値を読み込みます。
ReadLong メソッド	QABinaryMessage メッセージ本文未読部分の先頭から long 値を読み込みます。
ReadSbyte メソッド	QABinaryMessage メッセージ本文未読部分の先頭から signed byte 値を読み込みます。
ReadShort メソッド	QABinaryMessage メッセージ本文未読部分の先頭から short 値を読み込みます。
ReadString メソッド	QABinaryMessage メッセージ本文未読部分の先頭から string 値を読み込みます。
Reset メソッド	メッセージをリセットして、メッセージ本文の先頭から値の読み込みを開始できるようにします。
SetBooleanProperty メソッド	boolean 型のプロパティを設定します。
SetByteProperty メソッド	byte 型のプロパティを設定します。
SetDoubleProperty メソッド	double 型のプロパティを設定します。
SetFloatProperty メソッド	float 型のプロパティを設定します。
SetIntProperty メソッド	int 型のプロパティを設定します。
SetLongProperty メソッド	long 型のプロパティを設定します。
SetProperty メソッド	プロパティを設定します。
SetSbyteProperty メソッド	signed byte 型のプロパティを設定します。
SetShortProperty メソッド	short 型のプロパティを設定します。
SetStringProperty メソッド	string 型のプロパティを設定します。

名前	説明
WriteBinary メソッド	QABinaryMessage インスタンスのメッセージ本文に、特定のオフセットを先頭として、バイト配列から length で指定されたバイト数分を追加します。
WriteBoolean メソッド	QABinaryMessage インスタンスのメッセージ本文に boolean 値を追加します。
WriteChar メソッド	QABinaryMessage インスタンスのメッセージ本文に char 値を追加します。
WriteDouble メソッド	QABinaryMessage インスタンスのメッセージ本文に double 値を追加します。
WriteFloat メソッド	QABinaryMessage インスタンスのメッセージ本文に float 値を追加します。
WriteInt メソッド	QABinaryMessage インスタンスのメッセージ本文に int 値を追加します。
WriteLong メソッド	QABinaryMessage インスタンスのメッセージ本文に long 値を追加します。
WriteSbyte バイト	QABinaryMessage インスタンスのメッセージ本文に signed byte 値を追加します。
WriteShort メソッド	QABinaryMessage インスタンスのメッセージ本文に short 値を追加します。
WriteString メソッド	QABinaryMessage インスタンスのメッセージ本文に string 値を追加します。
Address プロパティ	QAMessage インスタンスの送信先アドレスです。
BodyLength プロパティ	メッセージ本文のサイズをバイト単位で返します。
Expiration プロパティ	メッセージの有効期限を取得します。
InReplyToID プロパティ	どのメッセージへの返信であるかを示すメッセージ ID です。
MessageID プロパティ	メッセージのグローバルにユニークな ID です。
Priority プロパティ	メッセージの優先度 (0 から 9) です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
Redelivered プロパティ	受信されたが受信確認されていないメッセージであるかどうかを示します。
ReplyToAddress プロパティ	メッセージの返信先アドレスです。
Timestamp プロパティ	メッセージのタイムスタンプです。

備考

これは QAMessage クラスを継承したもので、メッセージ本文にバイトストリームが追加されます。QABinaryMessage には、メッセージ本文からのバイトストリームの読み込み／書き込みを行うためのさまざまな関数があります。

メッセージが最初に作成された時点では、メッセージ本文は書き込み専用モードになっています。メッセージ送信元のクライアントは、メッセージを送信した後、そのメッセージを保持し変更できます。ただし、それによって、送信されたメッセージが変更されることはありません。同じメッセージオブジェクトを複数回送信できます。

メッセージが受信された時点でプロバイダーは QABinaryMessage.Reset() を呼び出します。これによりメッセージ本文が読み込み専用モードになり、メッセージ本文の先頭から値の読み込みが開始されます。

例

次の例では QABinaryMessage の writeString 関数を使用して、QABinaryMessage インスタンスのメッセージ本文に文字列 "Q" と "Anywhere" を書き込みます。

```
// create a binary message instance
QABinaryMessage binary_message;
binary_message = qa_manager.CreateBinaryMessage();

// set optional message properties ...
binary_message.ReplyToAddress = "my-queue-name";

// write to the message body
binary_message->WriteString("Q");
binary_messge->WriteString("Anywhere");

// put the message in the local database, ready for sending
if(!qa_manager->putMessage( "store-id¥¥queue-name", msg )) {
      handleError();
   }
```

メッセージは QAnywhere Agent から送信されます。受信が終了すると、最初の QABinaryMessage.ReadString の呼び出しから "Q" が返され、2 番目の QABinaryMessage.ReadString の呼び出しから "Anywhere" が返されます。

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

ReadBinary メソッド

QABinaryMessage インスタンスの本文未読部分の先頭から length で指定されたバイト数まで読み込んで、dest 配列に格納します。

オーバーロードリスト

名前	説明
ReadBinary(byte[]) メソッド	QABinaryMessage インスタンスの本文未読部分の先頭からいくつかのバイト数を読み込んで、**dest** 配列に格納します。
ReadBinary(byte[], int) メソッド	QABinaryMessage インスタンスの本文未読部分の先頭から length で指定されたバイト数まで読み込んで、**dest** 配列に格納します。
ReadBinary(byte[], int, int) メソッド	QABinaryMessage インスタンスの本文未読部分の先頭から length で指定されたバイト数まで読み込んで、dest[offset] を開始位置として **dest** 配列に格納します。

ReadBinary(byte[]) メソッド

QABinaryMessage インスタンスの本文未読部分の先頭からいくつかのバイト数を読み込んで、**dest** 配列に格納します。

Visual Basic 構文

```
Public Function ReadBinary(ByVal dest As Byte()) As Integer
```

C# 構文

```
public int ReadBinary(byte[] dest)
```

パラメーター

- **dest**　読み込まれるバイトに含まれるバイト配列。

戻り値

メッセージ本文から読み込まれたバイト数。読み込めるバイトが残っていない場合は -1。

例外

- **QAException クラス**　メッセージからのバイトの読み込み時にエラーが発生した場合にスローされます。

備考

ReadBinary(dest) メソッドには、ReadBinary(dest,0,dest.Length) と同じ効果があります。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.WriteBinary メソッド [QAnywhere .NET]211 ページ

ReadBinary(byte[], int) メソッド

QABinaryMessage インスタンスの本文未読部分の先頭から length で指定されたバイト数まで読み込んで、**dest** 配列に格納します。

Visual Basic 構文

```
Public Function ReadBinary(
     ByVal dest As Byte(),
    ByVal length As Integer
) As Integer
```

C# 構文

```
public int ReadBinary(byte[] dest, int length)
```

パラメーター

- **dest**　読み込まれるバイトに含まれるバイト配列。
- **length**　読み込むバイト数の最大値。

戻り値

メッセージ本文から読み込まれたバイト数。読み込めるバイトが残っていない場合は -1。

例外

- **QAException クラス**　バイトの読み込み時にエラーが発生した場合にスローされます。

備考

ReadBinary(dest,len) メソッドには、ReadBinary(dest,0,len) と同じ効果があります。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.WriteBinary メソッド [QAnywhere .NET]211 ページ

ReadBinary(byte[], int, int) メソッド

QABinaryMessage インスタンスの本文未読部分の先頭から length で指定されたバイト数まで読み込んで、dest[offset] を開始位置として **dest** 配列に格納します。

Visual Basic 構文

```
Public Function ReadBinary(
     ByVal dest As Byte(),
    ByVal offset As Integer,
```

```
    ByVal length As Integer
) As Integer
```

C# 構文

```
public int ReadBinary(byte[] dest, int offset, int length)
```

パラメーター

- **dest**　読み込まれるバイトに含まれるバイト配列。
- **offset**　コピー先の配列の開始オフセット。
- **length**　読み込むバイト数の最大値。

戻り値

メッセージ本文から読み込まれたバイト数。読み込めるバイトが残っていない場合は -1。

例外

- **QAException クラス**　値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

備考

dest が NULL の場合、ArgumentNullException がスローされます。offset が負の場合、length が負の場合、または offset+length が dest の長さを超過する場合は、ArgumentOutOfRangeException がスローされます。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.WriteBinary メソッド [QAnywhere .NET]211 ページ

ReadBoolean メソッド

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

Visual Basic 構文

```
Public Function ReadBoolean() As Boolean
```

C# 構文

```
public bool ReadBoolean()
```

戻り値

メッセージ本文から読み込まれる boolean 値。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例外

- **QAException クラス**　値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.WriteBoolean メソッド [QAnywhere .NET]214 ページ

ReadChar メソッド

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

Visual Basic 構文

```
Public Function ReadChar() As Char
```

C# 構文

```
public char ReadChar()
```

戻り値

メッセージ本文から読み込まれる char 値。

例外

- **QAException クラス**　値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.WriteChar メソッド [QAnywhere .NET]214 ページ

ReadDouble メソッド

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

Visual Basic 構文

```
Public Function ReadDouble() As Double
```

C# 構文

```
public double ReadDouble()
```

戻り値

メッセージ本文から読み込まれる double 値。

例外

- **QAException クラス**　値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.WriteDouble メソッド [QAnywhere .NET]215 ページ

ReadFloat メソッド

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

Visual Basic 構文

```
Public Function ReadFloat() As Single
```

C# 構文

```
public float ReadFloat()
```

戻り値

メッセージ本文から読み込まれる float 値。

例外

- **QAException クラス**　値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.WriteFloat メソッド [QAnywhere .NET]215 ページ

ReadInt メソッド

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

Visual Basic 構文

```
Public Function ReadInt() As Integer
```

C# 構文

```
public int ReadInt()
```

戻り値

メッセージ本文から読み込まれる int 値。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例外

- **QAException クラス**　値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.WriteInt メソッド [QAnywhere .NET]215 ページ

ReadLong メソッド

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

Visual Basic 構文

```
Public Function ReadLong() As Long
```

C# 構文

```
public long ReadLong()
```

戻り値

メッセージ本文から読み込まれる long 値。

例外

- **QAException クラス**　値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.WriteLong メソッド [QAnywhere .NET]216 ページ

ReadSbyte メソッド

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

Visual Basic 構文

```
Public Function ReadSbyte() As SByte
```

C# 構文

```
public sbyte ReadSbyte()
```

戻り値

メッセージ本文から読み込まれる signed byte 値。

例外

- **QAException クラス**　値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.WriteSbyte メソッド [QAnywhere .NET]216 ページ

ReadShort メソッド

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

Visual Basic 構文

```
Public Function ReadShort() As Short
```

C# 構文

```
public short ReadShort()
```

戻り値

メッセージ本文から読み込まれる short 値。

例外

- **QAException クラス**　値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.WriteShort メソッド [QAnywhere .NET]217 ページ

ReadString メソッド

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

Visual Basic 構文

```
Public Function ReadString() As String
```

C# 構文

```
public string ReadString()
```

戻り値

メッセージ本文から読み込まれる string 値。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例外

- **QAException クラス**　値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.WriteString メソッド [QAnywhere .NET]217 ページ

Reset メソッド

メッセージをリセットして、メッセージ本文の先頭から値の読み込みを開始できるようにします。

Visual Basic 構文

```
Public Sub Reset()
*
```

C# 構文

```
public void Reset()
```

備考

また、Reset メソッドは、QABinaryMessage のメッセージ本文を読み込み専用モードにします。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ

WriteBinary メソッド

QABinaryMessage インスタンスのメッセージ本文に、特定のオフセットを先頭として、バイト配列から length で指定されたバイト数分を追加します。

オーバーロードリスト

名前	説明
WriteBinary(byte[]) メソッド	QABinaryMessage インスタンスのメッセージ本文に byte 配列値を追加します。
WriteBinary(byte[], int) メソッド	QABinaryMessage インスタンスのメッセージ本文に、バイト配列から length で指定されたバイト数分を追加します。

名前	説明
WriteBinary(byte[], int, int) メソッド	QABinaryMessage インスタンスのメッセージ本文に、特定のオフセットを先頭として、バイト配列から length で指定されたバイト数分を追加します。

WriteBinary(byte[]) メソッド

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

Visual Basic 構文

```
Public Sub WriteBinary(ByVal val As Byte())
```

C# 構文

```
public void WriteBinary(byte[] val)
```

パラメーター

- **val** メッセージ本文に書き込む byte 配列値。

例外

- **QAException クラス** メッセージ本文へのバイト配列の追加で問題が発生した場合にスローされます。

備考

WriteBinary(val) メソッドには、WriteBinary(val,0,val.length) と同じ効果があります。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.ReadBinary メソッド [QAnywhere .NET]204 ページ

WriteBinary(byte[], int) メソッド

QABinaryMessage インスタンスのメッセージ本文に、バイト配列から length で指定されたバイト数分を追加します。

Visual Basic 構文

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

C# 構文

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

パラメーター

- **val** メッセージ本文に書き込む byte 配列値。

- **length**　書き込まれるバイト数。

例外

- **QAException クラス**　メッセージ本文へのバイト配列の追加で問題が発生した場合にスローされます。

備考

WriteBinary(val,len) メソッドには、WriteBinary(val,0,len) と同じ効果があります。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.ReadBinary メソッド [QAnywhere .NET]204 ページ

WriteBinary(byte[], int, int) メソッド

QABinaryMessage インスタンスのメッセージ本文に、特定のオフセットを先頭として、バイト配列から length で指定されたバイト数分を追加します。

Visual Basic 構文

```
Public Sub WriteBinary(
     ByVal val As Byte(),
    ByVal offset As Integer,
    ByVal length As Integer
)
```

C# 構文

```
public void WriteBinary(byte[] val, int offset, int length)
```

パラメーター

- **val**　メッセージ本文に書き込む byte 配列値。
- **offset**　byte 配列内でのオフセット (書き込み開始位置)。
- **length**　書き込まれるバイト数。

例外

- **QAException クラス**　メッセージ本文へのバイト配列の追加で問題が発生した場合にスローされます。

備考

val が NULL の場合、ArgumentNullException がスローされます。offset が負の場合、length が負の場合、または length+offset が val の長さを超過する場合は、ArgumentOutOfRangeException がスローされます。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.ReadBinary メソッド [QAnywhere .NET]204 ページ

WriteBoolean メソッド

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

Visual Basic 構文

```
Public Sub WriteBoolean(ByVal val As Boolean)
```

C# 構文

```
public void WriteBoolean(bool val)
```

パラメーター

- **val** メッセージ本文に書き込む boolean 値。

備考

boolean 値は 1 バイトの値で示されます。true は 1、false は 0 で示されます。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.ReadBoolean メソッド [QAnywhere .NET]206 ページ

WriteChar メソッド

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

Visual Basic 構文

```
Public Sub WriteChar(ByVal val As Char)
```

C# 構文

```
public void WriteChar(char val)
```

パラメーター

- **val** メッセージ本文に書き込む char 値。

備考

char は 2 バイトの値で示され、上位バイトが最初に追加されます。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.ReadChar メソッド [QAnywhere .NET]207 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

WriteDouble メソッド

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

Visual Basic 構文

```
Public Sub WriteDouble(ByVal val As Double)
```

C# 構文

```
public void WriteDouble(double val)
```

パラメーター

- **val** メッセージ本文に書き込む double 値。

備考

double 値は 8 バイトの long 値に変換されて、上位のバイトから追加されます。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.ReadDouble メソッド [QAnywhere .NET]207 ページ

WriteFloat メソッド

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

Visual Basic 構文

```
Public Sub WriteFloat(ByVal val As Single)
```

C# 構文

```
public void WriteFloat(float val)
```

パラメーター

- **val** メッセージ本文に書き込む float 値。

備考

float パラメーターは 4 バイトの int に変換され、上位のバイトから追加されます。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.ReadFloat メソッド [QAnywhere .NET]208 ページ

WriteInt メソッド

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

Visual Basic 構文

```
Public Sub WriteInt(ByVal val As Integer)
```

C# 構文

```
public void WriteInt(int val)
```

パラメーター

- **val** メッセージ本文に書き込む int 値。

備考

int パラメーターは 4 バイトの値で示され、上位のバイトから追加されます。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.ReadInt メソッド [QAnywhere .NET]208 ページ

WriteLong メソッド

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

Visual Basic 構文

```
Public Sub WriteLong(ByVal val As Long)
```

C# 構文

```
public void WriteLong(long val)
```

パラメーター

- **val** メッセージ本文に書き込む long 値。

備考

long パラメーターは 8 バイトの値で示され、上位のバイトから追加されます。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.ReadLong メソッド [QAnywhere .NET]209 ページ

WriteSbyte バイト

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

Visual Basic 構文

```
Public Sub WriteSbyte(ByVal val As SByte)
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

C# 構文

```
public void WriteSbyte(sbyte val)
```

パラメーター

- **val** メッセージ本文に書き込む signed byte 値。

備考

signed byte 値は 1 バイトの値で示されます。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.ReadSbyte メソッド [QAnywhere .NET]209 ページ

WriteShort メソッド

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

Visual Basic 構文

```
Public Sub WriteShort(ByVal val As Short)
```

C# 構文

```
public void WriteShort(short val)
```

パラメーター

- **val** メッセージ本文に書き込む short 値。

備考

short パラメーターは 2 バイトの値で示され、上位のバイトから追加されます。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.ReadShort メソッド [QAnywhere .NET]210 ページ

WriteString メソッド

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

Visual Basic 構文

```
Public Sub WriteString(ByVal val As String)
```

C# 構文

```
public void WriteString(string val)
```

パラメーター

- **val**　メッセージ本文に書き込む string 値。

備考

> **注意**
> 受信側アプリケーションは、WriteString が呼び出されるたびに QABinaryMessage.ReadString を呼び出す必要があります。

> **注意**
> 文字列の UTF-8 表記は、最大で 32767 バイトまで書き込み可能です。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QABinaryMessage.ReadString メソッド [QAnywhere .NET]210 ページ

BodyLength プロパティ

メッセージ本文のサイズをバイト単位で返します。

Visual Basic 構文

```
Public ReadOnly Property BodyLength As Long
```

C# 構文

```
public long BodyLength {get;}
```

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ

QAException クラス

QAnywhere クライアントアプリケーションの例外をカプセル化します。

Visual Basic 構文

```
Public MustInherit Class QAException Inherits System.Exception
```

C# 構文

```
public abstract class QAException : System.Exception
```

基本クラス

- System.Exception

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

メンバー

継承されたメンバーを含む QAException クラスのすべてのメンバー。

名前	説明
GetBaseException method(System.Exception から継承)	派生クラスで上書きされる場合は、後続の 1 つ以上の例外の根本原因となる System.Exception を返します。
GetObjectData method(System.Exception から継承)	派生クラスで上書きされる場合は、System.Runtime.Serialization.SerializationInfo に例外の情報を設定します。
GetType method(System.Exception から継承)	現在のインスタンスのランタイムタイプを取得します。
ToString method(System.Exception から継承)	現在の例外を表す文字列を作成して返します。
Data property(System.Exception から継承)	例外に関するユーザー定義の追加情報を提供するキー／値のペアのコレクションを取得します。
DetailedMessage プロパティ	例外の詳細エラーメッセージ。
ErrorCode プロパティ	例外のエラーコード。
HelpLink property(System.Exception から継承)	この例外に関連付けられたヘルプファイルへのリンクを取得または設定します。
HResult property(System.Exception から継承)	特定の例外に割り当てられた、コーディングされた数値である HRESULT を取得または設定します。
InnerException property(System.Exception から継承)	現在の例外を発生させた System.Exception インスタンスを取得します。
Message property(System.Exception から継承)	現在の例外を説明するメッセージを取得します。
NativeErrorCode プロパティ	例外のネイティブエラーコード。
Source property(System.Exception から継承)	エラーを発生させたアプリケーションまたはオブジェクトの名前を取得または設定します。
StackTrace property(System.Exception から継承)	現在の例外がスローされた時点のコールスタックのフレームを表す文字列を取得します。

名前	説明
TargetSite property(System.Exception から継承)	現在の例外をスローしたメソッドを取得します。
COMMON_ALREADY_OPEN_ERROR フィールド	QAManager はすでに開いています。
COMMON_GET_INIT_FILE_ERROR フィールド	クライアントのプロパティファイルにアクセスできません。
COMMON_GET_PROPERTY_ERROR フィールド	メッセージストアからプロパティを取得中にエラーが発生しました。
COMMON_GETQUEUEDEPTH_ERROR フィールド	キューの長さの取得中にエラーが発生しました。
COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG フィールド	フィルターが ALL の場合、指定された宛先には QAManagerBase.getQueueDepth を使用できません。
COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID フィールド	メッセージストア ID が設定されていない場合は、QAManagerBase.getQueueDepth を使用できません。
COMMON_INIT_ERROR フィールド	初期化エラーです。
COMMON_INIT_THREAD_ERROR フィールド	バックグラウンドスレッドを初期化中にエラーが発生しました。
COMMON_INVALID_PROPERTY フィールド	クライアントのプロパティファイル内に無効なプロパティが存在します。
COMMON_MSG_ACKNOWLEDGE_ERROR フィールド	メッセージの確認中にエラーが発生しました。
COMMON_MSG_CANCEL_ERROR フィールド	メッセージのキャンセル中にエラーが発生しました。
COMMON_MSG_CANCEL_ERROR_SENT フィールド	メッセージのキャンセル中にエラーが発生しました。
COMMON_MSG_NOT_WRITEABLE_ERROR フィールド	読み込み専用モードのメッセージには書き込みできません。
COMMON_MSG_RETRIEVE_ERROR フィールド	クライアントメッセージストアからメッセージを取得中にエラーが発生しました。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
COMMON_MSG_STORE_ERROR フィールド	メッセージをクライアントメッセージストアに格納中にエラーが発生しました。
COMMON_MSG_STORE_NOT_INITIALIZED フィールド	メッセージストアがメッセージを送るために初期化されていません。
COMMON_MSG_STORE_TOO_LARGE フィールド	メッセージストアがデバイスのディスク空き容量と比べて大きすぎます。
COMMON_NO_DEST_ERROR フィールド	宛先が指定されていません。
COMMON_NO_IMPLEMENTATION フィールド	メソッドが実装されていません。
COMMON_NOT_OPEN_ERROR フィールド	QAManager が開いていません。
COMMON_OPEN_ERROR フィールド	メッセージストアへの接続を開くときにエラーが発生しました。
COMMON_OPEN_LOG_FILE_ERROR フィールド	ログファイルを開くときにエラーが発生しました。
COMMON_OPEN_MAXTHREADS_ERROR フィールド	サーバーの最大同時要求数が少なすぎるため QAManager を開くことができません (データベースサーバーの -gn オプションを参照)。
COMMON_REOPEN_ERROR フィールド	メッセージストアへの接続を再び開くときにエラーが発生しました。
COMMON_SELECTOR_SYNTAX_ERROR フィールド	指定されたセレクターに構文エラーがあります。
COMMON_SET_PROPERTY_ERROR フィールド	プロパティをメッセージストアに格納中にエラーが発生しました。
COMMON_TERMINATE_ERROR フィールド	終了エラーです。
COMMON_UNEXPECTED_EOM_ERROR フィールド	予期しないメッセージの終わりに到達しました。
COMMON_UNREPRESENTABLE_TIMESTAMP フィールド	タイムスタンプが許容範囲外です。
QA_NO_ERROR フィールド	エラーはありません。

備考

QAException クラスを使用して QAnywhere の例外をキャッチできます。

例

次のメソッドでは、QAException クラスを使用して QAnywhere の例外が受信されます。

```
public static void startReceiver() {
    _mainWindow._messageList.Items.Clear();
    _mainWindow._detailWindow.Text = "";

    try {
        _qaManager = QAManagerFactory.Instance.CreateQAManager(null);
        _qaManager.Open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
        _qaManager.Start();
        _mainWindow.loadMessages();
        _qaManager.SetMessageListener(Options.getReceiveQueueName(),_receiveListener);
        _qaManager.SetMessageListener("system", _systemListener);
    }
    catch(QAException e) {
        MessageBox.Show("Error code: " + e.ErrorCode );
        MessageBox.Show("Error message: " + e.Message );
    }
}
```

DetailedMessage プロパティ

例外の詳細エラーメッセージ。

Visual Basic 構文

```
Public ReadOnly Property DetailedMessage As String
```

C# 構文

```
public abstract string DetailedMessage {get;}
```

ErrorCode プロパティ

例外のエラーコード。

Visual Basic 構文

```
Public ReadOnly Property ErrorCode As Integer
```

C# 構文

```
public abstract int ErrorCode {get;}
```

NativeErrorCode プロパティ

例外のネイティブエラーコード。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

Visual Basic 構文

```
Public ReadOnly Property NativeErrorCode As Integer
```

C# 構文

```
public abstract int NativeErrorCode {get;}
```

COMMON_ALREADY_OPEN_ERROR フィールド

QAManager はすでに開いています。

Visual Basic 構文

```
Public Const COMMON_ALREADY_OPEN_ERROR As Integer
```

C# 構文

```
public const int COMMON_ALREADY_OPEN_ERROR;
```

参照

- QAManager インターフェイス [QAnywhere .NET]231 ページ

COMMON_GET_INIT_FILE_ERROR フィールド

クライアントのプロパティファイルにアクセスできません。

Visual Basic 構文

```
Public Const COMMON_GET_INIT_FILE_ERROR As Integer
```

C# 構文

```
public const int COMMON_GET_INIT_FILE_ERROR;
```

COMMON_GET_PROPERTY_ERROR フィールド

メッセージストアからプロパティを取得中にエラーが発生しました。

Visual Basic 構文

```
Public Const COMMON_GET_PROPERTY_ERROR As Integer
```

C# 構文

```
public const int COMMON_GET_PROPERTY_ERROR;
```

COMMON_GETQUEUEDEPTH_ERROR フィールド

キューの長さの取得中にエラーが発生しました。

Visual Basic 構文

```
Public Const COMMON_GETQUEUEDEPTH_ERROR As Integer
```

C# 構文

```
public const int COMMON_GETQUEUEDEPTH_ERROR;
```

参照

- QAManagerBase.GetQueueDepth メソッド [QAnywhere .NET]258 ページ

COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG フィールド

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

Visual Basic 構文

```
Public Const COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG As Integer
```

C# 構文

```
public const int COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG;
```

参照

- QAManagerBase.GetQueueDepth メソッド [QAnywhere .NET]258 ページ

COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID フィールド

メッセージストア ID が設定されていない場合は、QAManagerBase.getQueueDepth を使用できません。

Visual Basic 構文

```
Public Const COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID As Integer
```

C# 構文

```
public const int COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID;
```

参照

- QAManagerBase.GetQueueDepth メソッド [QAnywhere .NET]258 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

COMMON_INIT_ERROR フィールド

初期化エラーです。

Visual Basic 構文

```
Public Const COMMON_INIT_ERROR As Integer
```

C# 構文

```
public const int COMMON_INIT_ERROR;
```

COMMON_INIT_THREAD_ERROR フィールド

バックグラウンドスレッドを初期化中にエラーが発生しました。

Visual Basic 構文

```
Public Const COMMON_INIT_THREAD_ERROR As Integer
```

C# 構文

```
public const int COMMON_INIT_THREAD_ERROR;
```

COMMON_INVALID_PROPERTY フィールド

クライアントのプロパティファイル内に無効なプロパティが存在します。

Visual Basic 構文

```
Public Const COMMON_INVALID_PROPERTY As Integer
```

C# 構文

```
public const int COMMON_INVALID_PROPERTY;
```

COMMON_MSG_ACKNOWLEDGE_ERROR フィールド

メッセージの確認中にエラーが発生しました。

Visual Basic 構文

```
Public Const COMMON_MSG_ACKNOWLEDGE_ERROR As Integer
```

C# 構文

```
public const int COMMON_MSG_ACKNOWLEDGE_ERROR;
```

COMMON_MSG_CANCEL_ERROR フィールド

メッセージのキャンセル中にエラーが発生しました。

Visual Basic 構文

```
Public Const COMMON_MSG_CANCEL_ERROR As Integer
```

C# 構文

```
public const int COMMON_MSG_CANCEL_ERROR;
```

COMMON_MSG_CANCEL_ERROR_SENT フィールド

メッセージのキャンセル中にエラーが発生しました。

Visual Basic 構文

```
Public Const COMMON_MSG_CANCEL_ERROR_SENT As Integer
```

C# 構文

```
public const int COMMON_MSG_CANCEL_ERROR_SENT;
```

備考

すでに送信されたメッセージをキャンセルすることはできません。

COMMON_MSG_NOT_WRITEABLE_ERROR フィールド

読み込み専用モードのメッセージには書き込みできません。

Visual Basic 構文

```
Public Const COMMON_MSG_NOT_WRITEABLE_ERROR As Integer
```

C# 構文

```
public const int COMMON_MSG_NOT_WRITEABLE_ERROR;
```

COMMON_MSG_RETRIEVE_ERROR フィールド

クライアントメッセージストアからメッセージを取得中にエラーが発生しました。

Visual Basic 構文

```
Public Const COMMON_MSG_RETRIEVE_ERROR As Integer
```

C# 構文

```
public const int COMMON_MSG_RETRIEVE_ERROR;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

COMMON_MSG_STORE_ERROR フィールド

メッセージをクライアントメッセージストアに格納中にエラーが発生しました。

Visual Basic 構文

```
Public Const COMMON_MSG_STORE_ERROR As Integer
```

C# 構文

```
public const int COMMON_MSG_STORE_ERROR;
```

COMMON_MSG_STORE_NOT_INITIALIZED フィールド

メッセージストアがメッセージを送るために初期化されていません。

Visual Basic 構文

```
Public Const COMMON_MSG_STORE_NOT_INITIALIZED As Integer
```

C# 構文

```
public const int COMMON_MSG_STORE_NOT_INITIALIZED;
```

COMMON_MSG_STORE_TOO_LARGE フィールド

メッセージストアがデバイスのディスク空き容量と比べて大きすぎます。

Visual Basic 構文

```
Public Const COMMON_MSG_STORE_TOO_LARGE As Integer
```

C# 構文

```
public const int COMMON_MSG_STORE_TOO_LARGE;
```

COMMON_NO_DEST_ERROR フィールド

宛先が指定されていません。

Visual Basic 構文

```
Public Const COMMON_NO_DEST_ERROR As Integer
```

C# 構文

```
public const int COMMON_NO_DEST_ERROR;
```

COMMON_NO_IMPLEMENTATION フィールド

メソッドが実装されていません。

Visual Basic 構文

```
Public Const COMMON_NO_IMPLEMENTATION As Integer
```

C# 構文

```
public const int COMMON_NO_IMPLEMENTATION;
```

COMMON_NOT_OPEN_ERROR フィールド

QAManager が開いていません。

Visual Basic 構文

```
Public Const COMMON_NOT_OPEN_ERROR As Integer
```

C# 構文

```
public const int COMMON_NOT_OPEN_ERROR;
```

参照

- QAManager インターフェイス [QAnywhere .NET]231 ページ

COMMON_OPEN_ERROR フィールド

メッセージストアへの接続を開くときにエラーが発生しました。

Visual Basic 構文

```
Public Const COMMON_OPEN_ERROR As Integer
```

C# 構文

```
public const int COMMON_OPEN_ERROR;
```

COMMON_OPEN_LOG_FILE_ERROR フィールド

ログファイルを開くときにエラーが発生しました。

Visual Basic 構文

```
Public Const COMMON_OPEN_LOG_FILE_ERROR As Integer
```

C# 構文

```
public const int COMMON_OPEN_LOG_FILE_ERROR;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

COMMON_OPEN_MAXTHREADS_ERROR フィールド

サーバーの最大同時要求数が少なすぎるため QAManager を開くことができません (データベースサーバーの -gn オプションを参照)。

Visual Basic 構文

```
Public Const COMMON_OPEN_MAXTHREADS_ERROR As Integer
```

C# 構文

```
public const int COMMON_OPEN_MAXTHREADS_ERROR;
```

COMMON_REOPEN_ERROR フィールド

メッセージストアへの接続を再び開くときにエラーが発生しました。

Visual Basic 構文

```
Public Const COMMON_REOPEN_ERROR As Integer
```

C# 構文

```
public const int COMMON_REOPEN_ERROR;
```

COMMON_SELECTOR_SYNTAX_ERROR フィールド

指定されたセレクターに構文エラーがあります。

Visual Basic 構文

```
Public Const COMMON_SELECTOR_SYNTAX_ERROR As Integer
```

C# 構文

```
public const int COMMON_SELECTOR_SYNTAX_ERROR;
```

COMMON_SET_PROPERTY_ERROR フィールド

プロパティをメッセージストアに格納中にエラーが発生しました。

Visual Basic 構文

```
Public Const COMMON_SET_PROPERTY_ERROR As Integer
```

C# 構文

```
public const int COMMON_SET_PROPERTY_ERROR;
```

COMMON_TERMINATE_ERROR フィールド

終了エラーです。

Visual Basic 構文

```
Public Const COMMON_TERMINATE_ERROR As Integer
```

C# 構文

```
public const int COMMON_TERMINATE_ERROR;
```

COMMON_UNEXPECTED_EOM_ERROR フィールド

予期しないメッセージの終わりに到達しました。

Visual Basic 構文

```
Public Const COMMON_UNEXPECTED_EOM_ERROR As Integer
```

C# 構文

```
public const int COMMON_UNEXPECTED_EOM_ERROR;
```

COMMON_UNREPRESENTABLE_TIMESTAMP フィールド

タイムスタンプが許容範囲外です。

Visual Basic 構文

```
Public Const COMMON_UNREPRESENTABLE_TIMESTAMP As Integer
```

C# 構文

```
public const int COMMON_UNREPRESENTABLE_TIMESTAMP;
```

QA_NO_ERROR フィールド

エラーはありません。

Visual Basic 構文

```
Public Const QA_NO_ERROR As Integer
```

C# 構文

```
public const int QA_NO_ERROR;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

QAManager インターフェイス

QAManager クラスは QAManagerBase から派生し、非トランザクション指向の QAnywhere メッセージング操作を管理します。

Visual Basic 構文

```
Public Interface QAManager Inherits QAManagerBase
```

C# 構文

```
public interface QAManager : QAManagerBase
```

基本クラス

- QAManagerBase インターフェイス [QAnywhere .NET]239 ページ

メンバー

継承されたメンバーを含む QAManager インターフェイスのすべてのメンバー。

名前	説明
Acknowledge メソッド	クライアントアプリケーションが QAnywhere メッセージを正常に受信したことを確認します。
AcknowledgeAll メソッド	クライアントアプリケーションが QAnywhere メッセージを正常に受信したことを確認します。
AcknowledgeUntil メソッド	指定された QAMessage インスタンスと、指定されたメッセージよりも前に受信されて受信確認されていないメッセージについて、すべて受信確認します。
BrowseMessages メソッド	この方法は使用されなくなりました。
BrowseMessagesByID メソッド	指定されたメッセージ ID を持つメッセージを参照します。
BrowseMessagesByQueue メソッド	指定されたアドレスに送信され、次以降に取得可能な一連の受信待機中メッセージを参照します。
BrowseMessagesBySelector メソッド	メッセージストアのキューに登録されているメッセージのうち、指定されたセレクターを満たすメッセージを参照します。
CancelMessage メソッド	指定されたメッセージ ID を持つメッセージをキャンセルします。

名前	説明
Close メソッド	QAnywhere メッセージシステムへの接続をクローズして、QAManagerBase で使用していたリソースをすべて解放します。
CreateBinaryMessage メソッド	QABinaryMessage オブジェクトを作成します。
CreateTextMessage メソッド	QATextMessage オブジェクトを作成します。
GetBooleanStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの boolean 値を取得します。
GetDoubleStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの double 値を取得します。
GetFloatStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの float 値を取得します。
GetIntStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの int 値を取得します。
GetLongStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの long 値を取得します。
GetMessage メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
GetMessageBySelector メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
GetMessageBySelectorNoWait メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
GetMessageBySelectorTimeout メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
GetMessageNoWait メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
GetMessageTimeout メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
GetQueueDepth メソッド	指定されたフィルターに基づいて、すべてのキューの長さの合計を返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
GetSbyteStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの signed byte 値を取得します。
GetShortStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの short 値を取得します。
GetStoreProperty メソッド	メッセージストアプロパティを示す System.Object を取得します。
GetStorePropertyNames メソッド	メッセージストアのプロパティ名の列挙子を取得します。
GetStringStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの string 値を取得します。
Open メソッド	指定された AcknowledgementMode 値を使用して QAManager をオープンします。
PropertyExists メソッド	指定されたプロパティに値が現在存在するかどうかをテストします。
PutMessage メソッド	別の QAnywhere クライアントに送信するメッセージを準備します。
PutMessageTimeToLive メソッド	別の QAnywhere クライアントに送信するメッセージを準備します。
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 StatusCodes.PENDING のステータスにします。
ReOpen メソッド	QAManagerBase を再度開きます。
SetBooleanStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを boolean 値に設定します。
SetDoubleStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを double 値に設定します。
SetExceptionListener メソッド	QAnywhere メッセージの非同期的な処理時に QAExceptions を受信するように、ExceptionListener デリゲートを設定します。
SetExceptionListener2 メソッド	QAnywhere メッセージの非同期的な処理時に QAExceptions を受信するように、ExceptionListener2 デリゲートを設定します。

名前	説明
SetFloatStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを float 値に設定します。
SetIntStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを int 値に設定します。
SetLongStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを long 値に設定します。
SetMessageListener メソッド	QAnywhere メッセージを非同期に受信するように、MessageListener デリゲートを設定します。
SetMessageListener2 メソッド	QAnywhere メッセージを非同期的に受信するように、MessageListener2 デリゲートを設定します。
SetMessageListenerBySelector メソッド	メッセージセレクターを使用して、QAnywhere メッセージを非同期的に受信するように、MessageListener デリゲートを設定します。
SetMessageListenerBySelector2 メソッド	メッセージセレクターを使用して、QAnywhere メッセージを非同期的に受信するように、MessageListener2 デリゲートを設定します。
SetProperty メソッド	QAnywhere Manager の設定プロパティをプログラムで設定できるようにします。
SetSbyteStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを sbyte (signed byte) 値に設定します。
SetShortStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを short 値に設定します。
SetStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを System.Object 値に設定します。
SetStringStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを string 値に設定します。
Start メソッド	着信メッセージをメッセージリスナーで受信するための QAManagerBase を起動します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
Stop メソッド	QAManagerBase による着信メッセージの受信を停止します。
TriggerSendReceive メソッド	QAnywhere メッセージサーバーとの同期処理を発生させて、他のクライアント宛てのメッセージをアップロードし、ローカルクライアント宛てのメッセージをダウンロードします。
Mode プロパティ	受信したメッセージの QAManager 受信確認モードを返します。

備考

この動作の完全な説明については、QAManagerBase を参照してください。

QAManager は、AcknowledgementMode クラスの定義のように、明示的に受信を確認するように設定することも、暗黙的に受信を確認するように設定することもできます。トランザクションの一部としてメッセージの受信を確認するには、QATransactionalManager を使用します。QAManager と QATransactionalManager オブジェクトを作成するには、QAManagerFactory を使用します。

参照

- AcknowledgementMode 列挙体 [QAnywhere .NET]317 ページ
- QATransactionalManager インターフェイス [QAnywhere .NET]309 ページ

Acknowledge メソッド

クライアントアプリケーションが QAnywhere メッセージを正常に受信したことを確認します。

Visual Basic 構文

```
Public Sub Acknowledge(ByVal msg As QAMessage)
```

C# 構文

```
public void Acknowledge(QAMessage msg)
```

パラメーター

- **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で問題が発生した場合にスローされます。

備考

> **注意**
> QAMessage が受信確認されると、そのステータスプロパティは StatusCodes.RECEIVED に変わります。QAMessage は、MessageProperties.STATUS メッセージプロパティが StatusCodes.RECEIVED に変わると、デフォルトの削除ルールを使用して削除できるようになります。

削除ルールの詳細については、「メッセージの削除ルール」786 ページを参照してください。

参照

- QAManager インターフェイス [QAnywhere .NET]231 ページ
- QAManager.AcknowledgeUntil メソッド [QAnywhere .NET]237 ページ
- StatusCodes 列挙体 [QAnywhere .NET]322 ページ
- MessageProperties クラス [QAnywhere .NET]189 ページ
- QAManager.AcknowledgeAll メソッド [QAnywhere .NET]236 ページ

AcknowledgeAll メソッド

クライアントアプリケーションが QAnywhere メッセージを正常に受信したことを確認します。

Visual Basic 構文

```
Public Sub AcknowledgeAll()
```

C# 構文

```
public void AcknowledgeAll()
```

例外

- **QAException クラス**　メッセージの受信確認で問題が発生した場合にスローされます。

備考

受信確認されていないメッセージが、すべて受信確認されます。

> **注意**
> QAMessage が受信確認されると、その MessageProperties.STATUS プロパティは StatusCodes.RECEIVED に変わります。QAMessage は、ステータスが StatusCodes.RECEIVED に変わると、デフォルトの削除ルールを使用して削除できるようになります。

削除ルールの詳細については、「メッセージの削除ルール」786 ページを参照してください。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- QAManager インターフェイス [QAnywhere .NET]231 ページ
- QAManager.Acknowledge メソッド [QAnywhere .NET]235 ページ
- QAManager.AcknowledgeUntil メソッド [QAnywhere .NET]237 ページ
- StatusCodes 列挙体 [QAnywhere .NET]322 ページ
- MessageProperties クラス [QAnywhere .NET]189 ページ

AcknowledgeUntil メソッド

指定された QAMessage インスタンスと、指定されたメッセージよりも前に受信されて受信確認されていないメッセージについて、すべて受信確認します。

Visual Basic 構文

```
Public Sub AcknowledgeUntil(ByVal msg As QAMessage)
```

C# 構文

```
public void AcknowledgeUntil(QAMessage msg)
```

パラメーター

- **msg**　受信確認するメッセージのうち最新のもの。それ以前の受信確認されていないメッセージも、すべて受信確認されます。

例外

- **QAException クラス**　メッセージの受信確認で問題が発生した場合にスローされます。

備考

> **注意**
> QAMessage が受信確認されると、その MessageProperties.STATUS プロパティは StatusCodes.RECEIVED に変わります。QAMessage は、ステータスが StatusCodes.RECEIVED に変わると、デフォルトの削除ルールを使用して削除できるようになります。

削除ルールの詳細については、「メッセージの削除ルール」786 ページを参照してください。

参照

- QAManager インターフェイス [QAnywhere .NET]231 ページ
- QAManager.Acknowledge メソッド [QAnywhere .NET]235 ページ
- QAManager.AcknowledgeAll メソッド [QAnywhere .NET]236 ページ
- StatusCodes 列挙体 [QAnywhere .NET]322 ページ
- MessageProperties クラス [QAnywhere .NET]189 ページ

Open メソッド

指定された AcknowledgementMode 値を使用して QAManager をオープンします。

Visual Basic 構文

```
Public Sub Open(ByVal mode As AcknowledgementMode)
```

C# 構文

```
public void Open(AcknowledgementMode mode)
```

パラメーター

- **mode**　受信確認モード。AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT または AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT のどちらかです。

例外

- **QAException クラス**　QAManager インスタンスのオープンで問題がある場合にスローされます。

備考

Open メソッドは、QAManager を作成した後、最初に呼び出す必要があるメソッドです。

参照

- QAManager インターフェイス [QAnywhere .NET]231 ページ

Recover メソッド

受信確認されていないメッセージを、すべて強制的に StatusCodes.PENDING のステータスにします。

Visual Basic 構文

```
Public Sub Recover()
```

C# 構文

```
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にスローされます。

備考

該当するメッセージは、QAManagerBase.GetMessage を使用して再度受信する必要があります。

参照

- QAManager インターフェイス [QAnywhere .NET]231 ページ
- QAManagerBase.GetMessage メソッド [QAnywhere .NET]253 ページ

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

QAManagerBase インターフェイス

このクラスは、QATransactionalManager と QAManager の基本クラスです。前者の派生クラスはトランザクション指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指向のメッセージングを管理します。

Visual Basic 構文

```
Public Interface QAManagerBase
```

C# 構文

```
public interface QAManagerBase
```

派生クラス

- QAManager インターフェイス [QAnywhere .NET]231 ページ
- QATransactionalManager インターフェイス [QAnywhere .NET]309 ページ

メンバー

継承されたメンバーを含む QAManagerBase インターフェイスのすべてのメンバー。

名前	説明
BrowseMessages メソッド	この方法は使用されなくなりました。
BrowseMessagesByID メソッド	指定されたメッセージ ID を持つメッセージを参照します。
BrowseMessagesByQueue メソッド	指定されたアドレスに送信され、次以降に取得可能な一連の受信待機中メッセージを参照します。
BrowseMessagesBySelector メソッド	メッセージストアのキューに登録されているメッセージのうち、指定されたセレクターを満たすメッセージを参照します。
CancelMessage メソッド	指定されたメッセージ ID を持つメッセージをキャンセルします。
Close メソッド	QAnywhere メッセージシステムへの接続をクローズして、QAManagerBase で使用していたリソースをすべて解放します。
CreateBinaryMessage メソッド	QABinaryMessage オブジェクトを作成します。
CreateTextMessage メソッド	QATextMessage オブジェクトを作成します。

名前	説明
GetBooleanStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの boolean 値を取得します。
GetDoubleStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの double 値を取得します。
GetFloatStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの float 値を取得します。
GetIntStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの int 値を取得します。
GetLongStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの long 値を取得します。
GetMessage メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
GetMessageBySelector メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
GetMessageBySelectorNoWait メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
GetMessageBySelectorTimeout メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
GetMessageNoWait メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
GetMessageTimeout メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
GetQueueDepth メソッド	指定されたフィルターに基づいて、すべてのキューの長さの合計を返します。
GetSbyteStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの signed byte 値を取得します。
GetShortStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの short 値を取得します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
GetStoreProperty メソッド	メッセージストアプロパティを示す System.Object を取得します。
GetStorePropertyNames メソッド	メッセージストアのプロパティ名の列挙子を取得します。
GetStringStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの string 値を取得します。
PropertyExists メソッド	指定されたプロパティに値が現在存在するかどうかをテストします。
PutMessage メソッド	別の QAnywhere クライアントに送信するメッセージを準備します。
PutMessageTimeToLive メソッド	別の QAnywhere クライアントに送信するメッセージを準備します。
ReOpen メソッド	QAManagerBase を再度開きます。
SetBooleanStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを boolean 値に設定します。
SetDoubleStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを double 値に設定します。
SetExceptionListener メソッド	QAnywhere メッセージの非同期的な処理時に QAExceptions を受信するように、ExceptionListener デリゲートを設定します。
SetExceptionListener2 メソッド	QAnywhere メッセージの非同期的な処理時に QAExceptions を受信するように、ExceptionListener2 デリゲートを設定します。
SetFloatStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを float 値に設定します。
SetIntStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを int 値に設定します。
SetLongStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを long 値に設定します。
SetMessageListener メソッド	QAnywhere メッセージを非同期に受信するように、MessageListener デリゲートを設定します。

名前	説明
SetMessageListener2 メソッド	QAnywhere メッセージを非同期的に受信するように、MessageListener2 デリゲートを設定します。
SetMessageListenerBySelector メソッド	メッセージセレクターを使用して、QAnywhere メッセージを非同期的に受信するように、MessageListener デリゲートを設定します。
SetMessageListenerBySelector2 メソッド	メッセージセレクターを使用して、QAnywhere メッセージを非同期的に受信するように、MessageListener2 デリゲートを設定します。
SetProperty メソッド	QAnywhere Manager の設定プロパティをプログラムで設定できるようにします。
SetSbyteStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを sbyte (signed byte) 値に設定します。
SetShortStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを short 値に設定します。
SetStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを System.Object 値に設定します。
SetStringStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを string 値に設定します。
Start メソッド	着信メッセージをメッセージリスナーで受信するための QAManagerBase を起動します。
Stop メソッド	QAManagerBase による着信メッセージの受信を停止します。
TriggerSendReceive メソッド	QAnywhere メッセージサーバーとの同期処理を発生させて、他のクライアント宛てのメッセージをアップロードし、ローカルクライアント宛てのメッセージをダウンロードします。
Mode プロパティ	受信したメッセージの QAManager 受信確認モードを返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

QAManagerBase インスタンスがメッセージを受信できるようにするには、QAManagerBase.Start() メソッドを使用します。QAManagerBase インスタンスは、アプリケーションのスレッドごとに 1 つだけにします。

このクラスのインスタンスを使用して、QAnywhere メッセージの作成と管理を行うことができます。適切な QAMessage インスタンスを作成するには、QAManagerBase.CreateBinaryMessage() メソッドと QAManagerBase.CreateTextMessage() メソッドを使用します。QAMessage インスタンスには、メッセージの内容とプロパティを設定するための、さまざまなメソッドがあります。

QAnywhere メッセージを送信するには、QAManagerBase.PutMessage メソッドを使用して、アドレス指定されたメッセージをローカルのメッセージストアキューに登録します。メッセージは、転送ポリシーに基づいて QAnywhere Agent によって転送されるか、QAManagerBase.TriggerSendReceive() が呼び出されたときに転送されます。

qaagent 転送ポリシーの詳細については、「クライアントにメッセージを転送するタイミングの決定」49 ページを参照してください。

QAManagerBase.Close メソッドを使用して QAManagerBase インスタンスがクローズされると、メッセージがメモリから解放されます。

QAManagerBase にも、メッセージストアプロパティを設定および取得するためのメソッドがあります。

詳細については、「クライアントメッセージストアプロパティ」29 ページと「MessageStoreProperties クラス」を参照してください。

参照

- QAManagerBase.CreateBinaryMessage メソッド [QAnywhere .NET]248 ページ
- QAManagerBase.TriggerSendReceive メソッド [QAnywhere .NET]278 ページ
- QAManagerBase.Close メソッド [QAnywhere .NET]248 ページ
- QAException クラス [QAnywhere .NET]218 ページ

BrowseMessages メソッド

この方法は使用されなくなりました。

オーバーロードリスト

名前	説明
BrowseMessages() メソッド	メッセージストア内にある、参照可能なメッセージをすべて参照します。
BrowseMessages(string) メソッド	この方法は使用されなくなりました。

BrowseMessages() メソッド

メッセージストア内にある、参照可能なメッセージをすべて参照します。

Visual Basic 構文

```
Public Function BrowseMessages() As System.Collections.IEnumerator
```

C# 構文

```
public System.Collections.IEnumerator BrowseMessages()
```

戻り値

参照可能な一連のメッセージの列挙子。

備考

メッセージは単に参照されるだけなので、受信確認はできません。メッセージを参照するとネイティブのリソースが割り当てられるため、参照後は列挙子の Reset() メソッドを呼び出す必要があります。このメソッドを呼び出さないと、この QAManagerBase オブジェクトが解放されないかぎりネイティブのリソースも解放されません。

メッセージを受信して受信確認できるようにする場合は、QAManagerBase.GetMessage を使用します。

参照

- QAManagerBase.BrowseMessagesByQueue メソッド [QAnywhere .NET]246 ページ
- QAManagerBase.BrowseMessagesByID メソッド [QAnywhere .NET]245 ページ
- QAManagerBase.BrowseMessages メソッド [QAnywhere .NET]243 ページ

BrowseMessages(string) メソッド

この方法は使用されなくなりました。

Visual Basic 構文

```
Public Function BrowseMessages(
     ByVal address As String
) As System.Collections.IEnumerator
```

C# 構文

```
public System.Collections.IEnumerator BrowseMessages(string address)
```

パラメーター

- **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。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

代わりに BrowseMessagesByQueue(string) メソッドを使用してください。

指定されたアドレスに送信された、次以降に取得可能な一連の受信待機中メッセージを参照します。address パラメーターには、store-id¥queue-name または queue-name の形式で指定できます。メッセージは単に参照されるだけなので、受信確認はできません。

メッセージを参照するとネイティブのリソースが割り当てられるため、参照後は列挙子の Reset() メソッドを呼び出す必要があります。このメソッドを呼び出さないと、この QAManagerBase オブジェクトが解放されないかぎりネイティブのリソースも解放されません。

メッセージを受信して受信確認できるようにする場合は、QAManagerBase.GetMessage を使用します。

参照

- QAManagerBase.BrowseMessagesByQueue メソッド [QAnywhere .NET]246 ページ
- QAManagerBase.BrowseMessagesByID メソッド [QAnywhere .NET]245 ページ
- QAManagerBase.BrowseMessagesBySelector メソッド [QAnywhere .NET]246 ページ
- QAManagerBase.BrowseMessages メソッド [QAnywhere .NET]243 ページ

BrowseMessagesByID メソッド

指定されたメッセージ ID を持つメッセージを参照します。

Visual Basic 構文

```
Public Function BrowseMessagesByID(
     ByVal msgid As String
) As System.Collections.IEnumerator
```

C# 構文

```
public System.Collections.IEnumerator BrowseMessagesByID(string msgid)
```

パラメーター

- **msgid**　メッセージのメッセージ ID。

戻り値

0 または 1 を含むメッセージの列挙子。

備考

メッセージは単に参照されるだけなので、受信確認はできません。メッセージを参照するとネイティブのリソースが割り当てられるため、参照後は列挙子の Reset() メソッドを呼び出す必要があります。このメソッドを呼び出さないと、この QAManagerBase オブジェクトが解放されないかぎりネイティブのリソースも解放されません。

メッセージを受信して受信確認できるようにする場合は、QAManagerBase.GetMessage を使用します。

参照

- QAManagerBase.BrowseMessagesByQueue メソッド [QAnywhere .NET]246 ページ
- QAManagerBase.BrowseMessages メソッド [QAnywhere .NET]243 ページ

BrowseMessagesByQueue メソッド

指定されたアドレスに送信され、次以降に取得可能な一連の受信待機中メッセージを参照します。

Visual Basic 構文

```
Public Function BrowseMessagesByQueue(
     ByVal address As String
) As System.Collections.IEnumerator
```

C# 構文

```
public System.Collections.IEnumerator BrowseMessagesByQueue(
     string address
)
```

パラメーター

- **address**　メッセージのアドレス。

戻り値

参照可能な一連のメッセージの列挙子。

備考

メッセージは単に参照されるだけなので、受信確認はできません。メッセージを参照するとネイティブのリソースが割り当てられるため、参照後は列挙子の Reset() メソッドを呼び出す必要があります。このメソッドを呼び出さないと、この QAManagerBase オブジェクトが解放されないかぎりネイティブのリソースも解放されません。

メッセージを受信して受信確認できるようにする場合は、QAManagerBase.GetMessage を使用します。

参照

- QAManagerBase.BrowseMessagesByID メソッド [QAnywhere .NET]245 ページ
- QAManagerBase.BrowseMessages メソッド [QAnywhere .NET]243 ページ

BrowseMessagesBySelector メソッド

メッセージストアのキューに登録されているメッセージのうち、指定されたセレクターを満たすメッセージを参照します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

Visual Basic 構文

```
Public Function BrowseMessagesBySelector(
     ByVal selector As String
) As System.Collections.IEnumerator
```

C# 構文

```
public System.Collections.IEnumerator BrowseMessagesBySelector(
     string selector
)
```

パラメーター

- **selector**　セレクター。

戻り値

参照可能な一連のメッセージの列挙子。

備考

メッセージは単に参照されるだけなので、受信確認はできません。メッセージを参照するとネイティブのリソースが割り当てられるため、参照後は列挙子の Reset() メソッドを呼び出す必要があります。このメソッドを呼び出さないと、この QAManagerBase オブジェクトが解放されないかぎりネイティブのリソースも解放されません。

メッセージを受信して受信確認できるようにする場合は、QAManagerBase.GetMessage を使用します。

参照

- QAManagerBase.BrowseMessagesByQueue メソッド [QAnywhere .NET]246 ページ
- QAManagerBase.BrowseMessages メソッド [QAnywhere .NET]243 ページ
- QAManagerBase.BrowseMessagesByID メソッド [QAnywhere .NET]245 ページ

CancelMessage メソッド

指定されたメッセージ ID を持つメッセージをキャンセルします。

Visual Basic 構文

```
Public Sub CancelMessage(ByVal msgid As String)
```

C# 構文

```
public void CancelMessage(string msgid)
```

パラメーター

- **msgid**　キャンセルするメッセージのメッセージ ID。

例外

- **QAException クラス**　メッセージのキャンセルで問題が発生した場合にスローされます。

備考

CancelMessage は、メッセージが転送される前にメッセージをキャンセル済みの状態にします。QAnywhere Agent のデフォルトの削除ルールにより、キャンセル済みメッセージは最終的にメッセージストアから削除されます。

メッセージがすでに最終ステータスになっている場合や、中央のメッセージングサーバーに転送済みである場合は、CancelMessage は失敗します。

削除ルールの詳細については、「メッセージの削除ルール」786 ページを参照してください。

Close メソッド

QAnywhere メッセージシステムへの接続をクローズして、QAManagerBase で使用していたリソースをすべて解放します。

Visual Basic 構文

```
Public Sub Close()
```

C# 構文

```
public void Close()
```

例外

- **QAException クラス**　QAManagerBase インスタンスのクローズで問題がある場合にスローされます。

備考

このメソッドは、メッセージリスナーと例外リスナーでは呼び出すことができません。

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。

CreateBinaryMessage メソッド

QABinaryMessage オブジェクトを作成します。

Visual Basic 構文

```
Public Function CreateBinaryMessage() As QABinaryMessage
```

C# 構文

```
public QABinaryMessage CreateBinaryMessage()
```

戻り値

新しい QABinaryMessage インスタンス。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例外

- **QAException クラス**　メッセージの作成で問題が発生した場合にスローされます。

備考

QABinaryMessage オブジェクトは、未解釈のバイトストリームのメッセージ本文が含まれるメッセージの送信に使用します。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ

CreateTextMessage メソッド

QATextMessage オブジェクトを作成します。

Visual Basic 構文

```
Public Function CreateTextMessage() As QATextMessage
```

C# 構文

```
public QATextMessage CreateTextMessage()
```

戻り値

新しい QATextMessage インスタンス。

例外

- **QAException クラス**　メッセージの作成で問題が発生した場合にスローされます。

備考

QATextMessage のオブジェクトは、string のメッセージ本文が含まれるメッセージを送信する場合に使用します。

参照

- QATextMessage インターフェイス [QAnywhere .NET]304 ページ

GetBooleanStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの boolean 値を取得します。

Visual Basic 構文

```
Public Function GetBooleanStoreProperty(
     ByVal propName As String
) As Boolean
```

C# 構文

```
public bool GetBooleanStoreProperty(string propName)
```

パラメーター

- **propName**　事前定義済みまたはカスタムのプロパティ名。

戻り値

boolean 型のプロパティの値。

例外

- **QAException クラス**　プロパティ値の取得で変換エラーまたはメッセージストアエラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

参照

- MessageStoreProperties クラス [QAnywhere .NET]198 ページ

GetDoubleStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの double 値を取得します。

Visual Basic 構文

```
Public Function GetDoubleStoreProperty(
     ByVal propName As String
) As Double
```

C# 構文

```
public double GetDoubleStoreProperty(string propName)
```

パラメーター

- **propName**　事前定義済みまたはカスタムのプロパティ名。

戻り値

double 型のプロパティの値。

例外

- **QAException クラス**　プロパティ値の取得で変換エラーまたはメッセージストアエラーがある場合、またはプロパティが存在しない場合にスローされます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

参照

- MessageStoreProperties クラス [QAnywhere .NET]198 ページ

GetFloatStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの float 値を取得します。

Visual Basic 構文

```
Public Function GetFloatStoreProperty(
     ByVal propName As String
) As Single
```

C# 構文

```
public float GetFloatStoreProperty(string propName)
```

パラメーター

- **propName**　事前定義済みまたはカスタムのプロパティ名。

戻り値

float 型のプロパティの値。

例外

- **QAException クラス**　プロパティ値の取得で変換エラーまたはメッセージストアエラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

参照

- MessageStoreProperties クラス [QAnywhere .NET]198 ページ

GetIntStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの int 値を取得します。

Visual Basic 構文

```
Public Function GetIntStoreProperty(ByVal propName As String) As Integer
```

C# 構文

```
public int GetIntStoreProperty(string propName)
```

パラメーター

- **propName**　事前定義済みまたはカスタムのプロパティ名。

戻り値

int 型のプロパティの値。

例外

- **QAException クラス**　プロパティ値の取得で変換エラーまたはメッセージストアエラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

参照

- MessageStoreProperties クラス [QAnywhere .NET]198 ページ

GetLongStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの long 値を取得します。

Visual Basic 構文

```
Public Function GetLongStoreProperty(ByVal propName As String) As Long
```

C# 構文

```
public long GetLongStoreProperty(string propName)
```

パラメーター

- **propName**　事前定義済みまたはカスタムのプロパティ名。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

戻り値

long 型のプロパティの値。

例外

- **QAException クラス**　プロパティ値の取得で変換エラーまたはメッセージストアエラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

参照

- MessageStoreProperties クラス [QAnywhere .NET]198 ページ

GetMessage メソッド

指定されたアドレスに送信された、次に取得可能な QAMessage を返します。

Visual Basic 構文

```
Public Function GetMessage(ByVal address As String) As QAMessage
```

C# 構文

```
public QAMessage GetMessage(string address)
```

パラメーター

- **address**　メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。

戻り値

該当する次の QAMessage。メッセージが存在しない場合は NULL。

例外

- **QAException クラス**　メッセージの取得で問題が発生した場合にスローされます。

備考

address パラメーターには、ローカルのキュー名を指定します。アドレスは、store-id¥queue-name または queue-name の形式で指定できます。

該当するメッセージが存在しない場合、新しいメッセージが着信するまでブロックされます。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (メッセージイベントハンドラーを使用する) 方法の詳細については、「非同期的なメッセージ受信」73 ページを参照してください。

参照

- QAMessage インターフェイス [QAnywhere .NET]285 ページ

GetMessageBySelector メソッド

指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。

Visual Basic 構文

```
Public Function GetMessageBySelector(
     ByVal address As String,
    ByVal selector As String
) As QAMessage
```

C# 構文

```
public QAMessage GetMessageBySelector(string address, string selector)
```

パラメーター

- **address**　このアドレスは、メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。
- **selector**　セレクター。

戻り値

該当する次の QAMessage。メッセージが存在しない場合は NULL。

例外

- **QAException クラス**　メッセージの取得で問題が発生した場合にスローされます。

備考

address パラメーターには、ローカルのキュー名を指定します。アドレスは、store-id¥queue-name または queue-name の形式で指定できます。

該当するメッセージが存在しない場合、新しいメッセージが着信するまでブロックされます。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (メッセージイベントハンドラーを使用する) 方法の詳細については、「非同期的なメッセージ受信」73 ページを参照してください。

参照

- QAMessage インターフェイス [QAnywhere .NET]285 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

GetMessageBySelectorNoWait メソッド

指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。

Visual Basic 構文

```
Public Function GetMessageBySelectorNoWait(
     ByVal address As String,
    ByVal selector As String
) As QAMessage
```

C# 構文

```
public QAMessage GetMessageBySelectorNoWait(
     string address,
    string selector
)
```

パラメーター

- **address** メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。
- **selector** セレクター。

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** メッセージの取得で問題が発生した場合にスローされます。

備考

address パラメーターには、ローカルのキュー名を指定します。アドレスは、store-id¥queue-name または queue-name の形式で指定できます。該当するメッセージがない場合は、このメソッドがすぐに返されます。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (メッセージイベントハンドラーを使用する) 方法の詳細については、「非同期的なメッセージ受信」73 ページを参照してください。

参照

- QAMessage インターフェイス [QAnywhere .NET]285 ページ

GetMessageBySelectorTimeout メソッド

指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。

Visual Basic 構文

```
Public Function GetMessageBySelectorTimeout(
     ByVal address As String,
    ByVal selector As String,
    ByVal timeout As Long
) As QAMessage
```

C# 構文

```
public QAMessage GetMessageBySelectorTimeout(
     string address,
    string selector,
    long timeout
)
```

パラメーター

- **address** メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。
- **selector** セレクター。
- **timeout** メッセージ着信を待機する時間 (ミリ秒単位)。

戻り値

該当する次の QAMessage。メッセージが存在しない場合は NULL。

例外

- **QAException クラス** メッセージの取得で問題が発生した場合にスローされます。

備考

address パラメーターには、ローカルのキュー名を指定します。アドレスは、store-id¥queue-name または queue-name の形式で指定できます。該当するメッセージがない場合、このメソッドは指定されたタイムアウト時間だけ待機してから返されます。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (メッセージイベントハンドラーを使用する) 方法の詳細については、「非同期的なメッセージ受信」73 ページを参照してください。

参照

- QAMessage インターフェイス [QAnywhere .NET]285 ページ

GetMessageNoWait メソッド

指定されたアドレスに送信された、次に取得可能な QAMessage を返します。

Visual Basic 構文

```
Public Function GetMessageNoWait(ByVal address As String) As QAMessage
```

C# 構文

```
public QAMessage GetMessageNoWait(string address)
```

パラメーター

- **address**　このアドレスは、メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。

戻り値

次の取得可能なメッセージ。該当するメッセージが存在しない場合は NULL。

例外

- **QAException クラス**　メッセージの取得で問題が発生した場合にスローされます。

備考

address パラメーターには、ローカルのキュー名を指定します。アドレスは、store-id¥queue-name または queue-name の形式で指定できます。該当するメッセージがない場合は、このメソッドがすぐに返されます。メッセージを同期的に受信する場合は、このメソッドを使用します。メッセージを非同期的に受信する (メッセージイベントハンドラーを使用する) 方法の詳細については、「非同期的なメッセージ受信」73 ページを参照してください。

参照

- QAMessage インターフェイス [QAnywhere .NET]285 ページ

GetMessageTimeout メソッド

指定されたアドレスに送信された、次に取得可能な QAMessage を返します。

Visual Basic 構文

```
Public Function GetMessageTimeout(
     ByVal address As String,
    ByVal timeout As Long
) As QAMessage
```

C# 構文

```
public QAMessage GetMessageTimeout(string address, long timeout)
```

パラメーター

- **address**　メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。
- **timeout**　メッセージ着信を待機する時間 (ミリ秒単位)。

戻り値

該当する次の QAMessage。メッセージが存在しない場合は NULL。

例外

- **QAException クラス**　メッセージの取得で問題が発生した場合にスローされます。

備考

address パラメーターには、ローカルのキュー名を指定します。アドレスは、store-id¥queue-name または queue-name の形式で指定できます。

該当するメッセージがない場合、このメソッドは指定されたタイムアウト時間だけ待機してから返されます。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (メッセージイベントハンドラーを使用する) 方法の詳細については、「非同期的なメッセージ受信」73 ページを参照してください。

GetQueueDepth メソッド

指定されたフィルターに基づいて、すべてのキューの長さの合計を返します。

オーバーロードリスト

名前	説明
GetQueueDepth(QueueDepthFilter) メソッド	指定されたフィルターに基づいて、すべてのキューの長さの合計を返します。
GetQueueDepth(string, QueueDepthFilter) メソッド	指定されたフィルターに基づいて、キューの長さを返します。

GetQueueDepth(QueueDepthFilter) メソッド

指定されたフィルターに基づいて、すべてのキューの長さの合計を返します。

Visual Basic 構文

```
Public Function GetQueueDepth(
     ByVal filter As QueueDepthFilter
) As Integer
```

C# 構文

```
public int GetQueueDepth(QueueDepthFilter filter)
```

パラメーター

- **filter**　着信メッセージ、送信メッセージ、またはすべてのメッセージを示すフィルター。

戻り値

メッセージ数。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例外

- **QAException クラス**　エラーが発生した場合にスローされます。

備考

キューの着信の深さは、受信されていない着信メッセージの数です (たとえば、QAManagerBase.GetMessage を使用)。キューの送信の深さは、サーバーに転送されていない送信メッセージ (コミットされていないメッセージを含む) の数です。

参照

- QueueDepthFilter 列挙体 [QAnywhere .NET]321 ページ

GetQueueDepth(string, QueueDepthFilter) メソッド

指定されたフィルターに基づいて、キューの長さを返します。

Visual Basic 構文

```
Public Function GetQueueDepth(
     ByVal address As String,
    ByVal filter As QueueDepthFilter
) As Integer
```

C# 構文

```
public int GetQueueDepth(string address, QueueDepthFilter filter)
```

パラメーター

- **filter**　着信メッセージ、送信メッセージ、またはすべてのメッセージを示すフィルター。
- **address**　キュー名。

戻り値

メッセージ数。

例外

- **QAException クラス**　エラーが発生した場合にスローされます。

備考

キューの着信の深さは、受信されていない着信メッセージの数です (たとえば、QAManagerBase.GetMessage を使用)。キューの送信の深さは、サーバーに転送されていない送信メッセージ (コミットされていないメッセージを含む) の数です。

参照

- QueueDepthFilter 列挙体 [QAnywhere .NET]321 ページ

GetSbyteStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの signed byte 値を取得します。

Visual Basic 構文

```
Public Function GetSbyteStoreProperty(ByVal propName As String) As SByte
```

C# 構文

```
public sbyte GetSbyteStoreProperty(string propName)
```

パラメーター

- **propName**　事前定義済みまたはカスタムのプロパティ名。

戻り値

signed byte 型のプロパティの値。

例外

- **QAException クラス**　プロパティ値の取得で変換エラーまたはメッセージストアエラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

参照

- MessageStoreProperties クラス [QAnywhere .NET]198 ページ

GetShortStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの short 値を取得します。

Visual Basic 構文

```
Public Function GetShortStoreProperty(ByVal propName As String) As Short
```

C# 構文

```
public short GetShortStoreProperty(string propName)
```

パラメーター

- **propName**　事前定義済みまたはカスタムのプロパティ名。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

戻り値

short 型のプロパティの値。

例外

- **QAException クラス**　プロパティ値の取得で変換エラーまたはメッセージストアエラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

参照

- MessageStoreProperties クラス [QAnywhere .NET]198 ページ

GetStoreProperty メソッド

メッセージストアプロパティを示す System.Object を取得します。

Visual Basic 構文

```
Public Function GetStoreProperty(ByVal propName As String) As Object
```

C# 構文

```
public Object GetStoreProperty(string propName)
```

パラメーター

- **propName**　事前定義済みまたはカスタムのプロパティ名。

戻り値

プロパティの値。

例外

- **QAException クラス**　プロパティの取得で問題が発生した場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

参照

- MessageStoreProperties クラス [QAnywhere .NET]198 ページ

GetStorePropertyNames メソッド

メッセージストアのプロパティ名の列挙子を取得します。

Visual Basic 構文

```
Public Function GetStorePropertyNames()
    As System.Collections.IEnumerator
```

C# 構文

```
public System.Collections.IEnumerator GetStorePropertyNames()
```

戻り値

メッセージストアのプロパティ名の列挙子。

備考

クライアントストアプロパティの詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

GetStringStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの string 値を取得します。

Visual Basic 構文

```
Public Function GetStringStoreProperty(
     ByVal propName As String
) As String
```

C# 構文

```
public string GetStringStoreProperty(string propName)
```

パラメーター

- **propName**　事前定義済みまたはカスタムのプロパティ名。

戻り値

string 型のプロパティの値。該当するプロパティが存在しない場合は NULL。

例外

- **QAException クラス**　string 値の取得で問題が発生した場合にスローされます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

参照

- MessageStoreProperties クラス [QAnywhere .NET]198 ページ

PropertyExists メソッド

指定されたプロパティに値が現在存在するかどうかをテストします。

Visual Basic 構文

```
Public Function PropertyExists(ByVal propName As String) As Boolean
```

C# 構文

```
public bool PropertyExists(string propName)
```

パラメーター

- **propName**　事前定義済みまたはカスタムのプロパティ名。

戻り値

プロパティにマッピングされた値がメッセージストアに存在する場合は true、それ以外の場合は false。

例外

- **QAException クラス**　プロパティ値の取得で問題が発生した場合にスローされます。

備考

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

参照

- MessageStoreProperties クラス [QAnywhere .NET]198 ページ

PutMessage メソッド

別の QAnywhere クライアントに送信するメッセージを準備します。

Visual Basic 構文

```
Public Sub PutMessage(ByVal address As String, ByVal msg As QAMessage)
```

C# 構文

```
public void PutMessage(string address, QAMessage msg)
```

パラメーター

- **address**　送信先のキュー名を指定するメッセージのアドレス。
- **msg**　転送用ローカルメッセージストアに格納するメッセージ。

例外

- **QAException クラス**　メッセージの登録で問題が発生した場合にスローされます。

備考

PutMessage メソッドは、メッセージと送信先アドレスをローカルメッセージストアに挿入します。メッセージが転送されるタイミングは、QAnywhere Agent の転送ポリシーで決まります。

詳細については、「クライアントにメッセージを転送するタイミングの決定」49 ページを参照してください。

アドレスは id¥queue-name の形式で指定します。id は送信先メッセージストアの ID、queue-name は送信先の QAnywhere クライアントがメッセージの受信で使用するキューを特定します。

QAnywhere アドレスの詳細については、「QAnywhere メッセージアドレス」60 ページを参照してください。

参照

- QAManagerBase.PutMessageTimeToLive メソッド [QAnywhere .NET]264 ページ

PutMessageTimeToLive メソッド

別の QAnywhere クライアントに送信するメッセージを準備します。

Visual Basic 構文

```
Public Sub PutMessageTimeToLive(
     ByVal address As String,
    ByVal msg As QAMessage,
    ByVal ttl As Long
)
```

C# 構文

```
public void PutMessageTimeToLive(
     string address,
    QAMessage msg,
    long ttl
)
```

パラメーター

- **address**　送信先のキュー名を指定するメッセージのアドレス。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- **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**　メッセージの登録で問題が発生した場合にスローされます。

備考

PutMessageTimeToLive メソッドは、メッセージと送信先アドレスをローカルメッセージストアに挿入します。メッセージが転送されるタイミングは、QAnywhere Agent の転送ポリシーで決まります。ただし、次のメッセージの転送時間が指定された存続時間を超えると、そのメッセージは期限切れになります。

詳細については、「クライアントにメッセージを転送するタイミングの決定」49 ページを参照してください。

アドレスは id¥queue-name の形式で指定します。id は送信先メッセージストアの ID、queue-name は送信先の QAnywhere クライアントがメッセージの受信で使用するキューを特定します。

QAnywhere アドレスの詳細については、「QAnywhere メッセージアドレス」60 ページを参照してください。

ReOpen メソッド

QAManagerBase を再度開きます。

Visual Basic 構文

```
Public Sub ReOpen()
```

C# 構文

```
public void ReOpen()
```

例外

- **QAException クラス**　QAManagerBase インスタンスを再度開くときに問題がある場合にスローされます。

備考

これにより、リソースを解放することなく、メッセージストアへの接続が再確立されます。このメソッドは、メッセージリスナーまたは例外リスナーで呼び出されることがありますが、その場合は Start() を再度呼び出す必要はありません。リスナーで呼び出されない場合、このメソッドでは Close() の後に Open() が実行されるだけなので、この場合は、Start() を呼び出して、メッセージの受信を再開する必要があります。

SetBooleanStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを boolean 値に設定します。

Visual Basic 構文

```
Public Sub SetBooleanStoreProperty(
     ByVal propName As String,
    ByVal val As Boolean
)
```

C# 構文

```
public void SetBooleanStoreProperty(string propName, bool val)
```

パラメーター

- **propName**　事前定義済みまたはカスタムのプロパティ名。
- **val**　boolean 型のプロパティの値。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

参照

- MessageStoreProperties クラス [QAnywhere .NET]198 ページ

SetDoubleStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを double 値に設定します。

Visual Basic 構文

```
Public Sub SetDoubleStoreProperty(
     ByVal propName As String,
    ByVal val As Double
)
```

C# 構文

```
public void SetDoubleStoreProperty(string propName, double val)
```

パラメーター

- **propName**　事前定義済みまたはカスタムのプロパティ名。
- **val**　double 型のプロパティの値。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

参照

- MessageStoreProperties クラス [QAnywhere .NET]198 ページ

SetExceptionListener メソッド

QAnywhere メッセージの非同期的な処理時に QAExceptions を受信するように、ExceptionListener デリゲートを設定します。

Visual Basic 構文

```
Public Sub SetExceptionListener(
     ByVal address As String,
    ByVal listener As ExceptionListener
)
```

C# 構文

```
public void SetExceptionListener(
     string address,
    ExceptionListener listener
)
```

パラメーター

- **address**　メッセージのアドレス。
- **listener**　登録する例外リスナー。

備考

ExceptionListener デリゲートには、QAException と QAMessage の各パラメーターを指定できます。指定したアドレスに対して ExceptionListener と MessageListener を設定できますが、Listener／Listener2 デリゲートを一致させる必要があります。つまり、同じアドレスに対して ExceptionListener と MessageListener2、または ExceptionListener2 と MessageListener を設定することはできません。

詳細については、「非同期的なメッセージ受信」73 ページを参照してください。

SetExceptionListener2 メソッド

QAnywhere メッセージの非同期的な処理時に QAExceptions を受信するように、ExceptionListener2 デリゲートを設定します。

Visual Basic 構文

```
Public Sub SetExceptionListener2(
     ByVal address As String,
    ByVal listener As ExceptionListener2
)
```

C# 構文

```
public void SetExceptionListener2(
     string address,
    ExceptionListener2 listener
)
```

パラメーター

- **address**　メッセージのアドレス。
- **listener**　登録する例外リスナー。

備考

ExceptionListener2 デリゲートには、QAManagerBase、QAException、QAMessage の各パラメーターを指定できます。指定したアドレスに対して ExceptionListener2 と MessageListener2 を設定できますが、Listener／Listener2 デリゲートを一致させる必要があります。つまり、同じアドレスに対して ExceptionListener と MessageListener2、または ExceptionListener2 と MessageListener を設定することはできません。

詳細については、「非同期的なメッセージ受信」73 ページを参照してください。

SetFloatStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを float 値に設定します。

Visual Basic 構文

```
Public Sub SetFloatStoreProperty(
     ByVal propName As String,
    ByVal val As Single
)
```

C# 構文

```
public void SetFloatStoreProperty(string propName, float val)
```

パラメーター

- **propName**　事前定義済みまたはカスタムのプロパティ名。
- **val**　float 型のプロパティの値。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

参照

- MessageStoreProperties クラス [QAnywhere .NET]198 ページ

SetIntStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを int 値に設定します。

Visual Basic 構文

```
Public Sub SetIntStoreProperty(
     ByVal propName As String,
    ByVal val As Integer
)
```

C# 構文

```
public void SetIntStoreProperty(string propName, int val)
```

パラメーター

- **propName**　事前定義済みまたはカスタムのプロパティ名。
- **val**　int 型のプロパティの値。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

参照

- MessageStoreProperties クラス [QAnywhere .NET]198 ページ

SetLongStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを long 値に設定します。

Visual Basic 構文

```
Public Sub SetLongStoreProperty(
     ByVal propName As String,
    ByVal val As Long
)
```

C# 構文

```
public void SetLongStoreProperty(string propName, long val)
```

パラメーター

- **propName**　事前定義済みまたはカスタムのプロパティ名。
- **val**　long 型のプロパティの値。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

参照

- MessageStoreProperties クラス [QAnywhere .NET]198 ページ

SetMessageListener メソッド

QAnywhere メッセージを非同期に受信するように、MessageListener デリゲートを設定します。

Visual Basic 構文

```
Public Sub SetMessageListener(
     ByVal address As String,
    ByVal listener As MessageListener
)
```

C# 構文

```
public void SetMessageListener(string address, MessageListener listener)
```

パラメーター

- **address**　メッセージのアドレス。
- **listener**　登録するリスナー。

備考

メッセージを非同期的に受信する場合は、このメソッドを使用します。

MessageListener デリゲートには、QAMessage パラメーターを 1 つだけ指定できます。

SetMessageListener の address パラメーターには、メッセージの受信で使用されるローカルキュー名を指定します。1 つのキューには、1 つのリスナーデリゲートだけを割り当てることができます。指定したアドレスに対して ExceptionListener と MessageListener を設定できますが、Listener／Listener2 デリゲートを一致させる必要があります。つまり、同じアドレスに対して

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ExceptionListener と MessageListener2、または ExceptionListener2 と MessageListener を設定することはできません。

Push 通知とネットワークステータス変更も含め、QAnywhere システムメッセージを受信したい場合は、キュー名に system を指定します。

詳細については、「非同期的なメッセージ受信」73 ページを参照してください。

参照

- MessageListener デリゲート [QAnywhere .NET]316 ページ

SetMessageListener2 メソッド

QAnywhere メッセージを非同期的に受信するように、MessageListener2 デリゲートを設定します。

Visual Basic 構文

```
Public Sub SetMessageListener2(
     ByVal address As String,
    ByVal listener As MessageListener2
)
```

C# 構文

```
public void SetMessageListener2(
     string address,
    MessageListener2 listener
)
```

パラメーター

- **address**　メッセージのアドレス。
- **listener**　登録するリスナー。

備考

メッセージを非同期的に受信する場合は、このメソッドを使用します。

MessageListener2 デリゲートには、QAManagerBase と QAMessage の各パラメーターを指定できます。

SetMessageListener2 の address パラメーターには、メッセージの受信で使用されるローカルキュー名を指定します。1 つのキューには、1 つのリスナーデリゲートだけを割り当てることができます。指定したアドレスに対して ExceptionListener2 と MessageListener2 を設定できますが、Listener／Listener2 デリゲートを一致させる必要があります。つまり、同じアドレスに対して ExceptionListener と MessageListener2、または ExceptionListener2 と MessageListener を設定することはできません。

Push 通知とネットワークステータス変更も含め、QAnywhere システムメッセージを受信したい場合は、キュー名に system を指定します。

詳細については、「非同期的なメッセージ受信」73 ページを参照してください。

SetMessageListenerBySelector メソッド

メッセージセレクターを使用して、QAnywhere メッセージを非同期的に受信するように、MessageListener デリゲートを設定します。

Visual Basic 構文

```
Public Sub SetMessageListenerBySelector(
     ByVal address As String,
    ByVal selector As String,
    ByVal listener As MessageListener
)
```

C# 構文

```
public void SetMessageListenerBySelector(
     string address,
    string selector,
    MessageListener listener
)
```

パラメーター

- **address**　メッセージのアドレス。
- **listener**　登録するリスナー。
- **selector**　受信されるメッセージをフィルタリングするために使用するセレクター。

備考

メッセージを非同期的に受信する場合は、このメソッドを使用します。

MessageListener デリゲートには、QAMessage パラメーターを 1 つだけ指定できます。

SetMessageListener の address パラメーターには、メッセージの受信で使用されるローカルキュー名を指定します。1 つのキューには、1 つのリスナーデリゲートだけを割り当てることができます。selector パラメーターは、指定されたアドレスで受信されるメッセージをフィルタリングするために使用するセレクターを指定します。指定したアドレスに対して ExceptionListener と MessageListener を設定できますが、Listener／Listener2 デリゲートを一致させる必要があります。つまり、同じアドレスに対して ExceptionListener と MessageListener2、または ExceptionListener2 と MessageListener を設定することはできません。

Push 通知とネットワークステータス変更も含め、QAnywhere システムメッセージを受信したい場合は、キュー名に system を指定します。

詳細については、「非同期的なメッセージ受信」73 ページと「システムキュー」60 ページを参照してください。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- MessageListener デリゲート [QAnywhere .NET]316 ページ

SetMessageListenerBySelector2 メソッド

メッセージセレクターを使用して、QAnywhere メッセージを非同期的に受信するように、MessageListener2 デリゲートを設定します。

Visual Basic 構文

```
Public Sub SetMessageListenerBySelector2(
     ByVal address As String,
    ByVal selector As String,
    ByVal listener As MessageListener2
)
```

C# 構文

```
public void SetMessageListenerBySelector2(
     string address,
    string selector,
    MessageListener2 listener
)
```

パラメーター

- **address**　メッセージのアドレス。
- **listener**　登録するリスナー。
- **selector**　受信されるメッセージをフィルタリングするために使用するセレクター。

備考

メッセージを非同期的に受信する場合は、このメソッドを使用します。

MessageListener2 デリゲートには、QAMessage パラメーターを 1 つだけ指定できます。

SetMessageListener2 の address パラメーターには、メッセージの受信で使用されるローカルキュー名を指定します。1 つのキューには、1 つのリスナーデリゲートだけを割り当てることができます。selector パラメーターは、指定されたアドレスで受信されるメッセージをフィルタリングするために使用するセレクターを指定します。指定したアドレスに対して ExceptionListener2 と MessageListener2 を設定できますが、Listener／Listener2 デリゲートを一致させる必要があります。つまり、同じアドレスに対して ExceptionListener と MessageListener2、または ExceptionListener2 と MessageListener を設定することはできません。

Push 通知とネットワークステータス変更も含め、QAnywhere システムメッセージを受信したい場合は、キュー名に system を指定します。

詳細については、「非同期的なメッセージ受信」73 ページと「システムキュー」60 ページを参照してください。

SetProperty メソッド

QAnywhere Manager の設定プロパティをプログラムで設定できるようにします。

Visual Basic 構文

```
Public Sub SetProperty(ByVal name As String, ByVal val As String)
```

C# 構文

```
public void SetProperty(string name, string val)
```

パラメーター

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

例外

- **QAException クラス** プロパティの設定で問題が発生した場合にスローされます。

備考

プロパティ名と値を指定してこのメソッドを使用することで、QAnywhere Manager のデフォルトの設定プロパティを無効にできます。プロパティのリストについては、「QAnywhere Manager の設定プロパティ」85 ページを参照してください。

QAnywhere Manager の設定プロパティは、プロパティファイルと QAManagerFactory.CreateQAManager メソッドを使用して設定することもできます。

注意
詳細については、「ファイルでの QAnywhere Manager 設定プロパティの設定」86 ページを参照してください。QAManager.Open または QATransactionalManager.Open() を呼び出す前に、必要なプロパティを設定してください。

参照

- QAManager.Open メソッド [QAnywhere .NET]237 ページ
- QATransactionalManager.Open メソッド [QAnywhere .NET]314 ページ

SetSbyteStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを sbyte (signed byte) 値に設定します。

Visual Basic 構文

```
Public Sub SetSbyteStoreProperty(
     ByVal propName As String,
    ByVal val As SByte
)
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

C# 構文

```
public void SetSbyteStoreProperty(string propName, sbyte val)
```

パラメーター

- **propName** 事前定義済みまたはカスタムのプロパティ名。
- **val** signed byte 型のプロパティの値。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

参照

- MessageStoreProperties クラス [QAnywhere .NET]198 ページ

SetShortStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを short 値に設定します。

Visual Basic 構文

```
Public Sub SetShortStoreProperty(
     ByVal propName As String,
    ByVal val As Short
)
```

C# 構文

```
public void SetShortStoreProperty(string propName, short val)
```

パラメーター

- **propName** 事前定義済みまたはカスタムのプロパティ名。
- **val** short 型のプロパティの値。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

参照

- MessageStoreProperties クラス [QAnywhere .NET]198 ページ

SetStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを System.Object 値に設定します。

Visual Basic 構文

```
Public Sub SetStoreProperty(
     ByVal propName As String,
    ByVal val As Object
)
```

C# 構文

```
public void SetStoreProperty(string propName, Object val)
```

パラメーター

- **propName**　事前定義済みまたはカスタムのプロパティ名。
- **val**　プロパティの値。

備考

プロパティ型は、使用可能ないずれかの単純型、文字列型、または DateTime 型でなければなりません。このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

参照

- MessageStoreProperties クラス [QAnywhere .NET]198 ページ

SetStringStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを string 値に設定します。

Visual Basic 構文

```
Public Sub SetStringStoreProperty(
     ByVal propName As String,
    ByVal val As String
)
```

C# 構文

```
public void SetStringStoreProperty(string propName, string val)
```

パラメーター

- **propName**　事前定義済みまたはカスタムのプロパティ名。
- **val**　string 型のプロパティの値。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

参照

- MessageStoreProperties クラス [QAnywhere .NET]198 ページ

Start メソッド

着信メッセージをメッセージリスナーで受信するための QAManagerBase を起動します。

Visual Basic 構文

```
Public Sub Start()
```

C# 構文

```
public void Start()
```

例外

- **QAException クラス**　QAManagerBase インスタンスの起動で問題がある場合にスローされます。

備考

メッセージリスナーセットがない場合、たとえばメッセージが GetMessage メソッドで受信される場合などは、QAManagerBase を起動する必要がありません。GetMessage メソッドとメッセージリスナーは、メッセージを受信するために併用しないでください。非同期 (メッセージリスナー) モデルまたは同期 (GetMessage) モデルのどちらかを使用してください。この Start() メソッドを繰り返し呼び出そうとしても、間に QAManagerBase.Stop() 呼び出しを挟まないかぎり 2 回目以降の呼び出しは無視されます。

参照

- QAManagerBase.Stop メソッド [QAnywhere .NET]277 ページ

Stop メソッド

QAManagerBase による着信メッセージの受信を停止します。

Visual Basic 構文

```
Public Sub Stop()
```

C# 構文

```
public void Stop()
```

例外

- **QAException クラス**　QAManagerBase インスタンスの停止で問題がある場合にスローされます。

備考

メッセージは失われません。メッセージは、Manager が再起動されるまで受信されません。この Stop() メソッドを繰り返し呼び出そうとしても、間に QAManagerBase.Start() 呼び出しを挟まないかぎり 2 回目以降の呼び出しは無視されます。

参照

- QAManagerBase.Start メソッド [QAnywhere .NET]277 ページ

TriggerSendReceive メソッド

QAnywhere メッセージサーバーとの同期処理を発生させて、他のクライアント宛てのメッセージをアップロードし、ローカルクライアント宛てのメッセージをダウンロードします。

Visual Basic 構文

```
Public Sub TriggerSendReceive()
```

C# 構文

```
public void TriggerSendReceive()
```

例外

- **QAException クラス**　送信／受信のトリガーで問題が発生した場合にスローされます。

備考

QAManagerBase の TriggerSendReceive では、QAnywhere Agent と中央のメッセージングサーバーの間でメッセージの同期処理がただちに行われます。手動の TriggerSendReceive 呼び出しでは、QAnywhere Agent の転送ポリシーとは無関係に、メッセージ転送がただちに行われます。

QAnywhere Agent の転送ポリシーは、メッセージ転送をどのようにして実行するかを決定します。たとえば、クライアントが Push 通知を受信した場合や、ユーザーが QAManagerBase.PutMessage メソッドを呼び出してメッセージを送信した場合に、一定の間隔でメッセージ転送を自動的に実行することができます。

詳細については、「クライアントにメッセージを転送するタイミングの決定」49 ページを参照してください。

参照

- QAManagerBase.PutMessage メソッド [QAnywhere .NET]263 ページ

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

Mode プロパティ

受信したメッセージの QAManager 受信確認モードを返します。

Visual Basic 構文

```
Public ReadOnly Property Mode As AcknowledgementMode
```

C# 構文

```
public AcknowledgementMode Mode {get;}
```

備考

使用可能な値のリストについては、AcknowledgementMode を参照してください。AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT と AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT は QAManager インスタンスで使用されます。AcknowledgementMode.TRANSACTIONAL は QATransactionalManager インスタンス用のモードです。

QAManagerFactory クラス

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

Visual Basic 構文

```
Public MustInherit Class QAManagerFactory
```

C# 構文

```
public abstract class QAManagerFactory
```

メンバー

継承されたメンバーを含む QAManagerFactory クラスのすべてのメンバー。

名前	説明
CreateQAManager メソッド	指定されたプロパティを持つ新しい QAManager インスタンスを返します。
CreateQATransactionalManager メソッド	指定されたプロパティを持つ新しい QATransactionalManager インスタンスを返します。
Instance プロパティ	QAManagerFactory のシングルトンインスタンスです。

備考

QAManagerFactory のインスタンスは 1 つだけ持つことができます。

CreateQAManager メソッド

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

オーバーロードリスト

名前	説明
CreateQAManager() メソッド	デフォルトのプロパティを持つ新しい QAManager インスタンスを返します。
CreateQAManager(Hashtable) メソッド	指定されたプロパティをハッシュテーブルとして持つ新しい QAManager インスタンスを返します。
CreateQAManager(string) メソッド	指定されたプロパティを持つ新しい QAManager インスタンスを返します。

CreateQAManager() メソッド

デフォルトのプロパティを持つ新しい QAManager インスタンスを返します。

Visual Basic 構文

```
Public Overridable Function CreateQAManager() As QAManager
```

C# 構文

```
public virtual abstract QAManager CreateQAManager()
```

戻り値

新しい QAManager インスタンス。

例外

- **QAException クラス**　Manager の作成で問題が発生した場合にスローされます。

備考

インスタンスの作成後に QAManagerBase.SetProperty を使用して、QAnywhere Manager の設定プロパティをプログラムで設定できます。

QAnywhere Manager の設定プロパティのリストについては、「QAnywhere Manager の設定プロパティ」85 ページを参照してください。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- QAManager インターフェイス [QAnywhere .NET]231 ページ

CreateQAManager(Hashtable) メソッド

指定されたプロパティをハッシュテーブルとして持つ新しい QAManager インスタンスを返します。

Visual Basic 構文

```
Public Overridable Function CreateQAManager(
     ByVal properties As Hashtable
) As QAManager
```

C# 構文

```
public virtual abstract QAManager CreateQAManager(Hashtable properties)
```

パラメーター

- **properties** QAManager のインスタンスを設定するためのハッシュテーブル。

戻り値

新しい QAManager インスタンス。

例外

- **QAException クラス** Manager の作成で問題が発生した場合にスローされます。

備考

ハッシュテーブルパラメーターが NULL の場合、QAManager はデフォルトのプロパティで作成されます。インスタンスの作成後に QAManagerBase.SetProperty を使用して、QAnywhere Manager の設定プロパティをプログラムで設定できます。

QAnywhere Manager の設定プロパティのリストについては、「QAnywhere Manager の設定プロパティ」85 ページを参照してください。

参照

- QAManager インターフェイス [QAnywhere .NET]231 ページ

CreateQAManager(string) メソッド

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

Visual Basic 構文

```
Public Overridable Function CreateQAManager(
     ByVal iniFile As String
) As QAManager
```

C# 構文

```
public virtual abstract QAManager CreateQAManager(string iniFile)
```

パラメーター

- **iniFile**　QAManager インスタンスを設定するためのプロパティファイル

戻り値

新しい QAManager インスタンス。

例外

- **QAException クラス**　Manager の作成で問題が発生した場合にスローされます。

備考

プロパティファイルパラメーターが NULL の場合、QAManager はデフォルトのプロパティを使用して作成されます。インスタンスの作成後に QAManagerBase.SetProperty を使用して、QAnywhere Manager の設定プロパティをプログラムで設定できます。

QAnywhere Manager の設定プロパティのリストについては、「QAnywhere Manager の設定プロパティ」85 ページを参照してください。

詳細については、「ファイルでの QAnywhere Manager 設定プロパティの設定」86 ページを参照してください。

参照

- QAManager インターフェイス [QAnywhere .NET]231 ページ

CreateQATransactionalManager メソッド

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

オーバーロードリスト

名前	説明
CreateQATransactionalManager() メソッド	デフォルトのプロパティを持つ新しい QATransactionalManager インスタンスを返します。
CreateQATransactionalManager(Hashtable) メソッド	指定されたプロパティをハッシュテーブルとして持つ新しい QATransactionalManager インスタンスを返します。
CreateQATransactionalManager(string) メソッド	指定されたプロパティを持つ新しい QATransactionalManager インスタンスを返します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

CreateQATransactionalManager() メソッド

デフォルトのプロパティを持つ新しい QATransactionalManager インスタンスを返します。

Visual Basic 構文

```
Public Overridable Function CreateQATransactionalManager()
    As QATransactionalManager
```

C# 構文

```
public virtual abstract QATransactionalManager
CreateQATransactionalManager()
```

戻り値

新しい QATransactionalManager インスタンス。

例外

- **QAException クラス**　Manager の作成で問題が発生した場合にスローされます。

備考

インスタンスの作成後に QAManagerBase.SetProperty を使用して、QAnywhere Manager の設定プロパティをプログラムで設定できます。

QAnywhere Manager の設定プロパティのリストについては、「QAnywhere Manager の設定プロパティ」85 ページを参照してください。

参照

- QATransactionalManager インターフェイス [QAnywhere .NET]309 ページ

CreateQATransactionalManager(Hashtable) メソッド

指定されたプロパティをハッシュテーブルとして持つ新しい QATransactionalManager インスタンスを返します。

Visual Basic 構文

```
Public Overridable Function CreateQATransactionalManager(
     ByVal properties As Hashtable
) As QATransactionalManager
```

C# 構文

```
public virtual abstract QATransactionalManager
CreateQATransactionalManager(
     Hashtable properties
)
```

パラメーター

- **properties**　QATransactionalManager のインスタンスを設定するためのハッシュテーブル。

戻り値

新しい QATransactionalManager。

例外

- **QAException クラス**　Manager の作成で問題が発生した場合にスローされます。

備考

ハッシュテーブルパラメーターが NULL の場合、QATransactionalManager はデフォルトのプロパティで作成されます。インスタンスの作成後に QAManagerBase.SetProperty を使用して、QAnywhere Manager の設定プロパティをプログラムで設定できます。

QAnywhere Manager の設定プロパティのリストについては、「QAnywhere Manager の設定プロパティ」85 ページを参照してください。

参照

- QATransactionalManager インターフェイス [QAnywhere .NET]309 ページ

CreateQATransactionalManager(string) メソッド

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

Visual Basic 構文

```
Public Overridable Function CreateQATransactionalManager(
     ByVal iniFile As String
) As QATransactionalManager
```

C# 構文

```
public virtual abstract QATransactionalManager
CreateQATransactionalManager(
     string iniFile
)
```

パラメーター

- **iniFile**　QATransactionalManager インスタンスを設定するためのプロパティファイル。QATransactionalManager インスタンスをデフォルトの設定で作成する場合は null。

戻り値

新しい QATransactionalManager。

例外

- **QAException クラス**　Manager の作成で問題が発生した場合にスローされます。

備考

プロパティファイルパラメーターが NULL の場合、QATransactionalManager はデフォルトのプロパティを使用して作成されます。インスタンスの作成後に QAManagerBase.SetProperty を使用して、QAnywhere Manager の設定プロパティをプログラムで設定できます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

QAnywhere Manager の設定プロパティのリストについては、「QAnywhere Manager の設定プロパティ」85 ページを参照してください。

詳細については、「ファイルでの QAnywhere Manager 設定プロパティの設定」86 ページを参照してください。

参照

- QATransactionalManager インターフェイス [QAnywhere .NET]309 ページ

Instance プロパティ

QAManagerFactory のシングルトンインスタンスです。

Visual Basic 構文

```
Public Shared ReadOnly Property Instance As QAManagerFactory
```

C# 構文

```
public QAManagerFactory Instance {get;}
```

例外

- **QAException クラス**　マネージャーファクトリの作成で問題が発生した場合にスローされます。

QAMessage インターフェイス

メッセージプロパティとヘッダーファイルを設定するためのインターフェイスがあります。

Visual Basic 構文

```
Public Interface QAMessage
```

C# 構文

```
public interface QAMessage
```

派生クラス

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QATextMessage インターフェイス [QAnywhere .NET]304 ページ

メンバー

継承されたメンバーを含む QAMessage インターフェイスのすべてのメンバー。

名前	説明
ClearBody メソッド	メッセージの本文をクリアします。

名前	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。
GetByteProperty メソッド	byte 型のメッセージプロパティを取得します。
GetDoubleProperty メソッド	double 型のメッセージプロパティを取得します。
GetFloatProperty メソッド	float 型のメッセージプロパティを取得します。
GetIntProperty メソッド	int 型のメッセージプロパティを取得します。
GetLongProperty メソッド	long 型のメッセージプロパティを取得します。
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	string 型のメッセージプロパティを取得します。
PropertyExists メソッド	指定されたプロパティがこのメッセージに設定されているかどうかを示します。
SetBooleanProperty メソッド	boolean 型のプロパティを設定します。
SetByteProperty メソッド	byte 型のプロパティを設定します。
SetDoubleProperty メソッド	double 型のプロパティを設定します。
SetFloatProperty メソッド	float 型のプロパティを設定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
SetIntProperty メソッド	int 型のプロパティを設定します。
SetLongProperty メソッド	long 型のプロパティを設定します。
SetProperty メソッド	プロパティを設定します。
SetSbyteProperty メソッド	signed byte 型のプロパティを設定します。
SetShortProperty メソッド	short 型のプロパティを設定します。
SetStringProperty メソッド	string 型のプロパティを設定します。
Address プロパティ	QAMessage インスタンスの送信先アドレスです。
Expiration プロパティ	メッセージの有効期限を取得します。
InReplyToID プロパティ	どのメッセージへの返信であるかを示すメッセージ ID です。
MessageID プロパティ	メッセージのグローバルにユニークな ID です。
Priority プロパティ	メッセージの優先度 (0 から 9) です。
Redelivered プロパティ	受信されたが受信確認されていないメッセージであるかどうかを示します。
ReplyToAddress プロパティ	メッセージの返信先アドレスです。
Timestamp プロパティ	メッセージのタイムスタンプです。

備考

派生クラスの QABinaryMessage と QATextMessage には、メッセージ本文の読み込み／書き込みを行うための特別なメソッドがあります。事前定義済みまたはカスタムのメッセージプロパティを設定するには、QAMessage のメソッドを使用します。

事前定義済みのプロパティ名のリストについては、MessageProperties を参照してください。

メッセージプロパティとヘッダーフィールドの設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QATextMessage インターフェイス [QAnywhere .NET]304 ページ

ClearBody メソッド

メッセージの本文をクリアします。

Visual Basic 構文

```
Public Sub ClearBody()
```

C# 構文

```
public void ClearBody()
```

ClearProperties メソッド

メッセージのすべてのプロパティをクリアします。

Visual Basic 構文

```
Public Sub ClearProperties()
```

C# 構文

```
public void ClearProperties()
```

GetBooleanProperty メソッド

boolean 型のメッセージプロパティを取得します。

Visual Basic 構文

```
Public Function GetBooleanProperty(ByVal propName As String) As Boolean
```

C# 構文

```
public bool GetBooleanProperty(string propName)
```

パラメーター

- **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で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ

GetByteProperty メソッド

byte 型のメッセージプロパティを取得します。

Visual Basic 構文

```
Public Function GetByteProperty(ByVal propName As String) As Byte
```

C# 構文

```
public byte GetByteProperty(string propName)
```

パラメーター

- **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で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ

GetDoubleProperty メソッド

double 型のメッセージプロパティを取得します。

Visual Basic 構文

```
Public Function GetDoubleProperty(ByVal propName As String) As Double
```

C# 構文

```
public double GetDoubleProperty(string propName)
```

パラメーター

- **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で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ

GetFloatProperty メソッド

float 型のメッセージプロパティを取得します。

Visual Basic 構文

```
Public Function GetFloatProperty(ByVal propName As String) As Single
```

C# 構文

```
public float GetFloatProperty(string propName)
```

パラメーター

- **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で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

GetIntProperty メソッド

int 型のメッセージプロパティを取得します。

Visual Basic 構文

```
Public Function GetIntProperty(ByVal propName As String) As Integer
```

C# 構文

```
public int GetIntProperty(string propName)
```

パラメーター

- **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で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ

GetLongProperty メソッド

long 型のメッセージプロパティを取得します。

Visual Basic 構文

```
Public Function GetLongProperty(ByVal propName As String) As Long
```

C# 構文

```
public long GetLongProperty(string propName)
```

パラメーター

- **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で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ

GetProperty メソッド

メッセージのプロパティを取得します。

Visual Basic 構文

```
Public Function GetProperty(ByVal propName As String) As Object
```

C# 構文

```
public Object GetProperty(string propName)
```

パラメーター

- **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にスローされます。

備考

プロパティ型は、使用可能ないずれかの単純型、string 型、または DateTime 型でなければなりません。

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。

Visual Basic 構文

```
Public Function GetPropertyNames() As System.Collections.IEnumerator
```

C# 構文

```
public System.Collections.IEnumerator GetPropertyNames()
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

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。

Visual Basic 構文

```
Public Function GetPropertyType(
     ByVal propName As String
) As PropertyType
```

C# 構文

```
public PropertyType GetPropertyType(string propName)
```

パラメーター

- **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。

Visual Basic 構文

```
Public Function GetSbyteProperty(ByVal propName As String) As SByte
```

C# 構文

```
public sbyte GetSbyteProperty(string propName)
```

パラメーター

- **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で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ

GetShortProperty メソッド

short 型のメッセージプロパティを取得します。

Visual Basic 構文

```
Public Function GetShortProperty(ByVal propName As String) As Short
```

C# 構文

```
public short GetShortProperty(string propName)
```

パラメーター

- **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で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ

GetStringProperty メソッド

string 型のメッセージプロパティを取得します。

Visual Basic 構文

```
Public Function GetStringProperty(ByVal propName As String) As String
```

C# 構文

```
public string GetStringProperty(string propName)
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- **propName**　プロパティ名。

戻り値

プロパティの値。該当するプロパティが存在しない場合は NULL。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ

PropertyExists メソッド

指定されたプロパティがこのメッセージに設定されているかどうかを示します。

Visual Basic 構文

```
Public Function PropertyExists(ByVal propName As String) As Boolean
```

C# 構文

```
public bool PropertyExists(string propName)
```

パラメーター

- **propName**　プロパティ名。

戻り値

プロパティが存在する場合は true。

SetBooleanProperty メソッド

boolean 型のプロパティを設定します。

Visual Basic 構文

```
Public Sub SetBooleanProperty(
     ByVal propName As String,
    ByVal val As Boolean
)
```

C# 構文

```
public void SetBooleanProperty(string propName, bool val)
```

パラメーター

- **propName**　プロパティ名。
- **val**　プロパティの値。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ

SetByteProperty メソッド

byte 型のプロパティを設定します。

Visual Basic 構文

```
Public Sub SetByteProperty(ByVal propName As String, ByVal val As Byte)
```

C# 構文

```
public void SetByteProperty(string propName, byte val)
```

パラメーター

- **propName**　プロパティ名。
- **val**　プロパティの値。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ

SetDoubleProperty メソッド

double 型のプロパティを設定します。

Visual Basic 構文

```
Public Sub SetDoubleProperty(
     ByVal propName As String,
    ByVal val As Double
)
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

C# 構文

```
public void SetDoubleProperty(string propName, double val)
```

パラメーター

- **propName**　プロパティ名。
- **val**　プロパティの値。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ

SetFloatProperty メソッド

float 型のプロパティを設定します。

Visual Basic 構文

```
Public Sub SetFloatProperty(
     ByVal propName As String,
    ByVal val As Single
)
```

C# 構文

```
public void SetFloatProperty(string propName, float val)
```

パラメーター

- **propName**　プロパティ名。
- **val**　プロパティの値。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ

SetIntProperty メソッド

int 型のプロパティを設定します。

Visual Basic 構文

```
Public Sub SetIntProperty(
     ByVal propName As String,
    ByVal val As Integer
)
```

C# 構文

```
public void SetIntProperty(string propName, int val)
```

パラメーター

- **propName**　プロパティ名。
- **val**　プロパティの値。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ

SetLongProperty メソッド

long 型のプロパティを設定します。

Visual Basic 構文

```
Public Sub SetLongProperty(ByVal propName As String, ByVal val As Long)
```

C# 構文

```
public void SetLongProperty(string propName, long val)
```

パラメーター

- **propName**　プロパティ名。
- **val**　プロパティの値。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

SetProperty メソッド

プロパティを設定します。

Visual Basic 構文

```
Public Sub SetProperty(ByVal propName As String, ByVal val As Object)
```

C# 構文

```
public void SetProperty(string propName, Object val)
```

パラメーター

- **propName** プロパティ名。
- **val** プロパティの値。

備考

プロパティ型は、使用可能ないずれかの単純型、文字列型、または DateTime 型でなければなりません。

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ

SetSbyteProperty メソッド

signed byte 型のプロパティを設定します。

Visual Basic 構文

```
Public Sub SetSbyteProperty(
     ByVal propName As String,
    ByVal val As SByte
)
```

C# 構文

```
public void SetSbyteProperty(string propName, sbyte val)
```

パラメーター

- **propName** プロパティ名。
- **val** プロパティの値。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ

SetShortProperty メソッド

short 型のプロパティを設定します。

Visual Basic 構文

```
Public Sub SetShortProperty(
     ByVal propName As String,
    ByVal val As Short
)
```

C# 構文

```
public void SetShortProperty(string propName, short val)
```

パラメーター

- **propName**　プロパティ名。
- **val**　プロパティの値。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ

SetStringProperty メソッド

string 型のプロパティを設定します。

Visual Basic 構文

```
Public Sub SetStringProperty(
     ByVal propName As String,
    ByVal val As String
)
```

C# 構文

```
public void SetStringProperty(string propName, string val)
```

パラメーター

- **propName**　プロパティ名。
- **val**　プロパティの値。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere .NET]189 ページ

Address プロパティ

QAMessage インスタンスの送信先アドレスです。

Visual Basic 構文

```
Public ReadOnly Property Address As String
```

C# 構文

```
public string Address {get;}
```

備考

このフィールドは、メッセージの送信時には無視されます。送信操作が完了すると、このフィールドには QAManagerBase.PutMessage で指定された送信先アドレスが入ります。

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- QAManagerBase.PutMessage メソッド [QAnywhere .NET]263 ページ

Expiration プロパティ

メッセージの有効期限を取得します。

Visual Basic 構文

```
Public ReadOnly Property Expiration As Date
```

C# 構文

```
public DateTime Expiration {get;}
```

備考

メッセージの Expiration ヘッダーフィールドは、メッセージの送信時には空のままです。このフィールドは、send メソッドが完了した後、メッセージの有効期限を保持します。

メッセージの有効期限は、QAManagerBase.PutMessageTimeToLive の存続時間の引数を現在の時間に追加することで設定されるため、このプロパティは読み込み専用です。

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

InReplyToID プロパティ

どのメッセージへの返信であるかを示すメッセージ ID です。

Visual Basic 構文

```
Public Property InReplyToID As String
```

C# 構文

```
public string InReplyToID {get;set;}
```

備考

NULL の場合もあります。

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

MessageID プロパティ

メッセージのグローバルにユニークな ID です。

Visual Basic 構文

```
Public ReadOnly Property MessageID As String
```

C# 構文

```
public string MessageID {get;}
```

備考

このプロパティは、メッセージがキューに登録されるまで NULL のままです。

QAManagerBase.PutMessage を使用してメッセージが送信されると、MessageID が NULL になるため無視できます。send メソッドが戻ると、割り当てられた値がここに格納されます。

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- QAManagerBase.PutMessage メソッド [QAnywhere .NET]263 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

Priority プロパティ

メッセージの優先度 (0 から 9) です。

Visual Basic 構文

```
Public Property Priority As Integer
```

C# 構文

```
public int Priority {get;set;}
```

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

Redelivered プロパティ

受信されたが受信確認されていないメッセージであるかどうかを示します。

Visual Basic 構文

```
Public ReadOnly Property Redelivered As Boolean
```

C# 構文

```
public bool Redelivered {get;}
```

備考

受信側の QAManager は、受信中のメッセージが以前に受信されたことを検知した場合に、Redelivered プロパティを設定します。

たとえば、AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT でオープンされた QAManager を使用してアプリケーションがメッセージを受信し、メッセージの受信確認を行わずに終了したとします。このアプリケーションは、次の起動時に同じメッセージを再受信し、Redelivered ヘッダーを true に設定します。

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- QAManager インターフェイス [QAnywhere .NET]231 ページ
- AcknowledgementMode 列挙体 [QAnywhere .NET]317 ページ

ReplyToAddress プロパティ

メッセージの返信先アドレスです。

Visual Basic 構文

```
Public Property ReplyToAddress As String
```

C# 構文

```
public string ReplyToAddress {get;set;}
```

備考

NULL の場合もあります。

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

Timestamp プロパティ

メッセージのタイムスタンプです。

Visual Basic 構文

```
Public ReadOnly Property Timestamp As Date
```

C# 構文

```
public DateTime Timestamp {get;}
```

備考

メッセージの Timestamp ヘッダーフィールドには、メッセージが作成された時刻が格納されます。

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

QATextMessage インターフェイス

QATextMessage は QAMessage クラスを継承したもので、メッセージ本文にテキストが追加されます。

Visual Basic 構文

```
Public Interface QATextMessage Inherits QAMessage
```

C# 構文

```
public interface QATextMessage : QAMessage
```

基本クラス

- QAMessage インターフェイス [QAnywhere .NET]285 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

メンバー

継承されたメンバーを含む QATextMessage インターフェイスのすべてのメンバー。

名前	説明
ClearBody メソッド	メッセージの本文をクリアします。
ClearProperties メソッド	メッセージのすべてのプロパティをクリアします。
GetBooleanProperty メソッド	boolean 型のメッセージプロパティを取得します。
GetByteProperty メソッド	byte 型のメッセージプロパティを取得します。
GetDoubleProperty メソッド	double 型のメッセージプロパティを取得します。
GetFloatProperty メソッド	float 型のメッセージプロパティを取得します。
GetIntProperty メソッド	int 型のメッセージプロパティを取得します。
GetLongProperty メソッド	long 型のメッセージプロパティを取得します。
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	string 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。
ReadText メソッド	未読テキストを指定されたバッファー内に読み込みます。

名前	説明
Reset メソッド	メッセージのテキスト位置を先頭にリセットします。
SetBooleanProperty メソッド	boolean 型のプロパティを設定します。
SetByteProperty メソッド	byte 型のプロパティを設定します。
SetDoubleProperty メソッド	double 型のプロパティを設定します。
SetFloatProperty メソッド	float 型のプロパティを設定します。
SetIntProperty メソッド	int 型のプロパティを設定します。
SetLongProperty メソッド	long 型のプロパティを設定します。
SetProperty メソッド	プロパティを設定します。
SetSbyteProperty メソッド	signed byte 型のプロパティを設定します。
SetShortProperty メソッド	short 型のプロパティを設定します。
SetStringProperty メソッド	string 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。
Address プロパティ	QAMessage インスタンスの送信先アドレスです。
Expiration プロパティ	メッセージの有効期限を取得します。
InReplyToID プロパティ	どのメッセージへの返信であるかを示すメッセージ ID です。
MessageID プロパティ	メッセージのグローバルにユニークな ID です。
Priority プロパティ	メッセージの優先度 (0 から 9) です。
Redelivered プロパティ	受信されたが受信確認されていないメッセージであるかどうかを示します。
ReplyToAddress プロパティ	メッセージの返信先アドレスです。
Text プロパティ	メッセージテキストです。
TextLength プロパティ	メッセージの文字数です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
Timestamp プロパティ	メッセージのタイムスタンプです。

備考

QATextMessage には、メッセージ本文からのテキストの読み込み／書き込みを行うためのメソッドがあります。

メッセージが最初に作成された時点では、メッセージ本文は書き込み専用モードになっています。メッセージ送信元のクライアントは、メッセージを送信した後、そのメッセージを保持し変更できます。ただし、それによって、送信されたメッセージが変更されることはありません。同じメッセージオブジェクトを複数回送信できます。

メッセージが受信された時点でプロバイダーは QATextMessage.Reset() を呼び出します。これによりメッセージ本文が読み込み専用モードになり、メッセージ本文の先頭から値の読み込みが開始されます。

参照

- QABinaryMessage インターフェイス [QAnywhere .NET]199 ページ
- QAMessage インターフェイス [QAnywhere .NET]285 ページ

ReadText メソッド

未読テキストを指定されたバッファー内に読み込みます。

Visual Basic 構文

```
Public Function ReadText(ByVal buf As StringBuilder) As Integer
```

C# 構文

```
public int ReadText(StringBuilder buf)
```

パラメーター

- **buf**　テキストの読み込み先バッファー。

戻り値

読み込まれた文字数。読み込めるテキストが残っていない場合は -1。

備考

未読テキストが追加された場合、そのテキストを読み込むにはこのメソッドを繰り返し呼び出す必要があります。読み込みは、未読テキストの先頭から実行されます。

Reset メソッド

メッセージのテキスト位置を先頭にリセットします。

Visual Basic 構文

```
Public Sub Reset()
```

C# 構文

```
public void Reset()
```

WriteText メソッド

メッセージテキストの末尾にテキストを追加します。

Visual Basic 構文

```
Public Sub WriteText(ByVal val As String)
```

C# 構文

```
public void WriteText(string val)
```

パラメーター

- **val** 追加するテキスト。

Text プロパティ

メッセージテキストです。

Visual Basic 構文

```
Public Property Text As String
```

C# 構文

```
public string Text {get;set;}
```

備考

メッセージのサイズが、QAManager.MAX_IN_MEMORY_MESSAGE_SIZE で指定された最大サイズを超える場合、このプロパティは NULL になります。この場合は、QATextMessage.ReadText メソッドを使用してテキストを読み込みます。

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

参照

- QATextMessage.ReadText メソッド [QAnywhere .NET]307 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

TextLength プロパティ

メッセージの文字数です。

Visual Basic 構文

```
Public ReadOnly Property TextLength As Long
```

C# 構文

```
public long TextLength {get;}
```

QATransactionalManager インターフェイス

QATransactionalManager クラスは QAManagerBase から派生し、トランザクション指向の QAnywhere メッセージング操作を管理します。

Visual Basic 構文

```
Public Interface QATransactionalManager Inherits QAManagerBase
```

C# 構文

```
public interface QATransactionalManager : QAManagerBase
```

基本クラス

- QAManagerBase インターフェイス [QAnywhere .NET]239 ページ

メンバー

継承されたメンバーを含む QATransactionalManager インターフェイスのすべてのメンバー。

名前	説明
BrowseMessages メソッド	この方法は使用されなくなりました。
BrowseMessagesByID メソッド	指定されたメッセージ ID を持つメッセージを参照します。
BrowseMessagesByQueue メソッド	指定されたアドレスに送信され、次以降に取得可能な一連の受信待機中メッセージを参照します。
BrowseMessagesBySelector メソッド	メッセージストアのキューに登録されているメッセージのうち、指定されたセレクターを満たすメッセージを参照します。
CancelMessage メソッド	指定されたメッセージ ID を持つメッセージをキャンセルします。

名前	説明
Close メソッド	QAnywhere メッセージシステムへの接続をクローズして、QAManagerBase で使用していたリソースをすべて解放します。
Commit メソッド	現在のトランザクションをコミットし、新しいトランザクションを開始します。
CreateBinaryMessage メソッド	QABinaryMessage オブジェクトを作成します。
CreateTextMessage メソッド	QATextMessage オブジェクトを作成します。
GetBooleanStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの boolean 値を取得します。
GetDoubleStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの double 値を取得します。
GetFloatStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの float 値を取得します。
GetIntStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの int 値を取得します。
GetLongStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの long 値を取得します。
GetMessage メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
GetMessageBySelector メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
GetMessageBySelectorNoWait メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
GetMessageBySelectorTimeout メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
GetMessageNoWait メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
GetMessageTimeout メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
GetQueueDepth メソッド	指定されたフィルターに基づいて、すべてのキューの長さの合計を返します。
GetSbyteStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの signed byte 値を取得します。
GetShortStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの short 値を取得します。
GetStoreProperty メソッド	メッセージストアプロパティを示す System.Object を取得します。
GetStorePropertyNames メソッド	メッセージストアのプロパティ名の列挙子を取得します。
GetStringStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの string 値を取得します。
Open メソッド	QATransactionalManager インスタンスをオープンします。
PropertyExists メソッド	指定されたプロパティに値が現在存在するかどうかをテストします。
PutMessage メソッド	別の QAnywhere クライアントに送信するメッセージを準備します。
PutMessageTimeToLive メソッド	別の QAnywhere クライアントに送信するメッセージを準備します。
ReOpen メソッド	QAManagerBase を再度開きます。
Rollback メソッド	現在のトランザクションをロールバックし、新しいトランザクションを開始します。
SetBooleanStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを boolean 値に設定します。
SetDoubleStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを double 値に設定します。
SetExceptionListener メソッド	QAnywhere メッセージの非同期的な処理時に QAExceptions を受信するように、ExceptionListener デリゲートを設定します。

名前	説明
SetExceptionListener2 メソッド	QAnywhere メッセージの非同期的な処理時に QAExceptions を受信するように、ExceptionListener2 デリゲートを設定します。
SetFloatStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを float 値に設定します。
SetIntStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを int 値に設定します。
SetLongStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを long 値に設定します。
SetMessageListener メソッド	QAnywhere メッセージを非同期に受信するように、MessageListener デリゲートを設定します。
SetMessageListener2 メソッド	QAnywhere メッセージを非同期的に受信するように、MessageListener2 デリゲートを設定します。
SetMessageListenerBySelector メソッド	メッセージセレクターを使用して、QAnywhere メッセージを非同期的に受信するように、MessageListener デリゲートを設定します。
SetMessageListenerBySelector2 メソッド	メッセージセレクターを使用して、QAnywhere メッセージを非同期的に受信するように、MessageListener2 デリゲートを設定します。
SetProperty メソッド	QAnywhere Manager の設定プロパティをプログラムで設定できるようにします。
SetSbyteStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを sbyte (signed byte) 値に設定します。
SetShortStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを short 値に設定します。
SetStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを System.Object 値に設定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
SetStringStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを string 値に設定します。
Start メソッド	着信メッセージをメッセージリスナーで受信するための QAManagerBase を起動します。
Stop メソッド	QAManagerBase による着信メッセージの受信を停止します。
TriggerSendReceive メソッド	QAnywhere メッセージサーバーとの同期処理を発生させて、他のクライアント宛てのメッセージをアップロードし、ローカルクライアント宛てのメッセージをダウンロードします。
Mode プロパティ	受信したメッセージの QAManager 受信確認モードを返します。

備考

この動作の完全な説明については、QAManagerBase を参照してください。

QATransactionalManager は、トランザクション指向の受信確認でのみ使用できます。QAManagerBase.PutMessage と QAManagerBase.GetMessage のすべての呼び出しをコミットする場合は、QATransactionalManager.Commit() メソッドを使用します。

詳細については、「トランザクション指向メッセージングの実装」66 ページを参照してください。

参照

- QATransactionalManager インターフェイス [QAnywhere .NET]309 ページ

Commit メソッド

現在のトランザクションをコミットし、新しいトランザクションを開始します。

Visual Basic 構文

```
Public Sub Commit()
```

C# 構文

```
public void Commit()
```

例外

- **QAException クラス**　コミットで問題が発生した場合にスローされます。

備考

> **注意**
> このメソッドは、QAManagerBase.PutMessage と QAManagerBase.GetMessage のすべての呼び出しをコミットします。最初のトランザクションは、QATransactionalManager.Open() の呼び出しで開始されます。

参照

- QATransactionalManager インターフェイス [QAnywhere .NET]309 ページ

Open メソッド

QATransactionalManager インスタンスをオープンします。

Visual Basic 構文

```
Public Sub Open()
```

C# 構文

```
public void Open()
```

例外

- **QAException クラス**　Manager のオープンで問題が発生した場合にスローされます。

備考

Open メソッドは、Manager を作成した後、最初に呼び出す必要があるメソッドです。

参照

- QATransactionalManager インターフェイス [QAnywhere .NET]309 ページ

Rollback メソッド

現在のトランザクションをロールバックし、新しいトランザクションを開始します。

Visual Basic 構文

```
Public Sub Rollback()
```

C# 構文

```
public void Rollback()
```

例外

- **QAException クラス**　ロールバックで問題が発生した場合にスローされます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

このメソッドは、コミットされていない QAManagerBase.PutMessage と QAManagerBase.GetMessage のすべての呼び出しをロールバックします。

参照

- QATransactionalManager インターフェイス [QAnywhere .NET]309 ページ

ExceptionListener デリゲート

ExceptionListener デリゲートの定義です。

Visual Basic 構文

```
Public Delegate Sub ExceptionListener(
     ByVal ex As QAException,
    ByVal msg As QAMessage
)
```

C# 構文

```
public delegate void ExceptionListener(QAException ex, QAMessage msg);
```

パラメーター

- **ex**　発生した例外。
- **msg**　受信したメッセージ。またはメッセージが構築できなかった場合は NULL。

備考

ExceptionListener は QAManagerBase.SetExceptionListener に渡します。

ExceptionListener2 デリゲート

ExceptionListener2 デリゲートの定義です。

Visual Basic 構文

```
Public Delegate Sub ExceptionListener2(
     ByVal mgr As QAManagerBase,
    ByVal ex As QAException,
    ByVal msg As QAMessage
)
```

C# 構文

```
public delegate void ExceptionListener2(
     QAManagerBase mgr,
    QAException ex,
    QAMessage msg
);
```

パラメーター

- **mgr** メッセージを処理した QAManagerBase。
- **ex** 発生した例外。
- **msg** 受信したメッセージ。またはメッセージが構築できなかった場合は NULL。

備考

ExceptionListener2 は QAManagerBase.SetExceptionListener2 に渡します。

MessageListener デリゲート

MessageListener デリゲートの定義です。

Visual Basic 構文

```
Public Delegate Sub MessageListener(ByVal msg As QAMessage)
```

C# 構文

```
public delegate void MessageListener(QAMessage msg);
```

パラメーター

- **msg** 受信したメッセージ。

備考

MessageListener は QAManagerBase.SetMessageListener メソッドに渡します。

参照

- QAManagerBase.SetMessageListener メソッド [QAnywhere .NET]270 ページ

MessageListener2 デリゲート

MessageListener2 デリゲートの定義です。

Visual Basic 構文

```
Public Delegate Sub MessageListener2(
     ByVal mgr As QAManagerBase,
    ByVal msg As QAMessage
)
```

C# 構文

```
public delegate void MessageListener2(QAManagerBase mgr, QAMessage msg);
```

パラメーター

- **mgr** メッセージを受信した QAManagerBase。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- **msg**　受信したメッセージ。

備考

MessageListener2 は QAManagerBase.SetMessageListener2 に渡します。

AcknowledgementMode 列挙体

QAnywhere クライアントアプリケーションによってどのようにメッセージが確認されるかを示します。

Visual Basic 構文

```
Public Enum AcknowledgementMode
```

C# 構文

```
public enum AcknowledgementMode
```

メンバー

メンバー名	説明
EXPLICIT_ACKNOWLEDGEMENT	受信メッセージは、QAManager のいずれかの受信確認メソッドを使用して確認されます。
IMPLICIT_ACKNOWLEDGEMENT	すべてのメッセージは、クライアントアプリケーションで受信されるとすぐに受信確認されます。 メッセージを同期的に受信する場合、メッセージは QAManagerBase.GetMessage(string) メソッドが返されるとすぐに受信確認されます。メッセージを非同期的に受信する場合、メッセージはイベント処理関数が返されるとすぐに受信確認されます。
TRANSACTIONAL	このモードでは、メッセージの受信確認はトランザクションの一部として行われます。それ以外では行われません。 このモードは、QATransactionalManager インスタンスに自動的に割り当てられます。

備考

暗黙的または明示的な受信確認モードは、QAManager.Open(AcknowledgementMode) メソッドを使用して、QAManager インスタンスに割り当てられます。

詳細については、「QAnywhere API の初期化」54 ページを参照してください。

暗黙的な受信確認では、メッセージは、クライアントアプリケーションで受信されるとすぐに受信確認されます。明示的な受信確認では、いずれかの QAManager 受信確認メソッドを呼び出してください。すべてのステータス変更は、サーバーによってクライアント間で伝達されます。

詳細については、「同期的なメッセージ受信」71 ページと「非同期的なメッセージ受信」73 ページを参照してください。

参照

- QAManager インターフェイス [QAnywhere .NET]231 ページ
- QATransactionalManager インターフェイス [QAnywhere .NET]309 ページ
- QAManagerBase インターフェイス [QAnywhere .NET]239 ページ

MessageType 列挙体

MessageProperties.MSG_TYPE メッセージプロパティの定数値を定義します。

Visual Basic 構文

```
Public Enum MessageType
```

C# 構文

```
public enum MessageType
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

メンバー

メンバー名	説明	値
NETWORK_STATUS_NOTIFICATION	QAnywhere クライアントアプリケーションにネットワークステータスの変更を通知する場合に使用する、QAnywhere システムメッセージを特定します。 ネットワークステータスの変更は、システムメッセージを受信するデバイスに適用されます。新しいネットワークステータス情報を特定するには、MessageProperties.ADAPTERS、MessageProperties.RASNAMES、MessageProperties.NETWORK_STATUS の各フィールドを使用します。 詳細については、「システムキュー」60 ページを参照してください。	14
PUSH_NOTIFICATION	QAnywhere クライアントアプリケーションに Push 通知を通知する場合に使用する、QAnywhere システムメッセージを特定します。 オンデマンドの qaagent ポリシーを使用した場合の一般的な応答は、QAManagerBase.TriggerSendReceive() を呼び出し、中央のメッセージサーバーで待機しているメッセージを受信することです。 詳細については、「システムキュー」60 ページを参照してください。	13

メンバー名	説明	値
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。	0

参照

- QAManagerBase.TriggerSendReceive メソッド [QAnywhere .NET]278 ページ
- MessageProperties クラス [QAnywhere .NET]189 ページ

例

次の例は、QAnywhere システムメッセージの処理で使用される onSystemMessage メソッドを示します。メッセージタイプは、MessageType.NETWORK_STATUS_NOTIFICATION と比較されます。

```
private void onSystemMessage(QAMessage msg) {
    QATextMessage   t_msg;
    MessageType     msg_type;
    String          network_adapters;
    String          network_names;
    String          network_info;

    t_msg = (QATextMessage)msg;
    if(t_msg != null) {
        // Evaluate message type.
        msg_type = (MessageType)t_msg.GetIntProperty( MessageProperties.MSG_TYPE );
      if(msg_type == MessageType.NETWORK_STATUS_NOTIFICATION) {
          // Handle network status notification...
      }
    }
}
```

PropertyType 列挙体

QAMessage プロパティ型列挙体。C# の型に自然に対応します。

Visual Basic 構文

```
Public Enum PropertyType
```

C# 構文

```
public enum PropertyType
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

メンバー

メンバー名	説明
BOOLEAN	プロパティ型が boolean であることを示します。
BYTE	プロパティ型が signed byte であることを示します。
DOUBLE	プロパティ型が double であることを示します。
FLOAT	プロパティ型が float であることを示します。
INT	プロパティ型が int であることを示します。
LONG	プロパティ型が long であることを示します。
SHORT	プロパティ型が short であることを示します。
STRING	プロパティ型が string であることを示します。
UNKNOWN	プロパティ型が不定であることを示します。通常はプロパティを認識できないことが原因です。

QueueDepthFilter 列挙体

QAManagerBase.GetQueueDepth(QueueDepthFilter) と QAManagerBase.GetQueueDepth(string, QueueDepthFilter) のキューの長さのフィルター値を提供します。

Visual Basic 構文

```
Public Enum QueueDepthFilter
```

C# 構文

```
public enum QueueDepthFilter
```

メンバー

メンバー名	説明
ALL	着信メッセージと送信メッセージの両方をカウントします。 システムメッセージと期限切れのメッセージは、キューの長さカウントに計上されません。
INCOMING	着信メッセージのみカウントします。 着信メッセージは、発信者がメッセージストアのエージェント ID ではないメッセージです。
LOCAL	メッセージストア内のローカルメッセージのみをカウントします。 ローカルメッセージは、発信者と送信先がメッセージストアのエージェント ID であるメッセージです。
OUTGOING	送信メッセージのみカウントします。 送信メッセージは、発信者がメッセージストアのエージェント ID であり、送信先がメッセージストアのエージェント ID ではないメッセージです。

参照

- QAManagerBase.GetQueueDepth メソッド [QAnywhere .NET]258 ページ

StatusCodes 列挙体

この列挙体は、メッセージのステータスコードセットを定義します。

Visual Basic 構文

```
Public Enum StatusCodes
```

C# 構文

```
public enum StatusCodes
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

メンバー

メンバー名	説明	値
CANCELED	メッセージはキャンセルされました。 このコードは MessageProperties.STATUS で使用されます。	40
EXPIRED	メッセージは有効期限が切れる前に受信されなかったため、期限切れになりました。 このコードは MessageProperties.STATUS で使用されます。	30
FINAL	この定数は、メッセージが最終ステータスになったかどうかを示します。 メッセージは、そのステータスがこの定数より大きい場合にのみ最終ステータスになったと見なされます。 このコードは MessageProperties.STATUS で使用されます。	20
LOCAL	メッセージはローカルメッセージストア宛てに送信され、サーバーに送信されません。 このコードは MessageProperties.TRANSMISSION_STATUS で使用されます。	2
PENDING	メッセージは送信されましたが、まだ受信および受信確認されていません。 このコードは MessageProperties.STATUS で使用されます。	1

メンバー名	説明	値
RECEIVED	メッセージが受信され、受信者によって受信確認されました。 このコードは MessageProperties.STATUS で使用されます。	60
RECEIVING	メッセージは受信中であるか、または受信されましたがまだ確認されていません。 このコードは MessageProperties.STATUS で使用されます。	10
TRANSMITTED	メッセージはサーバーに送信されました。 このコードは MessageProperties.TRANSMISSION_STATUS で使用されます。	1
TRANSMITTING	メッセージはサーバーに送信中です。 このコードは MessageProperties.TRANSMISSION_STATUS で使用されます。	3
UNRECEIVABLE	メッセージは受信不可のマークが付けられています。 メッセージに間違った形式があるか、失敗した試行回数が多すぎて配信されませんでした。 このコードは MessageProperties.STATUS で使用されます。	50

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

メンバー名	説明	値
UNTRANSMITTED	メッセージはサーバーに送信されていません。 このコードは MessageProperties.TRANSMISSION_STATUS で使用されます。	0

Web サービス用 QAnywhere .NET API リファレンス

ネームスペース

`iAnywhere.QAnywhere.WS`

WSBase クラス

これは、モバイル Web サービスのコンパイラーで生成されるメインの Web サービスプロキシクラスの基本クラスです。

Visual Basic の構文

```
Public Class WSBase
```

C# の構文

```
public class WSBase
```

メンバー

継承されたメンバーを含む WSBase クラスのすべてのメンバー。

名前	説明
WSBase コンストラクター	設定プロパティファイルで指定されたプロパティを使用して、WSBase インスタンスを構築します。
ClearRequestProperties メソッド	この WSBase に対して送信された要求プロパティをすべてクリアします。
GetResult メソッド	Web サービス要求の結果を表す WSResult オブジェクトを取得します。
GetServiceID メソッド	WSBase のこのインスタンスのサービス ID を取得します。

名前	説明
SetListener メソッド	指定された Web サービス要求の結果を受信するリスナーを設定します。
SetProperty メソッド	WSBase のこのインスタンスの設定プロパティを設定します。
SetQAManager メソッド	この Web サービスクライアントが Web サービス要求を処理するために使用する QAManagerBase を設定します。
SetRequestProperty メソッド	この WSBase が作成した Web サービス要求の要求プロパティを設定します。
SetServiceID メソッド	WSBase のこのインスタンスのユーザー定義 ID を設定します。

WSBase コンストラクター

設定プロパティファイルで指定されたプロパティを使用して、WSBase インスタンスを構築します。

オーバーロードリスト

名前	説明
WSBase() コンストラクター	デフォルトのプロパティを使用して WSBase インスタンスを構築します。
WSBase(String) コンストラクター	設定プロパティファイルで指定されたプロパティを使用して、WSBase インスタンスを構築します。

WSBase() コンストラクター

デフォルトのプロパティを使用して WSBase インスタンスを構築します。

Visual Basic の構文

```
Public Sub New()
```

C# の構文

```
public WSBase()
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例外

- **iAnywhere.QAnywhere.WS.WSException**　WSBase の構築で問題がある場合にスローされます。

WSBase(String) コンストラクター

設定プロパティファイルで指定されたプロパティを使用して、WSBase インスタンスを構築します。

Visual Basic の構文

```
Public Sub New(ByVal iniFile As String)
```

C# の構文

```
public WSBase(string iniFile)
```

パラメーター

- **iniFile**　設定プロパティが含まれているファイル。

例外

- **iAnywhere.QAnywhere.WS.WSException**　WSBase の構築で問題がある場合にスローされます。

備考

有効な設定プロパティには、次のものがあります。

LOG_FILE：ランタイム情報のログを記録するファイルです。

LOG_LEVEL：ログを記録する情報の冗長レベルを制御する、0 ～ 6 の値。6 が最高の冗長レベルです。

WS_CONNECTOR_ADDRESS : Mobile Link サーバーにおける Web サービスコネクターのアドレスです。

デフォルトの WS_CONNECTOR_ADDRESS は ianywhere.connector.webservices¥¥ です。

ClearRequestProperties メソッド

この WSBase に対して送信された要求プロパティをすべてクリアします。

Visual Basic の構文

```
Public Sub ClearRequestProperties()
```

C# の構文

```
public void ClearRequestProperties()
```

GetResult メソッド

Web サービス要求の結果を表す WSResult オブジェクトを取得します。

Visual Basic の構文

```
Public Function GetResult(ByVal requestID As String) As WSResult
```

C# の構文

```
public WSResult GetResult(String requestID)
```

パラメーター

- **requestID**　Web サービス要求の ID。

戻り値

Web サービス要求の結果を表す WSResult インスタンス。

参照

- WSStatus 列挙体 [QAnywhere .NET]374 ページ

GetServiceID メソッド

WSBase のこのインスタンスのサービス ID を取得します。

Visual Basic の構文

```
Public Function GetServiceID() As String
```

C# の構文

```
public String GetServiceID()
```

戻り値

サービス ID。

SetListener メソッド

指定された Web サービス要求の結果を受信するリスナーを設定します。

オーバーロードリスト

名前	説明
SetListener(string, WSListener) メソッド	指定された Web サービス要求の結果を受信するリスナーを設定します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
SetListener(WSListener) メソッド	WSBase のこのインスタンスが作成した、すべての Web サービス要求を受信するリスナーを設定します。

SetListener(string, WSListener) メソッド

指定された Web サービス要求の結果を受信するリスナーを設定します。

Visual Basic の構文

```
Public Sub SetListener(
     ByVal requestID As String,
    ByVal listener As WSListener
)
```

C# の構文

```
public void SetListener(string requestID, WSListener listener)
```

パラメーター

- **requestID**　結果を受信する Web サービス要求の ID。
- **listener**　指定された Web サービス要求が取得可能な場合に呼び出される Listener オブジェクト。

備考

一般にリスナーは、サービスの asyncXYZ メソッドの結果を取得する場合に使用されます。

リスナーを削除するには、listener に NULL を指定して SetListener を呼び出します。

> **注意**
> このメソッドは、以前に呼び出された SetListener で設定されたリスナーを置き換えます。

SetListener(WSListener) メソッド

WSBase のこのインスタンスが作成した、すべての Web サービス要求を受信するリスナーを設定します。

Visual Basic の構文

```
Public Sub SetListener(ByVal listener As WSListener)
```

C# の構文

```
public void SetListener(WSListener listener)
```

パラメーター

- **listener**　Web サービス要求が取得可能な場合に呼び出される Listener オブジェクト。

備考

一般にリスナーは、サービスの asyncXYZ メソッドの結果を取得する場合に使用されます。

リスナーを削除するには、listener に NULL を指定して SetListener を呼び出します。

> **注意**
> このメソッドは、以前に呼び出された SetListener で設定されたリスナーを置き換えます。

SetProperty メソッド

WSBase のこのインスタンスの設定プロパティを設定します。

Visual Basic の構文

```
Public Sub SetProperty(ByVal property As String, ByVal val As String)
```

C# の構文

```
public void SetProperty(string property, string val)
```

パラメーター

- **property**　設定するプロパティ名。
- **val**　プロパティの値。

備考

非同期または同期の Web サービス要求を出す前に、設定プロパティを作成してください。このメソッドは、Web サービス要求が出された後で呼び出された場合は無効です。

有効な設定プロパティには、次のものがあります。

LOG_FILE：ランタイム情報のログを記録するファイルです。

LOG_LEVEL：ログを記録する情報の冗長レベルを制御する、0 ～ 6 の値。6 が最高の冗長レベルです。

WS_CONNECTOR_ADDRESS : Mobile Link サーバーにおける Web サービスコネクターのアドレスです。デフォルトは ianywhere.connector.webservices¥¥ です。

SetQAManager メソッド

この Web サービスクライアントが Web サービス要求を処理するために使用する QAManagerBase を設定します。

Visual Basic の構文

```
Public Sub SetQAManager(ByVal mgr As QAManagerBase)
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

C# の構文

```
public void SetQAManager(QAManagerBase mgr)
```

パラメーター

- **mgr**　使用する QAManagerBase。

備考

> **注意**
> EXPLICIT_ACKNOWLEDGEMENT QAManager を使用する場合は、WSResult の acknowledge() メソッドを呼び出して、非同期の Web サービス要求の結果を受信確認できます。同期の Web サービス要求の結果は、EXPLICIT_ACKNOWLEDGEMENT QAManager を使用している場合であっても、自動的に受信確認されます。IMPLICIT_ACKNOWLEDGEMENT QAManager を使用する場合は、あらゆる Web サービス要求の結果が自動的に受信確認されます。

SetRequestProperty メソッド

この WSBase が作成した Web サービス要求の要求プロパティを設定します。

Visual Basic の構文

```
Public Sub SetRequestProperty(
     ByVal name As String,
    ByVal value As Object
)
```

C# の構文

```
public void SetRequestProperty(string name, Object value)
```

パラメーター

- **name**　設定するプロパティ名。
- **value**　プロパティの値。

備考

要求プロパティは、この WSBase によって送信される QAMessage ごとに設定されます。この処理は、プロパティがクリアされるまで行われます。要求プロパティをクリアするには、NULL 値を設定します。メッセージプロパティのタイプは、value パラメーターのクラスで決まります。たとえば、値が Int32 のインスタンスである場合は、SetIntProperty を使用して QAMessage のプロパティが設定されます。

SetServiceID メソッド

WSBase のこのインスタンスのユーザー定義 ID を設定します。

Visual Basic の構文

```
Public Sub SetServiceID(ByVal serviceID As String)
```

C# の構文

```
public void SetServiceID(String serviceID)
```

パラメーター

- **serviceID**　サービス ID。

備考

サービス ID には、WSBase のこのインスタンスにユニークな値を設定します。サービス ID は、Web サービス要求を送受信するためのキュー名を形成する場合に内部で使用されます。このため、前のセッションで作成された Web サービス要求の結果を取得できるように、サービス ID をアプリケーションのセッション間で保持する必要があります。

WSException クラス

このクラスは、Web サービス要求の処理中に発生した例外を表します。

Visual Basic の構文

```
Public Class WSException Inherits System.Exception
```

C# の構文

```
public class WSException : System.Exception
```

基本クラス

- System.Exception

派生クラス

- WSFaultException クラス [QAnywhere .NET]336 ページ

メンバー

継承されたメンバーを含む WSException クラスのすべてのメンバー。

名前	説明
WSException コンストラクター	指定されたエラーメッセージを使用して、新しい例外を構築します。
GetBaseException method(System.Exception から継承)	派生クラスで上書きされる場合は、後続の 1 つ以上の例外の根本原因となる System.Exception を返します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
GetObjectData method(System.Exception から継承)	派生クラスで上書きされる場合は、System.Runtime.Serialization.SerializationInfo に例外の情報を設定します。
GetType method(System.Exception から継承)	現在のインスタンスのランタイムタイプを取得します。
ToString method(System.Exception から継承)	現在の例外を表す文字列を作成して返します。
Data property(System.Exception から継承)	例外に関するユーザー定義の追加情報を提供するキー／値のペアのコレクションを取得します。
ErrorCode プロパティ	この例外に関連付けられているエラーコード。
HelpLink property(System.Exception から継承)	この例外に関連付けられたヘルプファイルへのリンクを取得または設定します。
HResult property(System.Exception から継承)	特定の例外に割り当てられた、コーディングされた数値である HRESULT を取得または設定します。
InnerException property(System.Exception から継承)	現在の例外を発生させた System.Exception インスタンスを取得します。
Message property(System.Exception から継承)	現在の例外を説明するメッセージを取得します。
Source property(System.Exception から継承)	エラーを発生させたアプリケーションまたはオブジェクトの名前を取得または設定します。
StackTrace property(System.Exception から継承)	現在の例外がスローされた時点のコールスタックのフレームを表す文字列を取得します。
TargetSite property(System.Exception から継承)	現在の例外をスローしたメソッドを取得します。
WS_STATUS_HTTP_ERROR フィールド	Web サービスコネクターが作成した Web サービス HTTP 要求にエラーがあったことを示すエラーコードです。

名前	説明
WS_STATUS_HTTP_OK フィールド	Web サービスコネクターからの Web サービス HTTP 要求が正常に行われたことを示すエラーコードです。
WS_STATUS_HTTP_RETRIES_EXCEEDED フィールド	HTTP の再試行回数が Web サービスコネクターを超えたこを示すエラーコードです。
WS_STATUS_SOAP_PARSE_ERROR フィールド	SOAP 応答または SOAP 要求の解析中に、Web サービスランタイムまたは Web サービスコネクターでエラーがあったことを示すエラーコードです。

WSException コンストラクター

指定されたエラーメッセージを使用して、新しい例外を構築します。

オーバーロードリスト

名前	説明
WSException(Exception) コンストラクター	新しい例外を構築します。
WSException(string) コンストラクター	指定されたエラーメッセージを使用して、新しい例外を構築します。
WSException(string, int) コンストラクター	指定されたエラーメッセージとエラーコードを使用して、新しい例外を構築します。

WSException(Exception) コンストラクター

新しい例外を構築します。

Visual Basic の構文

```
Public Sub New(ByVal ex As Exception)
```

C# の構文

```
public WSException(Exception ex)
```

パラメーター

- **ex**　例外。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

WSException(string) コンストラクター

指定されたエラーメッセージを使用して、新しい例外を構築します。

Visual Basic の構文

```
Public Sub New(ByVal msg As String)
```

C# の構文

```
public WSException(string msg)
```

パラメーター

- **msg** エラーメッセージ。

WSException(string, int) コンストラクター

指定されたエラーメッセージとエラーコードを使用して、新しい例外を構築します。

Visual Basic の構文

```
Public Sub New(ByVal msg As String, ByVal errorCode As Integer)
```

C# の構文

```
public WSException(string msg, int errorCode)
```

パラメーター

- **msg** エラーメッセージ。
- **errorCode** エラーコード。

ErrorCode プロパティ

この例外に関連付けられているエラーコード。

Visual Basic の構文

```
Public Property ErrorCode As Integer
```

C# の構文

```
public int ErrorCode {get;set;}
```

WS_STATUS_HTTP_ERROR フィールド

Web サービスコネクターが作成した Web サービス HTTP 要求にエラーがあったことを示すエラーコードです。

Visual Basic の構文

```
Public Shared WS_STATUS_HTTP_ERROR As Integer
```

C# の構文

```
public static int WS_STATUS_HTTP_ERROR;
```

WS_STATUS_HTTP_OK フィールド

Web サービスコネクターからの Web サービス HTTP 要求が正常に行われたことを示すエラーコードです。

Visual Basic の構文

```
Public Shared WS_STATUS_HTTP_OK As Integer
```

C# の構文

```
public static int WS_STATUS_HTTP_OK;
```

WS_STATUS_HTTP_RETRIES_EXCEEDED フィールド

HTTP の再試行回数が Web サービスコネクターを超えたこを示すエラーコードです。

Visual Basic の構文

```
Public Shared WS_STATUS_HTTP_RETRIES_EXCEEDED As Integer
```

C# の構文

```
public static int WS_STATUS_HTTP_RETRIES_EXCEEDED;
```

WS_STATUS_SOAP_PARSE_ERROR フィールド

SOAP 応答または SOAP 要求の解析中に、Web サービスランタイムまたは Web サービスコネクターでエラーがあったことを示すエラーコードです。

Visual Basic の構文

```
Public Shared WS_STATUS_SOAP_PARSE_ERROR As Integer
```

C# の構文

```
public static int WS_STATUS_SOAP_PARSE_ERROR;
```

WSFaultException クラス

このクラスは、Web サービスコネクターからの SOAP Fault 例外を表します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

Visual Basic の構文

```
Public Class WSFaultException Inherits WSException
```

C# の構文

```
public class WSFaultException : WSException
```

基本クラス

- WSException クラス [QAnywhere .NET]332 ページ

メンバー

継承されたメンバーを含む WSFaultException クラスのすべてのメンバー。

名前	説明
WSException コンストラクター	指定されたエラーメッセージを使用して、新しい例外を構築します。
WSFaultException コンストラクター	指定されたエラーメッセージを使用して、新しい例外を構築します。
GetBaseException method(System.Exception から継承)	派生クラスで上書きされる場合は、後続の 1 つ以上の例外の根本原因となる System.Exception を返します。
GetObjectData method(System.Exception から継承)	派生クラスで上書きされる場合は、System.Runtime.Serialization.SerializationInfo に例外の情報を設定します。
GetType method(System.Exception から継承)	現在のインスタンスのランタイムタイプを取得します。
ToString method(System.Exception から継承)	現在の例外を表す文字列を作成して返します。
Data property(System.Exception から継承)	例外に関するユーザー定義の追加情報を提供するキー／値のペアのコレクションを取得します。
ErrorCode プロパティ	この例外に関連付けられているエラーコード。
HelpLink property(System.Exception から継承)	この例外に関連付けられたヘルプファイルへのリンクを取得または設定します。
HResult property(System.Exception から継承)	特定の例外に割り当てられたコード化された数値 HRESULT を取得または設定します。

名前	説明
InnerException property(System.Exception から継承)	現在の例外を発生させた System.Exception インスタンスを取得します。
Message property(System.Exception から継承)	現在の例外を説明するメッセージを取得します。
Source property(System.Exception から継承)	エラーを発生させたアプリケーションまたはオブジェクトの名前を取得または設定します。
StackTrace property(System.Exception から継承)	現在の例外がスローされた時点のコールスタックのフレームを表す文字列を取得します。
TargetSite property(System.Exception から継承)	現在の例外をスローしたメソッドを取得します。
WS_STATUS_HTTP_ERROR フィールド	Web サービスコネクターが作成した Web サービス HTTP 要求にエラーがあったことを示すエラーコードです。
WS_STATUS_HTTP_OK フィールド	Web サービスコネクターからの Web サービス HTTP 要求が正常に行われたことを示すエラーコードです。
WS_STATUS_HTTP_RETRIES_EXCEEDED フィールド	HTTP の再試行回数が Web サービスコネクターを超えたこを示すエラーコードです。
WS_STATUS_SOAP_PARSE_ERROR フィールド	SOAP 応答または SOAP 要求の解析中に、Web サービスランタイムまたは Web サービスコネクターでエラーがあったことを示すエラーコードです。

WSFaultException コンストラクター

指定されたエラーメッセージを使用して、新しい例外を構築します。

Visual Basic の構文

```
Public Sub New(ByVal msg As String)
```

C# の構文

```
public WSFaultException(string msg)
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- **msg**　エラーメッセージ。

WSListener インターフェイス

このクラスは、Web サービス要求の結果を受信するリスナーを表します。

Visual Basic の構文

```
Public Interface WSListener
```

C# の構文

```
public interface WSListener
```

メンバー

継承されたメンバーを含む WSListener クラスのすべてのメンバー。

名前	説明
OnException メソッド	非同期の Web サービス要求の結果を処理中に例外が発生した場合に呼び出されます。
OnResult メソッド	非同期の Web サービス要求の結果を指定して呼び出されます。

OnException メソッド

非同期の Web サービス要求の結果を処理中に例外が発生した場合に呼び出されます。

Visual Basic の構文

```
Public Sub OnException(
     ByVal e As WSException,
    ByVal wsResult As WSResult
)
```

C# の構文

```
public void OnException(WSException e, WSResult wsResult)
```

パラメーター

- **e**　結果の処理中に発生した WSException。
- **wsResult**　WSResult。これから要求 ID を取得できる場合があります。この WSResult の値は定義されていません。

OnResult メソッド

非同期の Web サービス要求の結果を指定して呼び出されます。

Visual Basic の構文

```
Public Sub OnResult(ByVal wsResult As WSResult)
```

C# の構文

```
public void OnResult(WSResult wsResult)
```

パラメーター

- **wsResult**　Web サービス要求の結果を記述する WSResult。

WSResult クラス

このクラスは、Web サービス要求の結果を表します。

Visual Basic の構文

```
Public Class WSResult
```

C# の構文

```
public class WSResult
```

メンバー

継承されたメンバーを含む WSResult クラスのすべてのメンバー。

名前	説明
Acknowledge メソッド	この WSResult が処理されたことを確認します。
GetArrayValue メソッド	この WSResult から複合型の値の配列を取得します。
GetBoolArrayValue メソッド	この WSResult から bool 値の配列を取得します。
GetBooleanArrayValue メソッド	この WSResult から Boolean 値の配列を取得します。
GetBooleanValue メソッド	この WSResult から Boolean 値を取得します。
GetBoolValue メソッド	この WSResult から bool 値を取得します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
GetByteArrayValue メソッド	この WSResult から byte 値の配列を取得します。
GetByteValue メソッド	この WSResult から byte 値を取得します。
GetCharArrayValue メソッド	この WSResult から char 値の配列を取得します。
GetCharValue メソッド	この WSResult から char 値を取得します。
GetDecimalArrayValue メソッド	この WSResult から decimal 値の配列を取得します。
GetDecimalValue メソッド	この WSResult から decimal 値を取得します。
GetDoubleArrayValue メソッド	この WSResult から double 値の配列を取得します。
GetDoubleValue メソッド	この WSResult から double 値を取得します。
GetErrorMessage メソッド	エラーメッセージを取得します。
GetFloatArrayValue メソッド	この WSResult から float 値の配列を取得します。
GetFloatValue メソッド	この WSResult から float 値を取得します。
GetInt16ArrayValue メソッド	この WSResult から Int16 値の配列を取得します。
GetInt16Value メソッド	この WSResult から Int16 値を取得します。
GetInt32ArrayValue メソッド	この WSResult から Int32 値の配列を取得します。
GetInt32Value メソッド	この WSResult から Int32 値を取得します。
GetInt64ArrayValue メソッド	この WSResult から Int64 値の配列を取得します。
GetInt64Value メソッド	この WSResult から Int64 値を取得します。
GetIntArrayValue メソッド	この WSResult から int 値の配列を取得します。
GetIntValue メソッド	この WSResult から int 値を取得します。

名前	説明
GetLongArrayValue メソッド	この WSResult から long 値の配列を取得します。
GetLongValue メソッド	この WSResult から long 値を取得します。
GetNullableBoolArrayValue メソッド	この WSResult から bool 値の配列を取得します。
GetNullableBoolValue メソッド	この WSResult から bool 値を取得します。
GetNullableDecimalArrayValue メソッド	この WSResult から NullableDecimal 値の配列を取得します。
GetNullableDecimalValue メソッド	この WSResult から NullableDecimal 値を取得します。
GetNullableDoubleArrayValue メソッド	この WSResult から double 値の配列を取得します。
GetNullableDoubleValue メソッド	この WSResult から double 値を取得します。
GetNullableFloatArrayValue メソッド	この WSResult から float 値の配列を取得します。
GetNullableFloatValue メソッド	この WSResult から float 値を取得します。
GetNullableIntArrayValue メソッド	この WSResult から int 値の配列を取得します。
GetNullableIntValue メソッド	この WSResult から int 値を取得します。
GetNullableLongArrayValue メソッド	この WSResult から long 値の配列を取得します。
GetNullableLongValue メソッド	この WSResult から Int64 値を取得します。
GetNullableSByteArrayValue メソッド	この WSResult から sbyte (signed byte) 値の配列を取得します。
GetNullableSByteValue メソッド	この WSResult から sbyte (signed byte) 値を取得します。
GetNullableShortArrayValue メソッド	この WSResult から short 値の配列を取得します。
GetNullableShortValue メソッド	この WSResult から short 値を取得します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
GetObjectArrayValue メソッド	この WSResult から object 値の配列を取得します。
GetObjectValue メソッド	この WSResult から object 値を取得します。
GetRequestID メソッド	この WSResult が表す要求 ID を取得します。
GetSByteArrayValue メソッド	この WSResult から sbyte (signed byte) 値の配列を取得します。
GetSByteValue メソッド	この WSResult から sbyte (signed byte) 値を取得します。
GetShortArrayValue メソッド	この WSResult から short 値の配列を取得します。
GetShortValue メソッド	この WSResult から short 値を取得します。
GetSingleArrayValue メソッド	この WSResult から single 値の配列を取得します。
GetSingleValue メソッド	この WSResult から single 値を取得します。
GetStatus メソッド	この WSResult のステータスを取得します。
GetStringArrayValue メソッド	この WSResult から string 値の配列を取得します。
GetStringValue メソッド	この WSResult から string 値を取得します。
GetUIntArrayValue メソッド	この WSResult から unsigned int 値の配列を取得します。
GetUIntValue メソッド	この WSResult から unsigned int 値を取得します。
GetULongArrayValue メソッド	この WSResult から unsigned long 値の配列を取得します。
GetULongValue メソッド	この WSResult から unsigned long 値を取得します。
GetUShortArrayValue メソッド	この WSResult から unsigned short 値の配列を取得します。
GetUShortValue メソッド	この WSResult から unsigned short 値を取得します。

名前	説明
GetValue メソッド	この WSResult から複合型の値を取得します。
SetLogger メソッド	デバッグのオン／オフを切り替えます。

備考

WSResult オブジェクトは、次のいずれかの方法で取得されます。

●WSListener.onResult に渡されます。

●コンパイラーが生成したサービスプロキシの asyncXYZ メソッドから返されます。

●特定の要求 ID を指定して WSBase.getResult を呼び出して取得されます。

Acknowledge メソッド

この WSResult が処理されたことを確認します。

Visual Basic の構文

```
Public Sub Acknowledge()
```

C# の構文

```
public void Acknowledge()
```

備考

このメソッドは、EXPLICIT_ACKNOWLEDGEMENT QAManager が使用されている場合にのみ有効です。

GetArrayValue メソッド

この WSResult から複合型の値の配列を取得します。

Visual Basic の構文

```
Public Function GetArrayValue(
     ByVal parentName As String
) As WSSerializable()
```

C# の構文

```
public WSSerializable[] GetArrayValue(string parentName)
```

パラメーター

● **parentName**　この値の WSDL ドキュメント内の要素名。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetBoolArrayValue メソッド

この WSResult から bool 値の配列を取得します。

Visual Basic の構文

```
Public Function GetBoolArrayValue(
     ByVal elementName As String
) As Boolean()
```

C# の構文

```
public bool[] GetBoolArrayValue(string elementName)
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetBooleanArrayValue メソッド

この WSResult から Boolean 値の配列を取得します。

Visual Basic の構文

```
Public Function GetBooleanArrayValue(
     ByVal elementName As String
) As System.Boolean()
```

C# の構文

```
public System.Boolean[] GetBooleanArrayValue(string elementName)
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetBooleanValue メソッド

この WSResult から Boolean 値を取得します。

Visual Basic の構文

```
Public Function GetBooleanValue(
     ByVal childName As String
) As System.Boolean
```

C# の構文

```
public System.Boolean GetBooleanValue(string childName)
```

パラメーター

- **childName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetBoolValue メソッド

この WSResult から bool 値を取得します。

Visual Basic の構文

```
Public Function GetBoolValue(ByVal childName As String) As Boolean
```

C# の構文

```
public bool GetBoolValue(string childName)
```

パラメーター

- **childName**　この値の WSDL ドキュメント内の要素名。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetByteArrayValue メソッド

この WSResult から byte 値の配列を取得します。

Visual Basic の構文

```
Public Function GetByteArrayValue(ByVal elementName As String) As Byte()
```

C# の構文

```
public byte[] GetByteArrayValue(string elementName)
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetByteValue メソッド

この WSResult から byte 値を取得します。

Visual Basic の構文

```
Public Function GetByteValue(ByVal childName As String) As Byte
```

C# の構文

```
public byte GetByteValue(string childName)
```

パラメーター

- **childName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetCharArrayValue メソッド

この WSResult から char 値の配列を取得します。

Visual Basic の構文

```
Public Function GetCharArrayValue(ByVal elementName As String) As Char()
```

C# の構文

```
public char[] GetCharArrayValue(string elementName)
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetCharValue メソッド

この WSResult から char 値を取得します。

Visual Basic の構文

```
Public Function GetCharValue(ByVal childName As String) As Char
```

C# の構文

```
public char GetCharValue(string childName)
```

パラメーター

- **childName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

GetDecimalArrayValue メソッド

この WSResult から decimal 値の配列を取得します。

Visual Basic の構文

```
Public Function GetDecimalArrayValue(
     ByVal elementName As String
) As Decimal()
```

C# の構文

```
public decimal[] GetDecimalArrayValue(string elementName)
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException** 値の取得で問題が発生した場合にスローされます。

GetDecimalValue メソッド

この WSResult から decimal 値を取得します。

Visual Basic の構文

```
Public Function GetDecimalValue(ByVal childName As String) As Decimal
```

C# の構文

```
public decimal GetDecimalValue(string childName)
```

パラメーター

- **childName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException** 値の取得で問題が発生した場合にスローされます。

GetDoubleArrayValue メソッド

この WSResult から double 値の配列を取得します。

Visual Basic 構文

```
Public Function GetDoubleArrayValue(
     ByVal elementName As String
) As Double()
```

C# の構文

```
public double[] GetDoubleArrayValue(string elementName)
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException** 値の取得で問題が発生した場合にスローされます。

GetDoubleValue メソッド

この WSResult から double 値を取得します。

Visual Basic の構文

```
Public Function GetDoubleValue(ByVal childName As String) As Double
```

C# の構文

```
public double GetDoubleValue(string childName)
```

パラメーター

- **childName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException** 値の取得で問題が発生した場合にスローされます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

GetErrorMessage メソッド

エラーメッセージを取得します。

Visual Basic の構文

```
Public Function GetErrorMessage() As String
```

C# の構文

```
public string GetErrorMessage()
```

戻り値

エラーメッセージ。

GetFloatArrayValue メソッド

この WSResult から float 値の配列を取得します。

Visual Basic 構文

```
Public Function GetFloatArrayValue(
     ByVal elementName As String
) As Single()
```

C# の構文

```
public float[] GetFloatArrayValue(string elementName)
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetFloatValue メソッド

この WSResult から float 値を取得します。

Visual Basic の構文

```
Public Function GetFloatValue(ByVal childName As String) As Single
```

C# の構文

```
public float GetFloatValue(string childName)
```

パラメーター

- **childName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetInt16ArrayValue メソッド

この WSResult から Int16 値の配列を取得します。

Visual Basic の構文

```
Public Function GetInt16ArrayValue(
     ByVal elementName As String
) As System.Int16()
```

C# の構文

```
public System.Int16[] GetInt16ArrayValue(string elementName)
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetInt16Value メソッド

この WSResult から Int16 値を取得します。

Visual Basic の構文

```
Public Function GetInt16Value(ByVal childName As String) As System.Int16
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

C# の構文

```
public System.Int16 GetInt16Value(string childName)
```

パラメーター

- **childName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetInt32ArrayValue メソッド

この WSResult から Int32 値の配列を取得します。

Visual Basic の構文

```
Public Function GetInt32ArrayValue(
     ByVal elementName As String
) As System.Int32()
```

C# の構文

```
public System.Int32[] GetInt32ArrayValue(string elementName)
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetInt32Value メソッド

この WSResult から Int32 値を取得します。

Visual Basic の構文

```
Public Function GetInt32Value(ByVal childName As String) As System.Int32
```

C# の構文

```
public System.Int32 GetInt32Value(string childName)
```

パラメーター

- **childName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetInt64ArrayValue メソッド

この WSResult から Int64 値の配列を取得します。

Visual Basic の構文

```
Public Function GetInt64ArrayValue(
     ByVal elementName As String
) As System.Int64()
```

C# の構文

```
public System.Int64[] GetInt64ArrayValue(string elementName)
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetInt64Value メソッド

この WSResult から Int64 値を取得します。

Visual Basic の構文

```
Public Function GetInt64Value(ByVal childName As String) As System.Int64
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

C# の構文

```
public System.Int64 GetInt64Value(string childName)
```

パラメーター

- **childName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetIntArrayValue メソッド

この WSResult から int 値の配列を取得します。

Visual Basic 構文

```
Public Function GetIntArrayValue(
     ByVal elementName As String
) As Integer()
```

C# の構文

```
public int[] GetIntArrayValue(string elementName)
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetIntValue メソッド

この WSResult から int 値を取得します。

Visual Basic の構文

```
Public Function GetIntValue(ByVal childName As String) As Integer
```

C# の構文

```
public int GetIntValue(string childName)
```

パラメーター

- **childName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetLongArrayValue メソッド

この WSResult から long 値の配列を取得します。

Visual Basic の構文

```
Public Function GetLongArrayValue(ByVal elementName As String) As Long()
```

C# の構文

```
public long[] GetLongArrayValue(string elementName)
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetLongValue メソッド

この WSResult から long 値を取得します。

Visual Basic の構文

```
Public Function GetLongValue(ByVal childName As String) As Long
```

C# の構文

```
public long GetLongValue(string childName)
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- **childName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetNullableBoolArrayValue メソッド

この WSResult から bool 値の配列を取得します。

Visual Basic 構文

```
Public Function GetNullableBoolArrayValue(
     ByVal elementName As String
) As NullableBool()
```

C# の構文

```
public NullableBool[] GetNullableBoolArrayValue(string elementName)
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetNullableBoolValue メソッド

この WSResult から bool 値を取得します。

Visual Basic の構文

```
Public Function GetNullableBoolValue(
     ByVal childName As String
) As NullableBool
```

C# の構文

```
public NullableBool GetNullableBoolValue(string childName)
```

パラメーター

- **childName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException** 値の取得で問題が発生した場合にスローされます。

GetNullableDecimalArrayValue メソッド

この WSResult から NullableDecimal 値の配列を取得します。

Visual Basic の構文

```
Public Function GetNullableDecimalArrayValue(
     ByVal elementName As String
) As NullableDecimal()
```

C# 構文

```
public NullableDecimal[] GetNullableDecimalArrayValue(
     string elementName
)
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException** 値の取得で問題が発生した場合にスローされます。

GetNullableDecimalValue メソッド

この WSResult から NullableDecimal 値を取得します。

Visual Basic の構文

```
Public Function GetNullableDecimalValue(
     ByVal childName As String
) As NullableDecimal
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

C# の構文

```
public NullableDecimal GetNullableDecimalValue(string childName)
```

パラメーター

- **childName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException** 値の取得で問題が発生した場合にスローされます。

GetNullableDoubleArrayValue メソッド

この WSResult から double 値の配列を取得します。

Visual Basic 構文

```
Public Function GetNullableDoubleArrayValue(
     ByVal elementName As String
) As NullableDouble()
```

C# の構文

```
public NullableDouble[] GetNullableDoubleArrayValue(string elementName)
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException** 値の取得で問題が発生した場合にスローされます。

GetNullableDoubleValue メソッド

この WSResult から double 値を取得します。

Visual Basic の構文

```
Public Function GetNullableDoubleValue(
     ByVal childName As String
) As NullableDouble
```

C# の構文

```
public NullableDouble GetNullableDoubleValue(string childName)
```

パラメーター

- **childName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException** 値の取得で問題が発生した場合にスローされます。

GetNullableFloatArrayValue メソッド

この WSResult から float 値の配列を取得します。

Visual Basic 構文

```
Public Function GetNullableFloatArrayValue(
     ByVal elementName As String
) As NullableFloat()
```

C# の構文

```
public NullableFloat[] GetNullableFloatArrayValue(string elementName)
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException** 値の取得で問題が発生した場合にスローされます。

GetNullableFloatValue メソッド

この WSResult から float 値を取得します。

Visual Basic の構文

```
Public Function GetNullableFloatValue(
     ByVal childName As String
) As NullableFloat
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

C# の構文

```
public NullableFloat GetNullableFloatValue(string childName)
```

パラメーター

- **childName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetNullableIntArrayValue メソッド

この WSResult から int 値の配列を取得します。

Visual Basic 構文

```
Public Function GetNullableIntArrayValue(
     ByVal elementName As String
) As NullableInt()
```

C# の構文

```
public NullableInt[] GetNullableIntArrayValue(string elementName)
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetNullableIntValue メソッド

この WSResult から int 値を取得します。

Visual Basic の構文

```
Public Function GetNullableIntValue(
     ByVal childName As String
) As NullableInt
```

C# の構文

```
public NullableInt GetNullableIntValue(string childName)
```

パラメーター

- **childName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetNullableLongArrayValue メソッド

この WSResult から long 値の配列を取得します。

Visual Basic の構文

```
Public Function GetNullableLongArrayValue(
     ByVal elementName As String
) As NullableLong()
```

C# の構文

```
public NullableLong[] GetNullableLongArrayValue(string elementName)
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetNullableLongValue メソッド

この WSResult から Int64 値を取得します。

Visual Basic の構文

```
Public Function GetNullableLongValue(
     ByVal childName As String
) As NullableLong
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

C# の構文

```
public NullableLong GetNullableLongValue(string childName)
```

パラメーター

- **childName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException** 値の取得で問題が発生した場合にスローされます。

GetNullableSByteArrayValue メソッド

この WSResult から sbyte (signed byte) 値の配列を取得します。

Visual Basic の構文

```
Public Function GetNullableSByteArrayValue(
     ByVal elementName As String
) As NullableSByte()
```

C# の構文

```
public NullableSByte[] GetNullableSByteArrayValue(string elementName)
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException** 値の取得で問題が発生した場合にスローされます。

GetNullableSByteValue メソッド

この WSResult から sbyte (signed byte) 値を取得します。

Visual Basic の構文

```
Public Function GetNullableSByteValue(
     ByVal childName As String
) As NullableSByte
```

C# の構文

```
public NullableSByte GetNullableSByteValue(string childName)
```

パラメーター

- **childName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException** 値の取得で問題が発生した場合にスローされます。

GetNullableShortArrayValue メソッド

この WSResult から short 値の配列を取得します。

Visual Basic 構文

```
Public Function GetNullableShortArrayValue(
     ByVal elementName As String
) As NullableShort()
```

C# の構文

```
public NullableShort[] GetNullableShortArrayValue(string elementName)
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException** 値の取得で問題が発生した場合にスローされます。

GetNullableShortValue メソッド

この WSResult から short 値を取得します。

Visual Basic の構文

```
Public Function GetNullableShortValue(
     ByVal childName As String
) As NullableShort
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

C# の構文

```
public NullableShort GetNullableShortValue(string childName)
```

パラメーター

- **childName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetObjectArrayValue メソッド

この WSResult から object 値の配列を取得します。

Visual Basic の構文

```
Public Function GetObjectArrayValue(
     ByVal elementName As String
) As System.Object()
```

C# の構文

```
public System.Object[] GetObjectArrayValue(string elementName)
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetObjectValue メソッド

この WSResult から object 値を取得します。

Visual Basic の構文

```
Public Function GetObjectValue(
     ByVal childName As String
) As System.Object
```

C# の構文

```
public System.Object GetObjectValue(string childName)
```

パラメーター

- **childName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetRequestID メソッド

この WSResult が表す要求 ID を取得します。

Visual Basic の構文

```
Public Function GetRequestID() As String
```

C# の構文

```
public string GetRequestID()
```

戻り値

要求 ID。

備考

要求の作成時とは異なるアプリケーションで実行される Web サービス要求に対応する WSResult を取得するために、この要求 ID が必要な場合は、アプリケーションの実行と実行の間でこの要求 ID を保持する必要があります。

GetSByteArrayValue メソッド

この WSResult から sbyte (signed byte) 値の配列を取得します。

Visual Basic の構文

```
Public Function GetSByteArrayValue(
     ByVal elementName As String
) As SByte()
```

C# の構文

```
public sbyte[] GetSByteArrayValue(string elementName)
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException** 値の取得で問題が発生した場合にスローされます。

GetSByteValue メソッド

この WSResult から sbyte (signed byte) 値を取得します。

Visual Basic の構文

```
Public Function GetSByteValue(ByVal childName As String) As SByte
```

C# の構文

```
public sbyte GetSByteValue(string childName)
```

パラメーター

- **childName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException** 値の取得で問題が発生した場合にスローされます。

GetShortArrayValue メソッド

この WSResult から short 値の配列を取得します。

Visual Basic 構文

```
Public Function GetShortArrayValue(
     ByVal elementName As String
) As Short()
```

C# の構文

```
public short[] GetShortArrayValue(string elementName)
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetShortValue メソッド

この WSResult から short 値を取得します。

Visual Basic の構文

```
Public Function GetShortValue(ByVal childName As String) As Short
```

C# の構文

```
public short GetShortValue(string childName)
```

パラメーター

- **childName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetSingleArrayValue メソッド

この WSResult から single 値の配列を取得します。

Visual Basic の構文

```
Public Function GetSingleArrayValue(
     ByVal elementName As String
) As System.Single()
```

C# の構文

```
public System.Single[] GetSingleArrayValue(string elementName)
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException** 値の取得で問題が発生した場合にスローされます。

GetSingleValue メソッド

この WSResult から single 値を取得します。

Visual Basic の構文

```
Public Function GetSingleValue(
     ByVal childName As String
) As System.Single
```

C# の構文

```
public System.Single GetSingleValue(string childName)
```

パラメーター

- **childName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException** 値の取得で問題が発生した場合にスローされます。

GetStatus メソッド

この WSResult のステータスを取得します。

Visual Basic の構文

```
Public Function GetStatus() As WSStatus
```

C# の構文

```
public WSStatus GetStatus()
```

戻り値

ステータスコード。

参照

- WSStatus 列挙体 [QAnywhere .NET]374 ページ

GetStringArrayValue メソッド

この WSResult から string 値の配列を取得します。

Visual Basic の構文

```
Public Function GetStringArrayValue(
     ByVal elementName As String
) As String()
```

C# の構文

```
public string[] GetStringArrayValue(string elementName)
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetStringValue メソッド

この WSResult から string 値を取得します。

Visual Basic の構文

```
Public Function GetStringValue(ByVal childName As String) As String
```

C# の構文

```
public string GetStringValue(string childName)
```

パラメーター

- **childName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetUIntArrayValue メソッド

この WSResult から unsigned int 値の配列を取得します。

Visual Basic の構文

```
Public Function GetUIntArrayValue(
     ByVal elementName As String
) As UInteger()
```

C# の構文

```
public uint[] GetUIntArrayValue(string elementName)
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetUIntValue メソッド

この WSResult から unsigned int 値を取得します。

Visual Basic の構文

```
Public Function GetUIntValue(ByVal childName As String) As UInteger
```

C# の構文

```
public uint GetUIntValue(string childName)
```

パラメーター

- **childName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetULongArrayValue メソッド

この WSResult から unsigned long 値の配列を取得します。

Visual Basic の構文

```
Public Function GetULongArrayValue(
     ByVal elementName As String
) As ULong()
```

C# の構文

```
public ulong[] GetULongArrayValue(string elementName)
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetULongValue メソッド

この WSResult から unsigned long 値を取得します。

Visual Basic の構文

```
Public Function GetULongValue(ByVal childName As String) As ULong
```

C# の構文

```
public ulong GetULongValue(string childName)
```

パラメーター

- **childName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetUShortArrayValue メソッド

この WSResult から unsigned short 値の配列を取得します。

Visual Basic の構文

```
Public Function GetUShortArrayValue(
     ByVal elementName As String
) As UShort()
```

C# の構文

```
public ushort[] GetUShortArrayValue(string elementName)
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetUShortValue メソッド

この WSResult から unsigned short 値を取得します。

Visual Basic の構文

```
Public Function GetUShortValue(ByVal childName As String) As UShort
```

C# の構文

```
public ushort GetUShortValue(string childName)
```

パラメーター

- **childName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

GetValue メソッド

この WSResult から複合型の値を取得します。

Visual Basic の構文

```
Public Function GetValue(ByVal childName As String) As Object
```

C# の構文

```
public object GetValue(string childName)
```

パラメーター

- **childName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **iAnywhere.QAnywhere.WS.WSException**　値の取得で問題が発生した場合にスローされます。

SetLogger メソッド

デバッグのオン／オフを切り替えます。

Visual Basic の構文

```
Public Sub SetLogger(ByVal wsLogger As WSLogger)
```

C# の構文

```
public void SetLogger(WSLogger wsLogger)
```

WSStatus 列挙体

このクラスは、Web サービス要求のステータスコードを定義します。

Visual Basic の構文

```
Public Enum WSStatus
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

C# の構文

```
public enum WSStatus
```

メンバー

メンバー名	説明
STATUS_SUCCESS	要求は正常に処理されました。
STATUS_ERROR	要求の処理でエラーがありました。
STATUS_QUEUED	要求はサーバーへの配信用キューに登録されました。 要求の最終ステータスはまだ不明です。
STATUS_RESULT_AVAILABLE	要求の結果を取得できます。

クライアント用 QAnywhere C++ API リファレンス

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

ヘッダーファイル

```
qa.hpp
```

AcknowledgementMode クラス

QAnywhere クライアントアプリケーションによってどのようにメッセージが確認されるかを示します。

構文

```
public class AcknowledgementMode
```

メンバー

継承されたメンバーを含む AcknowledgementMode クラスのすべてのメンバー。

名前	説明
EXPLICIT_ACKNOWLEDGEMENT 変数	受信メッセージは、QAManager のいずれかの受信確認メソッドを使用して確認されます。
IMPLICIT_ACKNOWLEDGEMENT 変数	すべてのメッセージは、クライアントアプリケーションで受信されるとすぐに受信確認されます。

名前	説明
TRANSACTIONAL 変数	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。

備考

IMPLICIT_ACKNOWLEDGEMENT モードと EXPLICIT_ACKNOWLEDGEMENT モードは、QAManager::open() メソッドを使用して、QAManager インスタンスに割り当てられます。TRANSACTIONAL モードは、QATransactionalManager インスタンスに暗黙的に割り当てられます。

詳細については、「QAnywhere API の初期化」54 ページを参照してください。

暗黙的な受信確認モードでは、メッセージは、クライアントアプリケーションで受信されるとすぐに受信確認されます。明示的な受信確認モードでは、QAManager のいずれかの受信確認メソッドを呼び出してください。トランザクション指向モードでは、QATransactionalManager::commit() メソッドを呼び出して、未確認のすべてのメッセージの受信を確認します。すべてのステータス変更は、サーバーによってクライアント間で伝達されます。

詳細については、「同期的なメッセージ受信」71 ページと「非同期的なメッセージ受信」73 ページを参照してください。

トランザクション指向メッセージングの場合は、QATransactionalManager を使用します。この場合は、QATransactionalManager::commit メソッドを使用して、トランザクションに属するメッセージの受信を確認します。

QAManagerBase インスタンスのモードは、QAManagerBase::Mode プロパティを使用して判別できます。

参照

- QAManager クラス [QAnywhere C++]411 ページ
- QATransactionalManager クラス [QAnywhere C++]484 ページ
- QAManagerBase クラス [QAnywhere C++]418 ページ

EXPLICIT_ACKNOWLEDGEMENT 変数

受信メッセージは、QAManager のいずれかの受信確認メソッドを使用して確認されます。

構文

```
public static const qa_short EXPLICIT_ACKNOWLEDGEMENT;
```

IMPLICIT_ACKNOWLEDGEMENT 変数

すべてのメッセージは、クライアントアプリケーションで受信されるとすぐに受信確認されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
public static const qa_short IMPLICIT_ACKNOWLEDGEMENT;
```

備考

メッセージを同期的に受信する場合、メッセージは QAManagerBase::getMessage メソッドが返されるとすぐに受信確認されます。メッセージを非同期的に受信する場合、メッセージはイベント処理関数が返されるとすぐに受信確認されます。

TRANSACTIONAL 変数

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。

構文

```
public static const qa_short TRANSACTIONAL;
```

備考

このモードは、QATransactionalManager インスタンスに自動的に割り当てられます。

MessageProperties クラス

標準のメッセージプロパティ名を格納するフィールドを提供します。

構文

```
public class MessageProperties
```

メンバー

継承されたメンバーを含む MessageProperties クラスのすべてのメンバー。

名前	説明
ADAPTER 変数	このプロパティ名は、QAnywhere サーバーへの接続で使用されている、現在アクティブなネットワークアダプターを示します。
ADAPTERS 変数	このプロパティ名は、QAnywhere サーバーへの接続で使用できるネットワークアダプターのデリミター付きリストを示します。
DELIVERY_COUNT 変数	このプロパティ名は、メッセージを配信するために、これまでに実行された試行回数を示します。

名前	説明
IP 変数	このプロパティ名は、QAnywhere サーバーへの接続で使用されている、現在アクティブなネットワークアダプターの IP アドレスを示します。
MAC 変数	このプロパティ名は、QAnywhere サーバーへの接続で使用されている、現在アクティブなネットワークアダプターの MAC アドレスを示します。
MSG_TYPE 変数	このプロパティ名は、QAnywhere メッセージに関連付けられている MessageType 列挙値を示します。
NETWORK_STATUS 変数	このプロパティ名は、ネットワーク接続のステータスを示します。
ORIGINATOR 変数	このプロパティ名は、メッセージの発信者のメッセージストア ID を示します。
RAS 変数	このプロパティ名は、QAnywhere サーバーへの接続で使用されている、現在アクティブな RAS 名を示します。
RASNAMES 変数	このプロパティ名は、QAnywhere サーバーへの接続で使用できる RAS エントリ名のデリミター付きリストを示します。
STATUS 変数	このプロパティ名は、メッセージの現在のステータスを示します。
STATUS_TIME 変数	このプロパティ名は、メッセージが現在のステータスを受信した時刻を示します。
TRANSMISSION_STATUS 変数	このプロパティ名は、メッセージの現在の転送ステータスを示します。

備考

MessageProperties クラスは、標準のメッセージプロパティ名を提供します。MessageProperties フィールドは、メッセージプロパティの取得と設定で使用する QAMessage メソッドに渡すことができます。

詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- QAMessage クラス [QAnywhere C++]454 ページ

例

たとえば、次の QATextMessage インスタンスがあるとします。

```
QATextMessage * t_msg;
```

次の例では、QAMessage::getIntProperty メソッドを使用して MessageProperties::MSG_TYPE に対応する値を取得します。MessageType 列挙体は、int 値の結果を適切なメッセージタイプにマッピングします。

```
int msg_type;
t_msg->getIntProperty( MessageProperties::MSG_TYPE, &msg_type)
```

次の例では、MessageProperties::MSG_TYPE と MessageProperties::RASNAMES を使用して、それぞれメッセージタイプと RAS 名を評価します。

```
void SystemQueueListener::onMessage(QAMessage * msg) {
      QATextMessage *       t_msg;
      TCHAR                 buffer[512];
      int                   len;
      int                   msg_type;

      t_msg = msg->castToTextMessage();
      if( t_msg != NULL ) {
         t_msg->getIntProperty( MessageProperties::MSG_TYPE, &msg_type );

       if( msg_type == MessageType::NETWORK_STATUS_NOTIFICATION ) {
          // get RAS names using MessageProperties::RASNAMES
          len = t_msg->getStringProperty(MessageProperties::RASNAMES,buffer,sizeof(buffer));
       }
       //...
    }
}
```

ADAPTER 変数

このプロパティ名は、QAnywhere サーバーへの接続で使用されている、現在アクティブなネットワークアダプターを示します。

構文

```
public static const qa_string ADAPTER;
```

備考

これは、システムキューメッセージで使用されます。

このフィールドの値は "ias_Network.Adapter" です。

MessageProperties::ADAPTER は、最初のパラメーターとして QAMessage::getStringProperty メソッドに渡し、関連付けられているメッセージプロパティにアクセスします。

詳細については、「メッセージプロパティ」698 ページを参照してください。

参照

- QAMessage.getStringProperty メソッド [QAnywhere C++]467 ページ

ADAPTERS 変数

このプロパティ名は、QAnywhere サーバーへの接続で使用できるネットワークアダプターのデリミター付きリストを示します。

構文

```
public static const qa_string ADAPTERS;
```

備考

これは、システムキューメッセージで使用されます。

このフィールドの値は "ias_Adapters" です。

MessageProperties::ADAPTERS は、最初のパラメーターとして QAMessage::getStringProperty メソッドに渡し、関連付けられているメッセージプロパティにアクセスします。

詳細については、「メッセージプロパティ」698 ページを参照してください。

参照

- QAMessage.getStringProperty メソッド [QAnywhere C++]467 ページ

DELIVERY_COUNT 変数

このプロパティ名は、メッセージを配信するために、これまでに実行された試行回数を示します。

構文

```
public static const qa_string DELIVERY_COUNT;
```

備考

このフィールドの値は "ias_DeliveryCount" です。

MessageProperties::DELIVERY_COUNT は、最初のパラメーターとして QAMessage::setStringProperty メソッドまたは QAMessage::getStringProperty メソッドに渡し、関連付けられているメッセージプロパティにアクセスします。

参照

- QAMessage.setStringProperty メソッド [QAnywhere C++]476 ページ
- QAMessage.getStringProperty メソッド [QAnywhere C++]467 ページ

IP 変数

このプロパティ名は、QAnywhere サーバーへの接続で使用されている、現在アクティブなネットワークアダプターの IP アドレスを示します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
public static const qa_string IP;
```

備考

これは、システムキューメッセージで使用されます。

このフィールドの値は "ias_Network.IP" です。

MessageProperties::IP は、最初のパラメーターとして QAMessage::getStringProperty メソッドに渡し、関連付けられているメッセージプロパティにアクセスします。

詳細については、「メッセージプロパティ」698 ページを参照してください。

参照

- QAMessage.getStringProperty メソッド [QAnywhere C++]467 ページ

MAC 変数

このプロパティ名は、QAnywhere サーバーへの接続で使用されている、現在アクティブなネットワークアダプターの MAC アドレスを示します。

構文

```
public static const qa_string MAC;
```

備考

これは、システムキューメッセージで使用されます。

このフィールドの値は "ias_Network.MAC" です。

MessageProperties::MAC は、最初のパラメーターとして QAMessage::getStringProperty メソッドに渡し、関連付けられているメッセージプロパティにアクセスします。

詳細については、「メッセージプロパティ」698 ページを参照してください。

参照

- QAMessage.getStringProperty メソッド [QAnywhere C++]467 ページ

MSG_TYPE 変数

このプロパティ名は、QAnywhere メッセージに関連付けられている MessageType 列挙値を示します。

構文

```
public static const qa_string MSG_TYPE;
```

備考

このフィールドの値は "ias_MessageType" です。MessageProperties::MSG_TYPE は、最初のパラメーターとして QAMessage::setIntProperty メソッドまたは QAMessage::getIntProperty メソッドに渡し、関連付けられているプロパティを判断します。

参照

- MessageType クラス [QAnywhere C++]386 ページ
- QAMessage.setIntProperty メソッド [QAnywhere C++]473 ページ
- QAMessage.getIntProperty メソッド [QAnywhere C++]463 ページ

NETWORK_STATUS 変数

このプロパティ名は、ネットワーク接続のステータスを示します。

構文

```
public static const qa_string NETWORK_STATUS;
```

備考

このフィールドの値は "ias_NetworkStatus" です。

ネットワークが接続可能な場合、このプロパティの値は 1 です。それ以外の場合は 0 です。ネットワークステータスは、システムキューメッセージ (ネットワークステータスの変更など) で使用されます。

詳細については、「メッセージプロパティ」698 ページを参照してください。

MessageProperties::NETWORK_STATUS は、最初のパラメーターとして QAMessage::setStringProperty メソッドまたは QAMessage::getStringProperty メソッドに渡し、関連付けられているメッセージプロパティにアクセスします。

参照

- QAMessage.setStringProperty メソッド [QAnywhere C++]476 ページ
- QAMessage.getStringProperty メソッド [QAnywhere C++]467 ページ

ORIGINATOR 変数

このプロパティ名は、メッセージの発信者のメッセージストア ID を示します。

構文

```
public static const qa_string ORIGINATOR;
```

備考

このフィールドの値は "ias_Originator" です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

MessageProperties::ORIGINATOR は、最初のパラメーターとして QAMessage::setStringProperty メソッドまたは QAMessage::getStringProperty メソッドに渡し、関連付けられているメッセージプロパティにアクセスします。

参照

- QAMessage.setStringProperty メソッド [QAnywhere C++]476 ページ
- QAMessage.getStringProperty メソッド [QAnywhere C++]467 ページ

RAS 変数

このプロパティ名は、QAnywhere サーバーへの接続で使用されている、現在アクティブな RAS 名を示します。

構文

```
public static const qa_string RAS;
```

備考

これは、システムキューメッセージで使用されます。

このフィールドの値は "ias_Network.RAS" です。

MessageProperties::RAS は、最初のパラメーターとして QAMessage::getStringProperty メソッドに渡し、関連付けられているメッセージプロパティにアクセスします。

詳細については、「メッセージプロパティ」698 ページを参照してください。

参照

- QAMessage.getStringProperty メソッド [QAnywhere C++]467 ページ

RASNAMES 変数

このプロパティ名は、QAnywhere サーバーへの接続で使用できる RAS エントリ名のデリミター付きリストを示します。

構文

```
public static const qa_string RASNAMES;
```

備考

これは、システムキューメッセージで使用されます。

このフィールドの値は "ias_RASNames" です。

詳細については、「メッセージプロパティ」698 ページを参照してください。

MessageProperties::RASNAMES は、最初のパラメーターとして QAMessage::setStringProperty メソッドまたは QAMessage::getStringProperty メソッドに渡し、関連付けられているメッセージプロパティにアクセスします。

参照

- QAMessage.setStringProperty メソッド [QAnywhere C++]476 ページ
- QAMessage.getStringProperty メソッド [QAnywhere C++]467 ページ
- QAMessage.setIntProperty メソッド [QAnywhere C++]473 ページ
- QAMessage.getIntProperty メソッド [QAnywhere C++]463 ページ

STATUS 変数

このプロパティ名は、メッセージの現在のステータスを示します。

構文

```
public static const qa_string STATUS;
```

備考

値のリストについては、StatusCodes を参照してください。このフィールドの値は "ias_Status" です。

MessageProperties::STATUS は、最初のパラメーターとして QAMessage::setIntProperty メソッドまたは QAMessage::getIntProperty メソッドに渡し、関連付けられているメッセージプロパティにアクセスします。

参照

- StatusCodes クラス [QAnywhere C++]491 ページ
- QAMessage.setIntProperty メソッド [QAnywhere C++]473 ページ
- QAMessage.getIntProperty メソッド [QAnywhere C++]463 ページ

STATUS_TIME 変数

このプロパティ名は、メッセージが現在のステータスを受信した時刻を示します。

構文

```
public static const qa_string STATUS_TIME;
```

備考

タイムスタンプには、プラットフォームに合わせたフォーマットが使用されます。Windows/PocketPC プラットフォームのタイムスタンプは SYSTEMTIME フォーマットです。これが FILETIME フォーマットに変換されて、qa_long 値にコピーされます。これはローカル時間です。このフィールドの値は "ias_StatusTime" です。

MessageProperties::STATUS_TIME は、最初のパラメーターとして QAMessage::getLongProperty メソッドに渡し、関連付けられている読み込み専用のメッセージプロパティにアクセスします。

参照

- QAMessage.getLongProperty メソッド [QAnywhere C++]463 ページ

TRANSMISSION_STATUS 変数

このプロパティ名は、メッセージの現在の転送ステータスを示します。

構文

```
public static const qa_string TRANSMISSION_STATUS;
```

備考

値のリストについては、StatusCodes を参照してください。

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

MessageProperties::TRANSMISSION_STATUS は、最初のパラメーターとして QAMessage::setIntProperty メソッドまたは QAMessage::getIntProperty メソッドに渡し、関連付けられているメッセージプロパティにアクセスします。

参照

- StatusCodes クラス [QAnywhere C++]491 ページ
- QAMessage.setIntProperty メソッド [QAnywhere C++]473 ページ
- QAMessage.getIntProperty メソッド [QAnywhere C++]463 ページ

MessageStoreProperties クラス

MessageStoreProperties クラスは、標準のメッセージプロパティ名を提供します。

構文

```
public class MessageStoreProperties
```

メンバー

継承されたメンバーを含む MessageStoreProperties クラスのすべてのメンバー。

名前	説明
MAX_DELIVERY_ATTEMPTS 変数	このプロパティ名は、メッセージのステータスが StatusCodes::UNRECEIVABLE に設定されるまでに、明示的な受信確認がないままメッセージを受信できる最大回数を示します。

備考

MessageStoreProperties フィールドは、事前定義済みまたはカスタムのメッセージストアプロパティの取得と設定で使用する QAManagerBase メソッドに渡すことができます。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

MAX_DELIVERY_ATTEMPTS 変数

このプロパティ名は、メッセージのステータスが StatusCodes::UNRECEIVABLE に設定されるまでに、明示的な受信確認がないままメッセージを受信できる最大回数を示します。

構文

```
public static const qa_string MAX_DELIVERY_ATTEMPTS;
```

備考

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

参照

- StatusCodes クラス [QAnywhere C++]491 ページ

MessageType クラス

MessageProperties::MSG_TYPE メッセージプロパティの定数値を定義します。

構文

```
public class MessageType
```

メンバー

継承されたメンバーを含む MessageType クラスのすべてのメンバー。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
NETWORK_STATUS_NOTIFICATION 変数	QAnywhere クライアントアプリケーションにネットワークステータスの変更を通知する場合に使用する、QAnywhere システムメッセージを特定します。
PUSH_NOTIFICATION 変数	QAnywhere クライアントアプリケーションに Push 通知を通知する場合に使用する、QAnywhere システムメッセージを特定します。
REGULAR 変数	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。

備考

次の例は、QAnywhere システムメッセージの処理で使用される onSystemMessage メソッドを示します。

メッセージタイプは、MessageType.NETWORK_STATUS_NOTIFICATION と比較されます。

```
void SystemQueueListener::onMessage(QAMessage * msg) {
   QATextMessage *       t_msg;
   TCHAR                 buffer[512];
   int                   len;
   int                   msg_type;

   t_msg = msg->castToTextMessage();
   if( t_msg != NULL ) {
      t_msg->getIntProperty( MessageProperties::MSG_TYPE, &msg_type );

         if( msg_type == MessageType::NETWORK_STATUS_NOTIFICATION ) {
            // get network names using MessageProperties::NETWORK
          len = t_msg->getStringProperty(MessageProperties::NETWORK,buffer,sizeof(buffer));
      }
      //...
   }
}
```

NETWORK_STATUS_NOTIFICATION 変数

QAnywhere クライアントアプリケーションにネットワークステータスの変更を通知する場合に使用する、QAnywhere システムメッセージを特定します。

構文

```
public static const qa_int NETWORK_STATUS_NOTIFICATION;
```

備考

ネットワークステータスの変更は、システムメッセージを受信するデバイスに適用されます。新しいネットワークステータス情報を特定するには、MessageProperties::ADAPTER、MessageProperties::NETWORK、MessageProperties::NETWORK_STATUS の各フィールドを使用します。

詳細については、「事前に定義されたメッセージプロパティ」698 ページを参照してください。

PUSH_NOTIFICATION 変数

QAnywhere クライアントアプリケーションに Push 通知を通知する場合に使用する、QAnywhere システムメッセージを特定します。

構文

```
public static const qa_int PUSH_NOTIFICATION;
```

備考

オンデマンドの QAnywhere Agent ポリシーを使用した場合の一般的な応答は、QAManagerBase::triggerSendReceive() メソッドを呼び出し、中央のメッセージサーバーで待機しているメッセージを受信することです。

詳細については、「事前に定義されたメッセージプロパティ」698 ページを参照してください。

REGULAR 変数

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。

構文

```
public static const qa_int REGULAR;
```

備考

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

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。

構文

```
public class QABinaryMessage : QAMessage
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

基本クラス

- QAMessage クラス [QAnywhere C++]454 ページ

メンバー

継承されたメンバーを含む QABinaryMessage クラスのすべてのメンバー。

名前	説明
QABinaryMessage デコンストラクター	仮想デストラクターです。
beginEnumPropertyNames メソッド	メッセージプロパティ名の列挙を開始します。
castToBinaryMessage メソッド	QAMessage を QABinaryMessage にキャストします。
castToTextMessage メソッド	QAMessage を QATextMessage にキャストします。
clearProperties メソッド	メッセージのプロパティをクリアします。
endEnumPropertyNames メソッド	メッセージプロパティ名の列挙に関連付けられているリソースを解放します。
getAddress メソッド	QAMessage インスタンスの送信先アドレスを取得します。
getBodyLength メソッド	メッセージ本文のサイズをバイト単位で返します。
getBooleanProperty メソッド	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。
getByteProperty メソッド	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。
getDoubleProperty メソッド	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。
getExpiration メソッド	メッセージの有効期限を取得します。
getFloatProperty メソッド	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。
getInReplyToID メソッド	このメッセージが返信対象として指定されているメッセージの ID を取得します。

名前	説明
getIntProperty メソッド	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。
getLongProperty メソッド	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。
getMessageID メソッド	メッセージ ID を取得します。
getPriority メソッド	メッセージの優先度を取得します。
getPropertyType メソッド	指定された名前を持つプロパティの型を返します。
getRedelivered メソッド	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。
getReplyToAddress メソッド	メッセージの返信先アドレスを取得します。
getShortProperty メソッド	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。
getStringProperty メソッド	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。
getTimestamp メソッド	メッセージのタイムスタンプを取得します。
getTimestampAsString メソッド	メッセージのタイムスタンプをフォーマットされた string として取得します。
nextPropertyName メソッド	指定された列挙のメッセージプロパティ名を返します。該当するプロパティ名がない場合は -1 を返します。
propertyExists メソッド	プロパティ値が存在するかどうかを示します。
readBinary メソッド	QABinaryMessage インスタンスのメッセージ本文未読部分の先頭から、指定されたバイト数を読み込みます。
readBoolean メソッド	QABinaryMessage インスタンスのメッセージ本文未読部分の先頭から、boolean 値を読み込みます。
readByte メソッド	QABinaryMessage インスタンスのメッセージ本文未読部分の先頭から、符号付き 8 ビット値を読み込みます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
readChar メソッド	QABinaryMessage インスタンスのメッセージ本文未読部分の先頭から、char 値を読み込みます。
readDouble メソッド	QABinaryMessage インスタンスのメッセージ本文未読部分の先頭から、double 値を読み込みます。
readFloat メソッド	QABinaryMessage インスタンスのメッセージ本文未読部分の先頭から、float 値を読み込みます。
readInt メソッド	QABinaryMessage インスタンスのメッセージ本文未読部分の先頭から、符号付き 32 ビット int 値を読み込みます。
readLong メソッド	QABinaryMessage インスタンスのメッセージ本文未読部分の先頭から、符号付き 64 ビット int 値を読み込みます。
readShort メソッド	QABinaryMessage インスタンスのメッセージ本文未読部分の先頭から、符号付き 16 ビット値を読み込みます。
readString メソッド	QABinaryMessage インスタンスのメッセージ本文未読部分の先頭から、string 値を読み込みます。
reset メソッド	メッセージをリセットして、メッセージ本文の先頭から値の読み込みを開始できるようにします。
setAddress メソッド	メッセージの送信先アドレスを設定します。
setBooleanProperty メソッド	指定された名前を持つ qa_bool 型プロパティを指定された値に設定します。
setByteProperty メソッド	指定された名前を持つ qa_byte 型プロパティを指定された値に設定します。
setDoubleProperty メソッド	指定された名前を持つ qa_double 型プロパティを指定された値に設定します。
setFloatProperty メソッド	指定された名前を持つ qa_float 型プロパティを指定された値に設定します。

名前	説明
setInReplyToID メソッド	メッセージの In-Reply-To ID を設定します。
setIntProperty メソッド	指定された名前を持つ qa_int 型プロパティを指定された値に設定します。
setLongProperty メソッド	指定された名前を持つ qa_long 型プロパティを指定された値に設定します。
setMessageID メソッド	メッセージ ID を設定します。
setPriority メソッド	メッセージの優先度レベルを設定します。
setRedelivered メソッド	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。
setReplyToAddress メソッド	メッセージの返信先アドレスを設定します。
setShortProperty メソッド	指定された名前を持つ qa_short 型プロパティを指定された値に設定します。
setStringProperty メソッド	指定された名前を持つ qa_string 型プロパティを指定された値に設定します。
setTimestamp メソッド	メッセージのタイムスタンプを設定します。
writeBinary メソッド	QABinaryMessage インスタンスのメッセージ本文に byte 配列値を追加します。
writeBoolean メソッド	QABinaryMessage インスタンスのメッセージ本文に boolean 値を追加します。
writeByte メソッド	QABinaryMessage インスタンスのメッセージ本文に byte 値を追加します。
writeChar メソッド	QABinaryMessage インスタンスのメッセージ本文に char 値を追加します。
writeDouble メソッド	QABinaryMessage インスタンスのメッセージ本文に double 値を追加します。
writeFloat メソッド	QABinaryMessage インスタンスのメッセージ本文に float 値を追加します。
writeInt メソッド	QABinaryMessage インスタンスのメッセージ本文に int 値を追加します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
writeLong メソッド	QABinaryMessage インスタンスのメッセージ本文に long 値を追加します。
writeShort メソッド	QABinaryMessage インスタンスのメッセージ本文に short 値を追加します。
writeString メソッド	QABinaryMessage インスタンスのメッセージ本文に string 値を追加します。
DEFAULT_PRIORITY 変数	デフォルトのメッセージ優先度です。
DEFAULT_TIME_TO_LIVE 変数	デフォルトのメッセージの存続時間です。

備考

これは QAMessage クラスを継承したもので、メッセージ本文にバイトストリームが追加されます。QABinaryMessage には、メッセージ本文からのバイトストリームの読み込み／書き込みを行うためのさまざまなメソッドがあります。

メッセージが最初に作成された時点では、メッセージ本文は書き込み専用になっています。メッセージ送信元のクライアントは、メッセージを送信した後、そのメッセージを保持し変更できます。ただし、それによって、送信されたメッセージが変更されることはありません。同じメッセージオブジェクトを複数回送信できます。

メッセージが受信された時点でプロバイダーは QABinaryMessage::reset メソッドを呼び出します。これによりメッセージ本文が読み込み専用モードになり、メッセージ本文の先頭から値の読み込みが開始されます。クライアントが読み込み専用モードのメッセージに書き込もうとすると、COMMON_MSG_NOT_WRITEABLE_ERROR が設定されます。

例

次の例では QABinaryMessage::writeString メソッドを使用して、QABinaryMessage インスタンスのメッセージ本文に文字列 "Q" と "Anywhere" を書き込みます。

```
// Create a binary message instance.
QABinaryMessage * binary_message;
binary_message = qa_manager->createBinaryMessage();

// Set optional message properties.
binary_message->setReplyToAddress( "my-queue-name" );

// Write to the message body.
binary_message->writeString("Q");
binary_message->writeString("Anywhere");

// Put the message in the local database, ready for sending.

if( !qa_manager->putMessage( "store-id¥¥queue-name", msg ) ) {
    handleError();
}
```

受信が終了すると、最初の QABinaryMessage::readString() の呼び出しから "Q" が返され、2 番目の QABinaryMessage::readString() の呼び出しから "Anywhere" が返されます。

メッセージは QAnywhere Agent から送信されます。

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

QABinaryMessage デコンストラクター

仮想デストラクターです。

構文

```
public virtual ~QABinaryMessage()
```

getBodyLength メソッド

メッセージ本文のサイズをバイト単位で返します。

構文

```
public virtual qa_long getBodyLength()
```

戻り値

メッセージ本文のサイズ (バイト単位)。

readBinary メソッド

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

構文

```
public virtual qa_int readBinary(qa_bytes value, qa_int length)
```

パラメーター

- **value** データの読み込み先バッファー。
- **length** 読み込むバイト数の最大値。

戻り値

バッファー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。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- QABinaryMessage.writeBinary メソッド [QAnywhere C++]399 ページ

readBoolean メソッド

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

構文

```
public virtual qa_bool readBoolean(qa_bool * value)
```

パラメーター

- **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。

戻り値

処理が正常終了した場合のみ true。

参照

- QABinaryMessage.writeBoolean メソッド [QAnywhere C++]399 ページ

readByte メソッド

QABinaryMessage インスタンスのメッセージ本文未読部分の先頭から、符号付き 8 ビット値を読み込みます。

構文

```
public virtual qa_bool readByte(qa_byte * value)
```

パラメーター

- **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。

戻り値

処理が正常終了した場合のみ true。

参照

- QABinaryMessage.writeByte メソッド [QAnywhere C++]399 ページ

readChar メソッド

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

構文

```
public virtual qa_bool readChar(qa_char * value)
```

パラメーター

- **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。

戻り値

読み込まれた char 値。

参照

- QABinaryMessage.writeChar メソッド [QAnywhere C++]400 ページ

readDouble メソッド

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

構文

```
public virtual qa_bool readDouble(qa_double * value)
```

パラメーター

- **value** バイトメッセージストリームから読み込んだ double 値の格納先。

戻り値

処理が正常終了した場合のみ true。

参照

- QABinaryMessage.writeDouble メソッド [QAnywhere C++]400 ページ

readFloat メソッド

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

構文

```
public virtual qa_bool readFloat(qa_float * value)
```

パラメーター

- **value** バイトメッセージストリームから読み込んだ float 値の格納先。

戻り値

処理が正常終了した場合のみ true。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- QABinaryMessage.writeFloat メソッド [QAnywhere C++]401 ページ

readInt メソッド

QABinaryMessage インスタンスのメッセージ本文未読部分の先頭から、符号付き 32 ビット int 値を読み込みます。

構文

```
public virtual qa_bool readInt(qa_int * value)
```

パラメーター

- **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。

戻り値

処理が正常終了した場合のみ true。

参照

- QABinaryMessage.writeInt メソッド [QAnywhere C++]401 ページ

readLong メソッド

QABinaryMessage インスタンスのメッセージ本文未読部分の先頭から、符号付き 64 ビット int 値を読み込みます。

構文

```
public virtual qa_bool readLong(qa_long * value)
```

パラメーター

- **value**　バイトメッセージストリームから読み込んだ long 値の格納先。

戻り値

処理が正常終了した場合のみ true。

参照

- QABinaryMessage.writeLong メソッド [QAnywhere C++]401 ページ

readShort メソッド

QABinaryMessage インスタンスのメッセージ本文未読部分の先頭から、符号付き 16 ビット値を読み込みます。

構文

```
public virtual qa_bool readShort(qa_short * value)
```

パラメーター

- **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。

戻り値

処理が正常終了した場合のみ true。

参照

- QABinaryMessage.writeShort メソッド [QAnywhere C++]402 ページ

readString メソッド

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

構文

```
public virtual qa_int readString(qa_string dest, qa_int maxLen)
```

パラメーター

- **dest**　バイトメッセージストリームから読み込んだ qa_string 値の格納先。
- **maxLen**　読み込む文字の最大数 (NULL ターミネーターも含む)。

戻り値

バッファーに読み込まれた NULL 以外の qa_char の合計数。読み込めるデータが残っていない場合、またはエラーが発生した場合は -1。バッファーが小さすぎる場合は -2。

参照

- QABinaryMessage.writeString メソッド [QAnywhere C++]402 ページ

reset メソッド

メッセージをリセットして、メッセージ本文の先頭から値の読み込みを開始できるようにします。

構文

```
public virtual void reset()
```

備考

また、reset メソッドは、QABinaryMessage のメッセージ本文を読み込み専用モードにします。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

writeBinary メソッド

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

構文

```
public virtual void writeBinary(
     qa_const_bytes value,
    qa_int offset,
    qa_int length
)
```

パラメーター

- **value** メッセージ本文に書き込む byte 配列値。
- **offset** byte 配列内でのオフセット (書き込み開始位置)。
- **length** 書き込まれるバイト数。

参照

- QABinaryMessage.readBinary メソッド [QAnywhere C++]394 ページ

writeBoolean メソッド

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

構文

```
public virtual void writeBoolean(qa_bool value)
```

パラメーター

- **value** メッセージ本文に書き込む boolean 値。

備考

boolean 値は 1 バイトの値で示されます。true は 1、false は 0 で示されます。

参照

- QABinaryMessage.readBoolean メソッド [QAnywhere C++]395 ページ

writeByte メソッド

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

構文

```
public virtual void writeByte(qa_byte value)
```

パラメーター

- **value**　メッセージ本文に書き込む byte 配列値。

備考

byte 値は 1 バイトの値で示されます。

参照

- QABinaryMessage.readByte メソッド [QAnywhere C++]395 ページ

writeChar メソッド

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

構文

```
public virtual void writeChar(qa_char value)
```

パラメーター

- **value**　メッセージ本文に書き込む char 値。

備考

char パラメーターは 2 バイトの値で示され、上位のバイトから追加されます。

参照

- QABinaryMessage.readChar メソッド [QAnywhere C++]395 ページ

writeDouble メソッド

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

構文

```
public virtual void writeDouble(qa_double value)
```

パラメーター

- **value**　メッセージ本文に書き込む double 値。

備考

double パラメーターは 8 バイトの long 値に変換されます。上位のバイトから追加されます。

参照

- QABinaryMessage.readDouble メソッド [QAnywhere C++]396 ページ

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

writeFloat メソッド

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

構文

```
public virtual void writeFloat(qa_float value)
```

パラメーター

- **value** メッセージ本文に書き込む float 値。

備考

float パラメーターは 4 バイトの int に変換され、上位のバイトから追加されます。

参照

- QABinaryMessage.readFloat メソッド [QAnywhere C++]396 ページ

writeInt メソッド

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

構文

```
public virtual void writeInt(qa_int value)
```

パラメーター

- **value** メッセージ本文に書き込む int 値。

備考

int パラメーターは 4 バイトの値で示され、上位のバイトから追加されます。

参照

- QABinaryMessage.readInt メソッド [QAnywhere C++]397 ページ

writeLong メソッド

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

構文

```
public virtual void writeLong(qa_long value)
```

パラメーター

- **value** メッセージ本文に書き込む long 値。

備考

long パラメーターは 8 バイトの値で示され、上位のバイトから追加されます。

参照

- QABinaryMessage.readLong メソッド [QAnywhere C++]397 ページ

writeShort メソッド

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

構文

```
public virtual void writeShort(qa_short value)
```

パラメーター

- **value**　メッセージ本文に書き込む short 値。

備考

short パラメーターは 2 バイトの値で示され、上位のバイトから追加されます。

参照

- QABinaryMessage.readShort メソッド [QAnywhere C++]397 ページ

writeString メソッド

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

構文

```
public virtual void writeString(qa_const_string value)
```

パラメーター

- **value**　メッセージ本文に書き込む string 値。

備考

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

注意
文字列の UTF-8 表記は、最大で 32767 バイトまで可能です。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- QABinaryMessage.readString メソッド [QAnywhere C++]398 ページ

QAError クラス

このクラスは、QAnywhere クライアントアプリケーションに関連付けられているエラー定数を定義します。

構文

```
public class QAError
```

メンバー

継承されたメンバーを含む QAError クラスのすべてのメンバー。

名前	説明
COMMON_ALREADY_OPEN_ERROR 変数	QAManager はすでに開いています。
COMMON_GET_INIT_FILE_ERROR 変数	クライアントのプロパティファイルにアクセスできません。
COMMON_GET_PROPERTY_ERROR 変数	プロパティの取得中にエラーが発生しました。
COMMON_GETQUEUEDEPTH_ERROR 変数	キューの長さの取得中にエラーが発生しました。
COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG 変数	フィルターが ALL の場合、指定された宛先には QAManagerBase.getQueueDepth を使用できません。
COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID 変数	メッセージストア ID が設定されていない場合は QAManagerBase.getQueueDepth を使用できません。
COMMON_INIT_ERROR 変数	初期化エラーです。
COMMON_INIT_THREAD_ERROR 変数	バックグラウンドスレッドを初期化中にエラーが発生しました。
COMMON_INVALID_PROPERTY 変数	クライアントのプロパティファイルに無効なプロパティが存在します。
COMMON_MSG_ACKNOWLEDGE_ERROR 変数	メッセージの確認中にエラーが発生しました。

名前	説明
COMMON_MSG_CANCEL_ERROR 変数	メッセージのキャンセル中にエラーが発生しました。
COMMON_MSG_CANCEL_ERROR_SENT 変数	メッセージのキャンセル中にエラーが発生しました。
COMMON_MSG_NOT_WRITEABLE_ERROR 変数	読み込み専用モードであるため、メッセージに書き込みできません。
COMMON_MSG_RETRIEVE_ERROR 変数	クライアントメッセージストアからメッセージを取得するときにエラーが発生しました。
COMMON_MSG_STORE_ERROR 変数	クライアントメッセージストアにメッセージを格納するときにエラーが発生しました。
COMMON_MSG_STORE_NOT_INITIALIZED 変数	メッセージストアがメッセージを送るために初期化されていません。
COMMON_MSG_STORE_TOO_LARGE 変数	メッセージストアがデバイスのディスク空き容量と比べて大きすぎます。
COMMON_NO_DEST_ERROR 変数	宛先が指定されていません。
COMMON_NO_IMPLEMENTATION 変数	この関数は実装されていません。
COMMON_NOT_OPEN_ERROR 変数	QAManager が開いていません。
COMMON_OPEN_ERROR 変数	メッセージストアへの接続を開くときにエラーが発生しました。
COMMON_OPEN_LOG_FILE_ERROR 変数	ログファイルを開くときにエラーが発生しました。
COMMON_OPEN_MAXTHREADS_ERROR 変数	サーバーの最大同時要求数が少なすぎるため QAManager を開くことができません。
COMMON_SELECTOR_SYNTAX_ERROR 変数	指定されたセレクターに構文エラーがあります。
COMMON_SET_PROPERTY_ERROR 変数	プロパティの設定中にエラーが発生しました。
COMMON_TERMINATE_ERROR 変数	終了エラーです。
COMMON_UNEXPECTED_EOM_ERROR 変数	予期しないメッセージの終わりに到達しました。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
COMMON_UNREPRESENTABLE_TIMESTAMP 変数	タイムスタンプが許容範囲外です。
QA_NO_ERROR 変数	エラーはありません。

備考

QAError オブジェクトは、メッセージング操作と対応するエラーを追跡するために、QAManager オブジェクトによって内部的に使用されます。アプリケーションプログラマーが、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ーはこれらのエラー定数を使用して、QAManager::getLastError から返されたエラーコードを解釈します。

参照

- QAManagerBase.getLastErrorMsg メソッド [QAnywhere C++]433 ページ

例

qa_mgr という QAManager インスタンスがあるとします。次の例では、QAManager::getLastError メソッドを使用して、最後のエラーコードを QAError::QA_NO_ERROR と比較します。

```
if (qa_mgr->getLastError() != QAError::QA_NO_ERROR) {
    // Process error.
}
```

COMMON_ALREADY_OPEN_ERROR 変数

QAManager はすでに開いています。

構文

```
public static const qa_int COMMON_ALREADY_OPEN_ERROR;
```

COMMON_GET_INIT_FILE_ERROR 変数

クライアントのプロパティファイルにアクセスできません。

構文

```
public static const qa_int COMMON_GET_INIT_FILE_ERROR;
```

COMMON_GET_PROPERTY_ERROR 変数

プロパティの取得中にエラーが発生しました。

構文

```
public static const qa_int COMMON_GET_PROPERTY_ERROR;
```

COMMON_GETQUEUEDEPTH_ERROR 変数

キューの長さの取得中にエラーが発生しました。

構文

```
public static const qa_int COMMON_GETQUEUEDEPTH_ERROR;
```

COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG 変数

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

構文

```
public static const qa_int COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG;
```

COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID 変数

メッセージストア ID が設定されていない場合は、QAManagerBase.getQueueDepth を使用できません。

構文

```
public static const qa_int COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID;
```

COMMON_INIT_ERROR 変数

初期化エラーです。

構文

```
public static const qa_int COMMON_INIT_ERROR;
```

COMMON_INIT_THREAD_ERROR 変数

バックグラウンドスレッドを初期化中にエラーが発生しました。

構文

```
public static const qa_int COMMON_INIT_THREAD_ERROR;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

COMMON_INVALID_PROPERTY 変数

クライアントの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。

構文

```
public static const qa_int COMMON_INVALID_PROPERTY;
```

COMMON_MSG_ACKNOWLEDGE_ERROR 変数

メッセージの確認中にエラーが発生しました。

構文

```
public static const qa_int COMMON_MSG_ACKNOWLEDGE_ERROR;
```

COMMON_MSG_CANCEL_ERROR 変数

メッセージのキャンセル中にエラーが発生しました。

構文

```
public static const qa_int COMMON_MSG_CANCEL_ERROR;
```

COMMON_MSG_CANCEL_ERROR_SENT 変数

メッセージのキャンセル中にエラーが発生しました。

構文

```
public static const qa_int COMMON_MSG_CANCEL_ERROR_SENT;
```

備考

すでに送信されたメッセージをキャンセルすることはできません。

COMMON_MSG_NOT_WRITEABLE_ERROR 変数

読み込み専用モードであるため、メッセージに書き込みできません。

構文

```
public static const qa_int COMMON_MSG_NOT_WRITEABLE_ERROR;
```

COMMON_MSG_RETRIEVE_ERROR 変数

クライアントメッセージストアからメッセージを取得中にエラーが発生しました。

構文

```
public static const qa_int COMMON_MSG_RETRIEVE_ERROR;
```

COMMON_MSG_STORE_ERROR 変数

メッセージをクライアントメッセージストアに格納中にエラーが発生しました。

構文

```
public static const qa_int COMMON_MSG_STORE_ERROR;
```

COMMON_MSG_STORE_NOT_INITIALIZED 変数

メッセージストアがメッセージを送るために初期化されていません。

構文

```
public static const qa_int COMMON_MSG_STORE_NOT_INITIALIZED;
```

COMMON_MSG_STORE_TOO_LARGE 変数

メッセージストアがデバイスのディスク空き容量と比べて大きすぎます。

構文

```
public static const qa_int COMMON_MSG_STORE_TOO_LARGE;
```

COMMON_NO_DEST_ERROR 変数

宛先が指定されていません。

構文

```
public static const qa_int COMMON_NO_DEST_ERROR;
```

COMMON_NO_IMPLEMENTATION 変数

この関数は実装されていません。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
public static const qa_int COMMON_NO_IMPLEMENTATION;
```

COMMON_NOT_OPEN_ERROR 変数

QAManager が開いていません。

構文

```
public static const qa_int COMMON_NOT_OPEN_ERROR;
```

COMMON_OPEN_ERROR 変数

メッセージストアへの接続を開くときにエラーが発生しました。

構文

```
public static const qa_int COMMON_OPEN_ERROR;
```

COMMON_OPEN_LOG_FILE_ERROR 変数

ログファイルを開くときにエラーが発生しました。

構文

```
public static const qa_int COMMON_OPEN_LOG_FILE_ERROR;
```

COMMON_OPEN_MAXTHREADS_ERROR 変数

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

構文

```
public static const qa_int COMMON_OPEN_MAXTHREADS_ERROR;
```

備考

詳細については、「-gn dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』を参照してください。

COMMON_SELECTOR_SYNTAX_ERROR 変数

指定されたセレクターに構文エラーがあります。

構文

```
public static const qa_int COMMON_SELECTOR_SYNTAX_ERROR;
```

COMMON_SET_PROPERTY_ERROR 変数

プロパティの設定中にエラーが発生しました。

構文

```
public static const qa_int COMMON_SET_PROPERTY_ERROR;
```

COMMON_TERMINATE_ERROR 変数

終了エラーです。

構文

```
public static const qa_int COMMON_TERMINATE_ERROR;
```

COMMON_UNEXPECTED_EOM_ERROR 変数

予期しないメッセージの終わりに到達しました。

構文

```
public static const qa_int COMMON_UNEXPECTED_EOM_ERROR;
```

COMMON_UNREPRESENTABLE_TIMESTAMP 変数

タイムスタンプが許容範囲外です。

構文

```
public static const qa_int COMMON_UNREPRESENTABLE_TIMESTAMP;
```

QA_NO_ERROR 変数

エラーはありません。

構文

```
public static const qa_int QA_NO_ERROR;
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

QAManager クラス

QAManager クラスは QAManagerBase から派生し、非トランザクション指向の QAnywhere メッセージング操作を管理します。

構文

```
public class QAManager : QAManagerBase
```

基本クラス

- QAManagerBase クラス [QAnywhere C++]418 ページ

メンバー

継承されたメンバーを含む QAManager クラスのすべてのメンバー。

名前	説明
acknowledge メソッド	クライアントアプリケーションが QAnywhere メッセージを正常に受信したことを確認します。
acknowledgeAll メソッド	クライアントアプリケーションが、受信確認されていない QAnywhere メッセージを、すべて正常に受信したことを確認します。
acknowledgeUntil メソッド	指定された QAMessage インスタンスと、指定されたメッセージよりも前に受信されて受信確認されていないメッセージについて、すべて受信確認します。
beginEnumStorePropertyNames メソッド	メッセージストアプロパティ名の列挙を開始します。
browseClose メソッド	参照操作に関連付けられているリソースを解放します。
browseMessages メソッド	メッセージストアのキューに登録されているメッセージの参照を開始します。
browseMessagesByID メソッド	メッセージストアのキューに登録されているメッセージのうち、指定されたメッセージ ID を持つメッセージの参照を開始します。
browseMessagesByQueue メソッド	メッセージストアのキューに登録されているメッセージのうち、指定されたキューのメッセージの参照を開始します。

名前	説明
browseMessagesBySelector メソッド	メッセージストアのキューに登録されているメッセージのうち、指定されたセレクターを満たすメッセージの参照を開始します。
browseNextMessage メソッド	指定された参照操作の次のメッセージを返します。メッセージがない場合は NULL を返します。
cancelMessage メソッド	指定されたメッセージ ID を持つメッセージをキャンセルします。
close メソッド	QAnywhere メッセージシステムへの接続をクローズして、QAManagerBase で使用していたリソースをすべて解放します。
createBinaryMessage メソッド	QABinaryMessage インスタンスを作成します。
createTextMessage メソッド	QATextMessage インスタンスを作成します。
deleteMessage メソッド	QAMessage オブジェクトを削除します。
endEnumStorePropertyNames メソッド	メッセージストアプロパティ名の列挙に関連付けられているリソースを解放します。
getAllQueueDepth メソッド	指定されたフィルターに基づいて、すべてのキューの長さの合計を返します。
getBooleanStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの boolean 値を取得します。
getByteStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの byte 値を取得します。
getDoubleStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの double 値を取得します。
getFloatStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの float 値を取得します。
getIntStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの int 値を取得します。
getLastError メソッド	最後に実行された QAManagerBase メソッドに関連付けられているエラーコードです。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
getLastErrorMsg メソッド	最後に実行された QAManagerBase メソッドに関連付けられているエラーテキストです。
getLastNativeError メソッド	最後に実行された QAManagerBase メソッドに関連付けられているネイティブエラーコードです。
getLongStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの long 値を取得します。
getMessage メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
getMessageBySelector メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
getMessageBySelectorNoWait メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
getMessageBySelectorTimeout メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
getMessageNoWait メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
getMessageTimeout メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
getMode メソッド	受信したメッセージの QAManager 受信確認モードを返します。
getQueueDepth メソッド	指定されたフィルターに基づいて、キューの長さを返します。
getShortStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの short 値を取得します。
getStringStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの string 値を取得します。
nextStorePropertyName メソッド	指定された列挙のメッセージストアプロパティ名を返します。

名前	説明
open メソッド	指定された AcknowledgementMode 値を使用して QAManager をオープンします。
putMessage メソッド	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。
putMessageTimeToLive メソッド	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。
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。
setBooleanStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを boolean 値に設定します。
setByteStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを byte 値に設定します。
setDoubleStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを double 値に設定します。
setFloatStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを float 値に設定します。
setIntStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを int 値に設定します。
setLongStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを long 値に設定します。
setMessageListener メソッド	QAnywhere メッセージを非同期的に受信するようにメッセージリスナークラスを設定します。
setMessageListenerBySelector メソッド	メッセージセレクターを使用して、QAnywhere メッセージを非同期的に受信するように、メッセージリスナークラスを設定します。
setProperty メソッド	QAnywhere Manager の設定プロパティをプログラムで設定できるようにします。
setShortStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを short 値に設定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
setStringStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを string 値に設定します。
start メソッド	着信メッセージをメッセージリスナーで受信するための QAManagerBase を起動します。
stop メソッド	QAManagerBase による着信メッセージの受信を停止します。
triggerSendReceive メソッド	QAnywhere メッセージサーバーとの同期処理を発生させて、他のクライアント宛てのメッセージをアップロードし、ローカルクライアント宛てのメッセージをダウンロードします。

備考

この動作の完全な説明については、QAManagerBase を参照してください。

QAManager は、AcknowledgementMode 列挙体の定義のようにして、明示的に受信を確認するように設定することも、暗黙的に受信を確認するように設定することもできます。トランザクションの一部としてメッセージの受信を確認するには、QATransactionalManager を使用します。

QAManager と QATransactionalManager オブジェクトを作成するには、QAManagerFactory を使用します。

参照

- QAManagerFactory クラス [QAnywhere C++]450 ページ
- QAManagerBase クラス [QAnywhere C++]418 ページ
- QATransactionalManager クラス [QAnywhere C++]484 ページ
- AcknowledgementMode クラス [QAnywhere C++]375 ページ

acknowledge メソッド

クライアントアプリケーションが QAnywhere メッセージを正常に受信したことを確認します。

構文

```
public virtual qa_bool acknowledge(QAMessage * msg)
```

パラメーター

- **msg**　受信確認するメッセージ。

戻り値

処理が正常終了した場合のみ true。

備考

> **注意**
> QAMessage が受信を確認すると、その MessageProperties::STATUS プロパティが StatusCodes.RECEIVED に変わります。QAMessage は、ステータスが StatusCodes.RECEIVED に変わると、デフォルトの削除ルールを使用して削除できるようになります。

削除ルールの詳細については、「メッセージの削除ルール」786 ページを参照してください。

参照

- QAManager.acknowledgeAll メソッド [QAnywhere C++]416 ページ
- QAManager.acknowledgeUntil メソッド [QAnywhere C++]416 ページ

acknowledgeAll メソッド

クライアントアプリケーションが、受信確認されていない QAnywhere メッセージを、すべて正常に受信したことを確認します。

構文

```
public virtual qa_bool acknowledgeAll()
```

戻り値

処理が正常終了した場合のみ true。

備考

> **注意**
> QAMessage が受信確認されると、その MessageProperties::STATUS プロパティは StatusCodes.RECEIVED に変わります。QAMessage は、ステータスが StatusCodes.RECEIVED に変わると、デフォルトの削除ルールを使用して削除できるようになります。

削除ルールの詳細については、「メッセージの削除ルール」786 ページを参照してください。

参照

- QAManager.acknowledge メソッド [QAnywhere C++]415 ページ
- QAManager.acknowledgeUntil メソッド [QAnywhere C++]416 ページ

acknowledgeUntil メソッド

指定された QAMessage インスタンスと、指定されたメッセージよりも前に受信されて受信確認されていないメッセージについて、すべて受信確認します。

構文

```
public virtual qa_bool acknowledgeUntil(QAMessage * msg)
```

パラメーター

- **msg**　受信確認するメッセージのうち最新のもの。それ以前の受信確認されていないメッセージも、すべて受信確認されます。

戻り値

処理が正常終了した場合のみ true。

備考

> **注意**
> QAMessage が受信確認されると、その MessageProperties::STATUS プロパティは StatusCodes.RECEIVED に変わります。QAMessage は、ステータスが StatusCodes.RECEIVED に変わると、デフォルトの削除ルールを使用して削除できるようになります。

削除ルールの詳細については、「メッセージの削除ルール」786 ページを参照してください。

参照

- QAManager.acknowledge メソッド [QAnywhere C++]415 ページ
- QAManager.acknowledgeAll メソッド [QAnywhere C++]416 ページ

open メソッド

指定された AcknowledgementMode 値を使用して QAManager をオープンします。

構文

```
public virtual qa_bool open(qa_short mode)
```

パラメーター

- **mode**　受信確認モード。

戻り値

処理が正常終了した場合のみ true。

備考

open メソッドは、QAManager を作成した後、最初に呼び出す必要があるメソッドです。

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

参照

- AcknowledgementMode クラス [QAnywhere C++]375 ページ

recover メソッド

受信確認されていないメッセージを、すべて強制的に未受信に戻します。

構文

```
public virtual qa_bool recover()
```

戻り値

処理が正常終了した場合のみ true。

備考

つまり、該当するメッセージは、QAManager::getMessage() を使用して再度受信する必要があります。

QAManagerBase クラス

このクラスは、QATransactionalManager と QAManager の基本クラスです。前者の派生クラスはトランザクション指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指向のメッセージングを管理します。

構文

```
public class QAManagerBase
```

派生クラス

- QAManager クラス [QAnywhere C++]411 ページ
- QATransactionalManager クラス [QAnywhere C++]484 ページ

メンバー

継承されたメンバーを含む QAManagerBase クラスのすべてのメンバー。

名前	説明
beginEnumStorePropertyNames メソッド	メッセージストアプロパティ名の列挙を開始します。
browseClose メソッド	参照操作に関連付けられているリソースを解放します。
browseMessages メソッド	メッセージストアのキューに登録されているメッセージの参照を開始します。
browseMessagesByID メソッド	メッセージストアのキューに登録されているメッセージのうち、指定されたメッセージ ID を持つメッセージの参照を開始します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
browseMessagesByQueue メソッド	メッセージストアのキューに登録されているメッセージのうち、指定されたキューのメッセージの参照を開始します。
browseMessagesBySelector メソッド	メッセージストアのキューに登録されているメッセージのうち、指定されたセレクターを満たすメッセージの参照を開始します。
browseNextMessage メソッド	指定された参照操作の次のメッセージを返します。メッセージがない場合は NULL を返します。
cancelMessage メソッド	指定されたメッセージ ID を持つメッセージをキャンセルします。
close メソッド	QAnywhere メッセージシステムへの接続をクローズして、QAManagerBase で使用していたリソースをすべて解放します。
createBinaryMessage メソッド	QABinaryMessage インスタンスを作成します。
createTextMessage メソッド	QATextMessage インスタンスを作成します。
deleteMessage メソッド	QAMessage オブジェクトを削除します。
endEnumStorePropertyNames メソッド	メッセージストアプロパティ名の列挙に関連付けられているリソースを解放します。
getAllQueueDepth メソッド	指定されたフィルターに基づいて、すべてのキューの長さの合計を返します。
getBooleanStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの boolean 値を取得します。
getByteStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの byte 値を取得します。
getDoubleStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの double 値を取得します。
getFloatStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの float 値を取得します。
getIntStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの int 値を取得します。

名前	説明
getLastError メソッド	最後に実行された QAManagerBase メソッドに関連付けられているエラーコードです。
getLastErrorMsg メソッド	最後に実行された QAManagerBase メソッドに関連付けられているエラーテキストです。
getLastNativeError メソッド	最後に実行された QAManagerBase メソッドに関連付けられているネイティブエラーコードです。
getLongStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの long 値を取得します。
getMessage メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
getMessageBySelector メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
getMessageBySelectorNoWait メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
getMessageBySelectorTimeout メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
getMessageNoWait メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
getMessageTimeout メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
getMode メソッド	受信したメッセージの QAManager 受信確認モードを返します。
getQueueDepth メソッド	指定されたフィルターに基づいて、キューの長さを返します。
getShortStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの short 値を取得します。
getStringStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの string 値を取得します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
nextStorePropertyName メソッド	指定された列挙のメッセージストアプロパティ名を返します。
putMessage メソッド	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。
putMessageTimeToLive メソッド	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。
setBooleanStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを boolean 値に設定します。
setByteStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを byte 値に設定します。
setDoubleStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを double 値に設定します。
setFloatStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを float 値に設定します。
setIntStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを int 値に設定します。
setLongStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを long 値に設定します。
setMessageListener メソッド	QAnywhere メッセージを非同期的に受信するようにメッセージリスナークラスを設定します。
setMessageListenerBySelector メソッド	メッセージセレクターを使用して、QAnywhere メッセージを非同期的に受信するように、メッセージリスナークラスを設定します。
setProperty メソッド	QAnywhere Manager の設定プロパティをプログラムで設定できるようにします。
setShortStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを short 値に設定します。
setStringStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを string 値に設定します。

名前	説明
start メソッド	着信メッセージをメッセージリスナーで受信するための QAManagerBase を起動します。
stop メソッド	QAManagerBase による着信メッセージの受信を停止します。
triggerSendReceive メソッド	QAnywhere メッセージサーバーとの同期処理を発生させて、他のクライアント宛てのメッセージをアップロードし、ローカルクライアント宛てのメッセージをダウンロードします。

備考

QAManagerBase インスタンスがメッセージを受信できるようにするには、QAManagerBase::start() メソッドを使用します。QAManagerBase インスタンスは、アプリケーションのスレッドごとに 1 つだけにします。

このクラスのインスタンスを使用して、QAnywhere メッセージの作成と管理を行うことができます。適切な QAMessage インスタンスを作成するには、QAManagerBase::createBinaryMessage() メソッドと QAManagerBase::createTextMessage() メソッドを使用します。QAMessage インスタンスには、メッセージの内容とプロパティを設定するための、さまざまなメソッドがあります。QAnywhere メッセージを送信するには、QAManager::putMessage() を使用して、アドレス指定されたメッセージをローカルのメッセージストアキューに登録します。メッセージは、転送ポリシーに基づいて QAnywhere Agent によって転送されるか、QAManagerBase::triggerSendReceive() が呼び出されたときに転送されます。

qaagent 転送ポリシーの詳細については、「クライアントにメッセージを転送するタイミングの決定」49 ページを参照してください。

QAManagerBase::close() を使用して QAManagerBase インスタンスがクローズされると、メッセージがメモリから解放されます。

QAException が発生した場合にエラー情報を返すには、QAManagerBase::getLastError、QAManagerBase::getLastNativeError、QAManagerBase::getLastErrorMessage を使用します。QAManagerBase にも、メッセージストアプロパティを設定および取得するためのメソッドがあります。

詳細については、「クライアントメッセージストアプロパティ」29 ページと MessageStoreProperties を参照してください。

参照

- QATransactionalManager クラス [QAnywhere C++]484 ページ
- QAManager クラス [QAnywhere C++]411 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

beginEnumStorePropertyNames メソッド

メッセージストアプロパティ名の列挙を開始します。

構文

```
public virtual qa_store_property_enum_handle
beginEnumStorePropertyNames()
```

戻り値

QAManagerBase::nextStorePropertyName に渡されるハンドル。

備考

このメソッドから返されるハンドルは、QAManagerBase::nextStorePropertyName に渡されます。このメソッドと QAManagerBase::nextStorePropertyName を使用することで、このメソッドが呼び出されたときのメッセージストアプロパティ名を列挙できます。
QAManagerBase::beginEnumStorePropertyNames の呼び出しと QAManagerBase::endEnumStorePropertyNames の呼び出しの間は、メッセージストアプロパティを設定できません。

参照

- QAManagerBase.nextStorePropertyName メソッド [QAnywhere C++]441 ページ
- QAManagerBase.beginEnumStorePropertyNames メソッド [QAnywhere C++]423 ページ
- QAManagerBase.endEnumStorePropertyNames メソッド [QAnywhere C++]429 ページ

browseClose メソッド

参照操作に関連付けられているリソースを解放します。

構文

```
public virtual void browseClose(qa_browse_handle handle)
```

パラメーター

- **handle**　参照を開始するいずれかの操作から返されたハンドル。

browseMessages メソッド

メッセージストアのキューに登録されているメッセージの参照を開始します。

構文

```
public virtual qa_browse_handle browseMessages()
```

戻り値

QAManagerBase::browseNextMessage に渡されるハンドル

備考

このメソッドから返されるハンドルは、QAManagerBase::browseNextMessage に渡されます。このメソッドと QAManagerBase::browseNextMessage を使用することで、このメソッドが呼び出されたときにメッセージストア内にあるメッセージを列挙できます。

メッセージは単に参照されるだけなので、受信確認はできません。メッセージを受信して受信確認できるようにする場合は、QAManagerBase::getMessage を使用します。

参照

- QAManagerBase.browseNextMessage メソッド [QAnywhere C++]426 ページ
- QAManagerBase.browseMessagesByQueue メソッド [QAnywhere C++]425 ページ
- QAManagerBase.browseMessagesByID メソッド [QAnywhere C++]424 ページ
- QAManagerBase.browseClose メソッド [QAnywhere C++]423 ページ

browseMessagesByID メソッド

メッセージストアのキューに登録されているメッセージのうち、指定されたメッセージ ID を持つメッセージの参照を開始します。

構文

```
public virtual qa_browse_handle browseMessagesByID(
     qa_const_string msgid
)
```

パラメーター

- **msgid**　メッセージ ID。

戻り値

browseNextMessage に渡されるハンドル。

備考

このメソッドから返されるハンドルは、QAManagerBase::browseNextMessage に渡されます。このメソッドと QAManagerBase::browseNextMessage を使用することで、このメソッドが呼び出されたときにメッセージストア内にあるメッセージを列挙できます。

メッセージは単に参照されるだけなので、受信確認はできません。メッセージを受信して受信確認できるようにする場合は、QAManagerBase::getMessage を使用します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- QAManagerBase.browseNextMessage メソッド [QAnywhere C++]426 ページ
- QAManagerBase.browseMessagesByQueue メソッド [QAnywhere C++]425 ページ
- QAManagerBase.browseMessages メソッド [QAnywhere C++]423 ページ
- QAManagerBase.browseClose メソッド [QAnywhere C++]423 ページ

browseMessagesByQueue メソッド

メッセージストアのキューに登録されているメッセージのうち、指定されたキューのメッセージの参照を開始します。

構文

```
public virtual qa_browse_handle browseMessagesByQueue(
     qa_const_string address
)
```

パラメーター

- **address**　参照対象のキュー。

戻り値

browseNextMessage に渡されるハンドル。

備考

このメソッドから返されるハンドルは、QAManagerBase::browseNextMessage に渡されます。このメソッドと QAManagerBase::browseNextMessage を使用することで、このメソッドが呼び出されたときにメッセージストア内にあるメッセージを列挙できます。

メッセージは単に参照されるだけなので、受信確認はできません。メッセージを受信して受信確認できるようにする場合は、QAManagerBase::getMessage を使用します。

参照

- QAManagerBase.browseNextMessage メソッド [QAnywhere C++]426 ページ
- QAManagerBase.browseMessagesByID メソッド [QAnywhere C++]424 ページ
- QAManagerBase.browseMessages メソッド [QAnywhere C++]423 ページ
- QAManagerBase.browseClose メソッド [QAnywhere C++]423 ページ

browseMessagesBySelector メソッド

メッセージストアのキューに登録されているメッセージのうち、指定されたセレクターを満たすメッセージの参照を開始します。

構文

```
public virtual qa_browse_handle browseMessagesBySelector(
     qa_const_string selector
)
```

パラメーター

- **selector**　セレクター。

戻り値

browseNextMessage に渡されるハンドル。

備考

このメソッドから返されるハンドルは、QAManagerBase::browseNextMessage に渡されます。このメソッドと QAManagerBase::browseNextMessage を使用することで、このメソッドが呼び出されたときにメッセージストア内にあるメッセージを列挙できます。

メッセージは単に参照されるだけなので、受信確認はできません。

メッセージを受信して受信確認できるようにする場合は、QAManagerBase::getMessage を使用します。

参照

- QAManagerBase.browseNextMessage メソッド [QAnywhere C++]426 ページ
- QAManagerBase.browseMessagesByID メソッド [QAnywhere C++]424 ページ
- QAManagerBase.browseMessagesByQueue メソッド [QAnywhere C++]425 ページ
- QAManagerBase.browseMessages メソッド [QAnywhere C++]423 ページ
- QAManagerBase.browseClose メソッド [QAnywhere C++]423 ページ

browseNextMessage メソッド

指定された参照操作の次のメッセージを返します。メッセージがない場合は NULL を返します。

構文

```
public virtual QAMessage * browseNextMessage(qa_browse_handle handle)
```

パラメーター

- **handle**　参照を開始するいずれかの操作から返されたハンドル。

戻り値

次のメッセージ。メッセージがない場合は qa_null。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

参照対象メッセージのハンドルを取得するには、QAManagerBase::browseMessages を使用します。または、他の QAManagerBase メソッドを使用して、キューまたはメッセージ ID でメッセージを参照することもできます。

参照

- QAManagerBase.browseMessages メソッド [QAnywhere C++]423 ページ
- QAManagerBase.browseMessagesByQueue メソッド [QAnywhere C++]425 ページ
- QAManagerBase.browseMessagesByID メソッド [QAnywhere C++]424 ページ
- QAManagerBase.browseClose メソッド [QAnywhere C++]423 ページ

cancelMessage メソッド

指定されたメッセージ ID を持つメッセージをキャンセルします。

構文

```
public virtual qa_bool cancelMessage(qa_const_string msgid)
```

パラメーター

- **msgid**　キャンセルするメッセージの ID。

戻り値

処理が正常終了した場合のみ true。

備考

cancelMessage メソッドは、メッセージが転送される前にメッセージをキャンセル済みの状態にします。QAnywhere Agent のデフォルトの削除ルールにより、キャンセル済みメッセージは最終的にメッセージストアから削除されます。

メッセージがすでに最終ステータスになっている場合や、中央のメッセージングサーバーに転送済みである場合は、cancelMessage メソッドは失敗します。

削除ルールの詳細については、「メッセージの削除ルール」786 ページを参照してください。

close メソッド

QAnywhere メッセージシステムへの接続をクローズして、QAManagerBase で使用していたリソースをすべて解放します。

構文

```
public virtual qa_bool close()
```

戻り値

処理が正常終了した場合のみ true。

備考

この後で呼び出される close() は無視されます。いったんクローズした QAManagerBase のインスタンスをもう一度オープンすることはできません。この場合は、新規の QAManagerBase インスタンスを作成してから、オープンします。

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

参照

- QAManager.open メソッド [QAnywhere C++]417 ページ
- QATransactionalManager.open メソッド [QAnywhere C++]489 ページ

createBinaryMessage メソッド

QABinaryMessage インスタンスを作成します。

構文

```
public virtual QABinaryMessage * createBinaryMessage()
```

戻り値

新しい QABinaryMessage インスタンス。

備考

QABinaryMessage インスタンスは、未解釈のバイトストリームのメッセージ本文が含まれるメッセージの送信に使用します。

参照

- QABinaryMessage クラス [QAnywhere C++]388 ページ

createTextMessage メソッド

QATextMessage インスタンスを作成します。

構文

```
public virtual QATextMessage * createTextMessage()
```

戻り値

新しい QATextMessage インスタンス。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

QATextMessage のオブジェクトは、string のメッセージ本文が含まれるメッセージを送信する場合に使用します。

参照

- QATextMessage クラス [QAnywhere C++]478 ページ

deleteMessage メソッド

QAMessage オブジェクトを削除します。

構文

```
public virtual void deleteMessage(QAMessage * msg)
```

パラメーター

- **msg**　削除するメッセージ。

備考

デフォルトでは、QAManagerBase::createTextMessage または QAManagerBase::createBinaryMessag によって作成されたメッセージは、QAManagerBase をクローズすると自動的に削除されます。このメソッドを使用すると、メッセージを削除するタイミングをより柔軟に管理できます。

endEnumStorePropertyNames メソッド

メッセージストアプロパティ名の列挙に関連付けられているリソースを解放します。

構文

```
public virtual void endEnumStorePropertyNames(
     qa_store_property_enum_handle h
)
```

パラメーター

- **h**　beginEnumStorePropertyNames から返されるハンドル。

参照

- QAManagerBase.beginEnumStorePropertyNames メソッド [QAnywhere C++]423 ページ

getAllQueueDepth メソッド

指定されたフィルターに基づいて、すべてのキューの長さの合計を返します。

構文

```
public virtual qa_int getAllQueueDepth(qa_short filter)
```

パラメーター

- **filter**　着信メッセージ、送信メッセージ、またはすべてのメッセージを示すフィルター。

戻り値

メッセージの数。エラーが発生した場合は -1。

備考

キューの着信の深さは、受信されていない着信メッセージの数です (たとえば、QAManagerBase::getMessage を使用)。キューの送信の深さは、サーバーに転送されていない送信メッセージ (コミットされていないメッセージを含む) の数です。

参照

- QueueDepthFilter クラス [QAnywhere C++]489 ページ

getBooleanStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの boolean 値を取得します。

構文

```
public virtual qa_bool getBooleanStoreProperty(
     qa_const_string name,
    qa_bool * value
)
```

パラメーター

- **name**　事前定義済みまたはカスタムのプロパティ名。
- **value**　boolean 値の格納先。

戻り値

処理が正常終了した場合のみ true。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

getByteStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの byte 値を取得します。

構文

```
public virtual qa_bool getByteStoreProperty(
     qa_const_string name,
    qa_byte * value
)
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **value** byte 値の格納先。

戻り値

処理が正常終了した場合のみ true。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

getDoubleStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの double 値を取得します。

構文

```
public virtual qa_bool getDoubleStoreProperty(
     qa_const_string name,
    qa_double * value
)
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **value** double 値の格納先。

戻り値

処理が正常終了した場合のみ true。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

getFloatStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの float 値を取得します。

構文

```
public virtual qa_bool getFloatStoreProperty(
     qa_const_string name,
    qa_float * value
)
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **value** float 値の格納先。

戻り値

処理が正常終了した場合のみ true。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

getIntStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの int 値を取得します。

構文

```
public virtual qa_bool getIntStoreProperty(
     qa_const_string name,
    qa_int * value
)
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **value** int 値の格納先。

戻り値

処理が正常終了した場合のみ true。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

getLastError メソッド

最後に実行された QAManagerBase メソッドに関連付けられているエラーコードです。

構文

```
public virtual qa_int getLastError()
```

戻り値

エラーコード。

備考

0 はエラーがないことを示します。

参照

- QAManagerBase.getLastNativeError メソッド [QAnywhere C++]434 ページ
- QAManagerBase.getLastErrorMsg メソッド [QAnywhere C++]433 ページ
- QAError クラス [QAnywhere C++]403 ページ

getLastErrorMsg メソッド

最後に実行された QAManagerBase メソッドに関連付けられているエラーテキストです。

構文

```
public virtual qa_string getLastErrorMsg()
```

戻り値

エラーメッセージ。

備考

QAManagerBase::getLastError が 0 を返した場合、このメソッドは NULL を返します。このプロパティは、QAError をキャッチした後で取得できます。

参照

- QAManagerBase.getLastError メソッド [QAnywhere C++]433 ページ
- QAManagerBase.getLastNativeError メソッド [QAnywhere C++]434 ページ
- QAError クラス [QAnywhere C++]403 ページ

getLastNativeError メソッド

最後に実行された QAManagerBase メソッドに関連付けられているネイティブエラーコードです。

構文

```
public virtual an_sql_code getLastNativeError()
```

戻り値

ネイティブのエラーコード。

備考

0 はエラーがないことを示します。

参照

- QAManagerBase.getLastError メソッド [QAnywhere C++]433 ページ
- QAManagerBase.getLastErrorMsg メソッド [QAnywhere C++]433 ページ
- QAError クラス [QAnywhere C++]403 ページ

getLongStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの long 値を取得します。

構文

```
public virtual qa_bool getLongStoreProperty(
     qa_const_string name,
    qa_long * value
)
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **value** long 値の格納先。

戻り値

処理が正常終了した場合のみ true。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

getMessage メソッド

指定されたアドレスに送信された、次に取得可能な QAMessage を返します。

構文

```
public virtual QAMessage * getMessage(qa_const_string address)
```

パラメーター

- **address** 送信先。

戻り値

該当する次の QAMessage。メッセージが存在しない場合は NULL。

備考

address パラメーターには、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージが存在しない場合、新しいメッセージが着信するまでブロックされます。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (メッセージイベントハンドラーを使用する) 方法の詳細については、「非同期的なメッセージ受信」73 ページを参照してください。

getMessageBySelector メソッド

指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。

構文

```
public virtual QAMessage * getMessageBySelector(
     qa_const_string address,
    qa_const_string selector
)
```

パラメーター

- **address** 送信先。
- **selector** セレクター。

戻り値

該当する次の QAMessage。メッセージが存在しない場合は NULL。

備考

address パラメーターには、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージが存在しない場合、新しいメッセージが着信するまでブロックされます。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (メッセージイベントハンドラーを使用する) 方法の詳細については、「非同期的なメッセージ受信」73 ページを参照してください。

getMessageBySelectorNoWait メソッド

指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。

構文

```
public virtual QAMessage * getMessageBySelectorNoWait(
     qa_const_string address,
    qa_const_string selector
)
```

パラメーター

- **address**　送信先。
- **selector**　セレクター。

戻り値

該当する次のメッセージ。メッセージが存在しない場合は qa_null。

備考

address パラメーターには、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージがない場合は、このメソッドがすぐに返されます。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (メッセージイベントハンドラーを使用する) 方法の詳細については、「非同期的なメッセージ受信」73 ページを参照してください。

getMessageBySelectorTimeout メソッド

指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
public virtual QAMessage * getMessageBySelectorTimeout(
     qa_const_string address,
    qa_const_string selector,
    qa_long timeout
)
```

パラメーター

- **address** 送信先。
- **selector** セレクター。
- **timeout** 最大待ち時間 (ミリ秒)。

戻り値

該当する次の QAMessage。メッセージが存在しない場合は NULL。

備考

address パラメーターには、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージがない場合、このメソッドは指定されたタイムアウト時間だけ待機してから返されます。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (メッセージイベントハンドラーを使用する) 方法の詳細については、「非同期的なメッセージ受信」73 ページを参照してください。

getMessageNoWait メソッド

指定されたアドレスに送信された、次に取得可能な QAMessage を返します。

構文

```
public virtual QAMessage * getMessageNoWait(qa_const_string address)
```

パラメーター

- **address** 送信先。

戻り値

該当する次のメッセージ。メッセージが存在しない場合は qa_null。

備考

address パラメーターには、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージがない場合は、このメソッドがすぐに返されます。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (メッセージイベントハンドラーを使用する) 方法の詳細については、「非同期的なメッセージ受信」73 ページを参照してください。

getMessageTimeout メソッド

指定されたアドレスに送信された、次に取得可能な QAMessage を返します。

構文

```
public virtual QAMessage * getMessageTimeout(
     qa_const_string address,
    qa_long timeout
)
```

パラメーター

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

戻り値

該当する次の QAMessage。メッセージが存在しない場合は NULL。

備考

address パラメーターには、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージがない場合、このメソッドは指定されたタイムアウト時間だけ待機してから返されます。メッセージを同期的に受信する場合は、このメソッドを使用します。

メッセージを非同期的に受信する (メッセージイベントハンドラーを使用する) 方法の詳細については、「非同期的なメッセージ受信」73 ページを参照してください。

getMode メソッド

受信したメッセージの QAManager 受信確認モードを返します。

構文

```
public virtual qa_short getMode()
```

戻り値

受信確認モード。

備考

値のリストについては、AcknowledgementMode クラスを参照してください。

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

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- AcknowledgementMode クラス [QAnywhere C++]375 ページ

getQueueDepth メソッド

指定されたフィルターに基づいて、キューの長さを返します。

構文

```
public virtual qa_int getQueueDepth(
     qa_const_string address,
    qa_short filter
)
```

パラメーター

- **address** キュー名。
- **filter** 着信メッセージ、送信メッセージ、またはすべてのメッセージを示すフィルター。

戻り値

キューに登録されているメッセージの数。エラーが発生した場合は -1。

備考

キューの着信の深さは、受信されていない着信メッセージの数です (たとえば、QAManagerBase::getMessage を使用)。キューの送信の深さは、サーバーに転送されていない送信メッセージ (コミットされていないメッセージを含む) の数です。

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

参照

- QueueDepthFilter クラス [QAnywhere C++]489 ページ

getShortStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの short 値を取得します。

構文

```
public virtual qa_bool getShortStoreProperty(
     qa_const_string name,
    qa_short * value
)
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **value** short 値の格納先。

戻り値

処理が正常終了した場合のみ true。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

getStringStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの string 値を取得します。

構文

```
public virtual qa_int getStringStoreProperty(
     qa_const_string name,
    qa_string address,
    qa_int maxlen
)
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **address** qa_string 値の格納先。
- **maxlen** 取得した値からコピーする qa_char の最大数 (NULL ターミネーターも含む)。

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。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

nextStorePropertyName メソッド

指定された列挙のメッセージストアプロパティ名を返します。

構文

```
public virtual qa_int nextStorePropertyName(
     qa_store_property_enum_handle h,
    qa_string buffer,
    qa_int bufferLen
)
```

パラメーター

- **h**　beginEnumStorePropertyNames から返されるハンドル。
- **buffer**　プロパティ名が書き込まれるバッファー。
- **bufferLen**　プロパティ名を格納するバッファーの長さ。この長さには、NULL ターミネーターのスペースも含める必要があります。

戻り値

プロパティ名の長さ。プロパティ名がない場合は -1。プロパティ名。

備考

プロパティ名がない場合は -1 を返します。

参照

- QAManagerBase.beginEnumStorePropertyNames メソッド [QAnywhere C++]423 ページ

putMessage メソッド

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。

構文

```
public virtual qa_bool putMessage(
     qa_const_string address,
    QAMessage * msg
)
```

パラメーター

- **address**　送信先。
- **msg**　メッセージ。

戻り値

処理が正常終了した場合のみ true。

putMessageTimeToLive メソッド

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。

構文

```
public virtual qa_bool putMessageTimeToLive(
     qa_const_string address,
    QAMessage * msg,
    qa_long ttl
)
```

パラメーター

- **address** 送信先。
- **msg** メッセージ。
- **ttl** 存続時間 (ミリ秒)。

戻り値

処理が正常終了した場合のみ true。

setBooleanStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを boolean 値に設定します。

構文

```
public virtual qa_bool setBooleanStoreProperty(
     qa_const_string name,
    qa_bool value
)
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **value** プロパティの qa_bool 値。

戻り値

処理が正常終了した場合のみ true。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

setByteStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを byte 値に設定します。

構文

```
public virtual qa_bool setByteStoreProperty(
     qa_const_string name,
    qa_byte value
)
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **value** プロパティの qa_byte 値。

戻り値

処理が正常終了した場合のみ true。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

setDoubleStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを double 値に設定します。

構文

```
public virtual qa_bool setDoubleStoreProperty(
     qa_const_string name,
    qa_double value
)
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **value** プロパティの qa_double 値。

戻り値

処理が正常終了した場合のみ true。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

setFloatStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを float 値に設定します。

構文

```
public virtual qa_bool setFloatStoreProperty(
     qa_const_string name,
    qa_float value
)
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **value** プロパティの qa_float 値。

戻り値

処理が正常終了した場合のみ true。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

setIntStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを int 値に設定します。

構文

```
public virtual qa_bool setIntStoreProperty(
     qa_const_string name,
    qa_int value
)
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **value** プロパティの qa_int 値。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

戻り値

処理が正常終了した場合のみ true。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

setLongStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを long 値に設定します。

構文

```
public virtual qa_bool setLongStoreProperty(
     qa_const_string name,
    qa_long value
)
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **value** プロパティの qa_long 値。

戻り値

処理が正常終了した場合のみ true。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

setMessageListener メソッド

QAnywhere メッセージを非同期的に受信するようにメッセージリスナークラスを設定します。

構文

```
public virtual void setMessageListener(
     qa_const_string address,
    QAMessageListener * listener
)
```

パラメーター

- **address** リスナーに適用される送信先アドレス。
- **listener** 送信先アドレスに関連付けられるメッセージリスナー。

備考

リスナーは、QAMessageListener インターフェイスで定義される唯一のメソッドである QAMessageListener::onMessage を実装するクラスのインスタンスです。
QAMessageListener::onMessage は、QAMessage パラメーターを 1 つだけ指定できます。

setMessageListener の address パラメーターには、メッセージの受信で使用されるローカルキュー名を指定します。1 つのキューには、1 つのリスナーだけを割り当てることができます。

Push 通知とネットワークステータス変更も含め、QAnywhere システムメッセージを受信したい場合は、キュー名に "system" を指定します。メッセージを非同期的に受信する場合は、このメソッドを使用します。

詳細については、「非同期的なメッセージ受信」73 ページと「システムキュー」60 ページを参照してください。

setMessageListenerBySelector メソッド

メッセージセレクターを使用して、QAnywhere メッセージを非同期的に受信するように、メッセージリスナークラスを設定します。

構文

```
public virtual void setMessageListenerBySelector(
     qa_const_string address,
    qa_const_string selector,
    QAMessageListener * listener
)
```

パラメーター

- **address** リスナーに適用される送信先アドレス。
- **selector** 受信されるメッセージをフィルタリングするために使用するセレクター。
- **listener** 送信先アドレスに関連付けられるメッセージリスナー。

備考

リスナーは、QAMessageListener インターフェイスで定義される唯一のメソッドである QAMessageListener::onMessage を実装するクラスのインスタンスです。
QAMessageListener::onMessage は、QAMessage パラメーターを 1 つだけ指定できます。

setMessageListener の address パラメーターには、メッセージの受信で使用されるローカルキュー名を指定します。1 つのキューには、1 つのリスナーだけを割り当てることができます。selector パラメーターは、指定されたアドレスで受信されるメッセージをフィルタリングするために使用するセレクターを指定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

Push 通知とネットワークステータス変更も含め、QAnywhere システムメッセージを受信したい場合は、キュー名に "system" を指定します。メッセージを非同期的に受信する場合は、このメソッドを使用します。

詳細については、「非同期的なメッセージ受信」73 ページと「システムキュー」60 ページを参照してください。

setProperty メソッド

QAnywhere Manager の設定プロパティをプログラムで設定できるようにします。

構文

```
public virtual qa_bool setProperty(
     qa_const_string name,
    qa_const_string value
)
```

パラメーター

- **name** QAnywhere Manager の事前定義済みまたはカスタムの設定プロパティ名。
- **value** QAnywhere Manager の設定プロパティの値。

戻り値

処理が正常終了した場合のみ true。

備考

プロパティ名と値を指定してこのメソッドを使用することで、QAnywhere Manager のデフォルトの設定プロパティを無効にできます。

QAnywhere Manager の設定プロパティのリストについては、「QAnywhere Manager の設定プロパティ」85 ページを参照してください。

QAnywhere Manager の設定プロパティは、プロパティファイルと QAManagerFactory::createQAManager メソッドを使用して設定することもできます。

詳細については、「ファイルでの QAnywhere Manager 設定プロパティの設定」86 ページを参照してください。

注意
QAManager::open() または QATransactionalManager::open() を呼び出す前に、必要なプロパティを設定してください。

setShortStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを short 値に設定します。

構文

```
public virtual qa_bool setShortStoreProperty(
     qa_const_string name,
    qa_short value
)
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **value** プロパティの qa_short 値。

戻り値

処理が正常終了した場合のみ true。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

setStringStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを string 値に設定します。

構文

```
public virtual qa_bool setStringStoreProperty(
     qa_const_string name,
    qa_const_string value
)
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **value** プロパティの qa_string 値。

戻り値

処理が正常終了した場合のみ true。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

start メソッド

着信メッセージをメッセージリスナーで受信するための QAManagerBase を起動します。

構文

```
public virtual qa_bool start()
```

戻り値

処理が正常終了した場合のみ true。

備考

メッセージリスナーセットがない場合、たとえばメッセージが getMessage メソッドで受信される場合などは、QAManagerBase を起動する必要がありません。getMessage メソッドとメッセージリスナーは、メッセージを受信するために併用しないでください。非同期 (メッセージリスナー) モデルまたは同期 (getMessage) モデルのどちらかを使用してください。

この start メソッドを繰り返し呼び出そうとしても、間に QAManagerBase::stop() 呼び出しを挟まないかぎり 2 回目以降の呼び出しは無視されます。

参照

- QAManagerBase.stop メソッド [QAnywhere C++]449 ページ

stop メソッド

QAManagerBase による着信メッセージの受信を停止します。

構文

```
public virtual qa_bool stop()
```

戻り値

処理が正常終了した場合のみ true。

備考

メッセージは失われません。メッセージは、Manager が再起動されるまで受信されません。この stop メソッドを繰り返し呼び出そうとしても、間に QAManagerBase::start() を挟まないかぎり 2 回目以降の呼び出しは無視されます。

参照

- QAManagerBase.start メソッド [QAnywhere C++]449 ページ

triggerSendReceive メソッド

QAnywhere メッセージサーバーとの同期処理を発生させて、他のクライアント宛てのメッセージをアップロードし、ローカルクライアント宛てのメッセージをダウンロードします。

構文

```
public virtual qa_bool triggerSendReceive()
```

戻り値

処理が正常終了した場合のみ true。

備考

triggerSendReceive を呼び出すと、QAnywhere Agent と中央のメッセージングサーバー間でメッセージがただちに同期されます。手動の triggerSendReceive 呼び出しでは、QAnywhere Agent の転送ポリシーとは無関係に、メッセージ転送がただちに行われます。QAnywhere Agent の転送ポリシーは、メッセージ転送をどのようにして実行するかを決定します。たとえば、クライアントが Push 通知を受信した場合や、QAManagerBase::putMessage を呼び出してメッセージを送信した場合に、一定の間隔でメッセージ転送が自動的に実行されます。

詳細については、「クライアントにメッセージを転送するタイミングの決定」49 ページを参照してください。

参照

- QAManagerBase.putMessage メソッド [QAnywhere C++]441 ページ

QAManagerFactory クラス

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

構文

```
public class QAManagerFactory
```

メンバー

継承されたメンバーを含む QAManagerFactory クラスのすべてのメンバー。

名前	説明
createQAManager メソッド	指定されたプロパティを持つ新しい QAManager インスタンスを返します。
createQATransactionalManager メソッド	指定されたプロパティを持つ新しい QATransactionalManager インスタンスを返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
deleteQAManager メソッド	QAManager を破棄し、そのリソースを解放します。
deleteQATransactionalManager メソッド	QATransactionalManager インスタンスを破棄し、そのリソースを解放します。
getLastError メソッド	最後に実行された QAManagerFactory メソッドに関連付けられているエラーコードです。
getLastErrorMsg メソッド	最後に実行された QAManagerFactory メソッドに関連付けられているエラーテキスト。
getLastNativeError メソッド	最後に実行された QAManagerFactory メソッドに関連付けられているネイティブエラーコードです。

備考

QAManagerFactory のインスタンスは 1 つだけ持つことができます。

参照

- QAManager クラス [QAnywhere C++]411 ページ
- QATransactionalManager クラス [QAnywhere C++]484 ページ

createQAManager メソッド

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

構文

```
public virtual QAManager * createQAManager(qa_const_string iniFile)
```

パラメーター

- **iniFile**　プロパティファイルのパス。

戻り値

QAManager インスタンス。

備考

プロパティファイルパラメーターが NULL の場合、QAManager はデフォルトのプロパティを使用して作成されます。QAManager の作成後に QAManager::setProperty() メソッドを使用して、QAnywhere Manager のプロパティをプログラムで設定できます。

QAnywhere Manager の設定プロパティのリストについては、「QAnywhere Manager の設定プロパティ」85 ページを参照してください。

参照

- QAManager クラス [QAnywhere C++]411 ページ

createQATransactionalManager メソッド

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

構文

```
public virtual QATransactionalManager * createQATransactionalManager(
     qa_const_string iniFile
)
```

パラメーター

- **iniFile**　プロパティファイルのパス。

戻り値

QATransactionalManager インスタンス。

備考

プロパティファイルパラメーターが NULL の場合、QATransactionalManager はデフォルトのプロパティを使用して作成されます。QAManager の作成後に QATransactionalManager::setProperty() メソッドを使用して、QATransactionalManager のプロパティをプログラムで設定できます。

QAnywhere Manager の設定プロパティのリストについては、「QAnywhere Manager の設定プロパティ」85 ページを参照してください。

参照

- QATransactionalManager クラス [QAnywhere C++]484 ページ

deleteQAManager メソッド

QAManager を破棄し、そのリソースを解放します。

構文

```
public virtual void deleteQAManager(QAManager * mgr)
```

パラメーター

- **mgr**　破棄する QAManager インスタンス。

備考

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。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

詳細については、「QAnywhere の停止」84 ページを参照してください。

deleteQATransactionalManager メソッド

QATransactionalManager インスタンスを破棄し、そのリソースを解放します。

構文

```
public virtual void deleteQATransactionalManager(
     QATransactionalManager * mgr
)
```

パラメーター

- **mgr** 破棄する QATransactionalManager インスタンス。

備考

作成済みの QATransactionalManager インスタンスは QAnywhereFactory_term() が呼び出された時点ですべて破棄されるので、このメソッドを呼び出す必要はありません。ただし、すぐにリソースを解放する必要がある場合には、このメソッドを使用すると便利です。

詳細については、「QAnywhere の停止」84 ページを参照してください。

getLastError メソッド

最後に実行された QAManagerFactory メソッドに関連付けられているエラーコードです。

構文

```
public virtual qa_int getLastError()
```

戻り値

エラーコード。

備考

0 はエラーがないことを示します。

値のリストについては、QAError を参照してください。

参照

- QAManagerFactory.getLastNativeError メソッド [QAnywhere C++]454 ページ
- QAManagerFactory.getLastErrorMsg メソッド [QAnywhere C++]453 ページ

getLastErrorMsg メソッド

最後に実行された QAManagerFactory メソッドに関連付けられているエラーテキストです。

構文

```
public virtual qa_string getLastErrorMsg()
```

戻り値

エラーメッセージ。

備考

QAManagerFactory::getLastError が 0 を返した場合、このメソッドは NULL を返します。

このプロパティは、QAError をキャッチした後で取得できます。

参照

- QAManagerFactory.getLastError メソッド [QAnywhere C++]453 ページ
- QAManagerFactory.getLastNativeError メソッド [QAnywhere C++]454 ページ
- QAError クラス [QAnywhere C++]403 ページ

getLastNativeError メソッド

最後に実行された QAManagerFactory メソッドに関連付けられているネイティブエラーコードです。

構文

```
public virtual an_sql_code getLastNativeError()
```

戻り値

ネイティブのエラーコード。

備考

0 はエラーがないことを示します。

参照

- QAManagerFactory.getLastError メソッド [QAnywhere C++]453 ページ
- QAManagerFactory.getLastErrorMsg メソッド [QAnywhere C++]453 ページ

QAMessage クラス

QAMessage には、メッセージプロパティとヘッダーファイルを設定するためのインターフェイスがあります。

構文

```
public class QAMessage
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

派生クラス

- QABinaryMessage クラス [QAnywhere C++]388 ページ
- QATextMessage クラス [QAnywhere C++]478 ページ

メンバー

継承されたメンバーを含む QAMessage クラスのすべてのメンバー。

名前	説明
beginEnumPropertyNames メソッド	メッセージプロパティ名の列挙を開始します。
castToBinaryMessage メソッド	QAMessage を QABinaryMessage にキャストします。
castToTextMessage メソッド	QAMessage を QATextMessage にキャストします。
clearProperties メソッド	メッセージのプロパティをクリアします。
endEnumPropertyNames メソッド	メッセージプロパティ名の列挙に関連付けられているリソースを解放します。
getAddress メソッド	QAMessage インスタンスの送信先アドレスを取得します。
getBooleanProperty メソッド	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。
getByteProperty メソッド	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。
getDoubleProperty メソッド	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。
getExpiration メソッド	メッセージの有効期限を取得します。
getFloatProperty メソッド	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。
getInReplyToID メソッド	このメッセージが返信対象として指定されているメッセージの ID を取得します。
getIntProperty メソッド	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。
getLongProperty メソッド	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。

名前	説明
getMessageID メソッド	メッセージ ID を取得します。
getPriority メソッド	メッセージの優先度を取得します。
getPropertyType メソッド	指定された名前を持つプロパティの型を返します。
getRedelivered メソッド	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。
getReplyToAddress メソッド	メッセージの返信先アドレスを取得します。
getShortProperty メソッド	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。
getStringProperty メソッド	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。
getTimestamp メソッド	メッセージのタイムスタンプを取得します。
getTimestampAsString メソッド	メッセージのタイムスタンプをフォーマットされた string として取得します。
nextPropertyName メソッド	指定された列挙のメッセージプロパティ名を返します。該当するプロパティ名がない場合は -1 を返します。
propertyExists メソッド	プロパティ値が存在するかどうかを示します。
setAddress メソッド	メッセージの送信先アドレスを設定します。
setBooleanProperty メソッド	指定された名前を持つ qa_bool 型プロパティを指定された値に設定します。
setByteProperty メソッド	指定された名前を持つ qa_byte 型プロパティを指定された値に設定します。
setDoubleProperty メソッド	指定された名前を持つ qa_double 型プロパティを指定された値に設定します。
setFloatProperty メソッド	指定された名前を持つ qa_float 型プロパティを指定された値に設定します。
setInReplyToID メソッド	メッセージの In-Reply-To ID を設定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
setIntProperty メソッド	指定された名前を持つ qa_int 型プロパティを指定された値に設定します。
setLongProperty メソッド	指定された名前を持つ qa_long 型プロパティを指定された値に設定します。
setMessageID メソッド	メッセージ ID を設定します。
setPriority メソッド	メッセージの優先度レベルを設定します。
setRedelivered メソッド	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。
setReplyToAddress メソッド	メッセージの返信先アドレスを設定します。
setShortProperty メソッド	指定された名前を持つ qa_short 型プロパティを指定された値に設定します。
setStringProperty メソッド	指定された名前を持つ qa_string 型プロパティを指定された値に設定します。
setTimestamp メソッド	メッセージのタイムスタンプを設定します。
DEFAULT_PRIORITY 変数	デフォルトのメッセージ優先度です。
DEFAULT_TIME_TO_LIVE 変数	デフォルトのメッセージの存続時間です。

備考

派生クラスの QABinaryMessage と QATextMessage には、メッセージ本文の読み込み／書き込みを行うための特別なメソッドがあります。事前定義済みまたはカスタムのメッセージプロパティを設定するには、QAMessage のメソッドを使用します。

事前定義済みのプロパティ名のリストについては、MessageProperties を参照してください。

メッセージプロパティとヘッダーフィールドの設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

beginEnumPropertyNames メソッド

メッセージプロパティ名の列挙を開始します。

構文

```
public virtual qa_property_enum_handle beginEnumPropertyNames()
```

戻り値

nextPropertyName に渡されるハンドル。

備考

このメソッドから返されるハンドルは、nextPropertyName に渡されます。このメソッドと nextPropertyName を使用して、このメソッドが呼び出されたときのメッセージストアプロパティ名を列挙できます。メッセージプロパティは、beginEnumPropertyNames と endEnumPropertyNames の間では設定できません。

castToBinaryMessage メソッド

QAMessage を QABinaryMessage にキャストします。

構文

```
public virtual QABinaryMessage * castToBinaryMessage()
```

戻り値

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

備考

変換演算子を使用して、この QAMessage を QABinaryMessage に変換することもできます。

変換演算子を使用して QAMessage を QABinaryMessage に変換するには、次のようにします。

```
QAMessage *msg;
QABinaryMessage *bmsg;
...
bmsg = (QABinaryMessage *)(*msg);
```

castToTextMessage メソッド

QAMessage を QATextMessage にキャストします。

構文

```
public virtual QATextMessage * castToTextMessage()
```

戻り値

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

備考

変換演算子を使用して、QAMessage を QATextMessage に変換することもできます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

たとえば、変換演算子を使用して QAMessage を QATextMessage に変換するには、次のようにします。

```
QAMessage *msg;
QATextMessage *bmsg;
...
bmsg = (QATextMessage *)(*msg);
```

clearProperties メソッド

メッセージのプロパティをクリアします。

構文

```
public virtual void clearProperties()
```

備考

注意
メッセージのヘッダーフィールドと本文はクリアされません。

endEnumPropertyNames メソッド

メッセージプロパティ名の列挙に関連付けられているリソースを解放します。

構文

```
public virtual void endEnumPropertyNames(qa_property_enum_handle h)
```

パラメーター

- **h** beginEnumPropertyNames から返されるハンドル。

getAddress メソッド

QAMessage インスタンスの送信先アドレスを取得します。

構文

```
public virtual qa_const_string getAddress()
```

戻り値

送信先アドレス。

備考

このフィールドは、メッセージの送信時には無視されます。send メソッドが完了すると、このフィールドには QAManagerBase::putMessage() で指定された送信先アドレスが入ります。

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

getBooleanProperty メソッド

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。

構文

```
public virtual qa_bool getBooleanProperty(
     qa_const_string name,
    qa_bool * value
)
```

パラメーター

- **name** 取得するプロパティの名前。
- **value** qa_bool 値の格納先。

戻り値

処理が正常終了した場合のみ true。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere C++]377 ページ

getByteProperty メソッド

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。

構文

```
public virtual qa_bool getByteProperty(
     qa_const_string name,
    qa_byte * value
)
```

パラメーター

- **name** 取得するプロパティの名前。
- **value** qa_byte 値の格納先。

戻り値

処理が正常終了した場合のみ true。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere C++]377 ページ

getDoubleProperty メソッド

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。

構文

```
public virtual qa_bool getDoubleProperty(
     qa_const_string name,
    qa_double * value
)
```

パラメーター

- **name** 取得するプロパティの名前。
- **value** qa_double 値の格納先。

戻り値

処理が正常終了した場合のみ true。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere C++]377 ページ

getExpiration メソッド

メッセージの有効期限を取得します。

構文

```
public virtual qa_long getExpiration()
```

戻り値

有効期限。

備考

メッセージの Expiration ヘッダーフィールドは、メッセージの送信時には空のままです。send メソッドが完了すると、Expiration ヘッダーにはメッセージの有効期限が入ります。

メッセージの有効期限は、QAManagerBase::putMessageTimeToLive の存続時間の引数を現在の時間に追加することで設定されるため、このプロパティは読み込み専用です。

有効期限には、プラットフォームに合わせたフォーマットが使用されます。Windows/PocketPC プラットフォームの有効期限は SYSTEMTIME フォーマットです。これが FILETIME フォーマットに変換されて、qa_long 値にコピーされます。

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- QAMessage.getTimestamp メソッド [QAnywhere C++]468 ページ

getFloatProperty メソッド

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。

構文

```
public virtual qa_bool getFloatProperty(
     qa_const_string name,
    qa_float * value
)
```

パラメーター

- **name** 取得するプロパティの名前。
- **value** qa_float 値の格納先。

戻り値

処理が正常終了した場合のみ true。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere C++]377 ページ

getInReplyToID メソッド

このメッセージが返信対象として指定されているメッセージの ID を取得します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
public virtual qa_const_string getInReplyToID()
```

戻り値

In-Reply-To ID。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

getIntProperty メソッド

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。

構文

```
public virtual qa_bool getIntProperty(
     qa_const_string name,
    qa_int * value
)
```

パラメーター

- **name** 取得するプロパティの名前。
- **value** qa_int 値の格納先。

戻り値

処理が正常終了した場合のみ true。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere C++]377 ページ

getLongProperty メソッド

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。

構文

```
public virtual qa_bool getLongProperty(
     qa_const_string name,
    qa_long * value
)
```

パラメーター

- **name** 取得するプロパティの名前。
- **value** qa_long 値の格納先。

戻り値

処理が正常終了した場合のみ true。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere C++]377 ページ

getMessageID メソッド

メッセージ ID を取得します。

構文

```
public virtual qa_const_string getMessageID()
```

戻り値

メッセージ ID です。

備考

MessageID ヘッダーフィールドには、QAnywhere クライアントによって送信された各メッセージを一意に識別する値が格納されています。

QAManagerBase::putMessage メソッドを使用してメッセージが送信されると、MessageID ヘッダーが NULL になるため無視できます。send メソッドが戻ると、割り当てられた値がここに格納されます。

MessageID は、履歴レポジトリ内のメッセージを識別するためのユニークなキーとして使用できる qa_string 値です。

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

getPriority メソッド

メッセージの優先度を取得します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
public virtual qa_int getPriority()
```

戻り値

メッセージの優先度。

備考

QAnywhere クライアント API では、10 レベルの優先度が定義されています。0 が最低の優先度、9 が最高の優先度を表します。クライアントは、優先度 0 ～ 4 を通常のメッセージ、優先度 5 ～ 9 を緊急度の高いメッセージとみなす必要があります。

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

getPropertyType メソッド

指定された名前を持つプロパティの型を返します。

構文

```
public virtual qa_short getPropertyType(qa_const_string name)
```

パラメーター

- **name** プロパティ名。

戻り値

プロパティの型。

備考

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。

getRedelivered メソッド

受信されたが受信確認されていないメッセージであるかどうかを示します。

構文

```
public virtual qa_bool getRedelivered()
```

戻り値

メッセージが再配信された場合のみ true。

備考

受信側の QAManager は、受信中のメッセージが以前に受信されたことを検知した場合に、Redelivered ヘッダーを設定します。

たとえば、AcknowledgementMode::EXPLICIT_ACKNOWLEDGEMENT でオープンされた QAManager を使用してアプリケーションがメッセージを受信し、メッセージの受信確認を行わずに終了したとします。アプリケーションが次の起動時に同じメッセージを受信すると、Redelivered ヘッダーが true になります。

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

getReplyToAddress メソッド

メッセージの返信先アドレスを取得します。

構文

```
public virtual qa_const_string getReplyToAddress()
```

戻り値

返信先アドレス。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

getShortProperty メソッド

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。

構文

```
public virtual qa_bool getShortProperty(
     qa_const_string name,
    qa_short * value
)
```

パラメーター

- **name** 取得するプロパティの名前。
- **value** qa_short 値の格納先。

戻り値

処理が正常終了した場合のみ true。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere C++]377 ページ

getStringProperty メソッド

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。

オーバーロードリスト

名前	説明
getStringProperty(qa_const_string, qa_int, qa_string, qa_int) メソッド	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。
getStringProperty(qa_const_string, qa_string, qa_int) メソッド	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。

getStringProperty(qa_const_string, qa_int, qa_string, qa_int) メソッド

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。

構文

```
public virtual qa_int getStringProperty(
     qa_const_string name,
    qa_int offset,
    qa_string dest,
    qa_int maxlen
)
```

パラメーター

- **name** 取得するプロパティの名前。
- **offset** プロパティ値内でのオフセット (コピー開始位置)。
- **dest** qa_string 値の格納先。
- **maxlen** コピーする値の qa_char の最大数。この値には、NULL ターミネーターの qa_char も含まれます。

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。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere C++]377 ページ

getStringProperty(qa_const_string, qa_string, qa_int) メソッド

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。

構文

```
public virtual qa_int getStringProperty(
     qa_const_string name,
    qa_string dest,
    qa_int maxlen
)
```

パラメーター

- **name**　取得するプロパティの名前。
- **dest**　qa_string 値の格納先。
- **maxlen**　コピーする値の qa_char の最大数。この値には、null ターミネーターの qa_char も含まれます。

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。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere C++]377 ページ

getTimestamp メソッド

メッセージのタイムスタンプを取得します。

構文

```
public virtual qa_long getTimestamp()
```

戻り値

メッセージのタイムスタンプ。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

メッセージの Timestamp ヘッダーフィールドには、メッセージが作成された時刻が格納されます。これは、協定世界時 (UTC: Coordinated Universal Time) です。

この時刻は、メッセージが実際に送信された時刻ではないので注意してください。メッセージの実際の送信時刻は、トランザクションの進行状況やクライアント側のキューに登録されているその他のメッセージの影響で、作成時刻よりも遅れる可能性があります。タイムスタンプには、プラットフォームに合わせたフォーマットが使用されます。Windows/PocketPC プラットフォームのタイムスタンプは SYSTEMTIME フォーマットです。これが FILETIME フォーマットに変換されて、qa_long 値にコピーされます。

タイムスタンプ ts を SYSTEMTIME に変換してユーザーに表示するには、次のコードを実行します。

```
SYSTEMTIME stime;
FILETIME   ftime;
ULARGE_INTEGER time;
time.QuadPart = ts;
memcpy(&ftime, &time, sizeof(FILETIME));
FileTimeToSystemTime(&ftime, &stime);
```

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

getTimestampAsString メソッド

メッセージのタイムスタンプをフォーマットされた string として取得します。

構文

```
public virtual qa_int getTimestampAsString(
     qa_string buffer,
    qa_int bufferLen
)
```

パラメーター

- **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ー。
- **bufferLen**　バッファーのサイズ。

戻り値

バッファーに書き込まれた NULL 以外の qa_char の数。

備考

フォーマットは、"dow, MMM dd, yyyy hh:mm:ss.nnn GMT" です。

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

nextPropertyName メソッド

指定された列挙のメッセージプロパティ名を返します。該当するプロパティ名がない場合は -1 を返します。

構文

```
public virtual qa_int nextPropertyName(
     qa_property_enum_handle h,
    qa_string buffer,
    qa_int bufferLen
)
```

パラメーター

- **h**　beginEnumPropertyNames から返されるハンドル。
- **buffer**　プロパティ名が書き込まれるバッファー。
- **bufferLen**　プロパティ名を格納するバッファーの長さ。この長さには、NULL ターミネーターのスペースも含める必要があります。

戻り値

プロパティ名の長さ。プロパティ名がない場合は -1。

propertyExists メソッド

プロパティ値が存在するかどうかを示します。

構文

```
public virtual qa_bool propertyExists(qa_const_string name)
```

パラメーター

- **name**　プロパティ名。

戻り値

プロパティが存在する場合のみ true。

setAddress メソッド

メッセージの送信先アドレスを設定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
public virtual void setAddress(qa_const_string destination)
```

パラメーター

- **destination**　送信先アドレス。

備考

このメソッドを使用すると、受信したメッセージにすでに設定されている値を変更できます。

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

setBooleanProperty メソッド

指定された名前を持つ qa_bool 型プロパティを指定された値に設定します。

構文

```
public virtual void setBooleanProperty(
     qa_const_string name,
    qa_bool value
)
```

パラメーター

- **name**　設定するプロパティの名前。
- **value**　プロパティの qa_bool 値。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere C++]377 ページ

setByteProperty メソッド

指定された名前を持つ qa_byte 型プロパティを指定された値に設定します。

構文

```
public virtual void setByteProperty(qa_const_string name, qa_byte value)
```

パラメーター

- **name**　設定するプロパティの名前です。
- **value**　プロパティの qa_byte 値。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere C++]377 ページ

setDoubleProperty メソッド

指定された名前を持つ qa_double 型プロパティを指定された値に設定します。

構文

```
public virtual void setDoubleProperty(
     qa_const_string name,
    qa_double value
)
```

パラメーター

- **name** 設定するプロパティの名前です。
- **value** プロパティの qa_double 値です。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere C++]377 ページ

setFloatProperty メソッド

指定された名前を持つ qa_float 型プロパティを指定された値に設定します。

構文

```
public virtual void setFloatProperty(
     qa_const_string name,
    qa_float value
)
```

パラメーター

- **name** 設定するプロパティの名前です。
- **value** プロパティの qa_float 値です。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere C++]377 ページ

setInReplyToID メソッド

メッセージの In-Reply-To ID を設定します。

構文

```
public virtual void setInReplyToID(qa_const_string id)
```

パラメーター

- **id** In-Reply-To ID です。

備考

クライアントは、In-Reply-To ID ヘッダーフィールドを使用してメッセージ間リンクを設定できます。これは、応答メッセージをそれに対応する要求メッセージとリンクさせる場合によく使用されます。

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

setIntProperty メソッド

指定された名前を持つ qa_int 型プロパティを指定された値に設定します。

構文

```
public virtual void setIntProperty(qa_const_string name, qa_int value)
```

パラメーター

- **name** 設定するプロパティの名前です。
- **value** プロパティの qa_int 値です。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere C++]377 ページ

setLongProperty メソッド

指定された名前を持つ qa_long 型プロパティを指定された値に設定します。

構文

```
public virtual void setLongProperty(qa_const_string name, qa_long value)
```

パラメーター

- **name** 設定するプロパティの名前です。
- **value** プロパティの qa_long 値です。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere C++]377 ページ

setMessageID メソッド

メッセージ ID を設定します。

構文

```
public virtual void setMessageID(qa_const_string id)
```

パラメーター

- **id** メッセージ ID です。

備考

このメソッドを使用すると、受信したメッセージにすでに設定されている値を変更できます。

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

setPriority メソッド

メッセージの優先度レベルを設定します。

構文

```
public virtual void setPriority(qa_int priority)
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- **priority**　メッセージの優先度。

備考

このメソッドを使用すると、受信したメッセージにすでに設定されている値を変更できます。

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

setRedelivered メソッド

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。

構文

```
public virtual void setRedelivered(qa_bool redelivered)
```

パラメーター

- **redelivered**　再配信されたかどうかを示す値。

備考

このメソッドを使用すると、受信したメッセージにすでに設定されている値を変更できます。

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

setReplyToAddress メソッド

メッセージの返信先アドレスを設定します。

構文

```
public virtual void setReplyToAddress(qa_const_string replyTo)
```

パラメーター

- **replyTo**　返信先アドレス。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

setShortProperty メソッド

指定された名前を持つ qa_short 型プロパティを指定された値に設定します。

構文

```
public virtual void setShortProperty(
     qa_const_string name,
    qa_short value
)
```

パラメーター

- **name** 設定するプロパティの名前です。
- **value** プロパティの qa_short 値。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere C++]377 ページ

setStringProperty メソッド

指定された名前を持つ qa_string 型プロパティを指定された値に設定します。

構文

```
public virtual void setStringProperty(
     qa_const_string name,
    qa_const_string value
)
```

パラメーター

- **name** 設定するプロパティの名前です。
- **value** プロパティの qa_string 値。

備考

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- MessageProperties クラス [QAnywhere C++]377 ページ

setTimestamp メソッド

メッセージのタイムスタンプを設定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
public virtual void setTimestamp(qa_long timestamp)
```

パラメーター

- **timestamp** メッセージのタイムスタンプ。これは協定世界時 (UTC: Coordinated Universal Time) です。

備考

このメソッドを使用すると、受信したメッセージにすでに設定されている値を変更できます。

メッセージヘッダーとプロパティの取得と設定の詳細については、「QAnywhere メッセージ」15 ページを参照してください。

参照

- QAMessage.getTimestamp メソッド [QAnywhere C++]468 ページ

DEFAULT_PRIORITY 変数

デフォルトのメッセージ優先度です。

構文

```
public static const qa_int DEFAULT_PRIORITY;
```

備考

この値は 4 です。優先度の値が 0 ～ 4 は通常のメッセージ、値 5 ～ 9 は緊急度の高いメッセージです。

DEFAULT_TIME_TO_LIVE 変数

デフォルトのメッセージの存続時間です。

構文

```
public static const qa_long DEFAULT_TIME_TO_LIVE;
```

備考

この値は 0 で、メッセージの有効期限がないことを示します。

QAMessageListener クラス

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。

構文

```
public class QAMessageListener
```

メンバー

継承されたメンバーを含む QAMessageListener クラスのすべてのメンバー。

名前	説明
QAMessageListener デコンストラクター	仮想デストラクターです。
onMessage メソッド	メッセージをリスナーに渡します。

QAMessageListener デコンストラクター

仮想デストラクターです。

構文

```
public virtual ~QAMessageListener()
```

onMessage メソッド

メッセージをリスナーに渡します。

構文

```
public virtual void onMessage(QAMessage * message)
```

パラメーター

- **message**　リスナーに渡されるメッセージ。

QATextMessage クラス

QATextMessage は QAMessage クラスを継承したもので、メッセージ本文にテキストが追加されます。

構文

```
public class QATextMessage : QAMessage
```

基本クラス

- QAMessage クラス [QAnywhere C++]454 ページ

メンバー

継承されたメンバーを含む QATextMessage クラスのすべてのメンバー。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
beginEnumPropertyNames メソッド	メッセージプロパティ名の列挙を開始します。
castToBinaryMessage メソッド	QAMessage を QABinaryMessage にキャストします。
castToTextMessage メソッド	QAMessage を QATextMessage にキャストします。
clearProperties メソッド	メッセージのプロパティをクリアします。
endEnumPropertyNames メソッド	メッセージプロパティ名の列挙に関連付けられているリソースを解放します。
getAddress メソッド	QAMessage インスタンスの送信先アドレスを取得します。
getBooleanProperty メソッド	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。
getByteProperty メソッド	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。
getDoubleProperty メソッド	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。
getExpiration メソッド	メッセージの有効期限を取得します。
getFloatProperty メソッド	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。
getInReplyToID メソッド	このメッセージが返信対象として指定されているメッセージの ID を取得します。
getIntProperty メソッド	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。
getLongProperty メソッド	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。
getMessageID メソッド	メッセージ ID を取得します。
getPriority メソッド	メッセージの優先度を取得します。
getPropertyType メソッド	指定された名前を持つプロパティの型を返します。

名前	説明
getRedelivered メソッド	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。
getReplyToAddress メソッド	メッセージの返信先アドレスを取得します。
getShortProperty メソッド	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。
getStringProperty メソッド	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。
getText メソッド	メッセージのデータが格納されている string を取得します。
getTextLength メソッド	テキスト長を返します。
getTimestamp メソッド	メッセージのタイムスタンプを取得します。
getTimestampAsString メソッド	メッセージのタイムスタンプをフォーマットされた string として取得します。
nextPropertyName メソッド	指定された列挙のメッセージプロパティ名を返します。該当するプロパティ名がない場合は -1 を返します。
propertyExists メソッド	プロパティ値が存在するかどうかを示します。
readText メソッド	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ー内に読み込みます。
reset メソッド	現在のテキスト位置を先頭に戻します。
setAddress メソッド	メッセージの送信先アドレスを設定します。
setBooleanProperty メソッド	指定された名前を持つ qa_bool 型プロパティを指定された値に設定します。
setByteProperty メソッド	指定された名前を持つ qa_byte 型プロパティを指定された値に設定します。
setDoubleProperty メソッド	指定された名前を持つ qa_double 型プロパティを指定された値に設定します。
setFloatProperty メソッド	指定された名前を持つ qa_float 型プロパティを指定された値に設定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
setInReplyToID メソッド	メッセージの In-Reply-To ID を設定します。
setIntProperty メソッド	指定された名前を持つ qa_int 型プロパティを指定された値に設定します。
setLongProperty メソッド	指定された名前を持つ qa_long 型プロパティを指定された値に設定します。
setMessageID メソッド	メッセージ ID を設定します。
setPriority メソッド	メッセージの優先度レベルを設定します。
setRedelivered メソッド	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。
setReplyToAddress メソッド	メッセージの返信先アドレスを設定します。
setShortProperty メソッド	指定された名前を持つ qa_short 型プロパティを指定された値に設定します。
setStringProperty メソッド	指定された名前を持つ qa_string 型プロパティを指定された値に設定します。
setText メソッド	メッセージのデータが格納されている string を設定します。
setTimestamp メソッド	メッセージのタイムスタンプを設定します。
writeText メソッド	現在のテキストに、テキストを連結します。
DEFAULT_PRIORITY 変数	デフォルトのメッセージ優先度です。
DEFAULT_TIME_TO_LIVE 変数	デフォルトのメッセージの存続時間です。

備考

QATextMessage には、メッセージ本文からのテキストの読み込み／書き込みを行うためのメソッドがあります。

メッセージが最初に作成された時点では、メッセージ本文は書き込み専用モードになっています。メッセージ送信元のクライアントは、メッセージを送信した後、そのメッセージを保持し変更できます。ただし、それによって、送信されたメッセージが変更されることはありません。同じメッセージオブジェクトを複数回送信できます。

メッセージが受信された時点でプロバイダーは QATextMessage::reset を呼び出します。これによりメッセージ本文が読み込み専用モードになり、メッセージ本文の先頭から値の読み込みが開始されます。クライアントが読み込み専用モードのメッセージに書き込もうとすると、COMMON_MSG_NOT_WRITEABLE_ERROR が設定されます。

参照

- QABinaryMessage クラス [QAnywhere C++]388 ページ

getText メソッド

メッセージのデータが格納されている string を取得します。

構文

```
public virtual qa_string getText()
```

戻り値

メッセージのデータが格納されている string。

備考

デフォルト値は NULL です。

メッセージのサイズが、QAManager::MAX_IN_MEMORY_MESSAGE_SIZE プロパティで指定された最大サイズを超える場合、この関数は NULL を返します。この場合は、QATextMessage::readText メソッドを使用してテキストを読み込みます。

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

getTextLength メソッド

テキスト長を返します。

構文

```
public virtual qa_long getTextLength()
```

戻り値

テキスト長。

備考

注意
テキスト長がゼロ以外であるのに getText() が qa_null を返す場合があります。これは、そのテキストが大きすぎてメモリに収まらないためです。この場合、readText を使用して、テキストを分割して読み込みます。

readText メソッド

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ー内に読み込みます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
public virtual qa_int readText(qa_string string, qa_int length)
```

パラメーター

- **string** テキストの格納先。
- **length** 格納先バッファーに読み込む qa_char の最大数 (NULL 終了文字も含む)。

戻り値

実際に読み込まれた NULL 以外の qa_char の数。テキストストリーム全体が読み込まれた場合は -1。

reset メソッド

現在のテキスト位置を先頭に戻します。

構文

```
public virtual void reset()
```

setText メソッド

メッセージのデータが格納されている string を設定します。

構文

```
public virtual void setText(qa_const_string string)
```

パラメーター

- **string** 設定するメッセージデータが含まれている文字列。

writeText メソッド

現在のテキストに、テキストを連結します。

構文

```
public virtual void writeText(
     qa_const_string string,
    qa_int offset,
    qa_int length
)
```

パラメーター

- **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。

QATransactionalManager クラス

このクラスは、トランザクション指向メッセージング用のマネージャーです。

構文

```
public class QATransactionalManager : QAManagerBase
```

基本クラス

- QAManagerBase クラス [QAnywhere C++]418 ページ

メンバー

継承されたメンバーを含む QATransactionalManager クラスのすべてのメンバー。

名前	説明
beginEnumStorePropertyNames メソッド	メッセージストアプロパティ名の列挙を開始します。
browseClose メソッド	参照操作に関連付けられているリソースを解放します。
browseMessages メソッド	メッセージストアのキューに登録されているメッセージの参照を開始します。
browseMessagesByID メソッド	メッセージストアのキューに登録されているメッセージのうち、指定されたメッセージ ID を持つメッセージの参照を開始します。
browseMessagesByQueue メソッド	メッセージストアのキューに登録されているメッセージのうち、指定されたキューのメッセージの参照を開始します。
browseMessagesBySelector メソッド	メッセージストアのキューに登録されているメッセージのうち、指定されたセレクターを満たすメッセージの参照を開始します。
browseNextMessage メソッド	指定された参照操作の次のメッセージを返します。メッセージがない場合は NULL を返します。
cancelMessage メソッド	指定されたメッセージ ID を持つメッセージをキャンセルします。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
close メソッド	QAnywhere メッセージシステムへの接続をクローズして、QAManagerBase で使用していたリソースをすべて解放します。
commit メソッド	現在のトランザクションをコミットし、新しいトランザクションを開始します。
createBinaryMessage メソッド	QABinaryMessage インスタンスを作成します。
createTextMessage メソッド	QATextMessage インスタンスを作成します。
deleteMessage メソッド	QAMessage オブジェクトを削除します。
endEnumStorePropertyNames メソッド	メッセージストアプロパティ名の列挙に関連付けられているリソースを解放します。
getAllQueueDepth メソッド	指定されたフィルターに基づいて、すべてのキューの長さの合計を返します。
getBooleanStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの boolean 値を取得します。
getByteStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの byte 値を取得します。
getDoubleStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの double 値を取得します。
getFloatStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの float 値を取得します。
getIntStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの int 値を取得します。
getLastError メソッド	最後に実行された QAManagerBase メソッドに関連付けられているエラーコードです。
getLastErrorMsg メソッド	最後に実行された QAManagerBase メソッドに関連付けられているエラーテキストです。
getLastNativeError メソッド	最後に実行された QAManagerBase メソッドに関連付けられているネイティブエラーコードです。
getLongStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの long 値を取得します。

名前	説明
getMessage メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
getMessageBySelector メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
getMessageBySelectorNoWait メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
getMessageBySelectorTimeout メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
getMessageNoWait メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
getMessageTimeout メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
getMode メソッド	受信したメッセージの QAManager 受信確認モードを返します。
getQueueDepth メソッド	指定されたフィルターに基づいて、キューの長さを返します。
getShortStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの short 値を取得します。
getStringStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの string 値を取得します。
nextStorePropertyName メソッド	指定された列挙のメッセージストアプロパティ名を返します。
open メソッド	QATransactionalManager インスタンスをオープンします。
putMessage メソッド	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。
putMessageTimeToLive メソッド	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。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
rollback メソッド	現在のトランザクションをロールバックし、新しいトランザクションを開始します。
setBooleanStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを boolean 値に設定します。
setByteStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを byte 値に設定します。
setDoubleStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを double 値に設定します。
setFloatStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを float 値に設定します。
setIntStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを int 値に設定します。
setLongStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを long 値に設定します。
setMessageListener メソッド	QAnywhere メッセージを非同期的に受信するようにメッセージリスナークラスを設定します。
setMessageListenerBySelector メソッド	メッセージセレクターを使用して、QAnywhere メッセージを非同期的に受信するように、メッセージリスナークラスを設定します。
setProperty メソッド	QAnywhere Manager の設定プロパティをプログラムで設定できるようにします。
setShortStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを short 値に設定します。
setStringStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを string 値に設定します。
start メソッド	着信メッセージをメッセージリスナーで受信するための QAManagerBase を起動します。
stop メソッド	QAManagerBase による着信メッセージの受信を停止します。

名前	説明
triggerSendReceive メソッド	QAnywhere メッセージサーバーとの同期処理を発生させて、他のクライアント宛てのメッセージをアップロードし、ローカルクライアント宛てのメッセージをダウンロードします。

備考

QATransactionalManager クラスは QAManagerBase から派生し、トランザクション指向の QAnywhere メッセージング操作を管理します。

この動作の完全な説明については、QAManagerBase を参照してください。

QATransactionalManager は、トランザクション指向の受信確認でのみ使用できます。QAManagerBase::putMessage() と QAManagerBase::getMessage() のすべての呼び出しをコミットする場合は、QATransactionalManager::commit() メソッドを使用します。

詳細については、「トランザクション指向メッセージングの実装」66 ページを参照してください。

参照

● QATransactionalManager クラス [QAnywhere C++]484 ページ

commit メソッド

現在のトランザクションをコミットし、新しいトランザクションを開始します。

構文

```
public virtual qa_bool commit()
```

戻り値

commit 処理が正常終了した場合のみ true。

備考

このメソッドは、QAManagerBase::putMessage() と QAManagerBase::getMessage() のすべての呼び出しをコミットします。

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

参照

● QATransactionalManager クラス [QAnywhere C++]484 ページ

open メソッド

QATransactionalManager インスタンスをオープンします。

構文

```
public virtual qa_bool open()
```

戻り値

処理が正常終了した場合のみ true。

備考

open メソッドは、Manager を作成した後、最初に呼び出す必要があるメソッドです。

参照

- QATransactionalManager クラス [QAnywhere C++]484 ページ

rollback メソッド

現在のトランザクションをロールバックし、新しいトランザクションを開始します。

構文

```
public virtual qa_bool rollback()
```

戻り値

open 処理が正常終了した場合のみ true。

備考

このメソッドは、コミットされていない QAManagerBase::putMessage() と QAManagerBase::getMessage() のすべての呼び出しをロールバックします。

参照

- QATransactionalManager クラス [QAnywhere C++]484 ページ

QueueDepthFilter クラス

QAManagerBase のキューの長さ関連メソッドの QueueDepthFilter の値です。

構文

```
public class QueueDepthFilter
```

メンバー

継承されたメンバーを含む QueueDepthFilter クラスのすべてのメンバー。

名前	説明
ALL 変数	着信メッセージと送信メッセージの両方をカウントします。
INCOMING 変数	着信メッセージのみカウントします。
LOCAL 変数	ローカルメッセージのみカウントします。
OUTGOING 変数	送信メッセージのみカウントします。

ALL 変数

着信メッセージと送信メッセージの両方をカウントします。

構文

```
public static const qa_short ALL;
```

備考

システムメッセージと期限切れのメッセージは、キューの長さカウントに計上されません。

INCOMING 変数

着信メッセージのみカウントします。

構文

```
public static const qa_short INCOMING;
```

備考

着信メッセージは、発信者がメッセージストアのエージェント ID ではないメッセージです。

LOCAL 変数

ローカルメッセージのみカウントします。

構文

```
public static const qa_short LOCAL;
```

備考

ローカルメッセージは、発信者と送信先がメッセージストアのエージェント ID であるメッセージです。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

OUTGOING 変数

送信メッセージのみカウントします。

構文

```
public static const qa_short OUTGOING;
```

備考

送信メッセージは、発信者がメッセージストアのエージェント ID であり、送信先がメッセージストアのエージェント ID ではないメッセージです。

StatusCodes クラス

このインターフェイスは、メッセージのステータスコードセットを定義します。

構文

```
public class StatusCodes
```

メンバー

継承されたメンバーを含む StatusCodes クラスのすべてのメンバー。

名前	説明
CANCELED 変数	メッセージはキャンセルされました。
EXPIRED 変数	メッセージの有効期限が切れました。メッセージは、有効期限になる前に受信されませんでした。
FINAL 変数	この定数は、メッセージが最終ステータスになったかどうかを示します。
LOCAL 変数	メッセージはローカルメッセージストア宛てに送信され、サーバーに送信されません。
PENDING 変数	メッセージは送信されましたが、まだ受信および受信確認されていません。
RECEIVED 変数	メッセージが受信され、受信者によって受信確認されました。
RECEIVING 変数	メッセージは受信中であるか、または受信されましたがまだ確認されていません。
TRANSMITTED 変数	メッセージはサーバーに送信されました。

名前	説明
TRANSMITTING 変数	メッセージはサーバーに送信中です。
UNRECEIVABLE 変数	メッセージは受信不可のマークが付けられています。
UNTRANSMITTED 変数	メッセージはサーバーに送信されていません。

CANCELED 変数

メッセージはキャンセルされました。

構文

```
public static const qa_int CANCELED;
```

備考

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

EXPIRED 変数

メッセージの有効期限が切れました。メッセージは、有効期限になる前に受信されませんでした。

構文

```
public static const qa_int EXPIRED;
```

備考

このコードの値は 30 です。このコードは MessageProperties::STATUS で使用されます。

FINAL 変数

この定数は、メッセージが最終ステータスになったかどうかを示します。

構文

```
public static const qa_int FINAL;
```

備考

メッセージは、そのステータスがこの定数より大きい場合にのみ最終ステータスになったと見なされます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

このコードの値は 20 です。このコードは MessageProperties::STATUS で使用されます。

LOCAL 変数

メッセージはローカルメッセージストア宛てに送信され、サーバーに送信されません。

構文

```
public static const qa_int LOCAL;
```

備考

このコードの値は 2 です。このコードは MessageProperties::TRANSMISSION_STATUS で使用されます。

PENDING 変数

メッセージは送信されましたが、まだ受信および受信確認されていません。

構文

```
public static const qa_int PENDING;
```

備考

このコードの値は 1 です。このコードは MessageProperties::STATUS で使用されます。

RECEIVED 変数

メッセージが受信され、受信者によって受信確認されました。

構文

```
public static const qa_int RECEIVED;
```

備考

このコードの値は 60 です。このコードは MessageProperties::STATUS で使用されます。

RECEIVING 変数

メッセージは受信中であるか、または受信されましたがまだ確認されていません。

構文

```
public static const qa_int RECEIVING;
```

備考

このコードの値は 10 です。このコードは MessageProperties::STATUS で使用されます。

TRANSMITTED 変数

メッセージはサーバーに送信されました。

構文

```
public static const qa_int TRANSMITTED;
```

備考

このコードの値は 1 です。このコードは MessageProperties::TRANSMISSION_STATUS で使用されます。

TRANSMITTING 変数

メッセージはサーバーに送信中です。

構文

```
public static const qa_int TRANSMITTING;
```

備考

このコードの値は 3 です。このコードは MessageProperties::TRANSMISSION_STATUS で使用されます。

UNRECEIVABLE 変数

メッセージは受信不可のマークが付けられています。

構文

```
public static const qa_int UNRECEIVABLE;
```

備考

メッセージに間違った形式があるか、失敗した試行回数が多すぎて配信されませんでした。

このコードの値は 50 です。このコードは MessageProperties::STATUS で使用されます。

UNTRANSMITTED 変数

メッセージはサーバーに送信されていません。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
public static const qa_int UNTRANSMITTED;
```

備考

このコードの値は 0 です。このコードは MessageProperties::TRANSMISSION_STATUS で使用されます。

クライアント用 QAnywhere Java API リファレンス

パッケージ (通常のクライアントの場合)

```
ianywhere.qanywhere.client
```

パッケージ (スタンドアロンクライアントの場合)

```
ianywhere.qanywhere.standaloneclient
```

AcknowledgementMode インターフェイス

QAnywhere クライアントアプリケーションによってどのようにメッセージが確認されるかを示します。

構文

```
public interface AcknowledgementMode
```

メンバー

継承されたメンバーを含む AcknowledgementMode インターフェイスのすべてのメンバー。

名前	説明
EXPLICIT_ACKNOWLEDGEMENT 変数	受信メッセージは、QAManager のいずれかの受信確認メソッドを使用して確認されます。
IMPLICIT_ACKNOWLEDGEMENT 変数	すべてのメッセージは、クライアントアプリケーションで受信されるとすぐに受信確認されます。
TRANSACTIONAL 変数	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。

備考

暗黙的または明示的な受信確認モードは、QAManager.open(short) メソッドを使用して、QAManager インスタンスに割り当てられます。

暗黙的な受信確認では、メッセージは、クライアントアプリケーションで受信されるとすぐに受信確認されます。明示的な受信確認では、いずれかの QAManager 受信確認メソッドを呼び出してください。すべてのステータス変更は、サーバーによってクライアント間で伝達されます。

参照

- QAManager インターフェイス [QAnywhere Java]536 ページ
- QATransactionalManager インターフェイス [QAnywhere Java]609 ページ
- QAManagerBase インターフェイス [QAnywhere Java]544 ページ

EXPLICIT_ACKNOWLEDGEMENT 変数

受信メッセージは、QAManager のいずれかの受信確認メソッドを使用して確認されます。

構文

```
final short AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT
```

参照

- QAManager インターフェイス [QAnywhere Java]536 ページ

IMPLICIT_ACKNOWLEDGEMENT 変数

すべてのメッセージは、クライアントアプリケーションで受信されるとすぐに受信確認されます。

構文

```
final short AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT
```

備考

メッセージを同期的に受信する場合、メッセージは QAManagerBase.getMessage(String) メソッドが返されるとすぐに受信確認されます。メッセージを非同期的に受信する場合、メッセージはイベント処理メソッドが返されるとすぐに受信確認されます。

参照

- QAManagerBase.getMessage メソッド [QAnywhere Java]556 ページ

TRANSACTIONAL 変数

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。

構文

```
final short AcknowledgementMode.TRANSACTIONAL
```

備考

このモードは、QATransactionalManager インスタンスに自動的に割り当てられます。

参照

- QATransactionalManager インターフェイス [QAnywhere Java]609 ページ

MessageProperties インターフェイス

標準のメッセージプロパティ名を格納するフィールドを提供します。

構文

```
public interface MessageProperties
```

メンバー

継承されたメンバーを含む MessageProperties インターフェイスのすべてのメンバー。

名前	説明
ADAPTER 変数	"system" キューメッセージの場合は、QAnywhere サーバーへの接続で使用されているネットワークアダプターです。
ADAPTERS 変数	このプロパティ名は、QAnywhere サーバーへの接続で使用できるネットワークアダプターのデリミター付きリストを示します。
DELIVERY_COUNT 変数	このプロパティ名は、メッセージを配信するために、これまでに実行された試行回数を示します。
IP 変数	"system" キューメッセージの場合は、QAnywhere サーバーへの接続で使用されているネットワークアダプターの IP アドレスです。
MAC 変数	"system" キューメッセージの場合は、QAnywhere サーバーへの接続で使用されているネットワークアダプターの MAC アドレスです。
MSG_TYPE 変数	このプロパティ名は、QAnywhere メッセージに関連付けられている MessageType 列挙値を示します。

名前	説明
NETWORK_STATUS 変数	このプロパティ名は、ネットワーク接続のステータスを示します。
ORIGINATOR 変数	このプロパティ名は、メッセージの発信者のメッセージストア ID を示します。
RAS 変数	"system" キューメッセージの場合は、QAnywhere サーバーへの接続で使用されている RAS エントリ名です。
RASNAMES 変数	"system" キューメッセージの場合は、QAnywhere サーバーに接続するときに使用できる RAS エントリ名のデリミター付きリストです。
STATUS 変数	このプロパティ名は、メッセージの現在のステータスを示します。
STATUS_TIME 変数	このプロパティ名は、メッセージが現在のステータスになったと判断された時刻を示します。
TRANSMISSION_STATUS 変数	このプロパティ名は、メッセージの現在の転送ステータスを示します。

備考

MessageProperties クラスは、標準のメッセージプロパティ名を提供します。MessageProperties フィールドは、メッセージプロパティの取得と設定で使用する QAMessage メソッドに渡すことができます。

たとえば、次の QAMessage インスタンスがあるとします。

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

次の例では、QAMessage.getIntProperty(String) メソッドを使用して MessageProperties.MSG_TYPE に対応する値を取得します。MessageType 列挙体は、int 値の結果を適切なメッセージタイプにマッピングします。

```
int msg_type = t_msg.getIntProperty( MessageProperties.MSG_TYPE );
```

次の例は、QAnywhere システムメッセージの処理で使用される onSystemMessage(QAMessage) メソッドを示します。

メッセージタイプは、MessageProperties.MSG_TYPE 変数と QAMessage.getIntProperty(String) メソッドを使用して評価されます。

MessageProperties.RASNAMES と QAMessage.getStringProperty(String) メソッドを使用して、RAS エントリ名のリストが取得されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
private void onSystemMessage(QAMessage msg) {
  QATextMessage    t_msg;
  int              msg_type;
  String           network_adapters;
  String           network_names;
  String           network_info;

  t_msg = (QATextMessage)msg;
  if( t_msg != null ) {
     // Evaluate the message type.
     msg_type = (MessageType)t_msg.getIntProperty( MessageProperties.MSG_TYPE );
     if( msg_type == MessageType.NETWORK_STATUS_NOTIFICATION ) {
        // Handle network status notification.
        network_info = "";
        network_adapters = t_msg.getStringProperty( MessageProperties.ADAPTERS );
        if( network_adapters != null && network_adapters.length > 0) {
          network_info += network_adapters;
        }
        network_names = t_msg.getStringProperty( MessageProperties.RASNAMES );

         //...
     }
  }
}
```

ADAPTER 変数

"system" キューメッセージの場合は、QAnywhere サーバーへの接続で使用されているネットワークアダプターです。

構文

`final String` **`MessageProperties.ADAPTER`**

備考

このフィールドの値は "ias_Network.Adapter" です。

MessageProperties.ADAPTER を QAMessage.getStringProperty(String) メソッドに渡して、関連付けられているプロパティにアクセスできます。

このプロパティは読み込み専用です。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ
- QAMessage.getStringProperty メソッド [QAnywhere Java]593 ページ

ADAPTERS 変数

このプロパティ名は、QAnywhere サーバーへの接続で使用できるネットワークアダプターのデリミター付きリストを示します。

構文

```
final String MessageProperties.ADAPTERS
```

備考

これは、システムキューメッセージで使用されます。

MessageProperties.ADAPTERS を QAMessage.getStringProperty(String) メソッドに渡して、関連付けられているプロパティにアクセスできます。このプロパティは読み込み専用です。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ
- QAMessage.getStringProperty メソッド [QAnywhere Java]593 ページ

DELIVERY_COUNT 変数

このプロパティ名は、メッセージを配信するために、これまでに実行された試行回数を示します。

構文

```
final String MessageProperties.DELIVERY_COUNT
```

IP 変数

"system" キューメッセージの場合は、QAnywhere サーバーへの接続で使用されているネットワークアダプターの IP アドレスです。

構文

```
final String MessageProperties.IP
```

備考

このフィールドの値は "ias_Network.IP" です。

MessageProperties.IP を QAMessage.getStringProperty(String) メソッドに渡して、関連付けられているプロパティにアクセスできます。

このプロパティは読み込み専用です。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ
- QAMessage.getStringProperty メソッド [QAnywhere Java]593 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

MAC 変数

"system" キューメッセージの場合は、QAnywhere サーバーへの接続で使用されているネットワークアダプターの MAC アドレスです。

構文

```
final String MessageProperties.MAC
```

備考

このフィールドの値は "ias_Network.MAC" です。

MessageProperties.MAC を QAMessage.getStringProperty(String) メソッドに渡して、関連付けられているプロパティにアクセスできます。

このプロパティは読み込み専用です。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ
- QAMessage.getStringProperty メソッド [QAnywhere Java]593 ページ

MSG_TYPE 変数

このプロパティ名は、QAnywhere メッセージに関連付けられている MessageType 列挙値を示します。

構文

```
final String MessageProperties.MSG_TYPE
```

備考

このフィールドの値は "ias_MessageType" です。

MessageProperties.MSG_TYPE を QAMessage.getIntProperty(String) メソッドに渡して、関連付けられているプロパティにアクセスできます。

このプロパティは読み込み専用です。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ
- QAMessage.getIntProperty メソッド [QAnywhere Java]589 ページ

NETWORK_STATUS 変数

このプロパティ名は、ネットワーク接続のステータスを示します。

構文

```
final String MessageProperties.NETWORK_STATUS
```

備考

ネットワークがアクセス可能な場合は 1、それ以外の場合は 0 です。ネットワークステータスは、システムキューメッセージ (ネットワークステータスの変更など) で使用されます。

MessageProperties.NETWORK_STATUS を QAMessage.getIntProperty(String) メソッドに渡して、関連付けられているプロパティにアクセスできます。

このプロパティは読み込み専用です。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ
- QAMessage.getIntProperty メソッド [QAnywhere Java]589 ページ

ORIGINATOR 変数

このプロパティ名は、メッセージの発信者のメッセージストア ID を示します。

構文

```
final String MessageProperties.ORIGINATOR
```

RAS 変数

"system" キューメッセージの場合は、QAnywhere サーバーへの接続で使用されている RAS エントリ名です。

構文

```
final String MessageProperties.RAS
```

備考

このフィールドの値は "ias_Network.RAS" です。

MessageProperties.RAS を QAMessage.getStringProperty(String) メソッドに渡して、関連付けられているプロパティにアクセスできます。

このプロパティは読み込み専用です。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ
- QAMessage.getStringProperty メソッド [QAnywhere Java]593 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

RASNAMES 変数

"system" キューメッセージの場合は、QAnywhere サーバーに接続するときに使用できる RAS エントリ名のデリミター付きリストです。

構文

```
final String MessageProperties.RASNAMES
```

備考

このフィールドの値は "ias_RASNames" です。

MessageProperties.RASNAMES を QAMessage.getStringProperty(String) メソッドに渡して、関連付けられているプロパティにアクセスできます。

このプロパティは読み込み専用です。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ
- QAMessage.getStringProperty メソッド [QAnywhere Java]593 ページ

STATUS 変数

このプロパティ名は、メッセージの現在のステータスを示します。

構文

```
final String MessageProperties.STATUS
```

参照

- StatusCodes インターフェイス [QAnywhere Java]616 ページ

STATUS_TIME 変数

このプロパティ名は、メッセージが現在のステータスになったと判断された時刻を示します。

構文

```
final String MessageProperties.STATUS_TIME
```

備考

MessageProperties.STATUS_TIME を QAMessage.getProperty メソッドに渡すと、java.util.Date インスタンスが返されます。

参照

- QAMessage.getProperty メソッド [QAnywhere Java]590 ページ

TRANSMISSION_STATUS 変数

このプロパティ名は、メッセージの現在の転送ステータスを示します。

構文

```
final String MessageProperties.TRANSMISSION_STATUS
```

参照

- StatusCodes インターフェイス [QAnywhere Java]616 ページ

MessageStoreProperties インターフェイス

このクラスは、有用なメッセージストアプロパティ名の定数値を定義します。

構文

```
public interface MessageStoreProperties
```

メンバー

継承されたメンバーを含む MessageStoreProperties インターフェイスのすべてのメンバー。

名前	説明
MAX_DELIVERY_ATTEMPTS 変数	このプロパティ名は、メッセージのステータスが StatusCodes.UNRECEIVABLE に設定されるまでに、受信確認がないままメッセージを受信できる最大回数を示します。

備考

MessageStoreProperties クラスは、標準のメッセージプロパティ名を提供します。MessageStoreProperties フィールドは、事前定義済みまたはカスタムのメッセージストアプロパティの取得と設定で使用する QAManagerBase メソッドに渡すことができます。

参照

- QAManagerBase インターフェイス [QAnywhere Java]544 ページ

MAX_DELIVERY_ATTEMPTS 変数

このプロパティ名は、メッセージのステータスが StatusCodes.UNRECEIVABLE に設定されるまでに、受信確認がないままメッセージを受信できる最大回数を示します。

構文

```
final String MessageStoreProperties.MAX_DELIVERY_ATTEMPTS
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- StatusCodes.UNRECEIVABLE 変数 [QAnywhere Java]620 ページ

MessageType インターフェイス

MessageProperties.MSG_TYPE メッセージプロパティの定数値を定義します。

構文

```
public interface MessageType
```

メンバー

継承されたメンバーを含む MessageType インターフェイスのすべてのメンバー。

名前	説明
NETWORK_STATUS_NOTIFICATION 変数	QAnywhere クライアントアプリケーションにネットワークステータスの変更を通知する場合に使用する、QAnywhere システムメッセージを特定します。
PUSH_NOTIFICATION 変数	QAnywhere クライアントアプリケーションに Push 通知を通知する場合に使用する、QAnywhere システムメッセージを特定します。
REGULAR 変数	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。

備考

次の例は、QAnywhere システムメッセージの処理で使用される onSystemMessage(QAMessage) メソッドを示します。メッセージタイプは、MessageType.NETWORK_STATUS_NOTIFICATION と比較されます。

```
private void onSystemMessage(QAMessage msg)
{
  QATextMessage    t_msg;
  int              msg_type;
  String           network_adapters;
  String           network_names;
  String           network_info;

  t_msg = (QATextMessage)msg;
  if( t_msg != null )
  {
     // Evaluate message type.
     msg_type = t_msg.getIntProperty( MessageProperties.MSG_TYPE );
     if( msg_type == MessageType.NETWORK_STATUS_NOTIFICATION )
     {
```

```
            // Handle network status notification.
        }
    }
}
```

NETWORK_STATUS_NOTIFICATION 変数

QAnywhere クライアントアプリケーションにネットワークステータスの変更を通知する場合に使用する、QAnywhere システムメッセージを特定します。

構文

`final int` **`MessageType.NETWORK_STATUS_NOTIFICATION`**

備考

ネットワークステータスの変更は、システムメッセージを受信するデバイスに適用されます。新しいネットワークステータス情報を特定するには、MessageProperties.ADAPTER、MessageProperties.NETWORK、MessageProperties.NETWORK_STATUS の各フィールドを使用します。

PUSH_NOTIFICATION 変数

QAnywhere クライアントアプリケーションに Push 通知を通知する場合に使用する、QAnywhere システムメッセージを特定します。

構文

`final int` **`MessageType.PUSH_NOTIFICATION`**

備考

オンデマンドの QAnywhere Agent ポリシーを使用した場合の一般的な応答は、QAManagerBase.triggerSendReceive() メソッドを呼び出し、中央のメッセージサーバーで待機しているメッセージを受信することです。

REGULAR 変数

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。

構文

`final int` **`MessageType.REGULAR`**

備考

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。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

PropertyType インターフェイス

QAMessage プロパティ型列挙体。Java の型に自然に対応します。

構文

```
public interface PropertyType
```

メンバー

継承されたメンバーを含む PropertyType インターフェイスのすべてのメンバー。

名前	説明
PROPERTY_TYPE_BOOLEAN 変数	プロパティ型が boolean であることを示します。
PROPERTY_TYPE_BYTE 変数	プロパティ型が signed byte であることを示します。
PROPERTY_TYPE_DOUBLE 変数	プロパティ型が double であることを示します。
PROPERTY_TYPE_FLOAT 変数	プロパティ型が float であることを示します。
PROPERTY_TYPE_INT 変数	プロパティ型が int であることを示します。
PROPERTY_TYPE_LONG 変数	プロパティ型が long であることを示します。
PROPERTY_TYPE_SHORT 変数	プロパティ型が short であることを示します。
PROPERTY_TYPE_STRING 変数	プロパティ型が String であることを示します。
PROPERTY_TYPE_UNKNOWN 変数	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。

参照

- QAMessage インターフェイス [QAnywhere Java]582 ページ

PROPERTY_TYPE_BOOLEAN 変数

プロパティ型が boolean であることを示します。

構文

```
final short PropertyType.PROPERTY_TYPE_BOOLEAN
```

PROPERTY_TYPE_BYTE 変数

プロパティ型が signed byte であることを示します。

構文

```
final short PropertyType.PROPERTY_TYPE_BYTE
```

PROPERTY_TYPE_DOUBLE 変数

プロパティ型が double であることを示します。

構文

```
final short PropertyType.PROPERTY_TYPE_DOUBLE
```

PROPERTY_TYPE_FLOAT 変数

プロパティ型が float であることを示します。

構文

```
final short PropertyType.PROPERTY_TYPE_FLOAT
```

PROPERTY_TYPE_INT 変数

プロパティ型が int であることを示します。

構文

```
final short PropertyType.PROPERTY_TYPE_INT
```

PROPERTY_TYPE_LONG 変数

プロパティ型が long であることを示します。

構文

```
final short PropertyType.PROPERTY_TYPE_LONG
```

PROPERTY_TYPE_SHORT 変数

プロパティ型が short であることを示します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
final short PropertyType.PROPERTY_TYPE_SHORT
```

PROPERTY_TYPE_STRING 変数

プロパティ型が String であることを示します。

構文

```
final short PropertyType.PROPERTY_TYPE_STRING
```

PROPERTY_TYPE_UNKNOWN 変数

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。

構文

```
final short PropertyType.PROPERTY_TYPE_UNKNOWN
```

QABinaryMessage インターフェイス

QABinaryMessage オブジェクトは、未解釈のバイトストリームが含まれるメッセージの送信に使用します。

構文

```
public interface QABinaryMessage
```

基本クラス

- QAMessage インターフェイス [QAnywhere Java]582 ページ

メンバー

継承されたメンバーを含む QABinaryMessage インターフェイスのすべてのメンバー。

名前	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。
getAddress メソッド	QAMessage インスタンスの送信先アドレスを返します。
getBodyLength メソッド	メッセージ本文のサイズをバイト単位で返します。

名前	説明
getBooleanProperty メソッド	boolean 型のメッセージプロパティを取得します。
getByteProperty メソッド	signed byte 型のメッセージプロパティを取得します。
getDoubleProperty メソッド	double 型のメッセージプロパティを取得します。
getExpiration メソッド	メッセージの有効期限の値を返します。メッセージの有効期限がない場合や、まだ送信されていない場合は、NULL になります。
getFloatProperty メソッド	float 型のメッセージプロパティを取得します。
getInReplyToID メソッド	このメッセージの返信メッセージのメッセージ ID を返します。
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。
getMessageID メソッド	メッセージのグローバルでユニークなメッセージ ID を返します。
getPriority メソッド	メッセージの優先度 (0 から 9) を返します。
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。
getRedelivered メソッド	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。
getReplyToAddress メソッド	メッセージの返信アドレスを返します。
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	String 型のメッセージプロパティを取得します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
getTimestamp メソッド	メッセージの作成時刻を示すタイムスタンプを返します。
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。
readBinary メソッド	QABinaryMessage インスタンスの本文未読部分の先頭からいくつかのバイト数を読み込んで、**dest** 配列に格納します。
readBoolean メソッド	QABinaryMessage インスタンスのメッセージ本文未読部分の先頭から、boolean 値を読み込みます。
readByte メソッド	QABinaryMessage メッセージ本文未読部分の先頭から signed byte 値を読み込みます。
readChar メソッド	QABinaryMessage メッセージ本文未読部分の先頭から char 値を読み込みます。
readDouble メソッド	QABinaryMessage メッセージ本文未読部分の先頭から double 値を読み込みます。
readFloat メソッド	QABinaryMessage メッセージ本文未読部分の先頭から float 値を読み込みます。
readInt メソッド	QABinaryMessage メッセージ本文未読部分の先頭から int 値を読み込みます。
readLong メソッド	QABinaryMessage メッセージ本文未読部分の先頭から long 値を読み込みます。
readShort メソッド	QABinaryMessage メッセージ本文未読部分の先頭から short 値を読み込みます。
readString メソッド	QABinaryMessage メッセージ本文未読部分の先頭から string 値を読み込みます。
reset メソッド	メッセージをリセットして、メッセージ本文の先頭から値の読み込みを開始できるようにします。
setBooleanProperty メソッド	boolean 型のプロパティを設定します。
setByteProperty メソッド	signed byte 型のプロパティを設定します。
setDoubleProperty メソッド	double 型のプロパティを設定します。

名前	説明
setFloatProperty メソッド	float 型のプロパティを設定します。
setInReplyToID メソッド	このメッセージの返信対象メッセージを特定する、返信先 ID を設定します。
setIntProperty メソッド	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。
setPriority メソッド	メッセージの優先度 (0 から 9) を設定します。
setProperty メソッド	プロパティを設定します。
setReplyToAddress メソッド	返信先アドレスを設定します。
setShortProperty メソッド	short 型のプロパティを設定します。
setStringProperty メソッド	string 型のプロパティを設定します。
writeBinary メソッド	QABinaryMessage インスタンスのメッセージ本文に byte 配列値を追加します。
writeBoolean メソッド	QABinaryMessage インスタンスのメッセージ本文に boolean 値を追加します。
writeByte メソッド	QABinaryMessage インスタンスのメッセージ本文に signed byte 値を追加します。
writeChar メソッド	QABinaryMessage インスタンスのメッセージ本文に char 値を追加します。
writeDouble メソッド	QABinaryMessage インスタンスのメッセージ本文に double 値を追加します。
writeFloat メソッド	QABinaryMessage インスタンスのメッセージ本文に float 値を追加します。
writeInt メソッド	QABinaryMessage インスタンスのメッセージ本文に int 値を追加します。
writeLong メソッド	QABinaryMessage インスタンスのメッセージ本文に long 値を追加します。
writeShort メソッド	QABinaryMessage インスタンスのメッセージ本文に short 値を追加します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
writeString メソッド	QABinaryMessage インスタンスのメッセージ本文に string 値を追加します。
DEFAULT_PRIORITY 変数	デフォルトのメッセージ優先度です。
DEFAULT_TIME_TO_LIVE 変数	デフォルトの存続時間です。

備考

QABinaryMessage は QAMessage クラスを継承したもので、メッセージ本文にバイトストリームが追加されます。QABinaryMessage には、メッセージ本文からのバイトストリームの読み込み／書き込みを行うためのさまざまな関数があります。

メッセージが最初に作成された時点では、メッセージ本文は書き込み専用モードになっています。メッセージ送信元のクライアントは、メッセージを送信した後、そのメッセージを保持し変更できます。ただし、それによって、送信されたメッセージが変更されることはありません。同じメッセージオブジェクトを複数回送信できます。

メッセージが受信された時点でプロバイダーは QABinaryMessage.reset() を呼び出します。これによりメッセージ本文が読み込み専用モードになり、メッセージ本文の先頭から値の読み込みが開始されます。

次の例では QABinaryMessage.writeString(String) を使用して、QABinaryMessage インスタンスのメッセージ本文に文字列 "Q" と "Anywhere" を書き込みます。

```
// Create a binary message instance.
QABinaryMessage binary_message;
binary_message = qa_manager.createBinaryMessage();

// Set optional message properties.
binary_message.setReplyToAddress("my-queue-name");

// Write to the message body.
binary_message.writeString("Q");
binary_message.writeString("Anywhere");

// Put the message in the local database, ready for sending.
try {
   qa_manager.putMessage( "store-id¥¥queue-name", binary_message );
}
catch ( QAException e ) {
   handleError();
}
```

注意
受信が終了すると、最初の QABinaryMessage.readString() の呼び出しから "Q" が返され、2 番目の QABinaryMessage.readString() の呼び出しから "Anywhere" が返されます。

メッセージは QAnywhere Agent から送信されます。

参照

- QAMessage インターフェイス [QAnywhere Java]582 ページ
- QABinaryMessage.readString メソッド [QAnywhere Java]520 ページ

getBodyLength メソッド

メッセージ本文のサイズをバイト単位で返します。

構文

```
long QABinaryMessage.getBodyLength() throws QAException
```

戻り値

メッセージ本文のサイズ (バイト単位)。

例外

- **QAException クラス**　メッセージ本文のサイズの取得で問題が発生した場合にスローされます。

readBinary メソッド

QABinaryMessage インスタンスの本文未読部分の先頭からいくつかのバイト数を読み込んで、dest 配列に格納します。

オーバーロードリスト

名前	説明
readBinary(byte[]) メソッド	QABinaryMessage インスタンスの本文未読部分の先頭からいくつかのバイト数を読み込んで、**dest** 配列に格納します。
readBinary(byte[], int) メソッド	QABinaryMessage インスタンスの本文未読部分の先頭から length で指定されたバイト数まで読み込んで、**dest** 配列に格納します。
readBinary(byte[], int, int) メソッド	QABinaryMessage インスタンスの本文未読部分の先頭から length で指定されたバイト数まで読み込んで、dest[offset] を開始位置として **dest** 配列に格納します。

readBinary(byte[]) メソッド

QABinaryMessage インスタンスの本文未読部分の先頭からいくつかのバイト数を読み込んで、**dest** 配列に格納します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
int QABinaryMessage.readBinary(byte[] dest) throws QAException
```

パラメーター

- **dest**　読み込まれたバイトを保持する byte 配列。

戻り値

メッセージ本文から読み込まれたバイト数。読み込めるバイトが残っていない場合は -1。

例外

- **QAException クラス**　メッセージからのバイトの読み込み時にエラーが発生した場合にスローされます。

備考

readBinary(dest) メソッドには、readBinary(dest,0,dest.length) と同じ効果があります。

参照

- QABinaryMessage.writeBinary メソッド [QAnywhere Java]520 ページ

readBinary(byte[], int) メソッド

QABinaryMessage インスタンスの本文未読部分の先頭から length で指定されたバイト数まで読み込んで、**dest** 配列に格納します。

構文

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

パラメーター

- **dest**　読み込まれたバイトを保持する byte 配列。
- **length**　読み込むバイト数の最大値。

戻り値

メッセージ本文から読み込まれたバイト数。読み込めるバイトが残っていない場合は -1。

例外

- **QAException クラス**　メッセージからのバイトの読み込み時にエラーが発生した場合にスローされます。

備考

readBinary(dest, len) メソッドには、readBinary(dest,0,len) と同じ効果があります。

参照

- QABinaryMessage.writeBinary メソッド [QAnywhere Java]520 ページ

readBinary(byte[], int, int) メソッド

QABinaryMessage インスタンスの本文未読部分の先頭から length で指定されたバイト数まで読み込んで、dest[offset] を開始位置として **dest** 配列に格納します。

構文

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

パラメーター

- **dest**　読み込まれたバイトを保持する byte 配列。
- **offset**　コピー先の配列の開始オフセット。
- **length**　読み込むバイト数の最大値。

戻り値

メッセージ本文から読み込まれたバイト数。読み込めるバイトが残っていない場合は -1。

例外

- **QAException クラス**　メッセージからのバイトの読み込み時にエラーが発生した場合にスローされます。

備考

dest が NULL の場合、NullPointerException がスローされます。

offset が負の場合、length が負の場合、または offset+length が dest の長さを超過する場合は、IndexOutOfBoundsException がスローされます。

参照

- QABinaryMessage.writeBinary メソッド [QAnywhere Java]520 ページ

readBoolean メソッド

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

構文

```
boolean QABinaryMessage.readBoolean() throws QAException
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

戻り値

メッセージ本文から読み込まれる boolean 値。

例外

- **QAException クラス**　値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- QABinaryMessage.writeBoolean メソッド [QAnywhere Java]523 ページ

readByte メソッド

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

構文

```
byte QABinaryMessage.readByte() throws QAException
```

戻り値

メッセージ本文から読み込まれる signed byte 値。

例外

- **QAException クラス**　値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- QABinaryMessage.writeByte メソッド [QAnywhere Java]523 ページ

readChar メソッド

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

構文

```
char QABinaryMessage.readChar() throws QAException
```

戻り値

メッセージ本文から読み込まれる char 値。

例外

- **QAException クラス**　値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- QABinaryMessage.writeChar メソッド [QAnywhere Java]524 ページ

readDouble メソッド

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

構文

```
double QABinaryMessage.readDouble() throws QAException
```

戻り値

メッセージ本文から読み込まれる double 値。

例外

- **QAException クラス**　値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- QABinaryMessage.writeDouble メソッド [QAnywhere Java]524 ページ

readFloat メソッド

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

構文

```
float QABinaryMessage.readFloat() throws QAException
```

戻り値

メッセージ本文から読み込まれる float 値。

例外

- **QAException クラス**　値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- QABinaryMessage.writeFloat メソッド [QAnywhere Java]525 ページ

readInt メソッド

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

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
int QABinaryMessage.readInt() throws QAException
```

戻り値

メッセージ本文から読み込まれる int 値。

例外

- **QAException クラス**　値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- QABinaryMessage.writeInt メソッド [QAnywhere Java]525 ページ

readLong メソッド

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

構文

```
long QABinaryMessage.readLong() throws QAException
```

戻り値

メッセージ本文から読み込まれる long 値。

例外

- **QAException クラス**　値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- QABinaryMessage.writeLong メソッド [QAnywhere Java]526 ページ

readShort メソッド

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

構文

```
short QABinaryMessage.readShort() throws QAException
```

戻り値

メッセージ本文から読み込まれる short 値。

例外

- **QAException クラス**　値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- QABinaryMessage.writeShort メソッド [QAnywhere Java]526 ページ

readString メソッド

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

構文

```
String QABinaryMessage.readString() throws QAException
```

戻り値

メッセージ本文から読み込まれる string 値。

例外

- **QAException クラス**　値の読み込みで変換エラーが発生した場合、または読み込める入力が残っていない場合にスローされます。

参照

- QABinaryMessage.writeString メソッド [QAnywhere Java]527 ページ

reset メソッド

メッセージをリセットして、メッセージ本文の先頭から値の読み込みを開始できるようにします。

構文

```
void QABinaryMessage.reset() throws QAException
```

例外

- **QAException クラス**　メッセージのリセットで問題が発生した場合にスローされます。

備考

また、reset メソッドは、QABinaryMessage のメッセージ本文を読み込み専用モードにします。

writeBinary メソッド

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

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

オーバーロードリスト

名前	説明
writeBinary(byte[]) メソッド	QABinaryMessage インスタンスのメッセージ本文に byte 配列値を追加します。
writeBinary(byte[], int) メソッド	QABinaryMessage インスタンスのメッセージ本文に、バイト配列から length で指定されたバイト数分を追加します。
writeBinary(byte[], int, int) メソッド	QABinaryMessage インスタンスのメッセージ本文に、特定のオフセットを先頭として、バイト配列から length で指定されたバイト数分を追加します。

writeBinary(byte[]) メソッド

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

構文

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

パラメーター

- **val**　メッセージ本文に書き込む byte 配列値。

例外

- **QAException クラス**　メッセージ本文への byte 配列の追加で問題が発生した場合にスローされます。

備考

writeBinary(val) メソッドには、writeBinary(val,0,val.length) と同じ効果があります。

参照

- QABinaryMessage.readBinary メソッド [QAnywhere Java]514 ページ

writeBinary(byte[], int) メソッド

QABinaryMessage インスタンスのメッセージ本文に、バイト配列から length で指定されたバイト数分を追加します。

構文

```
void QABinaryMessage.writeBinary(
     byte[] val,
    int length
) throws QAException
```

パラメーター

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

例外

- **QAException クラス** メッセージ本文への byte 配列の追加で問題が発生した場合にスローされます。

備考

writeBinary(val,len) メソッドには、writeBinary(val,0,len) と同じ効果があります。

参照

- QABinaryMessage.readBinary メソッド [QAnywhere Java]514 ページ

writeBinary(byte[], int, int) メソッド

QABinaryMessage インスタンスのメッセージ本文に、特定のオフセットを先頭として、バイト配列から length で指定されたバイト数分を追加します。

構文

```
void QABinaryMessage.writeBinary(
     byte[] val,
    int offset,
    int length
) throws QAException
```

パラメーター

- **val** メッセージ本文に書き込む byte 配列値。
- **offset** byte 配列内でのオフセット (書き込み開始位置)。
- **length** 書き込まれるバイト数。

例外

- **QAException クラス** メッセージ本文への byte 配列の追加で問題が発生した場合にスローされます。

備考

val が NULL の場合、NullPointerException がスローされます。offset が負の場合、length が負の場合、または length+offset が val の長さを超過する場合は、IndexOutOfBoundsException がスローされます。

参照

- QABinaryMessage.readBinary メソッド [QAnywhere Java]514 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

writeBoolean メソッド

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

構文

```
void QABinaryMessage.writeBoolean(boolean val) throws QAException
```

パラメーター

- **val** メッセージ本文に書き込む boolean 値。

例外

- **QAException クラス** メッセージ本文への boolean 値の追加で問題が発生した場合にスローされます。

備考

boolean 値は 1 バイトの値で示されます。true は 1、false は 0 で示されます。

参照

- QABinaryMessage.readBoolean メソッド [QAnywhere Java]516 ページ

writeByte メソッド

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

構文

```
void QABinaryMessage.writeByte(byte val) throws QAException
```

パラメーター

- **val** メッセージ本文に書き込む signed byte 値。

例外

- **QAException クラス** メッセージ本文への signed byte 値の追加で問題が発生した場合にスローされます。

備考

signed byte 値は 1 バイトの値で示されます。

参照

- QABinaryMessage.readByte メソッド [QAnywhere Java]517 ページ

writeChar メソッド

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

構文

```
void QABinaryMessage.writeChar(char val) throws QAException
```

パラメーター

- **val** メッセージ本文に書き込む char 値。

例外

- **QAException クラス** メッセージ本文への char 値の追加で問題が発生した場合にスローされます。

備考

char は 2 バイトの値で示され、上位バイトが最初に追加されます。

参照

- QABinaryMessage.readChar メソッド [QAnywhere Java]517 ページ

writeDouble メソッド

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

構文

```
void QABinaryMessage.writeDouble(double val) throws QAException
```

パラメーター

- **val** メッセージ本文に書き込む double 値。

例外

- **QAException クラス** メッセージ本文への double 値の追加で問題が発生した場合にスローされます。

備考

double 値は 8 バイトの long 値に変換されて、上位のバイトから追加されます。

参照

- QABinaryMessage.readDouble メソッド [QAnywhere Java]518 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

writeFloat メソッド

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

構文

```
void QABinaryMessage.writeFloat(float val) throws QAException
```

パラメーター

- **val** メッセージ本文に書き込む float 値。

例外

- **QAException クラス** メッセージ本文への float 値の追加で問題が発生した場合にスローされます。

備考

float 値は 4 バイトの int に変換され、上位のバイトから追加されます。

参照

- QABinaryMessage.readFloat メソッド [QAnywhere Java]518 ページ

writeInt メソッド

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

構文

```
void QABinaryMessage.writeInt(int val) throws QAException
```

パラメーター

- **val** メッセージ本文に書き込む int 値。

例外

- **QAException クラス** メッセージ本文への int 値の追加で問題が発生した場合にスローされます。

備考

int パラメーターは 4 バイトの値で示され、上位のバイトから追加されます。

参照

- QABinaryMessage.readInt メソッド [QAnywhere Java]518 ページ

writeLong メソッド

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

構文

```
void QABinaryMessage.writeLong(long val) throws QAException
```

パラメーター

- **val** メッセージ本文に書き込む long 値。

例外

- **QAException クラス** メッセージ本文への long 値の追加で問題が発生した場合にスローされます。

備考

long パラメーターは 8 バイトの値で示され、上位のバイトから追加されます。

参照

- QABinaryMessage.readLong メソッド [QAnywhere Java]519 ページ

writeShort メソッド

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

構文

```
void QABinaryMessage.writeShort(short val) throws QAException
```

パラメーター

- **val** メッセージ本文に書き込む short 値。

例外

- **QAException クラス** メッセージ本文への short 値の追加で問題が発生した場合にスローされます。

備考

short パラメーターは 2 バイトの値で示され、上位のバイトから追加されます。

参照

- QABinaryMessage.readShort メソッド [QAnywhere Java]519 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

writeString メソッド

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

構文

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

パラメーター

- **val**　メッセージ本文に書き込む string 値。

例外

- **QAException クラス**　メッセージ本文への string 値の追加で問題が発生した場合にスローされます。

備考

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

> **注意**
> 文字列の UTF-8 表記は、最大で 32767 バイトまで書き込み可能です。

参照

- QABinaryMessage.readString メソッド [QAnywhere Java]520 ページ

QAException クラス

QAnywhere クライアントアプリケーションの例外をカプセル化します。

構文

```
public class QAException
```

メンバー

継承されたメンバーを含む QAException クラスのすべてのメンバー。

名前	説明
getDetailedMessage メソッド	例外の詳細な説明を返します。
getErrorCode メソッド	最後に発生した例外のエラーコードを返します。

名前	説明
getNativeErrorCode メソッド	最後に発生した例外のネイティブエラーコードを返します。
COMMON_ALREADY_OPEN_ERROR 変数	QAManager はすでに開いています。
COMMON_GET_INIT_FILE_ERROR 変数	クライアントのプロパティファイルにアクセスできません。
COMMON_GET_PROPERTY_ERROR 変数	メッセージストアからプロパティを取得するときにエラーが発生しました。
COMMON_GETQUEUEDEPTH_ERROR 変数	キューの長さの取得中にエラーが発生しました。
COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG 変数	フィルターが ALL の場合、指定された宛先には QAManagerBase.getQueueDepth を使用できません。
COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID 変数	メッセージストア ID が設定されていない場合は QAManagerBase.getQueueDepth を使用できません。
COMMON_INIT_ERROR 変数	初期化エラーです。
COMMON_INIT_THREAD_ERROR 変数	バックグラウンドスレッドを初期化中にエラーが発生しました。
COMMON_INVALID_PROPERTY 変数	クライアントのプロパティファイルに無効なプロパティが存在します。
COMMON_MSG_ACKNOWLEDGE_ERROR 変数	メッセージの確認中にエラーが発生しました。
COMMON_MSG_CANCEL_ERROR 変数	メッセージのキャンセル中にエラーが発生しました。
COMMON_MSG_CANCEL_ERROR_SENT 変数	メッセージのキャンセル中にエラーが発生しました。
COMMON_MSG_NOT_WRITEABLE_ERROR 変数	読み込み専用モードのメッセージには書き込みできません。
COMMON_MSG_RETRIEVE_ERROR 変数	クライアントメッセージストアからメッセージを取得するときにエラーが発生しました。
COMMON_MSG_STORE_ERROR 変数	クライアントメッセージストアにメッセージを格納するときにエラーが発生しました。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
COMMON_MSG_STORE_NOT_INITIALIZED 変数	メッセージストアがメッセージを送るために初期化されていません。
COMMON_MSG_STORE_TOO_LARGE 変数	メッセージストアがデバイスのディスク空き容量と比べて大きすぎます。
COMMON_NO_DEST_ERROR 変数	宛先が指定されていません。
COMMON_NO_IMPLEMENTATION 変数	メソッドが実装されていません。
COMMON_NOT_OPEN_ERROR 変数	QAManager が開いていません。
COMMON_OPEN_ERROR 変数	メッセージストアへの接続を開くときにエラーが発生しました。
COMMON_OPEN_LOG_FILE_ERROR 変数	ログファイルを開くときにエラーが発生しました。
COMMON_OPEN_MAXTHREADS_ERROR 変数	サーバーの最大同時要求数が少なすぎるため QAManager を開くことができません (データベースサーバーの -gn オプションを参照)。
COMMON_REOPEN_ERROR 変数	メッセージストアへの接続を再び開くときにエラーが発生しました。
COMMON_SELECTOR_SYNTAX_ERROR 変数	指定されたセレクターに構文エラーがあります。
COMMON_SET_PROPERTY_ERROR 変数	メッセージストアにプロパティを格納するときにエラーが発生しました。
COMMON_TERMINATE_ERROR 変数	終了エラーです。
COMMON_UNEXPECTED_EOM_ERROR 変数	予期しないメッセージの終わりに到達しました。
COMMON_UNREPRESENTABLE_TIMESTAMP 変数	タイムスタンプが許容範囲外です。
QA_NO_ERROR 変数	エラーはありません。

備考

QAException クラスを使用して QAnywhere の例外をキャッチできます。

次の例では QAnywhere 例外をキャッチします。

```
try {
    _qaManager = QAManagerFactory.getInstance().CreateQAManager();
```

```
        _qaManager.open( AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT );
        _qaManager.start();
    }
    catch( QAException e ) {
        // Handle exception.
        System.err.println("Error code: " + e.getErrorCode() );
        System.err.println("Error message: " + e.getMessage() );
    }
```

getDetailedMessage メソッド

例外の詳細な説明を返します。

構文

```
abstract String QAException.getDetailedMessage()
```

戻り値

例外の詳細な説明です。

getErrorCode メソッド

最後に発生した例外のエラーコードを返します。

構文

```
abstract int QAException.getErrorCode()
```

戻り値

最後に発生した例外のエラーコード

getNativeErrorCode メソッド

最後に発生した例外のネイティブエラーコードを返します。

構文

```
abstract int QAException.getNativeErrorCode()
```

戻り値

最後に発生した例外のネイティブエラーコード。

COMMON_ALREADY_OPEN_ERROR 変数

QAManager はすでに開いています。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
final int QAException.COMMON_ALREADY_OPEN_ERROR
```

参照

- QAManager インターフェイス [QAnywhere Java]536 ページ

COMMON_GET_INIT_FILE_ERROR 変数

クライアントのプロパティファイルにアクセスできません。

構文

```
final int QAException.COMMON_GET_INIT_FILE_ERROR
```

COMMON_GET_PROPERTY_ERROR 変数

メッセージストアからプロパティを取得中にエラーが発生しました。

構文

```
final int QAException.COMMON_GET_PROPERTY_ERROR
```

COMMON_GETQUEUEDEPTH_ERROR 変数

キューの長さの取得中にエラーが発生しました。

構文

```
final int QAException.COMMON_GETQUEUEDEPTH_ERROR
```

参照

- QAManagerBase.getQueueDepth メソッド [QAnywhere Java]561 ページ

COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG 変数

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

構文

```
final int QAException.COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG
```

参照

- QAManagerBase.getQueueDepth メソッド [QAnywhere Java]561 ページ

COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID 変数

メッセージストア ID が設定されていない場合は、QAManagerBase.getQueueDepth を使用できません。

構文

```
final int QAException.COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID
```

参照

- QAManagerBase.getQueueDepth メソッド [QAnywhere Java]561 ページ

COMMON_INIT_ERROR 変数

初期化エラーです。

構文

```
final int QAException.COMMON_INIT_ERROR
```

COMMON_INIT_THREAD_ERROR 変数

バックグラウンドスレッドを初期化中にエラーが発生しました。

構文

```
final int QAException.COMMON_INIT_THREAD_ERROR
```

COMMON_INVALID_PROPERTY 変数

クライアントの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。

構文

```
final int QAException.COMMON_INVALID_PROPERTY
```

COMMON_MSG_ACKNOWLEDGE_ERROR 変数

メッセージの確認中にエラーが発生しました。

構文

```
final int QAException.COMMON_MSG_ACKNOWLEDGE_ERROR
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

COMMON_MSG_CANCEL_ERROR 変数

メッセージのキャンセル中にエラーが発生しました。

構文

```
final int QAException.COMMON_MSG_CANCEL_ERROR
```

COMMON_MSG_CANCEL_ERROR_SENT 変数

メッセージのキャンセル中にエラーが発生しました。

構文

```
final int QAException.COMMON_MSG_CANCEL_ERROR_SENT
```

備考

すでに送信されたメッセージをキャンセルすることはできません。

COMMON_MSG_NOT_WRITEABLE_ERROR 変数

読み込み専用モードのメッセージには書き込みできません。

構文

```
final int QAException.COMMON_MSG_NOT_WRITEABLE_ERROR
```

COMMON_MSG_RETRIEVE_ERROR 変数

クライアントメッセージストアからメッセージを取得中にエラーが発生しました。

構文

```
final int QAException.COMMON_MSG_RETRIEVE_ERROR
```

COMMON_MSG_STORE_ERROR 変数

メッセージをクライアントメッセージストアに格納中にエラーが発生しました。

構文

```
final int QAException.COMMON_MSG_STORE_ERROR
```

COMMON_MSG_STORE_NOT_INITIALIZED 変数

メッセージストアがメッセージを送るために初期化されていません。

構文

```
final int QAException.COMMON_MSG_STORE_NOT_INITIALIZED
```

COMMON_MSG_STORE_TOO_LARGE 変数

メッセージストアがデバイスのディスク空き容量と比べて大きすぎます。

構文

```
final int QAException.COMMON_MSG_STORE_TOO_LARGE
```

COMMON_NO_DEST_ERROR 変数

宛先が指定されていません。

構文

```
final int QAException.COMMON_NO_DEST_ERROR
```

COMMON_NO_IMPLEMENTATION 変数

メソッドが実装されていません。

構文

```
final int QAException.COMMON_NO_IMPLEMENTATION
```

COMMON_NOT_OPEN_ERROR 変数

QAManager が開いていません。

構文

```
final int QAException.COMMON_NOT_OPEN_ERROR
```

参照

- QAManager インターフェイス [QAnywhere Java]536 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

COMMON_OPEN_ERROR 変数

メッセージストアへの接続を開くときにエラーが発生しました。

構文

```
final int QAException.COMMON_OPEN_ERROR
```

COMMON_OPEN_LOG_FILE_ERROR 変数

ログファイルを開くときにエラーが発生しました。

構文

```
final int QAException.COMMON_OPEN_LOG_FILE_ERROR
```

COMMON_OPEN_MAXTHREADS_ERROR 変数

サーバーの最大同時要求数が少なすぎるため QAManager を開くことができません (データベースサーバーの -gn オプションを参照)。

構文

```
final int QAException.COMMON_OPEN_MAXTHREADS_ERROR
```

COMMON_REOPEN_ERROR 変数

メッセージストアへの接続を再び開くときにエラーが発生しました。

構文

```
final int QAException.COMMON_REOPEN_ERROR
```

COMMON_SELECTOR_SYNTAX_ERROR 変数

指定されたセレクターに構文エラーがあります。

構文

```
final int QAException.COMMON_SELECTOR_SYNTAX_ERROR
```

COMMON_SET_PROPERTY_ERROR 変数

プロパティをメッセージストアに格納中にエラーが発生しました。

構文

```
final int QAException.COMMON_SET_PROPERTY_ERROR
```

COMMON_TERMINATE_ERROR 変数

終了エラーです。

構文

```
final int QAException.COMMON_TERMINATE_ERROR
```

COMMON_UNEXPECTED_EOM_ERROR 変数

予期しないメッセージの終わりに到達しました。

構文

```
final int QAException.COMMON_UNEXPECTED_EOM_ERROR
```

COMMON_UNREPRESENTABLE_TIMESTAMP 変数

タイムスタンプが許容範囲外です。

構文

```
final int QAException.COMMON_UNREPRESENTABLE_TIMESTAMP
```

QA_NO_ERROR 変数

エラーはありません。

構文

```
final int QAException.QA_NO_ERROR
```

QAManager インターフェイス

QAManager は、非トランザクション指向の QAnywhere メッセージング操作を管理します。

構文

```
public interface QAManager
```

基本クラス

- QAManagerBase インターフェイス [QAnywhere Java]544 ページ

メンバー

継承されたメンバーを含む QAManager インターフェイスのすべてのメンバー。

名前	説明
acknowledge メソッド	クライアントアプリケーションが QAnywhere メッセージを正常に受信したことを確認します。
acknowledgeAll メソッド	クライアントアプリケーションが QAnywhere メッセージを正常に受信したことを確認します。
acknowledgeUntil メソッド	指定された QAMessage インスタンスと、指定されたメッセージよりも前に受信されて受信確認されていないメッセージについて、すべて受信確認します。
browseMessages メソッド	メッセージストア内にある、参照可能なメッセージをすべて参照します。
browseMessagesByID メソッド	指定されたメッセージ ID を持つメッセージを参照します。
browseMessagesByQueue メソッド	指定されたアドレスに送信され、取得可能な一連の受信待機中メッセージを参照します。
browseMessagesBySelector メソッド	メッセージストアのキューに登録されているメッセージのうち、指定されたセレクターを満たすメッセージを参照します。
cancelMessage メソッド	指定されたメッセージ ID を持つメッセージをキャンセルします。
close メソッド	QAnywhere メッセージシステムへの接続をクローズして、QAManagerBase で使用していたリソースをすべて解放します。
createBinaryMessage メソッド	QABinaryMessage オブジェクトを作成します。
createTextMessage メソッド	QATextMessage オブジェクトを作成します。
getBooleanStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの boolean 値を取得します。

名前	説明
getByteStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの signed byte 値を取得します。
getDoubleStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの double 値を取得します。
getFloatStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの float 値を取得します。
getIntStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの int 値を取得します。
getLongStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの long 値を取得します。
getMessage メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
getMessageBySelector メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
getMessageBySelectorNoWait メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
getMessageBySelectorTimeout メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
getMessageNoWait メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
getMessageTimeout メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
getMode メソッド	受信したメッセージの QAManager 受信確認モードを返します。
getQueueDepth メソッド	指定されたフィルターに基づいて、すべてのキューの長さの合計を返します。
getShortStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの short 値を取得します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
getStoreProperty メソッド	メッセージストアプロパティを表すオブジェクトを取得します。
getStorePropertyNames メソッド	メッセージストアのプロパティ名の列挙子を取得します。
getStringStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの string 値を取得します。
open メソッド	指定された AcknowledgementMode 値を使用して QAManager をオープンします。
propertyExists メソッド	指定されたプロパティに値が現在存在するかどうかをテストします。
putMessage メソッド	別の QAnywhere クライアントに送信するメッセージを準備します。
putMessageTimeToLive メソッド	別の QAnywhere クライアントに送信するメッセージを準備します。
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 StatusCodes.PENDING のステータスにします。
reOpen メソッド	QAManagerBase を再度開きます。
setBooleanStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを boolean 値に設定します。
setByteStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを sbyte 値に設定します。
setDoubleStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを double 値に設定します。
setFloatStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを float 値に設定します。
setIntStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを int 値に設定します。
setLongStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを long 値に設定します。

名前	説明
setMessageListener メソッド	QAMessageListener オブジェクトを登録して QAnywhere メッセージを非同期的に受信します。
setMessageListener2 メソッド	QAMessageListener2 オブジェクトを登録して QAnywhere メッセージを非同期的に受信します。
setMessageListenerBySelector メソッド	QAMessageListener オブジェクトを登録し、メッセージセレクターを使用して QAnywhere メッセージを非同期的に受信します。
setMessageListenerBySelector2 メソッド	QAMessageListener2 オブジェクトを登録し、メッセージセレクターを使用して QAnywhere メッセージを非同期的に受信します。
setProperty メソッド	QAnywhere Manager の設定プロパティをプログラムで設定できるようにします。
setShortStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを short 値に設定します。
setStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを System.Object 値に設定します。
setStringStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを String 値に設定します。
start メソッド	着信メッセージを受信するための QAManagerBase を開始します。
stop メソッド	QAManagerBase による着信メッセージの受信を停止します。
triggerSendReceive メソッド	QAnywhere メッセージサーバーとの同期処理を発生させて、他のクライアント宛てのメッセージをアップロードし、ローカルクライアント宛てのメッセージをダウンロードします。

備考

これは QAManagerBase から派生します。

この動作の完全な説明については、QAManagerBase を参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

QAManager インスタンスは、AcknowledgementMode クラスの定義のように、明示的に受信を確認するように設定することも、暗黙的に受信を確認するように設定することもできます。トランザクションの一部としてメッセージの受信を確認するには、QATransactionalManager を使用します。

QAManager と QATransactionalManager オブジェクトを作成するには、QAManagerFactory クラスを使用します。

参照

- AcknowledgementMode インターフェイス [QAnywhere Java]495 ページ
- QAManagerFactory クラス [QAnywhere Java]578 ページ
- QATransactionalManager インターフェイス [QAnywhere Java]609 ページ

acknowledge メソッド

クライアントアプリケーションが QAnywhere メッセージを正常に受信したことを確認します。

構文

```
void QAManager.acknowledge(QAMessage msg) throws QAException
```

パラメーター

- **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で問題が発生した場合にスローされます。

備考

> **注意**
> QAMessage が受信を確認すると、そのステータスプロパティが StatusCodes.RECEIVED に変わり、デフォルトの削除ルールを使用して削除できるようになります。

参照

- StatusCodes.RECEIVED 変数 [QAnywhere Java]618 ページ
- QAManager.acknowledgeUntil メソッド [QAnywhere Java]542 ページ
- QAManager.acknowledgeAll メソッド [QAnywhere Java]541 ページ

acknowledgeAll メソッド

クライアントアプリケーションが QAnywhere メッセージを正常に受信したことを確認します。

構文

```
void QAManager.acknowledgeAll() throws QAException
```

例外

- **QAException クラス**　メッセージの受信確認で問題が発生した場合にスローされます。

備考

受信確認されていないメッセージが、すべて受信確認されます。

> **注意**
> QAMessage が受信確認されると、そのステータスプロパティが StatusCodes.RECEIVED に変わり、デフォルトの削除ルールを使用して削除できるようになります。

参照

- StatusCodes.RECEIVED 変数 [QAnywhere Java]618 ページ
- QAManager.acknowledge メソッド [QAnywhere Java]541 ページ
- QAManager.acknowledgeUntil メソッド [QAnywhere Java]542 ページ

acknowledgeUntil メソッド

指定された QAMessage インスタンスと、指定されたメッセージよりも前に受信されて受信確認されていないメッセージについて、すべて受信確認します。

構文

```
void QAManager.acknowledgeUntil(QAMessage msg) throws QAException
```

パラメーター

- **msg**　受信確認するメッセージのうち最新のもの。それ以前の受信確認されていないメッセージも、すべて受信確認されます。

例外

- **QAException クラス**　メッセージの受信確認で問題が発生した場合にスローされます。

備考

> **注意**
> QAMessage が受信確認されると、そのステータスプロパティが StatusCodes.RECEIVED に変わり、デフォルトの削除ルールを使用して削除できるようになります。

参照

- QAMessage インターフェイス [QAnywhere Java]582 ページ
- StatusCodes.RECEIVED 変数 [QAnywhere Java]618 ページ
- QAManager.acknowledge メソッド [QAnywhere Java]541 ページ
- QAManager.acknowledgeAll メソッド [QAnywhere Java]541 ページ

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

open メソッド

指定された AcknowledgementMode 値を使用して QAManager をオープンします。

構文

```
void QAManager.open(short mode) throws QAException
```

パラメーター

- **mode**　受信確認モード。AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT または AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT のどちらかです。

例外

- **QAException クラス**　QAManager インスタンスのオープンで問題がある場合にスローされます。

備考

open(short) メソッドは、QAManager を作成した後、最初に呼び出す必要があるメソッドです。

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

参照

- AcknowledgementMode インターフェイス [QAnywhere Java]495 ページ
- AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT 変数 [QAnywhere Java]496 ページ
- AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT 変数 [QAnywhere Java]496 ページ

recover メソッド

受信確認されていないメッセージを、すべて強制的に StatusCodes.PENDING のステータスにします。

構文

```
void QAManager.recover() throws QAException
```

例外

- **QAException クラス**　メッセージのステータスを戻すときに問題が発生した場合にスローされます。

備考

これらのメッセージは、QAManagerBase.getMessage(String) を使用して再受信します。

参照

- QAManagerBase.getMessage メソッド [QAnywhere Java]556 ページ

QAManagerBase インターフェイス

このクラスは、QATransactionalManager と QAManager の基本クラスです。前者の派生クラスはトランザクション指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指向のメッセージングを管理します。

構文

```
public interface QAManagerBase
```

派生クラス

- QAManager インターフェイス [QAnywhere Java]536 ページ
- QATransactionalManager インターフェイス [QAnywhere Java]609 ページ

メンバー

継承されたメンバーを含む QAManagerBase インターフェイスのすべてのメンバー。

名前	説明
browseMessages メソッド	メッセージストア内にある、参照可能なメッセージをすべて参照します。
browseMessagesByID メソッド	指定されたメッセージ ID を持つメッセージを参照します。
browseMessagesByQueue メソッド	指定されたアドレスに送信され、取得可能な一連の受信待機中メッセージを参照します。
browseMessagesBySelector メソッド	メッセージストアのキューに登録されているメッセージのうち、指定されたセレクターを満たすメッセージを参照します。
cancelMessage メソッド	指定されたメッセージ ID を持つメッセージをキャンセルします。
close メソッド	QAnywhere メッセージシステムへの接続をクローズして、QAManagerBase で使用していたリソースをすべて解放します。
createBinaryMessage メソッド	QABinaryMessage オブジェクトを作成します。
createTextMessage メソッド	QATextMessage オブジェクトを作成します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
getBooleanStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの boolean 値を取得します。
getByteStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの signed byte 値を取得します。
getDoubleStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの double 値を取得します。
getFloatStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの float 値を取得します。
getIntStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの int 値を取得します。
getLongStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの long 値を取得します。
getMessage メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
getMessageBySelector メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
getMessageBySelectorNoWait メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
getMessageBySelectorTimeout メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
getMessageNoWait メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
getMessageTimeout メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
getMode メソッド	受信したメッセージの QAManager 受信確認モードを返します。
getQueueDepth メソッド	指定されたフィルターに基づいて、すべてのキューの長さの合計を返します。

名前	説明
getShortStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの short 値を取得します。
getStoreProperty メソッド	メッセージストアプロパティを表すオブジェクトを取得します。
getStorePropertyNames メソッド	メッセージストアのプロパティ名の列挙子を取得します。
getStringStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの string 値を取得します。
propertyExists メソッド	指定されたプロパティに値が現在存在するかどうかをテストします。
putMessage メソッド	別の QAnywhere クライアントに送信するメッセージを準備します。
putMessageTimeToLive メソッド	別の QAnywhere クライアントに送信するメッセージを準備します。
reOpen メソッド	QAManagerBase を再度開きます。
setBooleanStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを boolean 値に設定します。
setByteStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを sbyte 値に設定します。
setDoubleStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを double 値に設定します。
setFloatStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを float 値に設定します。
setIntStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを int 値に設定します。
setLongStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを long 値に設定します。
setMessageListener メソッド	QAMessageListener オブジェクトを登録して QAnywhere メッセージを非同期的に受信します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
setMessageListener2 メソッド	QAMessageListener2 オブジェクトを登録して QAnywhere メッセージを非同期的に受信します。
setMessageListenerBySelector メソッド	QAMessageListener オブジェクトを登録し、メッセージセレクターを使用して QAnywhere メッセージを非同期的に受信します。
setMessageListenerBySelector2 メソッド	QAMessageListener2 オブジェクトを登録し、メッセージセレクターを使用して QAnywhere メッセージを非同期的に受信します。
setProperty メソッド	QAnywhere Manager の設定プロパティをプログラムで設定できるようにします。
setShortStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを short 値に設定します。
setStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを System.Object 値に設定します。
setStringStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを String 値に設定します。
start メソッド	着信メッセージを受信するための QAManagerBase を開始します。
stop メソッド	QAManagerBase による着信メッセージの受信を停止します。
triggerSendReceive メソッド	QAnywhere メッセージサーバーとの同期処理を発生させて、他のクライアント宛てのメッセージをアップロードし、ローカルクライアント宛てのメッセージをダウンロードします。

備考

QAManagerBase インスタンスがメッセージを受信できるようにするには、QAManagerBase.start() メソッドを使用します。特定の QAManagerBase インスタンスを使用できるのは、そのインスタンスを生成したスレッドだけです。

このクラスのインスタンスを使用して、QAnywhere メッセージの作成と管理を行うことができます。適切な QAMessage インスタンスを作成するには、QAManagerBase.createBinaryMessage() メソッドと QAManagerBase.createTextMessage() メソッドを使用します。QAMessage インスタンスには、メッセージの内容とプロパティを設定するための、さまざまなメソッドがあります。

QAnywhere メッセージを送信するには、QAManagerBase.putMessage(String, QAMessage) メソッドを使用して、アドレス指定されたメッセージをローカルのメッセージストアキューに登録します。メッセージは、転送ポリシーに基づいて QAnywhere Agent によって転送されるか、QAManagerBase.triggerSendReceive() が呼び出されたときに転送されます。

QAManagerBase にも、メッセージストアプロパティを設定および取得するためのメソッドがあります。

参照

- QATransactionalManager インターフェイス [QAnywhere Java]609 ページ
- QAManager インターフェイス [QAnywhere Java]536 ページ

browseMessages メソッド

メッセージストア内にある、参照可能なメッセージをすべて参照します。

構文

```
java.util.Enumeration QAManagerBase.browseMessages() throws QAException
```

戻り値

参照可能な一連のメッセージの列挙子。

例外

- **QAException クラス**　メッセージの参照で問題が発生した場合にスローされます。

備考

メッセージは単に参照されるだけなので、受信確認はできません。

メッセージを受信して受信確認できるようにする場合は、QAManagerBase.getMessage(String) メソッドを使用します。

参照

- QAManagerBase.browseMessagesByQueue メソッド [QAnywhere Java]549 ページ
- QAManagerBase.browseMessagesByID メソッド [QAnywhere Java]548 ページ
- QAManagerBase.getMessage メソッド [QAnywhere Java]556 ページ

browseMessagesByID メソッド

指定されたメッセージ ID を持つメッセージを参照します。

構文

```
java.util.Enumeration QAManagerBase.browseMessagesByID(
     String id
) throws QAException
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- **id**　メッセージのメッセージ ID。

戻り値

0 または 1 を含むメッセージの列挙子。

例外

- **QAException クラス**　メッセージの参照で問題が発生した場合にスローされます。

備考

メッセージは単に参照されるだけなので、受信確認はできません。メッセージを受信して受信確認できるようにする場合は、QAManagerBase.getMessage(String) を使用します。

参照

- QAManagerBase.browseMessagesByQueue メソッド [QAnywhere Java]549 ページ
- QAManagerBase.browseMessages メソッド [QAnywhere Java]548 ページ
- QAManagerBase.getMessage メソッド [QAnywhere Java]556 ページ

browseMessagesByQueue メソッド

指定されたアドレスに送信され、取得可能な一連の受信待機中メッセージを参照します。

構文

```
java.util.Enumeration QAManagerBase.browseMessagesByQueue(
     String address
) throws QAException
```

パラメーター

- **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。

例外

- **QAException クラス**　メッセージの参照で問題が発生した場合にスローされます。

備考

メッセージは単に参照されるだけなので、受信確認はできません。

メッセージを受信して受信確認できるようにする場合は、QAManagerBase.getMessage(String) メソッドを使用します。

参照

- QAManagerBase.browseMessagesByID メソッド [QAnywhere Java]548 ページ
- QAManagerBase.browseMessages メソッド [QAnywhere Java]548 ページ
- QAManagerBase.getMessage メソッド [QAnywhere Java]556 ページ

browseMessagesBySelector メソッド

メッセージストアのキューに登録されているメッセージのうち、指定されたセレクターを満たすメッセージを参照します。

構文

```
java.util.Enumeration QAManagerBase.browseMessagesBySelector(
     String selector
) throws QAException
```

パラメーター

- **selector**　セレクター。

戻り値

参照可能な一連のメッセージの列挙子。

例外

- **QAException クラス**　メッセージの参照で問題が発生した場合にスローされます。

備考

メッセージは単に参照されるだけなので、受信確認はできません。メッセージを受信して受信確認できるようにする場合は、QAManagerBase.getMessage(String) を使用します。

参照

- QAManagerBase.browseMessagesByQueue メソッド [QAnywhere Java]549 ページ
- QAManagerBase.browseMessages メソッド [QAnywhere Java]548 ページ
- QAManagerBase.browseMessagesByID メソッド [QAnywhere Java]548 ページ
- QAManagerBase.getMessage メソッド [QAnywhere Java]556 ページ

cancelMessage メソッド

指定されたメッセージ ID を持つメッセージをキャンセルします。

構文

```
boolean QAManagerBase.cancelMessage(String id) throws QAException
```

パラメーター

- **id**　キャンセルするメッセージのメッセージ ID。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例外

- **QAException クラス**　メッセージのキャンセルで問題が発生した場合にスローされます。

備考

メッセージが転送される前にメッセージをキャンセル済みの状態にします。

QAnywhere Agent のデフォルトの削除ルールにより、キャンセル済みメッセージは最終的にメッセージストアから削除されます。

メッセージがすでに最終ステータスになっている場合や、中央のメッセージングサーバーに転送済みである場合は、メッセージをキャンセルできません。

close メソッド

QAnywhere メッセージシステムへの接続をクローズして、QAManagerBase で使用していたリソースをすべて解放します。

構文

```
void QAManagerBase.close() throws QAException
```

例外

- **QAException クラス**　QAManagerBase インスタンスのクローズで問題がある場合にスローされます。

備考

いったんクローズした後は、何度この close() メソッドを呼び出しても無視されます。このメソッドは、メッセージリスナーと例外リスナーでは呼び出すことができません。

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

createBinaryMessage メソッド

QABinaryMessage オブジェクトを作成します。

構文

```
QABinaryMessage QAManagerBase.createBinaryMessage() throws QAException
```

戻り値

新しい QABinaryMessage インスタンス。

例外

- **QAException クラス**　メッセージの作成で問題が発生した場合にスローされます。

備考

QABinaryMessage オブジェクトは、未解釈のバイトストリームのメッセージ本文が含まれるメッセージの送信に使用します。

参照

- QABinaryMessage インターフェイス [QAnywhere Java]509 ページ

createTextMessage メソッド

QATextMessage オブジェクトを作成します。

構文

```
QATextMessage QAManagerBase.createTextMessage() throws QAException
```

戻り値

新しい QATextMessage インスタンス。

例外

- **QAException クラス**　メッセージの作成で問題が発生した場合にスローされます。

備考

QATextMessage のオブジェクトは、string のメッセージ本文が含まれるメッセージを送信する場合に使用します。

参照

- QATextMessage インターフェイス [QAnywhere Java]603 ページ

getBooleanStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの boolean 値を取得します。

構文

```
boolean QAManagerBase.getBooleanStoreProperty(
     String name
) throws QAException
```

パラメーター

- **name**　事前定義済みまたはカスタムのプロパティ名。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

戻り値

boolean 型のプロパティの値。

例外

- **QAException クラス**　プロパティ値の取得で変換エラーまたはメッセージストアエラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

参照

- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ

getByteStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの signed byte 値を取得します。

構文

```
byte QAManagerBase.getByteStoreProperty(String name) throws QAException
```

パラメーター

- **name**　事前定義済みまたはカスタムのプロパティ名。

戻り値

signed byte 型のプロパティの値。

例外

- **QAException クラス**　プロパティ値の取得で変換エラーまたはメッセージストアエラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

参照

- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ

getDoubleStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの double 値を取得します。

構文

```
double QAManagerBase.getDoubleStoreProperty(
     String name
) throws QAException
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。

戻り値

double 型のプロパティの値。

例外

- **QAException クラス** プロパティ値の取得で変換エラーまたはメッセージストアエラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

参照

- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ

getFloatStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの float 値を取得します。

構文

```
float QAManagerBase.getFloatStoreProperty(
     String name
) throws QAException
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。

戻り値

float 型のプロパティの値。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例外

- **QAException クラス**　プロパティ値の取得で変換エラーまたはメッセージストアエラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

参照

- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ

getIntStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの int 値を取得します。

構文

```
int QAManagerBase.getIntStoreProperty(String name) throws QAException
```

パラメーター

- **name**　事前定義済みまたはカスタムのプロパティ名。

戻り値

int 型のプロパティの値。

例外

- **QAException クラス**　プロパティ値の取得で変換エラーまたはメッセージストアエラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

参照

- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ

getLongStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの long 値を取得します。

構文

```
long QAManagerBase.getLongStoreProperty(String name) throws QAException
```

パラメーター

- **name**　事前定義済みまたはカスタムのプロパティ名。

戻り値

long 型のプロパティの値。

例外

- **QAException クラス**　プロパティ値の取得で変換エラーまたはメッセージストアエラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

参照

- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ

getMessage メソッド

指定されたアドレスに送信された、次に取得可能な QAMessage を返します。

構文

```
QAMessage QAManagerBase.getMessage(String address) throws QAException
```

パラメーター

- **address**　このアドレスは、メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。

戻り値

該当する次の QAMessage。メッセージが存在しない場合は NULL。

例外

- **QAException クラス**　メッセージの取得で問題が発生した場合にスローされます。

備考

address パラメーターには、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージが存在しない場合、新しいメッセージが着信するまでブロックされます。メッセージを同期的に受信する場合は、このメソッドを使用します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- QAMessage インターフェイス [QAnywhere Java]582 ページ

getMessageBySelector メソッド

指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。

構文

```
QAMessage QAManagerBase.getMessageBySelector(
     String address,
    String selector
) throws QAException
```

パラメーター

- **address**　このアドレスは、メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。
- **selector**　セレクター。

戻り値

該当する次の QAMessage。メッセージが存在しない場合は NULL。

例外

- **QAException クラス**　メッセージの取得で問題が発生した場合にスローされます。

備考

address パラメーターには、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージが存在しない場合、新しいメッセージが着信するまでブロックされます。

メッセージを同期的に受信する場合は、このメソッドを使用します。

参照

- QAMessage インターフェイス [QAnywhere Java]582 ページ

getMessageBySelectorNoWait メソッド

指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。

構文

```
QAMessage QAManagerBase.getMessageBySelectorNoWait(
     String address,
```

```
    String selector
) throws QAException
```

パラメーター

- **address**　このアドレスは、メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。
- **selector**　セレクター。

戻り値

次の取得可能な QAMessage。該当するメッセージが存在しない場合は NULL。

例外

- **QAException クラス**　メッセージの取得で問題が発生した場合にスローされます。

備考

address パラメーターには、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージがない場合は、このメソッドがすぐに返されます。

メッセージを同期的に受信する場合は、このメソッドを使用します。

参照

- QAMessage インターフェイス [QAnywhere Java]582 ページ

getMessageBySelectorTimeout メソッド

指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。

構文

```
QAMessage QAManagerBase.getMessageBySelectorTimeout(
     String address,
    String selector,
    long timeout
) throws QAException
```

パラメーター

- **address**　このアドレスは、メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。
- **selector**　セレクター。
- **timeout**　メッセージ着信を待機する時間 (ミリ秒単位)。

戻り値

次の取得可能な QAMessage。メッセージが存在しない場合は NULL。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例外

- **QAException クラス**　メッセージの取得で問題が発生した場合にスローされます。

備考

address パラメーターには、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージがない場合、このメソッドは指定されたタイムアウト時間だけ待機してから返されます。

メッセージを同期的に受信する場合は、このメソッドを使用します。

参照

- QAMessage インターフェイス [QAnywhere Java]582 ページ

getMessageNoWait メソッド

指定されたアドレスに送信された、次に取得可能な QAMessage を返します。

構文

```
QAMessage QAManagerBase.getMessageNoWait(
     String address
) throws QAException
```

パラメーター

- **address**　このアドレスは、メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。

戻り値

次の取得可能な QAMessage。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**　メッセージの取得で問題が発生した場合にスローされます。

備考

address パラメーターには、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージがない場合は、このメソッドがすぐに返されます。

メッセージを同期的に受信する場合は、このメソッドを使用します。

参照

- QAMessage インターフェイス [QAnywhere Java]582 ページ

getMessageTimeout メソッド

指定されたアドレスに送信された、次に取得可能な QAMessage を返します。

構文

```
QAMessage QAManagerBase.getMessageTimeout(
     String address,
    long timeout
) throws QAException
```

パラメーター

- **address**　このアドレスは、メッセージの受信で QAnywhere クライアントが使用するキュー名を指定します。
- **timeout**　メッセージ着信を待機する時間 (ミリ秒単位)。

戻り値

該当する次の QAMessage。メッセージが存在しない場合は NULL。

例外

- **QAException クラス**　メッセージの取得で問題が発生した場合にスローされます。

備考

address パラメーターには、ローカルのキュー名を指定します。アドレスは、'store-id¥queue-name' または 'queue-name' の形式で指定できます。該当するメッセージがない場合、このメソッドは指定されたタイムアウト時間だけ待機してから返されます。メッセージを同期的に受信する場合は、このメソッドを使用します。

参照

- QAMessage インターフェイス [QAnywhere Java]582 ページ

getMode メソッド

受信したメッセージの QAManager 受信確認モードを返します。

構文

```
short QAManagerBase.getMode() throws QAException
```

戻り値

受信メッセージの QAManager 受信確認モード。

例外

- **QAException クラス**　QAManager 受信確認モードの取得で問題が発生した場合にスローされます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

戻り値のリストについては、AcknowledgementMode を参照してください。

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

参照

- AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT 変数 [QAnywhere Java]496 ページ
- AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT 変数 [QAnywhere Java]496 ページ
- QAManager インターフェイス [QAnywhere Java]536 ページ
- QATransactionalManager インターフェイス [QAnywhere Java]609 ページ

getQueueDepth メソッド

指定されたフィルターに基づいて、すべてのキューの長さの合計を返します。

オーバーロードリスト

名前	説明
getQueueDepth(short) メソッド	指定されたフィルターに基づいて、すべてのキューの長さの合計を返します。
getQueueDepth(String, short) メソッド	指定されたフィルターに基づいて、キューの長さを返します。

getQueueDepth(short) メソッド

指定されたフィルターに基づいて、すべてのキューの長さの合計を返します。

構文

```
int QAManagerBase.getQueueDepth(short filter) throws QAException
```

パラメーター

- **filter**　着信メッセージ、送信メッセージ、またはすべてのメッセージを示すフィルター。

戻り値

すべてのキューに登録されているメッセージのうち、指定されたフィルターを満たすメッセージの数。

例外

- **QAException クラス**　エラーが発生した場合にスローされます。

備考

キューの着信の深さは、受信されていない着信メッセージの数です (たとえば、QAManagerBase.getMessage(String) メソッドを使用)。キューの送信の深さは、サーバーに転送されていない送信メッセージ (コミットされていないメッセージを含む) の数です。

使用可能なフィルターの値のリストについては、QueueDepthFilter を参照してください。

参照

- QAManagerBase.getMessage メソッド [QAnywhere Java]556 ページ

getQueueDepth(String, short) メソッド

指定されたフィルターに基づいて、キューの長さを返します。

構文

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

パラメーター

- **filter**　着信メッセージ、送信メッセージ、またはすべてのメッセージを示すフィルター。
- **queue**　キュー名。

戻り値

メッセージ数

例外

- **QAException クラス**　エラーが発生した場合にスローされます。

備考

キューの着信の深さは、受信されていない着信メッセージの数です (たとえば、QAManagerBase.getMessage(String) メソッドを使用)。キューの送信の深さは、サーバーに転送されていない送信メッセージ (コミットされていないメッセージを含む) の数です。

使用可能なフィルターの値のリストについては、QueueDepthFilter を参照してください。

getShortStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの short 値を取得します。

構文

```
short QAManagerBase.getShortStoreProperty(
     String name
) throws QAException
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- **name**　事前定義済みまたはカスタムのプロパティ名。

戻り値

short 型のプロパティの値。

例外

- **QAException クラス**　プロパティ値の取得で変換エラーまたはメッセージストアエラーがある場合、またはプロパティが存在しない場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

参照

- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ

getStoreProperty メソッド

メッセージストアプロパティを表すオブジェクトを取得します。

構文

```
Object QAManagerBase.getStoreProperty(String name) throws QAException
```

パラメーター

- **name**　事前定義済みまたはカスタムのプロパティ名。

戻り値

プロパティの値。

例外

- **QAException クラス**　プロパティの取得で問題が発生した場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

参照

- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ

getStorePropertyNames メソッド

メッセージストアのプロパティ名の列挙子を取得します。

構文

```
java.util.Enumeration QAManagerBase.getStorePropertyNames()
    throws QAException
```

戻り値

メッセージストアのプロパティ名の列挙子。

例外

- **QAException クラス**　列挙子の取得で問題が発生した場合にスローされます。

getStringStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティの string 値を取得します。

構文

```
String QAManagerBase.getStringStoreProperty(
     String name
) throws QAException
```

パラメーター

- **name**　事前定義済みまたはカスタムのプロパティ名。

戻り値

string 型のプロパティの値。該当するプロパティが存在しない場合は NULL。

例外

- **QAException クラス**　string 値の取得で問題が発生した場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティにアクセスできます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

参照

- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

propertyExists メソッド

指定されたプロパティに値が現在存在するかどうかをテストします。

構文

```
boolean QAManagerBase.propertyExists(String name) throws QAException
```

パラメーター

- **name**　事前定義済みまたはカスタムのプロパティ名。

戻り値

プロパティにマッピングされた値がメッセージストアに存在する場合は true、それ以外の場合は false。

例外

- **QAException クラス**　プロパティ値の取得で問題が発生した場合にスローされます。

備考

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

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

参照

- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ

putMessage メソッド

別の QAnywhere クライアントに送信するメッセージを準備します。

構文

```
void QAManagerBase.putMessage(
     String address,
    QAMessage msg
) throws QAException
```

パラメーター

- **address**　送信先のキュー名を指定するメッセージのアドレス。
- **msg**　転送用ローカルメッセージストアに格納するメッセージ。

例外

- **QAException クラス**　メッセージの登録で問題が発生した場合にスローされます。

備考

このメソッドは、メッセージと送信先アドレスをローカルメッセージストアに挿入します。メッセージが転送されるタイミングは、QAnywhere Agent の転送ポリシーで決まります。

アドレスは 'id¥queue-name' の形式で指定します。'id' は送信先メッセージストアの ID、'queue-name' は送信先の QAnywhere クライアントがメッセージの受信で使用するキューを特定します。

参照

- QAManagerBase.putMessageTimeToLive メソッド [QAnywhere Java]566 ページ

putMessageTimeToLive メソッド

別の QAnywhere クライアントに送信するメッセージを準備します。

構文

```
void QAManagerBase.putMessageTimeToLive(
     String address,
    QAMessage msg,
    long ttl
) throws QAException
```

パラメーター

- **address**　送信先のキュー名を指定するメッセージのアドレス。
- **msg**　登録するメッセージ。
- **ttl**　メッセージの有効期限 (ミリ秒単位)。この期限を過ぎても配信されなかったメッセージは期限切れになります。値 0 は、有効期限なしを意味します。

例外

- **QAException クラス**　メッセージの登録で問題が発生した場合にスローされます。

備考

このメソッドは、メッセージと送信先アドレスをローカルメッセージストアに挿入します。メッセージが転送されるタイミングは、QAnywhere Agent の転送ポリシーで決まります。ただし、次のメッセージの転送時間が指定された存続時間を超えると、そのメッセージは期限切れになります。

アドレスは 'id¥queue-name' の形式で指定します。'id' は送信先メッセージストアの ID、'queue-name' は送信先の QAnywhere クライアントがメッセージの受信で使用するキューを特定します。

reOpen メソッド

QAManagerBase を再度開きます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
void QAManagerBase.reOpen() throws QAException
```

例外

- **QAException クラス**　QAManagerBase インスタンスを再度開くときに問題がある場合にスローされます。

備考

これにより、リソースを解放することなく、メッセージストアへの接続が再確立されます。このメソッドは、メッセージリスナーまたは例外リスナーで呼び出されることがありますが、その場合は start() を再度呼び出す必要はありません。リスナーで呼び出されない場合、このメソッドでは close() の後に open() が実行されるだけなので、この場合は、start() を呼び出して、メッセージの受信を再開する必要があります。

setBooleanStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを boolean 値に設定します。

構文

```
void QAManagerBase.setBooleanStoreProperty(
     String name,
    boolean value
) throws QAException
```

パラメーター

- **name**　事前定義済みまたはカスタムのプロパティ名。
- **value**　boolean 型のプロパティの値。

例外

- **QAException クラス**　メッセージストアプロパティの設定で問題が発生した場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

参照

- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ

setByteStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを sbyte 値に設定します。

構文

```
void QAManagerBase.setByteStoreProperty(
      String name,
     byte value
) throws QAException
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **value** sbyte 型のプロパティの値。

例外

- **QAException クラス** メッセージストアプロパティの設定で問題が発生した場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

参照

- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ

setDoubleStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを double 値に設定します。

構文

```
void QAManagerBase.setDoubleStoreProperty(
      String name,
     double value
) throws QAException
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **value** double 型のプロパティの値。

例外

- **QAException クラス** メッセージストアプロパティの設定で問題が発生した場合にスローされます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

参照

- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ

setFloatStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを float 値に設定します。

構文

```
void QAManagerBase.setFloatStoreProperty(
     String name,
    float value
) throws QAException
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **value** float 型のプロパティの値。

例外

- **QAException クラス** メッセージストアプロパティの設定で問題が発生した場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

参照

- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ

setIntStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを int 値に設定します。

構文

```
void QAManagerBase.setIntStoreProperty(
     String name,
    int value
) throws QAException
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **value** int 型のプロパティの値。

例外

- **QAException クラス** メッセージストアプロパティの設定で問題が発生した場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

参照

- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ

setLongStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを long 値に設定します。

構文

```
void QAManagerBase.setLongStoreProperty(
     String name,
    long value
) throws QAException
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **value** long 型のプロパティの値。

例外

- **QAException クラス** メッセージストアプロパティの設定で問題が発生した場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

参照

- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

setMessageListener メソッド

QAMessageListener オブジェクトを登録して QAnywhere メッセージを非同期的に受信します。

構文

```
void QAManagerBase.setMessageListener(
     String address,
    QAMessageListener listener
) throws QAException
```

パラメーター

- **address**　メッセージの受信で使用されるローカルのキュー名のアドレス、または QAnywhere システムメッセージを受信するシステムのアドレス。
- **listener**　リスナー。

例外

- **QAException クラス**　QAMessageListener オブジェクトの登録で問題が発生した場合にスローされます。

備考

address パラメーターには、メッセージの受信で使用されるローカルキュー名を指定します。1 つのキューには、1 つの Listener オブジェクトだけを割り当てることができます。Push 通知とネットワークステータス変更も含め、QAnywhere システムメッセージを受信したい場合は、キュー名に "system" を指定します。

メッセージを非同期的に受信する場合は、このメソッドを使用します。

参照

- QAMessageListener インターフェイス [QAnywhere Java]600 ページ

setMessageListener2 メソッド

QAMessageListener2 オブジェクトを登録して QAnywhere メッセージを非同期的に受信します。

構文

```
void QAManagerBase.setMessageListener2(
     String address,
    QAMessageListener2 listener
) throws QAException
```

パラメーター

- **address**　メッセージの受信で使用されるローカルのキュー名のアドレス、または QAnywhere システムメッセージを受信するシステムのアドレス。
- **listener**　リスナー。

例外

- **QAException クラス**　QAMessageListener2 オブジェクトの登録で問題が発生した場合にスローされます。

備考

address パラメーターには、メッセージの受信で使用されるローカルキュー名を指定します。1 つのキューには、1 つの Listener オブジェクトだけを割り当てることができます。Push 通知とネットワークステータス変更も含め、QAnywhere システムメッセージを受信したい場合は、キュー名に "system" を指定します。

メッセージを非同期的に受信する場合は、このメソッドを使用します。

参照

- QAMessageListener2 インターフェイス [QAnywhere Java]601 ページ

setMessageListenerBySelector メソッド

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

構文

```
void QAManagerBase.setMessageListenerBySelector(
     String address,
    String selector,
    QAMessageListener listener
) throws QAException
```

パラメーター

- **address**　メッセージの受信で使用されるローカルのキュー名のアドレス、または QAnywhere システムメッセージを受信するシステムのアドレス。
- **selector**　受信されるメッセージをフィルタリングするために使用するセレクター。
- **listener**　リスナー。

例外

- **QAException クラス**　指定されたキューに Listener オブジェクトがすでに割り当てられているなどの理由により、QAMessageListener オブジェクトの登録で問題が発生した場合にスローされます。

備考

address パラメーターには、メッセージの受信で使用されるローカルキュー名を指定します。1 つのキューには、1 つの Listener オブジェクトだけを割り当てることができます。selector パラメーターは、指定されたアドレスで受信されるメッセージをフィルタリングするために使用するセレクターを指定します。Push 通知とネットワークステータス変更も含め、QAnywhere システムメッセージを受信したい場合は、キュー名に "system" を指定します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

メッセージを非同期的に受信する場合は、このメソッドを使用します。

setMessageListenerBySelector2 メソッド

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

構文

```
void QAManagerBase.setMessageListenerBySelector2(
     String address,
    String selector,
    QAMessageListener2 listener
) throws QAException
```

パラメーター

- **address**　メッセージの受信で使用されるローカルのキュー名のアドレス、または QAnywhere システムメッセージを受信するシステムのアドレス。
- **selector**　受信されるメッセージをフィルタリングするために使用するセレクター。
- **listener**　リスナー。

例外

- **QAException クラス**　QAMessageListener2 オブジェクトの登録で問題が発生した場合にスローされます。

備考

address パラメーターには、メッセージの受信で使用されるローカルキュー名を指定します。1 つのキューには、1 つの Listener オブジェクトだけを割り当てることができます。selector パラメーターは、指定されたアドレスで受信されるメッセージをフィルタリングするために使用するセレクターを指定します。Push 通知とネットワークステータス変更も含め、QAnywhere システムメッセージを受信したい場合は、キュー名に "system" を指定します。

メッセージを非同期的に受信する場合は、このメソッドを使用します。

参照

- QAMessageListener2 インターフェイス [QAnywhere Java]601 ページ

setProperty メソッド

QAnywhere Manager の設定プロパティをプログラムで設定できるようにします。

構文

```
void QAManagerBase.setProperty(
     String name,
```

```
    String val
) throws QAException
```

パラメーター

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

例外

- **QAException クラス** プロパティを値に設定するときに問題が発生した場合にスローされます。

備考

プロパティ名と値を指定してこのメソッドを使用することで、QAnywhere Manager のデフォルトの設定プロパティを無効にできます。プロパティのリストについては、MessageStoreProperties を参照してください。

QAnywhere Manager の設定プロパティは、プロパティファイルと QAManagerFactory.CreateQAManager メソッドを使用して設定することもできます。詳細については、プロパティファイルと QAManagerFactory.CreateQAManager メソッドを使用したプロパティの設定に関する項を参照してください。

> **注意**
> QAManager.Open または QATransactionalManager.Open() を呼び出す前に、必要なプロパティを設定してください。

参照

- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ

setShortStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを short 値に設定します。

構文

```
void QAManagerBase.setShortStoreProperty(
     String name,
    short value
) throws QAException
```

パラメーター

- **name** 事前定義済みまたはカスタムのプロパティ名。
- **value** short 型のプロパティの値。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例外

- **QAException クラス**　メッセージストアプロパティの設定で問題が発生した場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

参照

- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ

setStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを System.Object 値に設定します。

構文

```
void QAManagerBase.setStoreProperty(
      String name,
     Object value
) throws QAException
```

パラメーター

- **name**　事前定義済みまたはカスタムのプロパティ名。
- **value**　プロパティの値。

例外

- **QAException クラス**　メッセージストアプロパティを value パラメーターで指定した型の値に設定するときに問題が発生した場合にスローされます。

備考

プロパティ型は、使用可能ないずれかのプリミティブ型、または String 型でなければなりません。このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

参照

- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ

setStringStoreProperty メソッド

事前定義済みまたはカスタムのメッセージストアプロパティを String 値に設定します。

構文

```
void QAManagerBase.setStringStoreProperty(
     String name,
    String value
) throws QAException
```

パラメーター

- **name**　事前定義済みまたはカスタムのプロパティ名。
- **value**　string 型のプロパティの値。

例外

- **QAException クラス**　メッセージストアプロパティを string 値に設定するときに問題が発生した場合にスローされます。

備考

このメソッドを使用して、事前定義済みまたはユーザー定義のクライアントストアプロパティを設定できます。

事前定義済みプロパティのリストについては、MessageStoreProperties を参照してください。

参照

- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ

start メソッド

着信メッセージを受信するための QAManagerBase を開始します。

構文

```
void QAManagerBase.start() throws QAException
```

例外

- **QAException クラス**　QAManagerBase インスタンスの起動で問題がある場合にスローされます。

備考

このメソッドを繰り返し呼び出そうとしても、間に QAManagerBase.stop() 呼び出しを挟まないかぎり 2 回目以降の呼び出しは無視されます。

参照

- QAManagerBase.stop メソッド [QAnywhere Java]577 ページ

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

stop メソッド

QAManagerBase による着信メッセージの受信を停止します。

構文

```
void QAManagerBase.stop() throws QAException
```

例外

- **QAException クラス**　QAManagerBase インスタンスの停止で問題がある場合にスローされます。

備考

メッセージは失われません。メッセージは、Manager が再起動されるまで受信されません。この stop() メソッドを繰り返し呼び出そうとしても、間に QAManagerBase.start() 呼び出しを挟まないかぎり 2 回目以降の呼び出しは無視されます。

参照

- QAManagerBase.start メソッド [QAnywhere Java]576 ページ

triggerSendReceive メソッド

QAnywhere メッセージサーバーとの同期処理を発生させて、他のクライアント宛てのメッセージをアップロードし、ローカルクライアント宛てのメッセージをダウンロードします。

構文

```
void QAManagerBase.triggerSendReceive() throws QAException
```

例外

- **QAException クラス**　送信／受信のトリガーで問題が発生した場合にスローされます。

備考

このメソッドを呼び出すと、QAnywhere Agent と中央のメッセージングサーバー間でメッセージがただちに同期されます。手動の triggerSendReceive() 呼び出しでは、QAnywhere Agent の転送ポリシーとは無関係に、メッセージ転送がただちに行われます。

QAnywhere Agent の転送ポリシーは、メッセージ転送をどのようにして実行するかを決定します。たとえば、クライアントが Push 通知を受信した場合や、ユーザーが QAManagerBase.putMessage() メソッドを呼び出してメッセージを送信した場合に、一定の間隔でメッセージ転送を自動的に実行することができます。

参照

- QAManagerBase.putMessage メソッド [QAnywhere Java]565 ページ

QAManagerFactory クラス

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

構文

```
public class QAManagerFactory
```

メンバー

継承されたメンバーを含む QAManagerFactory クラスのすべてのメンバー。

名前	説明
createQAManager メソッド	指定されたプロパティを持つ新しい QAManager インスタンスを返します。
createQATransactionalManager メソッド	指定されたプロパティを持つ新しい QATransactionalManager インスタンスを返します。
getInstance メソッド	QAManagerFactory のシングルトンインスタンスを返します。

備考

QAManagerFactory のインスタンスは 1 つだけ持つことができます。

参照

- QAManager インターフェイス [QAnywhere Java]536 ページ
- QATransactionalManager インターフェイス [QAnywhere Java]609 ページ

createQAManager メソッド

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

オーバーロードリスト

名前	説明
createQAManager() メソッド	デフォルトのプロパティを持つ新しい QAManager インスタンスを返します。
createQAManager(Hashtable) メソッド	指定されたプロパティをハッシュテーブルとして持つ新しい QAManager インスタンスを返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
createQAManager(String) メソッド	指定されたプロパティを持つ新しい QAManager インスタンスを返します。

createQAManager() メソッド

デフォルトのプロパティを持つ新しい QAManager インスタンスを返します。

構文

```
abstract QAManager QAManagerFactory.createQAManager() throws QAException
```

戻り値

新しい QAManager インスタンス。

例外

- **QAException クラス**　Manager の作成で問題が発生した場合にスローされます。

参照

- QAManager インターフェイス [QAnywhere Java]536 ページ

createQAManager(Hashtable) メソッド

指定されたプロパティをハッシュテーブルとして持つ新しい QAManager インスタンスを返します。

構文

```
abstract QAManager QAManagerFactory.createQAManager(
     java.util.Hashtable properties
) throws QAException
```

パラメーター

- **properties**　QAManager のインスタンスを設定するためのハッシュテーブル。

戻り値

新しい QAManager インスタンス。

例外

- **QAException クラス**　Manager の作成で問題が発生した場合にスローされます。

参照

- QAManager インターフェイス [QAnywhere Java]536 ページ

createQAManager(String) メソッド

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

構文

```
abstract QAManager QAManagerFactory.createQAManager(
     String iniFile
) throws QAException
```

パラメーター

- **iniFile** QAManager のインスタンスを設定するためのプロパティファイル。QAManager のインスタンスをデフォルトの設定で作成する場合は NULL。

戻り値

新しい QAManager インスタンス。

例外

- **QAException クラス** Manager の作成で問題が発生した場合にスローされます。

備考

iniFile パラメーターが NULL の場合、QAManager はデフォルトのプロパティを使用して作成されます。インスタンスの作成後に QAManagerBase の各プロパティ設定メソッドを使用して、QAManager プロパティをプログラムで設定できます。

参照

- QAManager インターフェイス [QAnywhere Java]536 ページ

createQATransactionalManager メソッド

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

オーバーロードリスト

名前	説明
createQATransactionalManager() メソッド	デフォルトのプロパティを持つ新しい QATransactionalManager インスタンスを返します。
createQATransactionalManager(Hashtable) メソッド	指定されたプロパティを持つ新しい QATransactionalManager インスタンスを返します。
createQATransactionalManager(String) メソッド	指定されたプロパティを持つ新しい QATransactionalManager インスタンスを返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

createQATransactionalManager() メソッド

デフォルトのプロパティを持つ新しい QATransactionalManager インスタンスを返します。

構文

```
abstract QATransactionalManager
QAManagerFactory.createQATransactionalManager()
    throws QAException
```

戻り値

新しい QATransactionalManager。

例外

- **QAException クラス**　Manager の作成で問題が発生した場合にスローされます。

参照

- QATransactionalManager インターフェイス [QAnywhere Java]609 ページ

createQATransactionalManager(Hashtable) メソッド

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

構文

```
abstract QATransactionalManager
QAManagerFactory.createQATransactionalManager(
     java.util.Hashtable properties
) throws QAException
```

パラメーター

- **properties**　QATransactionalManager のインスタンスを設定するためのハッシュテーブル。

戻り値

設定された QATransactionalManager。

例外

- **QAException クラス**　Manager の作成で問題が発生した場合にスローされます。

参照

- QATransactionalManager インターフェイス [QAnywhere Java]609 ページ

createQATransactionalManager(String) メソッド

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

構文

```
abstract QATransactionalManager
QAManagerFactory.createQATransactionalManager(
      String iniFile
) throws QAException
```

パラメーター

- **iniFile**　QATransactionalManager のインスタンスを設定するためのプロパティファイル。

戻り値

設定された QATransactionalManager。

例外

- **QAException クラス**　Manager の作成で問題が発生した場合にスローされます。

備考

iniFile パラメーターが NULL の場合、QATransactionalManager はデフォルトのプロパティを使用して作成されます。インスタンスの作成後に QAManagerBase の各プロパティ設定メソッドを使用して、QATransactionalManager のプロパティをプログラムで設定できます。

参照

- QATransactionalManager インターフェイス [QAnywhere Java]609 ページ

getInstance メソッド

QAManagerFactory のシングルトンインスタンスを返します。

構文

```
QAManagerFactory QAManagerFactory.getInstance() throws QAException
```

戻り値

QAManagerFactory のシングルトンインスタンス。

例外

- **QAException クラス**　マネージャーファクトリの作成で問題が発生した場合にスローされます。

QAMessage インターフェイス

QAMessage には、メッセージプロパティとヘッダーファイルを設定するためのインターフェイスがあります。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
public interface QAMessage
```

派生クラス

- QABinaryMessage インターフェイス [QAnywhere Java]509 ページ
- QATextMessage インターフェイス [QAnywhere Java]603 ページ

メンバー

継承されたメンバーを含む QAMessage インターフェイスのすべてのメンバー。

名前	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。
getAddress メソッド	QAMessage インスタンスの送信先アドレスを返します。
getBooleanProperty メソッド	boolean 型のメッセージプロパティを取得します。
getByteProperty メソッド	signed byte 型のメッセージプロパティを取得します。
getDoubleProperty メソッド	double 型のメッセージプロパティを取得します。
getExpiration メソッド	メッセージの有効期限の値を返します。メッセージの有効期限がない場合や、まだ送信されていない場合は、NULL になります。
getFloatProperty メソッド	float 型のメッセージプロパティを取得します。
getInReplyToID メソッド	このメッセージの返信メッセージのメッセージ ID を返します。
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。
getMessageID メソッド	メッセージのグローバルにユニークなメッセージ ID を返します。
getPriority メソッド	メッセージの優先度 (0 から 9) を返します。
getProperty メソッド	メッセージのプロパティを取得します。

名前	説明
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。
getRedelivered メソッド	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。
getReplyToAddress メソッド	メッセージの返信アドレスを返します。
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	String 型のメッセージプロパティを取得します。
getTimestamp メソッド	メッセージの作成時刻を示すタイムスタンプを返します。
propertyExists メソッド	指定されたプロパティがこのメッセージに設定されているかどうかを示します。
setBooleanProperty メソッド	boolean 型のプロパティを設定します。
setByteProperty メソッド	signed byte 型のプロパティを設定します。
setDoubleProperty メソッド	double 型のプロパティを設定します。
setFloatProperty メソッド	float 型のプロパティを設定します。
setInReplyToID メソッド	このメッセージの返信対象メッセージを特定する、返信先 ID を設定します。
setIntProperty メソッド	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。
setPriority メソッド	メッセージの優先度 (0 から 9) を設定します。
setProperty メソッド	プロパティを設定します。
setReplyToAddress メソッド	返信先アドレスを設定します。
setShortProperty メソッド	short 型のプロパティを設定します。
setStringProperty メソッド	string 型のプロパティを設定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
DEFAULT_PRIORITY 変数	デフォルトのメッセージ優先度です。
DEFAULT_TIME_TO_LIVE 変数	デフォルトの存続時間です。

備考

派生クラスの QABinaryMessage と QATextMessage には、メッセージ本文の読み込み／書き込みを行うための特別な関数があります。事前定義済みまたはカスタムのメッセージプロパティを設定するには、QAMessage の関数を使用します。

事前定義済みのプロパティ名のリストについては、MessageProperties を参照してください。

参照

- QABinaryMessage インターフェイス [QAnywhere Java]509 ページ
- QATextMessage インターフェイス [QAnywhere Java]603 ページ

clearProperties メソッド

メッセージのすべてのプロパティをクリアします。

構文

```
void QAMessage.clearProperties() throws QAException
```

例外

- **QAException クラス**　メッセージプロパティのクリアで問題が発生した場合にスローされます。

getAddress メソッド

QAMessage インスタンスの送信先アドレスを返します。

構文

```
String QAMessage.getAddress() throws QAException
```

戻り値

QAMessage インスタンスの送信先アドレス。

例外

- **QAException クラス**　送信先アドレスの取得で問題が発生した場合にスローされます。

備考

このフィールドは、メッセージの送信時には無視されます。送信操作が完了すると、このフィールドには QAManagerBase.putMessage(String, QAMessage) で指定された送信先アドレスが入ります。

getBooleanProperty メソッド

boolean 型のメッセージプロパティを取得します。

構文

```
boolean QAMessage.getBooleanProperty(String name) throws QAException
```

パラメーター

- **name** 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で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ

getByteProperty メソッド

signed byte 型のメッセージプロパティを取得します。

構文

```
byte QAMessage.getByteProperty(String name) throws QAException
```

パラメーター

- **name** 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で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ

getDoubleProperty メソッド

double 型のメッセージプロパティを取得します。

構文

```
double QAMessage.getDoubleProperty(String name) throws QAException
```

パラメーター

- **name**　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で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ

getExpiration メソッド

メッセージの有効期限の値を返します。メッセージの有効期限がない場合や、まだ送信されていない場合は、NULL になります。

構文

```
java.util.Date QAMessage.getExpiration() throws QAException
```

戻り値

メッセージの有効期限の値。メッセージの有効期限がない場合や、まだ送信されていない場合は NULL。

例外

- **QAException クラス**　有効期限の取得で問題が発生した場合にスローされます。

備考

メッセージの送信時には、有効期限は空のままです。送信操作が終了すると、メッセージの有効期限が入ります。

メッセージの有効期限は、QAManagerBase.putMessageTimeToLive(String, QAMessage, long) の存続時間の引数を現在の時間に追加することで設定されるため、このプロパティは読み込み専用です。

参照

- QAManagerBase.putMessageTimeToLive メソッド [QAnywhere Java]566 ページ

getFloatProperty メソッド

float 型のメッセージプロパティを取得します。

構文

```
float QAMessage.getFloatProperty(String name) throws QAException
```

パラメーター

- **name**　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で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ

getInReplyToID メソッド

このメッセージの返信メッセージのメッセージ ID を返します。

構文

```
String QAMessage.getInReplyToID() throws QAException
```

戻り値

このメッセージの返信メッセージのメッセージ ID。このメッセージが返信しない場合は NULL。

例外

- **QAException クラス**　このメッセージの返信メッセージのメッセージ ID の取得で問題が発生した場合にスローされます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

getIntProperty メソッド

int 型のメッセージプロパティを取得します。

構文

```
int QAMessage.getIntProperty(String name) throws QAException
```

パラメーター

- **name** 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で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ

getLongProperty メソッド

long 型のメッセージプロパティを取得します。

構文

```
long QAMessage.getLongProperty(String name) throws QAException
```

パラメーター

- **name** 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で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ

getMessageID メソッド

メッセージのグローバルにユニークなメッセージ ID を返します。

構文

```
String QAMessage.getMessageID() throws QAException
```

戻り値

メッセージのメッセージ ID。メッセージが登録されていない場合は NULL。

例外

- **QAException クラス**　メッセージ ID の取得で問題が発生した場合にスローされます。

備考

このプロパティは、メッセージがキューに登録されるまで NULL のままです。

QAManagerBase.putMessage(String, QAMessage) を使用してメッセージが送信されると、メッセージ ID が NULL になるため無視できます。send メソッドが戻ると、割り当てられた値がここに格納されます。

参照

- QAManagerBase.putMessage メソッド [QAnywhere Java]565 ページ

getPriority メソッド

メッセージの優先度 (0 から 9) を返します。

構文

```
int QAMessage.getPriority() throws QAException
```

戻り値

メッセージの優先度。

例外

- **QAException クラス**　メッセージの優先度の取得で問題が発生した場合にスローされます。

getProperty メソッド

メッセージのプロパティを取得します。

構文

```
Object QAMessage.getProperty(String name) throws QAException
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- **name**　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。

例外

- **QAException クラス**　プロパティ値の取得で変換エラーが発生した場合にスローされます。

getPropertyNames メソッド

メッセージのプロパティ名の列挙子を取得します。

構文

```
java.util.Enumeration QAMessage.getPropertyNames() throws QAException
```

戻り値

メッセージのプロパティ名の列挙子。

例外

- **QAException クラス**　メッセージのプロパティ名の列挙子を取得するときに問題が発生した場合にスローされます。

getPropertyType メソッド

指定されたプロパティのプロパティ型を返します。

構文

```
short QAMessage.getPropertyType(String name) throws QAException
```

パラメーター

- **name**　プロパティ名。

戻り値

プロパティ型。

例外

- **QAException クラス**　プロパティ型の取得で問題が発生した場合にスローされます。

参照

- PropertyType インターフェイス [QAnywhere Java]507 ページ

getRedelivered メソッド

受信されたが受信確認されていないメッセージであるかどうかを示します。

構文

```
boolean QAMessage.getRedelivered() throws QAException
```

戻り値

受信されたが受信確認されていないメッセージである場合は true。

例外

- **QAException クラス**　再配信ステータスの取得で問題が発生した場合にスローされます。

備考

受信側の QAManager は、受信中のメッセージが以前に受信されたことを検知した場合に、Redelivered プロパティを設定します。

たとえば、AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT でオープンされた QAManager を使用してアプリケーションがメッセージを受信し、メッセージの受信確認を行わずに終了したとします。このアプリケーションが再起動されて同じメッセージを受信すると、メッセージに再配信のマークが付けられます。

getReplyToAddress メソッド

メッセージの返信アドレスを返します。

構文

```
String QAMessage.getReplyToAddress() throws QAException
```

戻り値

このメッセージの返信アドレス。返信アドレスがない場合は NULL。

例外

- **QAException クラス**　返信アドレスの取得で問題が発生した場合にスローされます。

getShortProperty メソッド

short 型のメッセージプロパティを取得します。

構文

```
short QAMessage.getShortProperty(String name) throws QAException
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- **name**　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で変換エラーがある場合、またはプロパティが存在しない場合にスローされます。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ

getStringProperty メソッド

String 型のメッセージプロパティを取得します。

構文

```
String QAMessage.getStringProperty(String name) throws QAException
```

パラメーター

- **name**　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。

例外

- **QAException クラス**　メッセージプロパティの取得で問題が発生した場合にスローされます。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ

getTimestamp メソッド

メッセージの作成時刻を示すタイムスタンプを返します。

構文

```
java.util.Date QAMessage.getTimestamp() throws QAException
```

戻り値

メッセージのタイムスタンプ。

例外

- **QAException クラス**　メッセージのタイムスタンプの取得で問題が発生した場合にスローされます。

propertyExists メソッド

指定されたプロパティがこのメッセージに設定されているかどうかを示します。

構文

```
boolean QAMessage.propertyExists(String name) throws QAException
```

パラメーター

- **name**　プロパティ名。

戻り値

プロパティが存在する場合は true。

例外

- **QAException クラス**　プロパティが設定されているかどうかの確認で問題が発生した場合にスローされます。

setBooleanProperty メソッド

boolean 型のプロパティを設定します。

構文

```
void QAMessage.setBooleanProperty(
     String name,
    boolean value
) throws QAException
```

パラメーター

- **name**　プロパティ名。
- **value**　プロパティの値。

例外

- **QAException クラス**　プロパティの設定で問題が発生した場合にスローされます。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

setByteProperty メソッド

signed byte 型のプロパティを設定します。

構文

```
void QAMessage.setByteProperty(
     String name,
    byte value
) throws QAException
```

パラメーター

- **name** プロパティ名。
- **value** プロパティの値。

例外

- **QAException クラス** プロパティの設定で問題が発生した場合にスローされます。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ

setDoubleProperty メソッド

double 型のプロパティを設定します。

構文

```
void QAMessage.setDoubleProperty(
     String name,
    double value
) throws QAException
```

パラメーター

- **name** プロパティ名。
- **value** プロパティの値。

例外

- **QAException クラス** プロパティの設定で問題が発生した場合にスローされます。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ

setFloatProperty メソッド

float 型のプロパティを設定します。

構文

```
void QAMessage.setFloatProperty(
     String name,
    float value
) throws QAException
```

パラメーター

- **name** プロパティ名。
- **value** プロパティの値。

例外

- **QAException クラス** プロパティの設定で問題が発生した場合にスローされます。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ

setInReplyToID メソッド

このメッセージの返信対象メッセージを特定する、返信先 ID を設定します。

構文

```
void QAMessage.setInReplyToID(String id) throws QAException
```

パラメーター

- **id** このメッセージの返信メッセージの ID。

例外

- **QAException クラス** 返信先 ID の設定で問題が発生した場合にスローされます。

setIntProperty メソッド

int 型のプロパティを設定します。

構文

```
void QAMessage.setIntProperty(String name, int value) throws QAException
```

パラメーター

- **name** プロパティ名。
- **value** プロパティの値。

例外

- **QAException クラス** プロパティの設定で問題が発生した場合にスローされます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ

setLongProperty メソッド

long 型のプロパティを設定します。

構文

```
void QAMessage.setLongProperty(
     String name,
    long value
) throws QAException
```

パラメーター

- **name** プロパティ名。
- **value** プロパティの値。

例外

- **QAException クラス** プロパティの設定で問題が発生した場合にスローされます。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ

setPriority メソッド

メッセージの優先度 (0 から 9) を設定します。

構文

```
void QAMessage.setPriority(int priority) throws QAException
```

パラメーター

- **priority** メッセージの優先度。

例外

- **QAException クラス** 優先度の設定で問題が発生した場合にスローされます。

setProperty メソッド

プロパティを設定します。

構文

```
void QAMessage.setProperty(String name, Object value) throws QAException
```

パラメーター

- **name** プロパティ名。
- **value** プロパティの値。

例外

- **QAException クラス** プロパティの設定で問題が発生した場合にスローされます。

備考

プロパティ型は、使用可能ないずれかのプリミティブ型、または String 型でなければなりません。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ

setReplyToAddress メソッド

返信先アドレスを設定します。

構文

```
void QAMessage.setReplyToAddress(String address) throws QAException
```

パラメーター

- **address** 返信先アドレス。

例外

- **QAException クラス** 返信アドレスの設定で問題が発生した場合にスローされます。

setShortProperty メソッド

short 型のプロパティを設定します。

構文

```
void QAMessage.setShortProperty(
     String name,
    short value
) throws QAException
```

パラメーター

- **name** プロパティ名。
- **value** プロパティの値。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例外

- **QAException クラス**　プロパティの設定で問題が発生した場合にスローされます。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ

setStringProperty メソッド

string 型のプロパティを設定します。

構文

```
void QAMessage.setStringProperty(
     String name,
    String value
) throws QAException
```

パラメーター

- **name**　プロパティ名。
- **value**　プロパティの値。

例外

- **QAException クラス**　プロパティの設定で問題が発生した場合にスローされます。

参照

- MessageProperties インターフェイス [QAnywhere Java]497 ページ

DEFAULT_PRIORITY 変数

デフォルトのメッセージ優先度です。

構文

```
final int QAMessage.DEFAULT_PRIORITY
```

DEFAULT_TIME_TO_LIVE 変数

デフォルトの存続時間です。

構文

```
final long QAMessage.DEFAULT_TIME_TO_LIVE
```

QAMessageListener インターフェイス

メッセージを受信するには、QAMangerBase.setMessageListener(String,QAMessageListener) を呼び出して、このインターフェイスを実装し、実装を登録します。

構文

```
public interface QAMessageListener
```

メンバー

継承されたメンバーを含む QAMessageListener インターフェイスのすべてのメンバー。

名前	説明
onException メソッド	このメソッドは、メッセージの受信中に例外が発生すると呼び出されます。
onMessage メソッド	このメソッドは、メッセージが受信されると呼び出されます。

参照

- QAManagerBase.setMessageListener メソッド [QAnywhere Java]571 ページ

onException メソッド

このメソッドは、メッセージの受信中に例外が発生すると呼び出されます。

構文

```
void QAMessageListener.onException(
     QAException exception,
    QAMessage message
)
```

パラメーター

- **exception**　発生した例外。
- **message**　メッセージが onMessage(QAMessage) に渡された後で例外が発生した場合は、処理されたメッセージ。それ以外の場合は NULL。

備考

このメソッドを使用して QAManagerBase インスタンスを自動的に終了することはできません。すべてのメッセージリスナーの処理が終了するまでは、QAManagerBase.close() メソッドでブロックされるためです。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- QAManagerBase インターフェイス [QAnywhere Java]544 ページ
- QAManagerBase.close メソッド [QAnywhere Java]551 ページ

onMessage メソッド

このメソッドは、メッセージが受信されると呼び出されます。

構文

```
void QAMessageListener.onMessage(QAMessage message) throws QAException
```

パラメーター

- **message**　受信したメッセージ。

QAMessageListener2 インターフェイス

メッセージを受信するには、QAMangerBase.setMessageListener2(String,QAMessageListener2) を呼び出して、このインターフェイスを実装し、実装を登録します。

構文

```
public interface QAMessageListener2
```

メンバー

継承されたメンバーを含む QAMessageListener2 インターフェイスのすべてのメンバー。

名前	説明
onException メソッド	このメソッドは、メッセージの受信中に例外が発生すると呼び出されます。
onMessage メソッド	このメソッドは、メッセージが受信されると呼び出されます。

参照

- QAManagerBase.setMessageListener2 メソッド [QAnywhere Java]571 ページ

onException メソッド

このメソッドは、メッセージの受信中に例外が発生すると呼び出されます。

構文

```
void QAMessageListener2.onException(
     QAManagerBase mgr,
    QAException exception,
    QAMessage message
)
```

パラメーター

- **mgr**　メッセージを処理した QAManagerBase。
- **exception**　発生した例外。
- **message**　メッセージが onMessage(QAMessage) に渡された後で例外が発生した場合は、処理されたメッセージ。それ以外の場合は NULL。

備考

このメソッドを使用して QAManagerBase インスタンスを自動的に終了することはできません。すべてのメッセージリスナーの処理が終了するまでは、QAManagerBase.close() メソッドでブロックされるためです。

参照

- QAManagerBase インターフェイス [QAnywhere Java]544 ページ
- QAManagerBase.close メソッド [QAnywhere Java]551 ページ

onMessage メソッド

このメソッドは、メッセージが受信されると呼び出されます。

構文

```
void QAMessageListener2.onMessage(
     QAManagerBase mgr,
    QAMessage message
) throws QAException
```

パラメーター

- **mgr**　メッセージを受信した QAManagerBase。
- **message**　受信したメッセージ。

参照

- QAManagerBase インターフェイス [QAnywhere Java]544 ページ

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

QATextMessage インターフェイス

QATextMessage は QAMessage クラスを継承したもので、メッセージ本文にテキストが追加されます。また、メッセージ本文からのテキストの読み込み／書き込みを行うためのメソッドがあります。

構文

```
public interface QATextMessage
```

基本クラス

- QAMessage インターフェイス [QAnywhere Java]582 ページ

メンバー

継承されたメンバーを含む QATextMessage インターフェイスのすべてのメンバー。

名前	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。
getAddress メソッド	QAMessage インスタンスの送信先アドレスを返します。
getBooleanProperty メソッド	boolean 型のメッセージプロパティを取得します。
getByteProperty メソッド	signed byte 型のメッセージプロパティを取得します。
getDoubleProperty メソッド	double 型のメッセージプロパティを取得します。
getExpiration メソッド	メッセージの有効期限の値を返します。メッセージの有効期限がない場合や、まだ送信されていない場合は、NULL になります。
getFloatProperty メソッド	float 型のメッセージプロパティを取得します。
getInReplyToID メソッド	このメッセージの返信メッセージのメッセージ ID を返します。
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。

名前	説明
getMessageID メソッド	メッセージのグローバルにユニークなメッセージ ID を返します。
getPriority メソッド	メッセージの優先度 (0 から 9) を返します。
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。
getRedelivered メソッド	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。
getReplyToAddress メソッド	メッセージの返信アドレスを返します。
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	String 型のメッセージプロパティを取得します。
getText メソッド	メッセージテキストを返します。
getTextLength メソッド	メッセージの文字数を返します。
getTimestamp メソッド	メッセージの作成時刻を示すタイムスタンプを返します。
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。
readText メソッド	メッセージの未読テキストを返します。
reset メソッド	メッセージのテキスト位置を先頭にリセットします。
setBooleanProperty メソッド	boolean 型のプロパティを設定します。
setByteProperty メソッド	signed byte 型のプロパティを設定します。
setDoubleProperty メソッド	double 型のプロパティを設定します。
setFloatProperty メソッド	float 型のプロパティを設定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
setInReplyToID メソッド	このメッセージの返信対象メッセージを特定する、返信先 ID を設定します。
setIntProperty メソッド	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。
setPriority メソッド	メッセージの優先度 (0 から 9) を設定します。
setProperty メソッド	プロパティを設定します。
setReplyToAddress メソッド	返信先アドレスを設定します。
setShortProperty メソッド	short 型のプロパティを設定します。
setStringProperty メソッド	string 型のプロパティを設定します。
setText メソッド	メッセージテキストを上書きします。
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。
DEFAULT_PRIORITY 変数	デフォルトのメッセージ優先度です。
DEFAULT_TIME_TO_LIVE 変数	デフォルトの存続時間です。

備考

メッセージが最初に作成された時点では、メッセージ本文は書き込み専用モードになっています。メッセージ送信元のクライアントは、メッセージを送信した後、そのメッセージを保持し変更できます。ただし、それによって、送信されたメッセージが変更されることはありません。同じメッセージオブジェクトを複数回送信できます。

メッセージが受信された時点でプロバイダーは QATextMessage.reset() を呼び出します。これによりメッセージ本文が読み込み専用モードになり、メッセージ本文の先頭から値の読み込みが開始されます。

参照

- QAMessageListener.onMessage メソッド [QAnywhere Java]601 ページ

getText メソッド

メッセージテキストを返します。

構文

```
String QATextMessage.getText() throws QAException
```

戻り値

メッセージテキストまたは NULL。

例外

- **QAException クラス**　メッセージテキストの取得で問題が発生した場合にスローされます。

備考

メッセージテキストのサイズが、QAManager.MAX_IN_MEMORY_MESSAGE_SIZE プロパティで指定された最大サイズを超える場合、このメソッドは NULL を返します。この場合は、QATextMessage.readText(int) メソッドを使用してテキストを読み込みます。

参照

- QATextMessage.readText メソッド [QAnywhere Java]606 ページ

getTextLength メソッド

メッセージの文字数を返します。

構文

```
long QATextMessage.getTextLength() throws QAException
```

戻り値

メッセージの文字数。

例外

- **QAException クラス**　メッセージの長さの取得で問題が発生した場合にスローされます。

readText メソッド

メッセージの未読テキストを返します。

構文

```
String QATextMessage.readText(int maxLength) throws QAException
```

パラメーター

- **maxLength**　読み込まれる最大文字数。

戻り値

テキスト。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例外

- **QAException クラス**　未読テキストの取得で問題が発生した場合にスローされます。

備考

未読テキストが追加された場合、そのテキストを読み込むにはこのメソッドを繰り返し呼び出す必要があります。読み込みは、未読テキストの先頭から実行されます。

reset メソッド

メッセージのテキスト位置を先頭にリセットします。

構文

```
void QATextMessage.reset() throws QAException
```

例外

- **QAException クラス**　メッセージのテキスト位置のリセットで問題が発生した場合にスローされます。

setText メソッド

メッセージテキストを上書きします。

構文

```
void QATextMessage.setText(String value) throws QAException
```

パラメーター

- **value**　メッセージ本文に書き込むテキスト。

例外

- **QAException クラス**　メッセージテキストの上書きで問題が発生した場合にスローされます。

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。

オーバーロードリスト

名前	説明
writeText(String) 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。

名前	説明
writeText(String, int) 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。
writeText(String, int, int) 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。

writeText(String) メソッド

メッセージテキストの末尾にテキストを追加します。

構文

```
void QATextMessage.writeText(String value) throws QAException
```

パラメーター

- **value** 追加するテキスト。

例外

- **QAException クラス** メッセージテキストの追加で問題が発生した場合にスローされます。

writeText(String, int) メソッド

メッセージテキストの末尾にテキストを追加します。

構文

```
void QATextMessage.writeText(
     String value,
    int length
) throws QAException
```

パラメーター

- **value** 追加するテキスト。
- **length** 追加するテキストの文字数。

例外

- **QAException クラス** メッセージテキストの追加で問題が発生した場合にスローされます。

writeText(String, int, int) 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。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
void QATextMessage.writeText(
     String value,
    int offset,
    int length
) throws QAException
```

パラメーター

- **value**　追加するテキスト。
- **offset**　追加するテキスト値内でのオフセット。
- **length**　追加するテキストの文字数。

例外

- **QAException クラス**　メッセージテキストの追加で問題が発生した場合にスローされます。

QATransactionalManager インターフェイス

QATransactionalManager クラスは QAManagerBase から派生し、トランザクション指向の QAnywhere メッセージング操作を管理します。

構文

```
public interface QATransactionalManager
```

基本クラス

- QAManagerBase インターフェイス [QAnywhere Java]544 ページ

メンバー

継承されたメンバーを含む QATransactionalManager インターフェイスのすべてのメンバー。

名前	説明
browseMessages メソッド	メッセージストア内にある、参照可能なメッセージをすべて参照します。
browseMessagesByID メソッド	指定されたメッセージ ID を持つメッセージを参照します。
browseMessagesByQueue メソッド	指定されたアドレスに送信され、取得可能な一連の受信待機中メッセージを参照します。
browseMessagesBySelector メソッド	メッセージストアのキューに登録されているメッセージのうち、指定されたセレクターを満たすメッセージを参照します。

名前	説明
cancelMessage メソッド	指定されたメッセージ ID を持つメッセージをキャンセルします。
close メソッド	QAnywhere メッセージシステムへの接続をクローズして、QAManagerBase で使用していたリソースをすべて解放します。
commit メソッド	現在のトランザクションをコミットし、新しいトランザクションを開始します。
createBinaryMessage メソッド	QABinaryMessage オブジェクトを作成します。
createTextMessage メソッド	QATextMessage オブジェクトを作成します。
getBooleanStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの boolean 値を取得します。
getByteStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの signed byte 値を取得します。
getDoubleStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの double 値を取得します。
getFloatStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの float 値を取得します。
getIntStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの int 値を取得します。
getLongStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの long 値を取得します。
getMessage メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
getMessageBySelector メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
getMessageBySelectorNoWait メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
getMessageBySelectorTimeout メソッド	指定されたアドレスに送信され、かつ指定されたセレクターを満たす、次に取得可能な QAMessage を返します。
getMessageNoWait メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
getMessageTimeout メソッド	指定されたアドレスに送信された、次に取得可能な QAMessage を返します。
getMode メソッド	受信したメッセージの QAManager 受信確認モードを返します。
getQueueDepth メソッド	指定されたフィルターに基づいて、すべてのキューの長さの合計を返します。
getShortStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの short 値を取得します。
getStoreProperty メソッド	メッセージストアプロパティを表すオブジェクトを取得します。
getStorePropertyNames メソッド	メッセージストアのプロパティ名の列挙子を取得します。
getStringStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティの string 値を取得します。
open メソッド	QATransactionalManager インスタンスをオープンします。
propertyExists メソッド	指定されたプロパティに値が現在存在するかどうかをテストします。
putMessage メソッド	別の QAnywhere クライアントに送信するメッセージを準備します。
putMessageTimeToLive メソッド	別の QAnywhere クライアントに送信するメッセージを準備します。
reOpen メソッド	QAManagerBase を再度開きます。
rollback メソッド	現在のトランザクションをロールバックし、新しいトランザクションを開始します。
setBooleanStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを boolean 値に設定します。

名前	説明
setByteStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを sbyte 値に設定します。
setDoubleStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを double 値に設定します。
setFloatStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを float 値に設定します。
setIntStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを int 値に設定します。
setLongStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを long 値に設定します。
setMessageListener メソッド	QAMessageListener オブジェクトを登録して QAnywhere メッセージを非同期的に受信します。
setMessageListener2 メソッド	QAMessageListener2 オブジェクトを登録して QAnywhere メッセージを非同期的に受信します。
setMessageListenerBySelector メソッド	QAMessageListener オブジェクトを登録し、メッセージセレクターを使用して QAnywhere メッセージを非同期的に受信します。
setMessageListenerBySelector2 メソッド	QAMessageListener2 オブジェクトを登録し、メッセージセレクターを使用して QAnywhere メッセージを非同期的に受信します。
setProperty メソッド	QAnywhere Manager の設定プロパティをプログラムで設定できるようにします。
setShortStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを short 値に設定します。
setStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを System.Object 値に設定します。
setStringStoreProperty メソッド	事前定義済みまたはカスタムのメッセージストアプロパティを String 値に設定します。
start メソッド	着信メッセージを受信するための QAManagerBase を開始します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
stop メソッド	QAManagerBase による着信メッセージの受信を停止します。
triggerSendReceive メソッド	QAnywhere メッセージサーバーとの同期処理を発生させて、他のクライアント宛てのメッセージをアップロードし、ローカルクライアント宛てのメッセージをダウンロードします。

備考

この動作の完全な説明については、QAManagerBase を参照してください。

QATransactionalManager インスタンスは、トランザクション指向の受信確認でのみ使用できます。QAManagerBase.putMessage(String, QAMessage) と QAManagerBase.getMessage(String) のすべての呼び出しをコミットする場合は、QATransactionalManager.commit() メソッドを使用します。

参照

- QATransactionalManager.commit メソッド [QAnywhere Java]613 ページ
- QAManagerBase.putMessage メソッド [QAnywhere Java]565 ページ
- QAManagerBase.getMessage メソッド [QAnywhere Java]556 ページ

commit メソッド

現在のトランザクションをコミットし、新しいトランザクションを開始します。

構文

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

例外

- **QAException クラス**　コミットで問題が発生した場合にスローされます。

備考

このメソッドは、QAManagerBase.putMessage(String, QAMessage) と QAManagerBase.getMessage(String) のすべての呼び出しをコミットします。

> **注意**
> 最初のトランザクションは、QATransactionalManager.open() の呼び出しで開始されます。

open メソッド

QATransactionalManager インスタンスをオープンします。

構文

```
void QATransactionalManager.open() throws QAException
```

例外

- **QAException クラス**　Manager のオープンで問題が発生した場合にスローされます。

備考

これは、Manager を作成した後、最初に呼び出す必要があるメソッドです。

rollback メソッド

現在のトランザクションをロールバックし、新しいトランザクションを開始します。

構文

```
void QATransactionalManager.rollback() throws QAException
```

例外

- **QAException クラス**　ロールバックで問題が発生した場合にスローされます。

備考

このメソッドは、QAManagerBase.putMessage(String, QAMessage) と QAManagerBase.getMessage(String) のすべてのコミットされていない呼び出しをロールバックします。

QueueDepthFilter インターフェイス

QAManagerBase.getQueueDepth(short) と QAManagerBase.getQueueDepth(String, short) のキューの長さのフィルター値を提供します。

構文

```
public interface QueueDepthFilter
```

メンバー

継承されたメンバーを含む QueueDepthFilter インターフェイスのすべてのメンバー。

名前	説明
ALL 変数	このフィルターは、着信メッセージと送信メッセージの両方を指定します。
INCOMING 変数	このフィルターは、着信メッセージのみ指定します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
LOCAL 変数	このフィルターは、ローカルメッセージのみ指定します。
OUTGOING 変数	このフィルターは、送信メッセージのみ指定します。

参照

- QAManagerBase.getQueueDepth メソッド [QAnywhere Java]561 ページ

ALL 変数

このフィルターは、着信メッセージと送信メッセージの両方を指定します。

構文

```
final short QueueDepthFilter.ALL
```

備考

システムメッセージと期限切れのメッセージは、キューの長さカウントに計上されません。

INCOMING 変数

このフィルターは、着信メッセージのみ指定します。

構文

```
final short QueueDepthFilter.INCOMING
```

備考

着信メッセージは、発信者がメッセージストアのエージェント ID ではないメッセージです。

LOCAL 変数

このフィルターは、ローカルメッセージのみ指定します。

構文

```
final short QueueDepthFilter.LOCAL
```

備考

ローカルメッセージは、発信者と送信先がメッセージストアのエージェント ID であるメッセージです。

OUTGOING 変数

このフィルターは、送信メッセージのみ指定します。

構文

```
final short QueueDepthFilter.OUTGOING
```

備考

送信メッセージは、発信者がメッセージストアのエージェント ID であり、送信先がメッセージストアのエージェント ID ではないメッセージです。

StatusCodes インターフェイス

このインターフェイスは、メッセージのステータスコードセットを定義します。

構文

```
public interface StatusCodes
```

メンバー

継承されたメンバーを含む StatusCodes インターフェイスのすべてのメンバー。

名前	説明
CANCELED 変数	メッセージはキャンセルされました。
EXPIRED 変数	メッセージの有効期限が切れました。メッセージは、有効期限になる前に受信されませんでした。
FINAL 変数	この定数は、メッセージが最終ステータスになったかどうかを示します。
LOCAL 変数	メッセージはローカルメッセージストア宛てに送信され、サーバーに送信されません。
PENDING 変数	メッセージは送信されましたが、まだ受信および受信確認されていません。
RECEIVED 変数	メッセージが受信され、受信者によって受信確認されました。
RECEIVING 変数	メッセージは受信中であるか、または受信されましたがまだ確認されていません。
TRANSMITTED 変数	メッセージはサーバーに送信されました。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
TRANSMITTING 変数	メッセージはサーバーに送信中です。
UNRECEIVABLE 変数	メッセージは受信不可のマークが付けられています。
UNTRANSMITTED 変数	メッセージはサーバーに送信されていません。

CANCELED 変数

メッセージはキャンセルされました。

構文

```
final int StatusCodes.CANCELED
```

備考

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

参照

- MessageProperties.STATUS 変数 [QAnywhere Java]503 ページ

EXPIRED 変数

メッセージの有効期限が切れました。メッセージは、有効期限になる前に受信されませんでした。

構文

```
final int StatusCodes.EXPIRED
```

備考

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

参照

- MessageProperties.STATUS 変数 [QAnywhere Java]503 ページ

FINAL 変数

この定数は、メッセージが最終ステータスになったかどうかを示します。

構文

```
final int StatusCodes.FINAL
```

備考

メッセージは、そのステータスがこの定数より大きい場合にのみ最終ステータスになったと見なされます。

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

参照

- MessageProperties.STATUS 変数 [QAnywhere Java]503 ページ

LOCAL 変数

メッセージはローカルメッセージストア宛てに送信され、サーバーに送信されません。

構文

```
final int StatusCodes.LOCAL
```

備考

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

参照

- MessageProperties.TRANSMISSION_STATUS 変数 [QAnywhere Java]504 ページ

PENDING 変数

メッセージは送信されましたが、まだ受信および受信確認されていません。

構文

```
final int StatusCodes.PENDING
```

備考

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

参照

- MessageProperties.STATUS 変数 [QAnywhere Java]503 ページ

RECEIVED 変数

メッセージが受信され、受信者によって受信確認されました。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
final int StatusCodes.RECEIVED
```

備考

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

参照

- MessageProperties.STATUS 変数 [QAnywhere Java]503 ページ

RECEIVING 変数

メッセージは受信中であるか、または受信されましたがまだ確認されていません。

構文

```
final int StatusCodes.RECEIVING
```

備考

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

参照

- MessageProperties.STATUS 変数 [QAnywhere Java]503 ページ

TRANSMITTED 変数

メッセージはサーバーに送信されました。

構文

```
final int StatusCodes.TRANSMITTED
```

備考

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

参照

- MessageProperties.TRANSMISSION_STATUS 変数 [QAnywhere Java]504 ページ

TRANSMITTING 変数

メッセージはサーバーに送信中です。

構文

```
final int StatusCodes.TRANSMITTING
```

備考

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

参照

- MessageProperties.TRANSMISSION_STATUS 変数 [QAnywhere Java]504 ページ

UNRECEIVABLE 変数

メッセージは受信不可のマークが付けられています。

構文

```
final int StatusCodes.UNRECEIVABLE
```

備考

メッセージに間違った形式があるか、失敗した試行回数が多すぎて配信されませんでした。

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

参照

- MessageProperties.STATUS 変数 [QAnywhere Java]503 ページ

UNTRANSMITTED 変数

メッセージはサーバーに送信されていません。

構文

```
final int StatusCodes.UNTRANSMITTED
```

備考

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

参照

- MessageProperties.TRANSMISSION_STATUS 変数 [QAnywhere Java]504 ページ

Web サービス用 QAnywhere Java API リファレンス

パッケージ

```
ianywhere.qanywhere.ws
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

WSBase クラス

これは、モバイル Web サービスのコンパイラーで生成されるメインの Web サービスプロキシクラスの基本クラスです。

構文

```
public class WSBase
```

メンバー

継承されたメンバーを含む WSBase クラスのすべてのメンバー。

名前	説明
WSBase コンストラクター	設定プロパティファイルが指定されたコンストラクターです。
clearRequestProperties メソッド	この WSBase に対して送信された要求プロパティをすべてクリアします。
getResult メソッド	Web サービス要求の結果を表す WSResult オブジェクトを取得します。
getServiceID メソッド	WSBase のこのインスタンスのサービス ID を取得します。
setListener メソッド	指定された Web サービス要求の結果を受信するリスナーを設定します。
setProperty メソッド	WSBase のこのインスタンスの設定プロパティを設定します。
setQAManager メソッド	この Web サービスクライアントが Web サービス要求を処理するために使用する QAManagerBase を設定します。
setRequestProperty メソッド	この WSBase のインスタンスが作成した Web サービス要求の要求プロパティを設定します。
setServiceID メソッド	WSBase のこのインスタンスのユーザー定義 ID を設定します。

WSBase コンストラクター

設定プロパティファイルが指定されたコンストラクターです。

オーバーロードリスト

名前	説明
WSBase() コンストラクター	コンストラクターです。
WSBase(String) コンストラクター	設定プロパティファイルが指定されたコンストラクターです。

WSBase() コンストラクター

コンストラクターです。

構文

```
WSBase.WSBase() throws WSException
```

例外

- **WSException クラス**　WSBase の構築で問題がある場合にスローされます。

WSBase(String) コンストラクター

設定プロパティファイルが指定されたコンストラクターです。

構文

```
WSBase.WSBase(String iniFile) throws WSException
```

パラメーター

- **iniFile**　設定プロパティが含まれているファイル。

例外

- **WSException クラス**　WSBase の構築で問題がある場合にスローされます。

備考

有効な設定プロパティには、次のものがあります。

LOG_FILE : ランタイム情報のログを記録するファイルです。

LOG_LEVEL : ログを記録する情報の冗長レベルを制御する、0 ～ 6 の値。6 が最高の冗長レベルです。

WS_CONNECTOR_ADDRESS : Mobile Link サーバーにおける Web サービスコネクターのアドレスです。デフォルトの WS_CONNECTOR_ADDRESS は "ianywhere.connector.webservices¥¥" です。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

clearRequestProperties メソッド

この WSBase に対して送信された要求プロパティをすべてクリアします。

構文

```
void WSBase.clearRequestProperties()
```

getResult メソッド

Web サービス要求の結果を表す WSResult オブジェクトを取得します。

構文

```
WSResult WSBase.getResult(String requestID) throws QAException
```

パラメーター

- **requestID**　Web サービス要求の ID。

戻り値

Web サービス要求の結果を表す WSResult インスタンス。

参照

- WSStatus クラス [QAnywhere Java]653 ページ

getServiceID メソッド

WSBase のこのインスタンスのサービス ID を取得します。

構文

```
String WSBase.getServiceID()
```

戻り値

サービス ID。

setListener メソッド

指定された Web サービス要求の結果を受信するリスナーを設定します。

オーバーロードリスト

名前	説明
setListener(String, WSListener) メソッド	指定された Web サービス要求の結果を受信するリスナーを設定します。
setListener(WSListener) メソッド	WSBase のこのインスタンスが作成した、すべての Web サービス要求を受信するリスナーを設定します。

setListener(String, WSListener) メソッド

指定された Web サービス要求の結果を受信するリスナーを設定します。

構文

```
void WSBase.setListener(
     String requestID,
    WSListener listener
) throws QAException
```

パラメーター

- **requestID**　結果を受信する Web サービス要求の ID。
- **listener**　指定された Web サービス要求が取得可能な場合に呼び出される Listener オブジェクト。

備考

一般にリスナーは、サービスの asyncXYZ メソッドの結果を取得する場合に使用されます。

リスナーを削除するには、リスナーとして *null* を指定し setListener を呼び出します。

> **注意**
> このメソッドは、以前に呼び出された setListener で設定されたリスナーを置き換えます。

setListener(WSListener) メソッド

WSBase のこのインスタンスが作成した、すべての Web サービス要求を受信するリスナーを設定します。

構文

```
void WSBase.setListener(WSListener listener) throws QAException
```

パラメーター

- **listener**　Web サービス要求が取得可能な場合に呼び出される Listener オブジェクト。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

一般にリスナーは、サービスの asyncXYZ メソッドの結果を取得する場合に使用されます。

リスナーを削除するには、リスナーとして *null* を指定し setListener を呼び出します。

> **注意**
> このメソッドは、以前に呼び出された setListener で設定されたリスナーを置き換えます。

setProperty メソッド

WSBase のこのインスタンスの設定プロパティを設定します。

構文

```
void WSBase.setProperty(String property, String val) throws WSException
```

パラメーター

- **property**　設定するプロパティ名。
- **val**　プロパティの値。

備考

設定プロパティは、非同期または同期の Web サービス要求を作成する前に設定してください。Web サービス要求の作成後は、このメソッドは無効になります。

有効な設定プロパティには、次のものがあります。

LOG_FILE : ランタイム情報のログを記録するファイルです。

LOG_LEVEL : ログを記録する情報の冗長レベルを制御する、0 ～ 6 の値。6 が最高の冗長レベルです。

WS_CONNECTOR_ADDRESS : Mobile Link サーバーにおける Web サービスコネクターのアドレスです。デフォルトは "ianywhere.connector.webservices¥¥" です。

setQAManager メソッド

この Web サービスクライアントが Web サービス要求を処理するために使用する QAManagerBase を設定します。

構文

```
void WSBase.setQAManager(QAManagerBase mgr)
```

パラメーター

- **mgr**　使用する QAManagerBase。

備考

> **注意**
> EXPLICIT_ACKNOWLEDGEMENT QAManager を使用する場合は、WSResult の acknowledge() メソッドを呼び出して、非同期の Web サービス要求の結果を受信確認できます。同期の Web サービス要求の結果は、EXPLICIT_ACKNOWLEDGEMENT QAManager を使用している場合であっても、自動的に受信確認されます。IMPLICIT_ACKNOWLEDGEMENT QAManager を使用する場合は、あらゆる Web サービス要求の結果が自動的に受信確認されます。

setRequestProperty メソッド

この WSBase のインスタンスが作成した Web サービス要求の要求プロパティを設定します。

構文

```
void WSBase.setRequestProperty(String name, Object value)
```

パラメーター

- **name** 設定するプロパティ名。
- **value** プロパティの値。

備考

要求プロパティは、この WSBase によって送信される QAMessage ごとに設定されます。この処理は、プロパティがクリアされるまで行われます。要求プロパティをクリアするには、NULL 値を設定します。メッセージプロパティのタイプは、value パラメーターのクラスで決まります。たとえば、値が Integer のインスタンスである場合は、setIntProperty を使用して QAMessage のプロパティが設定されます。

setServiceID メソッド

WSBase のこのインスタンスのユーザー定義 ID を設定します。

構文

```
void WSBase.setServiceID(String serviceID)
```

パラメーター

- **serviceID** サービス ID。

備考

サービス ID には、WSBase のこのインスタンスにユニークな値を設定します。サービス ID は、Web サービス要求を送受信するためのキュー名を形成する場合に内部で使用されます。このため、前のセッションで作成された Web サービス要求の結果を取得できるように、サービス ID をアプリケーションのセッション間で保持する必要があります。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

WSException クラス

このクラスは、Web サービス要求の処理中に発生した例外を表します。

構文

```
public class WSException
```

派生クラス

- WSFaultException クラス [QAnywhere Java]630 ページ

メンバー

継承されたメンバーを含む WSException クラスのすべてのメンバー。

名前	説明
WSException コンストラクター	指定されたエラーメッセージを使用して、新しい例外を構築します。
getErrorCode メソッド	この例外に関連付けられているエラーコードを取得します。
WS_STATUS_HTTP_ERROR 変数	Web サービスコネクターが作成した Web サービス HTTP 要求にエラーがあったことを示すエラーコードです。
WS_STATUS_HTTP_OK 変数	Web サービスコネクターからの Web サービス HTTP 要求が正常に行われたことを示すエラーコードです。
WS_STATUS_HTTP_RETRIES_EXCEEDED 変数	HTTP の再試行回数が Web サービスコネクターを超えたこを示すエラーコードです。
WS_STATUS_SOAP_PARSE_ERROR 変数	SOAP 応答または SOAP 要求の解析中に、Web サービスランタイムまたは Web サービスコネクターでエラーがあったことを示すエラーコードです。

WSException コンストラクター

指定されたエラーメッセージを使用して、新しい例外を構築します。

オーバーロードリスト

名前	説明
WSException(Exception) コンストラクター	新しい例外を構築します。
WSException(String) コンストラクター	指定されたエラーメッセージを使用して、新しい例外を構築します。
WSException(String, int) コンストラクター	指定されたエラーメッセージとエラーコードを使用して、新しい例外を構築します。

WSException(Exception) コンストラクター

新しい例外を構築します。

構文

```
WSException.WSException(Exception exception)
```

パラメーター

- **exception**　例外。

WSException(String) コンストラクター

指定されたエラーメッセージを使用して、新しい例外を構築します。

構文

```
WSException.WSException(String msg)
```

パラメーター

- **msg**　エラーメッセージ。

WSException(String, int) コンストラクター

指定されたエラーメッセージとエラーコードを使用して、新しい例外を構築します。

構文

```
WSException.WSException(String msg, int errorCode)
```

パラメーター

- **msg**　エラーメッセージ。
- **errorCode**　エラーコード。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

getErrorCode メソッド

この例外に関連付けられているエラーコードを取得します。

構文

```
int WSException.getErrorCode()
```

戻り値

この例外に関連付けられているエラーコード。

WS_STATUS_HTTP_ERROR 変数

Web サービスコネクターが作成した Web サービス HTTP 要求にエラーがあったことを示すエラーコードです。

構文

```
final int WSException.WS_STATUS_HTTP_ERROR
```

WS_STATUS_HTTP_OK 変数

Web サービスコネクターからの Web サービス HTTP 要求が正常に行われたことを示すエラーコードです。

構文

```
final int WSException.WS_STATUS_HTTP_OK
```

WS_STATUS_HTTP_RETRIES_EXCEEDED 変数

HTTP の再試行回数が Web サービスコネクターを超えたこを示すエラーコードです。

構文

```
final int WSException.WS_STATUS_HTTP_RETRIES_EXCEEDED
```

WS_STATUS_SOAP_PARSE_ERROR 変数

SOAP 応答または SOAP 要求の解析中に、Web サービスランタイムまたは Web サービスコネクターでエラーがあったことを示すエラーコードです。

構文

```
final int WSException.WS_STATUS_SOAP_PARSE_ERROR
```

WSFaultException クラス

このクラスは、Web サービスコネクターからの SOAP Fault 例外を表します。

構文

```
public class WSFaultException
```

基本クラス

- WSException クラス [QAnywhere Java]627 ページ

メンバー

継承されたメンバーを含む WSFaultException クラスのすべてのメンバー。

名前	説明
WSException コンストラクター	指定されたエラーメッセージを使用して、新しい例外を構築します。
WSFaultException コンストラクター	指定されたエラーメッセージを使用して、新しい例外を構築します。
getErrorCode メソッド	この例外に関連付けられているエラーコードを取得します。
WS_STATUS_HTTP_ERROR 変数	Web サービスコネクターが作成した Web サービス HTTP 要求にエラーがあったことを示すエラーコードです。
WS_STATUS_HTTP_OK 変数	Web サービスコネクターからの Web サービス HTTP 要求が正常に行われたことを示すエラーコードです。
WS_STATUS_HTTP_RETRIES_EXCEEDED 変数	HTTP の再試行回数が Web サービスコネクターを超えたこを示すエラーコードです。
WS_STATUS_SOAP_PARSE_ERROR 変数	SOAP 応答または SOAP 要求の解析中に、Web サービスランタイムまたは Web サービスコネクターでエラーがあったことを示すエラーコードです。

WSFaultException コンストラクター

指定されたエラーメッセージを使用して、新しい例外を構築します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
WSFaultException.WSFaultException(String msg)
```

パラメーター

- **msg**　エラーメッセージ。

WSListener インターフェイス

このクラスは、Web サービス要求の結果を受信するリスナーを表します。

構文

```
public interface WSListener
```

メンバー

継承されたメンバーを含む WSListener クラスのすべてのメンバー。

名前	説明
onException メソッド	非同期の Web サービス要求の結果を処理中に例外が発生した場合に呼び出されます。
onResult メソッド	非同期の Web サービス要求の結果を指定して呼び出されます。

onException メソッド

非同期の Web サービス要求の結果を処理中に例外が発生した場合に呼び出されます。

構文

```
void WSListener.onException(WSException e, WSResult wsResult)
```

パラメーター

- **e**　結果の処理中に発生した WSException。
- **wsResult**　WSResult。これから要求 ID を取得できる場合があります。この WSResult の値は定義されていません。

参照

- WSException クラス [QAnywhere Java]627 ページ
- WSResult クラス [QAnywhere Java]632 ページ

onResult メソッド

非同期の Web サービス要求の結果を指定して呼び出されます。

構文

```
void WSListener.onResult(WSResult wsResult)
```

パラメーター

- **wsResult**　Web サービス要求の結果を記述する WSResult。

参照

- WSResult クラス [QAnywhere Java]632 ページ

WSResult クラス

このクラスは、Web サービス要求の結果を表します。

構文

```
public class WSResult
```

メンバー

継承されたメンバーを含む WSResult クラスのすべてのメンバー。

名前	説明
acknowledge メソッド	この WSResult が処理されたことを確認します。
getArrayValue メソッド	この WSResult から複合型の値の配列を取得します。
getBigDecimalArrayValue メソッド	この WSResult から BigDecimal 配列値を取得します。
getBigDecimalValue メソッド	この WSResult から BigDecimal 値を取得します。
getBigIntegerArrayValue メソッド	この WSResult から BigInteger 配列値を取得します。
getBigIntegerValue メソッド	この WSResult から BigInteger 値を取得します。
getBooleanArrayValue メソッド	この WSResult から java.lang.Boolean 配列値を取得します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
getBooleanValue メソッド	この WSResult から java.lang.Boolean 値を取得します。
getByteArrayValue メソッド	この WSResult から java.lang.Byte 配列値を取得します。
getByteValue メソッド	この WSResult から java.lang.Byte 値を取得します。
getCharacterArrayValue メソッド	この WSResult から java.lang.Character 配列値を取得します。
getCharacterValue メソッド	この WSResult から java.lang.Character 値を取得します。
getDoubleArrayValue メソッド	この WSResult から java.lang.Double 配列値を取得します。
getDoubleValue メソッド	この WSResult から java.lang.Double 値を取得します。
getErrorMessage メソッド	エラーメッセージを取得します。
getFloatArrayValue メソッド	この WSResult から java.lang.Float 配列値を取得します。
getFloatValue メソッド	この WSResult から java.lang.Float 値を取得します。
getIntegerArrayValue メソッド	この WSResult から java.lang.Integer 配列値を取得します。
getIntegerValue メソッド	この WSResult から java.lang.Integer 値を取得します。
getLongArrayValue メソッド	この WSResult から java.lang.Long 配列値を取得します。
getLongValue メソッド	この WSResult から java.lang.Long 値を取得します。
getObjectArrayValue メソッド	この WSResult から複合型の値の配列を取得します。
getObjectValue メソッド	この WSResult から複合型の値を取得します。

名前	説明
getPrimitiveBooleanArrayValue メソッド	この WSResult から boolean 配列値を取得します。
getPrimitiveBooleanValue メソッド	この WSResult から boolean 値を取得します。
getPrimitiveByteArrayValue メソッド	この WSResult から byte 配列値を取得します。
getPrimitiveByteValue メソッド	この WSResult から byte 値を取得します。
getPrimitiveCharArrayValue メソッド	この WSResult から char 配列値を取得します。
getPrimitiveCharValue メソッド	この WSResult から char 値を取得します。
getPrimitiveDoubleArrayValue メソッド	この WSResult から double 配列値を取得します。
getPrimitiveDoubleValue メソッド	この WSResult から double 値を取得します。
getPrimitiveFloatArrayValue メソッド	この WSResult から float 配列値を取得します。
getPrimitiveFloatValue メソッド	この WSResult から float 値を取得します。
getPrimitiveIntArrayValue メソッド	この WSResult から int 配列値を取得します。
getPrimitiveIntValue メソッド	この WSResult から int 値を取得します。
getPrimitiveLongArrayValue メソッド	この WSResult から long 配列値を取得します。
getPrimitiveLongValue メソッド	この WSResult から long 値を取得します。
getPrimitiveShortArrayValue メソッド	この WSResult から short 配列値を取得します。
getPrimitiveShortValue メソッド	この WSResult から short 値を取得します。
getRequestID メソッド	この WSResult が表す要求 ID を取得します。
getShortArrayValue メソッド	この WSResult から java.lang.Short 配列値を取得します。
getShortValue メソッド	この WSResult から java.lang.Short 値を取得します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

名前	説明
getStatus メソッド	この WSResult のステータスを取得します。
getStringArrayValue メソッド	この WSResult から String 配列値を取得します。
getStringValue メソッド	この WSResult から String 値を取得します。
getValue メソッド	この WSResult から複合型の値を取得します。

備考

WSResult オブジェクトは、次のいずれかの方法で取得されます。

●WSListener.onResult に渡されます。

●コンパイラーが生成したサービスプロキシの asyncXYZ メソッドから返されます。

●特定の要求 ID を指定して WSBase.getResult を呼び出して取得されます。

acknowledge メソッド

この WSResult が処理されたことを確認します。

構文

```
void WSResult.acknowledge() throws WSException, QAException
```

備考

このメソッドは、EXPLICIT_ACKNOWLEDGEMENT QAManager が使用されている場合にのみ有効です。

getArrayValue メソッド

この WSResult から複合型の値の配列を取得します。

構文

```
WSSerializable[] WSResult.getArrayValue(
     String parentName
) throws WSException
```

パラメーター

● **parentName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

getBigDecimalArrayValue メソッド

この WSResult から BigDecimal 配列値を取得します。

構文

```
BigDecimal[] WSResult.getBigDecimalArrayValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

getBigDecimalValue メソッド

この WSResult から BigDecimal 値を取得します。

構文

```
BigDecimal WSResult.getBigDecimalValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

getBigIntegerArrayValue メソッド

この WSResult から BigInteger 配列値を取得します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
BigInteger[] WSResult.getBigIntegerArrayValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス** 値の取得で問題が発生した場合にスローされます。

getBigIntegerValue メソッド

この WSResult から BigInteger 値を取得します。

構文

```
BigInteger WSResult.getBigIntegerValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス** 値の取得で問題が発生した場合にスローされます。

getBooleanArrayValue メソッド

この WSResult から java.lang.Boolean 配列値を取得します。

構文

```
Boolean[] WSResult.getBooleanArrayValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

getBooleanValue メソッド

この WSResult から java.lang.Boolean 値を取得します。

構文

```
Boolean WSResult.getBooleanValue(String elementName) throws WSException
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

getByteArrayValue メソッド

この WSResult から java.lang.Byte 配列値を取得します。

構文

```
Byte[] WSResult.getByteArrayValue(String elementName) throws WSException
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

getByteValue メソッド

この WSResult から java.lang.Byte 値を取得します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
Byte WSResult.getByteValue(String elementName) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス** 値の取得で問題が発生した場合にスローされます。

getCharacterArrayValue メソッド

この WSResult から java.lang.Character 配列値を取得します。

構文

```
Character[] WSResult.getCharacterArrayValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス** 値の取得で問題が発生した場合にスローされます。

getCharacterValue メソッド

この WSResult から java.lang.Character 値を取得します。

構文

```
Character WSResult.getCharacterValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

getDoubleArrayValue メソッド

この WSResult から java.lang.Double 配列値を取得します。

構文

```
Double[] WSResult.getDoubleArrayValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

getDoubleValue メソッド

この WSResult から java.lang.Double 値を取得します。

構文

```
Double WSResult.getDoubleValue(String elementName) throws WSException
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

getErrorMessage メソッド

エラーメッセージを取得します。

構文

```
String WSResult.getErrorMessage()
```

戻り値

エラーメッセージ。

getFloatArrayValue メソッド

この WSResult から java.lang.Float 配列値を取得します。

構文

```
Float[] WSResult.getFloatArrayValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス** 値の取得で問題が発生した場合にスローされます。

getFloatValue メソッド

この WSResult から java.lang.Float 値を取得します。

構文

```
Float WSResult.getFloatValue(String elementName) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス** 値の取得で問題が発生した場合にスローされます。

getIntegerArrayValue メソッド

この WSResult から java.lang.Integer 配列値を取得します。

構文

```
Integer[] WSResult.getIntegerArrayValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス** 値の取得で問題が発生した場合にスローされます。

getIntegerValue メソッド

この WSResult から java.lang.Integer 値を取得します。

構文

```
Integer WSResult.getIntegerValue(String elementName) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス** 値の取得で問題が発生した場合にスローされます。

getLongArrayValue メソッド

この WSResult から java.lang.Long 配列値を取得します。

構文

```
Long[] WSResult.getLongArrayValue(String elementName) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

getLongValue メソッド

この WSResult から java.lang.Long 値を取得します。

構文

```
Long WSResult.getLongValue(String elementName) throws WSException
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

getObjectArrayValue メソッド

この WSResult から複合型の値の配列を取得します。

構文

```
Object[] WSResult.getObjectArrayValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

getObjectValue メソッド

この WSResult から複合型の値を取得します。

構文

```
Object WSResult.getObjectValue(String elementName) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス** 値の取得で問題が発生した場合にスローされます。

getPrimitiveBooleanArrayValue メソッド

この WSResult から boolean 配列値を取得します。

構文

```
boolean[] WSResult.getPrimitiveBooleanArrayValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス** 値の取得で問題が発生した場合にスローされます。

getPrimitiveBooleanValue メソッド

この WSResult から boolean 値を取得します。

構文

```
boolean WSResult.getPrimitiveBooleanValue(
     String elementName
) throws WSException
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス** 値の取得で問題が発生した場合にスローされます。

getPrimitiveByteArrayValue メソッド

この WSResult から byte 配列値を取得します。

構文

```
byte[] WSResult.getPrimitiveByteArrayValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス** 値の取得で問題が発生した場合にスローされます。

getPrimitiveByteValue メソッド

この WSResult から byte 値を取得します。

構文

```
byte WSResult.getPrimitiveByteValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

getPrimitiveCharArrayValue メソッド

この WSResult から char 配列値を取得します。

構文

```
char[] WSResult.getPrimitiveCharArrayValue(
      String elementName
) throws WSException
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

getPrimitiveCharValue メソッド

この WSResult から char 値を取得します。

構文

```
char WSResult.getPrimitiveCharValue(
      String elementName
) throws WSException
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

getPrimitiveDoubleArrayValue メソッド

この WSResult から double 配列値を取得します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

```
double[] WSResult.getPrimitiveDoubleArrayValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス** 値の取得で問題が発生した場合にスローされます。

getPrimitiveDoubleValue メソッド

この WSResult から double 値を取得します。

構文

```
double WSResult.getPrimitiveDoubleValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス** 値の取得で問題が発生した場合にスローされます。

getPrimitiveFloatArrayValue メソッド

この WSResult から float 配列値を取得します。

構文

```
float[] WSResult.getPrimitiveFloatArrayValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

getPrimitiveFloatValue メソッド

この WSResult から float 値を取得します。

構文

```
float WSResult.getPrimitiveFloatValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

getPrimitiveIntArrayValue メソッド

この WSResult から int 配列値を取得します。

構文

```
int[] WSResult.getPrimitiveIntArrayValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

getPrimitiveIntValue メソッド

この WSResult から int 値を取得します。

構文

```
int WSResult.getPrimitiveIntValue(String elementName) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス** 値の取得で問題が発生した場合にスローされます。

getPrimitiveLongArrayValue メソッド

この WSResult から long 配列値を取得します。

構文

```
long[] WSResult.getPrimitiveLongArrayValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス** 値の取得で問題が発生した場合にスローされます。

getPrimitiveLongValue メソッド

この WSResult から long 値を取得します。

構文

```
long WSResult.getPrimitiveLongValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス** 値の取得で問題が発生した場合にスローされます。

getPrimitiveShortArrayValue メソッド

この WSResult から short 配列値を取得します。

構文

```
short[] WSResult.getPrimitiveShortArrayValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス** 値の取得で問題が発生した場合にスローされます。

getPrimitiveShortValue メソッド

この WSResult から short 値を取得します。

構文

```
short WSResult.getPrimitiveShortValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

getRequestID メソッド

この WSResult が表す要求 ID を取得します。

構文

```
String WSResult.getRequestID()
```

戻り値

要求 ID。

備考

要求の作成時とは異なるアプリケーションで実行される Web サービス要求に対応する WSResult を取得するために、この要求 ID が必要な場合は、アプリケーションの実行と実行の間でこの要求 ID を保持する必要があります。

getShortArrayValue メソッド

この WSResult から java.lang.Short 配列値を取得します。

構文

```
Short[] WSResult.getShortArrayValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

getShortValue メソッド

この WSResult から java.lang.Short 値を取得します。

構文

```
Short WSResult.getShortValue(String elementName) throws WSException
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

getStatus メソッド

この WSResult のステータスを取得します。

構文

```
int WSResult.getStatus() throws QAException, WSException
```

戻り値

ステータスコード。

参照

- WSStatus クラス [QAnywhere Java]653 ページ

getStringArrayValue メソッド

この WSResult から String 配列値を取得します。

構文

```
String[] WSResult.getStringArrayValue(
     String elementName
) throws WSException
```

パラメーター

- **elementName**　この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス**　値の取得で問題が発生した場合にスローされます。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

getStringValue メソッド

この WSResult から String 値を取得します。

構文

```
String WSResult.getStringValue(String elementName) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス** 値の取得で問題が発生した場合にスローされます。

getValue メソッド

この WSResult から複合型の値を取得します。

構文

```
Object WSResult.getValue(String elementName) throws WSException
```

パラメーター

- **elementName** この値の WSDL ドキュメント内の要素名。

戻り値

値。

例外

- **WSException クラス** 値の取得で問題が発生した場合にスローされます。

WSStatus クラス

このクラスは、Web サービス要求のステータスコードを定義します。

構文

```
public class WSStatus
```

メンバー

継承されたメンバーを含む WSStatus クラスのすべてのメンバー。

名前	説明
STATUS_ERROR 変数	要求の処理でエラーがありました。
STATUS_QUEUED 変数	要求はサーバーへの配信用キューに登録されました。
STATUS_RESULT_AVAILABLE 変数	要求の結果を取得できます。
STATUS_SUCCESS 変数	要求は正常に処理されました。

STATUS_ERROR 変数

要求の処理でエラーがありました。

構文

```
final int WSStatus.STATUS_ERROR
```

STATUS_QUEUED 変数

要求はサーバーへの配信用キューに登録されました。

構文

```
final int WSStatus.STATUS_QUEUED
```

備考

要求の最終ステータスはまだ不明です。

STATUS_RESULT_AVAILABLE 変数

要求の結果を取得できます。

構文

```
final int WSStatus.STATUS_RESULT_AVAILABLE
```

STATUS_SUCCESS 変数

要求は正常に処理されました。

構文

```
final int WSStatus.STATUS_SUCCESS
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

QAnywhere SQL API リファレンス

メッセージのプロパティ、ヘッダー、内容

この項では、メッセージヘッダー、メッセージの内容、メッセージプロパティを容易に設定できる QAnywhere SQL ストアドプロシージャーについて説明します。

メッセージヘッダー

メッセージヘッダー情報の取得と設定は、次のストアドプロシージャーを使用して行います。

「メッセージヘッダー」695 ページを参照してください。

ml_qa_getaddress

メッセージの QAnywhere アドレスを返します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

戻り値

VARCHAR(128) 型の QAnywhere のメッセージアドレス。QAnywhere のメッセージアドレスは、*id*¥*queue-name* の形式を取ります。

このヘッダーは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「QAnywhere メッセージアドレス」60 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ

例

次の例では、メッセージを受信して、そのアドレスをデータベースサーバーメッセージウィンドウに出力します。

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

```
    declare @addr varchar(128);
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @addr = ml_qa_getaddress( @msgid );
    message 'message to address ' || @addr || ' received';
    commit;
end
```

ml_qa_getexpiration

メッセージの有効期限を返します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

戻り値

TIMESTAMP 型の有効期限。有効期限がない場合は NULL が返されます。

備考

ml_qa_putmessage の処理が完了した後で、対象となる受信者によって指定の時間内にメッセージが受信されない場合、メッセージは期限切れになります。期限切れになったメッセージは、QAnywhere のデフォルトの削除ルールを使って削除できます。

このヘッダーは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「メッセージの削除ルール」786 ページ
- 「QAnywhere メッセージの送信」63 ページ
- 「ml_qa_setexpiration」662 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ

例

次の例では、メッセージを受信して、その有効期限をデータベースサーバーメッセージウィンドウに出力します。

```
begin
    declare @msgid varchar(128);
    declare @expires timestamp;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @expires = ml_qa_getexpiration( @msgid );
    message 'message would have expired at ' || @expires || ' if it had not been received';
```

```
    commit;
end
```

ml_qa_getinreplytoid

メッセージの in-reply-to ID を返します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

戻り値

VARCHAR(128) 型の in-reply-to ID。

備考

クライアントは、In-Reply-To ID ヘッダーフィールドを使用してメッセージ間リンクを設定できます。これは、応答メッセージをそれに対応する要求メッセージとリンクさせる場合によく使用されます。

in-reply-to ID は、返信対象メッセージの ID です。

このヘッダーは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_setinreplytoid」663 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ

例

次の例では、メッセージを受信して、その in-reply-to ID をデータベースサーバーメッセージウィンドウに出力します。

```
begin
   declare @msgid varchar(128);
   declare @inreplytoid varchar(128);
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @inreplytoid = ml_qa_getinreplytoid( @msgid );
   message 'message is likely a reply to the message with id ' || @inreplytoid;
   commit;
end
```

ml_qa_getpriority

メッセージの優先度レベルを返します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

戻り値

INTEGER 型の優先度レベル。

備考

QAnywhere API では、10 レベルの優先度が定義されています。0 が最低の優先度、9 が最高の優先度を表します。クライアントは、優先度 0 ～ 4 を通常のメッセージ、優先度 5 ～ 9 を緊急度の高いメッセージとみなす必要があります。

このヘッダーは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_setpriority」664 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ

例

次の例では、メッセージを受信して、その優先度をデータベースサーバーメッセージウィンドウに出力します。

```
begin
   declare @msgid varchar(128);
   declare @priority integer;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @priority = ml_qa_getpriority( @msgid );
   message 'a message with priority ' || @priority || ' has been received';
   commit;
end
```

ml_qa_getredelivered

受信されたが受信確認されていないメッセージであるかどうかを示す値を返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

戻り値

再配信のステータスを示す BIT 型の値。**1** はメッセージが再配信中であることを示し、**0** は再配信中ではないことを示します。

備考

受信されたが受信確認されていないメッセージは、再配信される場合があります。たとえば、メッセージは受信されたものの、メッセージを受信したアプリケーションがメッセージの内容の処理を完了する前にクラッシュした場合などです。このような場合には、メッセージは再配信とマーク付けされて、メッセージの処理が完了していない可能性があることを示す警告が受信側に送信されます。

たとえば、次のような手順を経てメッセージが受信されるとします。

1. 非トランザクション指向の QAnywhere Manager を使用するアプリケーションがメッセージを受信します。
2. このアプリケーションは、T1 というデータベーステーブルにメッセージの内容とメッセージ ID を書き込み、変更をコミットします。
3. アプリケーションがメッセージの受信を確認します。

手順 1 と 2、または手順 2 と 3 の間でアプリケーションに障害が発生した場合は、アプリケーションの再起動時にメッセージが再配信されます。

手順 1 と 2 の間で障害が発生した場合は、手順 2 と 3 を実行してメッセージを再配信してください。手順 2 と 3 の間で障害が発生した場合は、メッセージはすでに処理されているので、メッセージの確認のみ必要です。

アプリケーションの障害時に何が起きたかを特定するには、アプリケーションで ml_qa_getredelivered を呼び出して、メッセージがすでに再配信されていたかどうかをチェックします。テーブル T1 で検索する必要があるのは、再配信されるメッセージだけです。アプリケーションで障害が発生することはほとんどないので、この方法は、受信したメッセージのメッセージ ID にアプリケーションからアクセスして、メッセージがテーブル T1 に格納されているかどうかをチェックするよりも効率的です。

このヘッダーは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ

例

次の例では、メッセージを受信し、そのメッセージが以前に配信されたが受信されていなかった場合に、メッセージ ID をデータベースサーバーメッセージウィンドウに出力します。

```
begin
   declare @msgid varchar(128);
   declare @redelivered bit;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @redelivered = ml_qa_getredelivered( @msgid );
   if @redelivered = 1 then
      message 'message with message ID ' || @msgid || ' has been redelivered';
   end if;
   commit;
end
```

ml_qa_getreplytoaddress

メッセージの返信先アドレスを返します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

戻り値

VARCHAR(128) 型の返信アドレス。

備考

このヘッダーは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_setreplytoaddress」664 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

次の例では、受信されたメッセージに返信アドレスがある場合に、「message received」という内容のメッセージを返信アドレスに送信します。

```
begin
   declare @msgid varchar(128);
   declare @rmsgid varchar(128);
   declare @replytoaddr varchar(128);
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @replytoaddr = ml_qa_getreplytoaddress( @msgid );
   if @replytoaddr is not null then
      set @rmsgid = ml_qa_createmessage();
      call ml_qa_settextcontent( @rmsgid, 'message received' );
      call ml_qa_putmessage( @rmsgid, @replytoaddr );
   end if;
   commit;
end
```

ml_qa_gettimestamp

メッセージの作成時刻を返します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

戻り値

TIMESTAMP 型のメッセージ作成時刻。

備考

メッセージの Timestamp ヘッダーフィールドには、メッセージが作成された時刻が格納されます。これは、協定世界時 (UTC: Coordinated Universal Time) です。この時刻は、メッセージが実際に送信された時刻ではないので注意してください。メッセージの実際の送信時刻は、トランザクションの進行状況やクライアント側のキューに登録されているその他のメッセージの影響で、作成時刻よりも遅れる可能性があります。

このヘッダーは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ

例

次の例では、メッセージを受信して、その作成時刻をデータベースサーバーメッセージウィンドウに出力します。

```
begin
   declare @msgid varchar(128);
   declare @ts timestamp;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @ts = ml_qa_gettimestamp( @msgid );
   message 'message received with create time: ' || @ts ;
   commit;
end
```

ml_qa_setexpiration

メッセージの有効期限を設定します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	有効期限	TIMESTAMP

備考

このヘッダーは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_getexpiration」656 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ

例

次の例では、メッセージが作成されて 3 日以内に配信されない場合に、メッセージが期限切れになります。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setexpiration( @msgid, dateadd( day, 3, current timestamp ) );
   call ml_qa_settextcontent( @msgid, 'time-limited offer' );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ml_qa_setinreplytoid

メッセージの in-reply-to ID を設定します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	in-reply-to ID	VARCHAR(128)

備考

in-reply-to ID は、返信を追跡するために電子メールシステムで使用する in-reply-to ID のようなものです。

通常は、返信対象メッセージがある場合に、そのメッセージ ID として in-reply-to ID を設定します。

クライアントは、In-Reply-To ID ヘッダーフィールドを使用してメッセージ間リンクを設定できます。これは、応答メッセージをそれに対応する要求メッセージとリンクさせる場合によく使用されます。

メッセージの送信後にこのヘッダーを変更することはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_getinreplytoid」657 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ

例

次の例では、返信アドレスが指定されたメッセージが受信されたら、in-reply-to-id にメッセージ ID を指定して返信メッセージを作成し、送信します。

```
begin
   declare @msgid varchar(128);
   declare @rmsgid varchar(128);
   declare @replyaddr varchar(128);
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @replyaddr = ml_qa_getreplyaddress( @msgid );
   if @replyaddr is not null then
      set @rmsgid = ml_qa_createmessage();
      call ml_qa_settextcontent( @rmsgid, 'message received' );
      call ml_qa_setinreplytoid( @rmsgid, @msgid );
      call ml_qa_putmessage( @rmsgid, @replyaddr );
   end if;
   commit;
end
```

ml_qa_setpriority

メッセージの優先度を設定します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	優先度	INTEGER

備考

QAnywhere API では、10 レベルの優先度が定義されています。0 が最低の優先度、9 が最高の優先度を表します。クライアントは、優先度 0 ～ 4 を通常のメッセージ、優先度 5 ～ 9 を緊急度の高いメッセージとみなす必要があります。

メッセージの送信後にこのヘッダーを変更することはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_getpriority」658 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ

例

次の例では、優先度の高いメッセージを送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setpriority( @msgid, 9 );
   call ml_qa_settextcontent( @msgid, 'priority content' );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

ml_qa_setreplytoaddress

メッセージの返信アドレスを設定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	返信アドレス	VARCHAR(128)

備考

メッセージの送信後にこのヘッダーを変更することはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_getreplytoaddress」660 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ

例

次の例では、返信アドレスをメッセージに追加します。メッセージの受信者はこの返信アドレスを使用して、返信メッセージを作成できます。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setreplytoaddress( @msgid, 'myaddress' );
   call ml_qa_settextcontent( @msgid, 'some content' );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

メッセージプロパティ

カスタムメッセージプロパティの取得と設定、および事前定義済みメッセージプロパティの取得は、次のストアドプロシージャーを使用して行います。

参照

- 「メッセージプロパティ」698 ページ

ml_qa_getbooleanproperty

指定されたメッセージプロパティを SQL BIT データ型で返します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)

戻り値

BIT 型のプロパティ値。

備考

メッセージプロパティ値が範囲外の場合は、SQL エラー SQLSTATE 22003 が発生します。

このプロパティは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_setbooleanproperty」674 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ
- 「カスタムのメッセージプロパティ」700 ページ

例

次の例では、メッセージを受信して、boolean 型プロパティ mybooleanproperty の値をデータベースサーバーメッセージウィンドウに出力します。

```
begin
   declare @msgid varchar(128);
   declare @prop bit;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @prop = ml_qa_getbooleanproperty( @msgid, 'mybooleanproperty' );
   message 'message property mybooleanproperty is set to ' || @prop;
   commit;
end
```

ml_qa_getbyteproperty

指定されたメッセージプロパティを SQL TINYINT データ型で返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)

戻り値

TINYINT 型のプロパティ値。

備考

メッセージプロパティ値が範囲外の場合は、SQL エラー SQLSTATE 22003 が発生します。

このプロパティは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_setbyteproperty」675 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ
- 「カスタムのメッセージプロパティ」700 ページ

例

次の例では、メッセージを受信して、byte 型プロパティ mybyteproperty の値をデータベースサーバーメッセージウィンドウに出力します。

```
begin
   declare @msgid varchar(128);
   declare @prop tinyint;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @prop = ml_qa_getbyteproperty( @msgid, 'mybyteproperty' );
   message 'message property mybyteproperty is set to ' || @prop;
   commit;
end
```

ml_qa_getdoubleproperty

指定されたメッセージプロパティを SQL DOUBLE データ型で返します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)

戻り値

DOUBLE 型のプロパティ値。

備考

メッセージプロパティ値が範囲外の場合は、SQL エラー SQLSTATE 22003 が発生します。

このプロパティは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_setdoubleproperty」676 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ
- 「カスタムのメッセージプロパティ」700 ページ

例

次の例では、メッセージを受信して、double 型プロパティ mydoubleproperty の値をデータベースサーバーメッセージウィンドウに出力します。

```
begin
   declare @msgid varchar(128);
   declare @prop double;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @prop = ml_qa_getdoubleproperty( @msgid, 'mydoubleproperty' );
   message 'message property mydoubleproperty is set to ' || @prop;
   commit;
end
```

ml_qa_getfloatproperty

指定されたメッセージプロパティを SQL FLOAT データ型で返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)

戻り値

FLOAT 型のプロパティ値。

備考

メッセージプロパティ値が範囲外の場合は、SQL エラー SQLSTATE 22003 が発生します。

このプロパティは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_setfloatproperty」677 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ
- 「カスタムのメッセージプロパティ」700 ページ

例

次の例では、メッセージを受信して、float 型プロパティ myfloatproperty の値をデータベースサーバーメッセージウィンドウに出力します。

```
begin
   declare @msgid varchar(128);
   declare @prop float;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @prop = ml_qa_getfloatproperty( @msgid, 'myfloatproperty' );
   message 'message property myfloatproperty is set to ' || @prop;
   commit;
end
```

ml_qa_getintproperty

指定されたメッセージプロパティを SQL INTEGER データ型で返します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)

戻り値

INTEGER 型のプロパティ値。

備考

メッセージプロパティ値が範囲外の場合は、SQL エラー SQLSTATE 22003 が発生します。

このプロパティは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_setintproperty」677 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ
- 「カスタムのメッセージプロパティ」700 ページ

例

次の例では、メッセージを受信して、integer 型プロパティ myintproperty の値をデータベースサーバーメッセージウィンドウに出力します。

```
begin
   declare @msgid varchar(128);
   declare @prop integer;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @prop = ml_qa_getintproperty( @msgid, 'myintproperty' );
   message 'message property myintproperty is set to ' || @prop;
   commit;
end
```

ml_qa_getlongproperty

指定されたメッセージプロパティを SQL BIGINT データ型で返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)

戻り値

BIGINT 型のプロパティ値。

備考

メッセージプロパティ値が範囲外の場合は、SQL エラー SQLSTATE 22003 が発生します。

このプロパティは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_setlongproperty」678 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ
- 「カスタムのメッセージプロパティ」700 ページ

ml_qa_getpropertynames

指定されたメッセージのプロパティ名を取得します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

備考

このストアドプロシージャーは、指定されたメッセージのプロパティ名の結果セットを開きます。メッセージ ID パラメーターが、受信されたメッセージの ID であることが必要です。

結果セットは単一の VARCHAR(128) カラムです。このカラムの各行にメッセージプロパティ名が含まれています。QAnywhere の予約済みプロパティ名 ("ias_" または "QA" のプレフィクスが付いているプロパティ名) は返されません。

このプロパティは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ
- 「カスタムのメッセージプロパティ」700 ページ

例

次の例では、メッセージ ID が msgid であるメッセージに対するプロパティ名の結果セット上でカーソルを宣言します。次に、アドレスが clientid¥queuename であるメッセージを取得し、カーソルを開いてそのメッセージのプロパティ名にアクセスしてから、次のプロパティ名をフェッチします。

```
begin
 declare prop_name_cursor cursor for
  call ml_qa_getpropertynames( @msgid );
 declare @msgid varchar(128);
 declare @name varchar(128);

 set @msgid = ml_qa_getmessage( 'clientid¥queuename' );
 open prop_name_cursor;
 lp: loop
  fetch next prop_name_cursor into name;
  if sqlcode <> 0 then leave lp end if;
  ...
 end loop;
 close prop_name_cursor;
end
```

ml_qa_getshortproperty

指定されたメッセージプロパティを SQL SMALLINT データ型で返します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)

戻り値

SMALLINT 型のプロパティ値。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

メッセージプロパティ値が範囲外の場合は、SQL エラー SQLSTATE 22003 が発生します。

このプロパティは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_setshortproperty」679 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ
- 「カスタムのメッセージプロパティ」700 ページ

例

次の例では、メッセージを受信して、short 型プロパティ myshortproperty の値をデータベースサーバーメッセージウィンドウに出力します。

```
begin
   declare @msgid varchar(128);
   declare @prop smallint;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @prop = ml_qa_getshortproperty( @msgid, 'myshortproperty' );
   message 'message property myshortproperty is set to ' || @prop;
   commit;
end
```

ml_qa_getstringproperty

指定されたメッセージプロパティを SQL LONG VARCHAR データ型で返します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)

戻り値

LONG VARCHAR 型のプロパティ値。

備考

メッセージプロパティ値が範囲外の場合は、SQL エラー SQLSTATE 22003 が発生します。

このプロパティは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_setstringproperty」680 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ
- 「カスタムのメッセージプロパティ」700 ページ

例

次の例では、メッセージを受信して、string 型プロパティ mystringproperty の値をデータベースサーバーメッセージウィンドウに出力します。

```
begin
   declare @msgid varchar(128);
   declare @prop long varchar;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @prop = ml_qa_getstringproperty( @msgid, 'mystringproperty' );
   message 'message property mystringproperty is set to ' || @prop;
   commit;
end
```

ml_qa_setbooleanproperty

指定されたメッセージプロパティを SQL BIT データ型で設定します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)
3	プロパティ値	BIT

備考

メッセージの送信後にこのプロパティを変更することはできません。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_getbooleanproperty」665 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ
- 「カスタムのメッセージプロパティ」700 ページ

例

次の例では、メッセージを作成し、boolean 型プロパティ mybooleanproperty1 と mybooleanproperty2 を設定して、アドレス clientid¥queuename に送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setbooleanproperty( @msgid, 'mybooleanproperty1', 0 );
   call ml_qa_setbooleanproperty( @msgid, 'mybooleanproperty2', 1 );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

ml_qa_setbyteproperty

指定されたメッセージプロパティを SQL TINYINT データ型で設定します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)
3	プロパティ値	TINYINT

備考

メッセージの送信後にこのプロパティを変更することはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_getbyteproperty」666 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ
- 「カスタムのメッセージプロパティ」700 ページ

例

次の例では、メッセージを作成し、byte 型プロパティ mybyteproperty1 と mybyteproperty2 を設定して、アドレス clientid¥queuename に送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setbyteproperty( @msgid, 'mybyteproperty1', 0 );
   call ml_qa_setbyteproperty( @msgid, 'mybyteproperty2', 255 );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

ml_qa_setdoubleproperty

指定されたメッセージプロパティを SQL DOUBLE データ型で設定します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)
3	プロパティ値	DOUBLE

備考

メッセージの送信後にこのプロパティを変更することはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_getdoubleproperty」667 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ
- 「カスタムのメッセージプロパティ」700 ページ

例

次の例では、メッセージを作成し、double 型プロパティ mydoubleproperty1 と mydoubleproperty2 を設定して、アドレス clientid¥queuename に送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setdoubleproperty( @msgid, 'mydoubleproperty1', -12.34e-56 );
   call ml_qa_setdoubleproperty( @msgid, 'mydoubleproperty2', 12.34e56 );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
    commit;
end
```

ml_qa_setfloatproperty

指定されたメッセージプロパティを SQL FLOAT データ型で設定します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)
3	プロパティ値	FLOAT

備考

メッセージの送信後にこのプロパティを変更することはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_getfloatproperty」668 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ
- 「カスタムのメッセージプロパティ」700 ページ

例

次の例では、メッセージを作成し、float 型プロパティ myfloatproperty1 と myfloatproperty2 を設定して、アドレス clientid¥queuename に送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setfloatproperty( @msgid, 'myfloatproperty1', -1.3e-5 );
   call ml_qa_setfloatproperty( @msgid, 'myfloatproperty2', 1.3e5 );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

ml_qa_setintproperty

指定されたメッセージプロパティを SQL INTEGER データ型で設定します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)
3	プロパティ値	INTEGER

備考

メッセージの送信後にこのプロパティを変更することはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_getintproperty」669 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ
- 「カスタムのメッセージプロパティ」700 ページ

例

次の例では、メッセージを作成し、integer 型プロパティ myintproperty1 と myintproperty2 を設定して、アドレス clientid¥queuename に送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setintproperty( @msgid, 'myintproperty1', -1234567890 );
   call ml_qa_setintproperty( @msgid, 'myintproperty2',  1234567890 );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

ml_qa_setlongproperty

指定されたメッセージプロパティを SQL BIGINT データ型で設定します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

項目	説明	備考
3	プロパティ値	BIGINT

備考

メッセージの送信後にこのプロパティを変更することはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_getlongproperty」670 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ
- 「カスタムのメッセージプロパティ」700 ページ

例

次の例では、メッセージを作成し、long 型プロパティ mylongproperty1 と mylongproperty2 を設定して、アドレス clientid¥queuename に送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setlongproperty( @msgid, 'mylongproperty1', -12345678900987654321 );
   call ml_qa_setlongproperty( @msgid, 'mylongproperty2',  12345678900987654321 );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

ml_qa_setshortproperty

指定されたメッセージプロパティを SQL SMALLINT データ型で設定します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)
3	プロパティ値	SMALLINT

備考

メッセージの送信後にこのプロパティを変更することはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_getshortproperty」672 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ
- 「カスタムのメッセージプロパティ」700 ページ

例

次の例では、メッセージを作成し、short 型プロパティ myshortproperty1 と myshortproperty2 を設定して、アドレス clientid¥queuename に送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setshortproperty( @msgid, 'myshortproperty1', -12345 );
   call ml_qa_setshortproperty( @msgid, 'myshortproperty2',  12345 );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

ml_qa_setstringproperty

指定されたメッセージプロパティを SQL LONG VARCHAR データ型で設定します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	プロパティ名	VARCHAR(128)
3	プロパティ値	LONG VARCHAR

備考

メッセージの送信後にこのプロパティを変更することはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_getstringproperty」673 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ
- 「カスタムのメッセージプロパティ」700 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

次の例では、メッセージを作成し、string 型プロパティ mystringproperty1 と mystringproperty2 を設定して、アドレス clientid¥queuename に送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setstringproperty( @msgid, 'mystringproperty1', 'c:¥¥temp' );
   call ml_qa_setstringproperty( @msgid, 'mystringproperty2',  'first line¥nsecond line' );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

メッセージの内容

メッセージの内容の取得と設定は、次のストアドプロシージャーを使用して行います。

ml_qa_getbinarycontent

バイナリメッセージの内容を返します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

戻り値

LONG BINARY 型のメッセージの内容。

メッセージの内容がバイナリではなくテキストの場合は、NULL を返します。

この内容は、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_setbinarycontent」684 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ
- 「ml_qa_getcontentclass」682 ページ

例

次の例では、メッセージの暗号化された内容を復号化してデータベースサーバーメッセージウィンドウに出力します。

```
begin
   declare @msgid varchar(128);
   declare @content long binary;
   declare @plaintext long varchar;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @content = ml_qa_getbinarycontent( @msgid );
   set @plaintext = decrypt( @content, 'mykey' );
   message 'message content decrypted: ' || @plaintext;
   commit;
end
```

ml_qa_getcontentclass

メッセージタイプ (テキストまたはバイナリ) を返します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

戻り値

INTEGER 型の内容クラス。

次のいずれかの値を返します。

- **1**　メッセージの内容がバイナリであり、ストアドプロシージャー ml_qa_getbinarycontent を使用して読み込む必要があることを示します。
- **2**　メッセージの内容がテキストであり、ストアドプロシージャー ml_qa_gettextcontent を使用して読み込む必要があることを示します。

備考

この内容は、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ
- 「ml_qa_getbinarycontent」681 ページ
- 「ml_qa_gettextcontent」683 ページ

例

次の例では、メッセージを受信して、その内容をデータベースサーバーメッセージウィンドウに出力します。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
begin
   declare @msgid varchar(128);
   declare @contentclass integer;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @contentclass = ml_qa_getcontentclass( @msgid );
   if @contentclass = 1 then
      message 'message  binary is ' || ml_qa_getbinarycontent( @msgid );
   elseif @contentclass = 2 then
      message 'message  text is ' || ml_qa_gettextcontent( @msgid );
   end if;
   commit;
end
```

ml_qa_gettextcontent

テキストメッセージの内容を返します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

戻り値

LONG VARCHAR 型のメッセージの内容。

メッセージの内容がテキストではなくバイナリの場合は、NULL を返します。

備考

この内容は、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_settextcontent」684 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ
- 「ml_qa_getcontentclass」682 ページ

例

次の例では、メッセージの内容をデータベースサーバーメッセージウィンドウに出力します。

```
begin
   declare @msgid varchar(128);
   declare @content long binary;
   set @msgid = ml_qa_getmessage( 'myaddress' );
   set @content = ml_qa_gettextcontent( @msgid );
   message 'message content: ' || @content ;
```

```
    commit;
end
```

ml_qa_setbinarycontent

メッセージにバイナリの内容を設定します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	内容	LONG BINARY

メッセージの送信後にこの内容を変更することはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_getbinarycontent」681 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ

例

次の例では、暗号化した内容を含むメッセージを作成してから送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_setbinarycontent( @msgid, encrypt( 'my secret message', 'mykey' ) );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

ml_qa_settextcontent

メッセージにテキストの内容を設定します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

項目	説明	備考
2	内容	LONG VARCHAR

備考

メッセージの送信後にこの内容を変更することはできません。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_gettextcontent」683 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ

例

次の例では、メッセージを作成してから、指定された内容を設定します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_settextcontent( @msgid, 'my simple message' );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

メッセージストアプロパティ

クライアントメッセージストアのプロパティの取得と設定は、次のストアドプロシージャーを使用して行います。

メッセージストアプロパティの詳細については、「クライアントメッセージストアプロパティ」29 ページを参照してください。

ml_qa_getstoreproperty

クライアントメッセージストアのプロパティを返します。

パラメーター

項目	説明	備考
1	プロパティ名	VARCHAR(128)

戻り値

LONG VARCHAR 型のプロパティ値。

備考

クライアントメッセージストアのプロパティは、このクライアントメッセージストアへのすべての接続から読み込むことができます。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_setstoreproperty」686 ページ

例

次の例では、メッセージストアの現在の同期ポリシーを取得して、データベースサーバーメッセージウィンドウに出力します。

```
begin
   declare @policy varchar(128);
   set @policy = ml_qa_getstoreproperty( 'policy' );
   message 'the current policy for synchronizing this message store is ' || @policy;
end
```

ml_qa_setstoreproperty

クライアントメッセージストアのプロパティを設定します。

パラメーター

項目	説明	備考
1	プロパティ名	VARCHAR(128)
2	プロパティ値	SMALLINT

備考

クライアントメッセージストアのプロパティは、このクライアントメッセージストアへのすべての接続から読み込むことができます。プロパティ値は、サーバーに対しても同期されます。サーバーでは転送ルールでプロパティ値を使用できます。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_getstoreproperty」685 ページ

例

次の例では、メッセージストアの同期ポリシーを自動に設定します。

```
begin
   call ml_qa_setstoreproperty( 'policy', 'automatic' );
   commit;
end
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

メッセージの管理

QAnywhere クライアントトランザクションを管理するには、次のストアドプロシージャーを使用します。

ml_qa_createmessage

新規メッセージのメッセージ ID を返します。

戻り値

新規メッセージのメッセージ ID。

備考

このストアドプロシージャーは、メッセージを作成するために使用します。作成したメッセージは、内容、プロパティ、ヘッダーを関連付けて送信できます。

内容、プロパティ、ヘッダーを関連付けるには、ml_qa_set で始まる QAnywhere ストアドプロシージャーを使用します。たとえば、ml_qa_setbinarycontent を使用してバイナリメッセージを作成したり、ml_qa_settextcontent を使用してテキストメッセージを作成します。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「メッセージヘッダー」655 ページ
- 「メッセージプロパティ」665 ページ
- 「メッセージの内容」681 ページ

例

次の例では、メッセージを作成して内容を設定し、アドレス clientid¥queuename に送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_settextcontent( @msgid, 'some content' );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

ml_qa_getmessage

キューに登録されている、指定されたアドレスを持つメッセージのうち、次に取り出せるもののメッセージ ID を返します。メッセージがキューに登録されるまで処理をブロックします。

パラメーター

項目	説明	備考
1	アドレス	VARCHAR(128)

戻り値

VARCHAR(128) 型のメッセージ ID。

このアドレスを持つ、キューに登録されたメッセージがない場合は、NULL を返します。

備考

指定された QAnywhere メッセージアドレス宛ての待機中のメッセージがあるかどうかを同期的にチェックするには、このストアドプロシージャーを使用します。指定された QAnywhere アドレス宛てのメッセージが発生した場合に、非同期で SQL プロシージャーを呼び出すには、Mobile Link Listener を使用します。

このストアドプロシージャーは、メッセージがキューに登録されるまで処理をブロックします。

ブロックを回避する方法の詳細については、「ml_qa_getmessagenowait」689 ページまたは「ml_qa_getmessagetimeout」690 ページを参照してください。

返されるメッセージ ID に対応するメッセージが受信されたとみなされるのは、現在のトランザクションがコミットされた後です。受信がコミットされたメッセージは、このストアドプロシージャーやいかなる QAnywhere API でも受信することができません。同様に、現在のトランザクションがロールバックされると、メッセージが受信されていないことを意味します。このため、以降の ml_qa_getmessage の呼び出しで同じメッセージ ID が返される可能性があります。

受信されたメッセージのプロパティと内容は、現在のトランザクションに対してコミットまたはロールバックが実行されるまでの間、各種の ml_qa_get ストアドプロシージャーを使用して読み込むことができます。現在のトランザクションに対してコミットまたはロールバックが実行された時点で、そのメッセージデータは読み込めなくなります。コミットを実行する前に、メッセージ内の必要なデータを表形式のデータとして、または SQL 変数に保存してください。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_getmessagenowait」689 ページ
- 「ml_qa_getmessagetimeout」690 ページ
- 「メッセージヘッダー」655 ページ
- 「メッセージプロパティ」665 ページ
- 「メッセージの内容」681 ページ

例

次の例では、アドレス myaddress に送信されたすべてのメッセージの内容を表示します。

```
begin
   declare @msgid varchar(128);
   loop
      set @msgid = ml_qa_getmessage( 'myaddress' );
```

```
        message 'a message with content ' || ml_qa_gettextcontent( @msgid ) || ' has been received';
        commit;
    end loop;
end
```

ml_qa_getmessagenowait

現在キューに登録されている、指定されたアドレスを持つメッセージのうち、次に取り出せるもののメッセージ ID を返します。

パラメーター

項目	説明	備考
1	アドレス	VARCHAR(128)

戻り値

VARCHAR(128) 型のメッセージ ID。

キューに登録されている、指定されたアドレスを持つメッセージのうち、次に取り出せるもののメッセージ ID を返します。このアドレスを持つ、キューに登録されたメッセージがない場合は、NULL を返します。

備考

指定された QAnywhere メッセージアドレス宛ての待機中のメッセージがあるかどうかを同期的にチェックするには、このストアドプロシージャーを使用します。指定された QAnywhere アドレス宛てのメッセージが発生した場合に、非同期で SQL プロシージャーを呼び出すには、Mobile Link Listener を使用します。

メッセージが準備できるまで処理をブロックする方法の詳細については、「ml_qa_getmessage」687 ページと「ml_qa_getmessagetimeout」690 ページを参照してください。

返されるメッセージ ID に対応するメッセージが受信されたとみなされるのは、現在のトランザクションがコミットされた後です。受信がコミットされたメッセージは、このストアドプロシージャーやいかなる QAnywhere API でも受信することができません。同様に、現在のトランザクションがロールバックされると、メッセージが受信されていないことを意味します。このため、以降の ml_qa_getmessagenowait の呼び出しで同じメッセージ ID が返される可能性があります。

受信されたメッセージのプロパティと内容は、現在のトランザクションに対してコミットまたはロールバックが実行されるまでの間、各種の ml_qa_get ストアドプロシージャーを使用して読み込むことができます。現在のトランザクションに対してコミットまたはロールバックが実行された時点で、そのメッセージデータは読み込めなくなります。コミットを実行する前に、メッセージ内の必要なデータを表形式のデータとして、または SQL 変数に保存してください。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「QAnywhere メッセージアドレス」60 ページ
- 「Listener」『Mobile Link サーバー起動同期』
- 「ml_qa_getmessagetimeout」690 ページ
- 「メッセージヘッダー」655 ページ
- 「メッセージプロパティ」665 ページ
- 「メッセージの内容」681 ページ

例

次の例では、キューに登録されている、アドレス myaddress を持つメッセージがすべて読み込まれるまで、これらすべてのメッセージの内容を表示します (一般には、各メッセージが読み込まれた後ではなく、最後のメッセージが読み込まれた後でコミットする方が効率的です)。

```
begin
   declare @msgid varchar(128);
   loop
      set @msgid = ml_qa_getmessagenowait( 'myaddress' );
      if @msgid is null then leave end if;
      message 'a message with content ' || ml_qa_gettextcontent( @msgid ) || ' has been received';
   end loop;
   commit;
end
```

ml_qa_getmessagetimeout

指定されたタイムアウト期間だけ待機してから、キューに登録されている、指定されたアドレスを持つメッセージのうち、次に取り出せるもののメッセージ ID を返します。

パラメーター

項目	説明	備考
1	アドレス	VARCHAR(128)
2	タイムアウト (ミリ秒単位)	整数

戻り値

VARCHAR(128) 型のメッセージ ID。

タイムアウト時間内に、このアドレスを持つキューに登録されたメッセージがない場合は、NULL を返します。

備考

指定された QAnywhere メッセージアドレス宛ての待機中のメッセージがあるかどうかを同期的にチェックするには、このストアドプロシージャーを使用します。指定された QAnywhere アド

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

レス宛てのメッセージが発生した場合に、非同期で SQL プロシージャーを呼び出すには、Mobile Link Listener を使用します。

返されるメッセージ ID に対応するメッセージが受信されたとみなされるのは、現在のトランザクションがコミットされた後です。受信がコミットされたメッセージは、このストアドプロシージャーやいかなる QAnywhere API でも受信することができません。同様に、現在のトランザクションがロールバックされると、メッセージが受信されていないことを意味します。このため、以降の ml_qa_getmessage の呼び出しで同じメッセージ ID が返される可能性があります。

受信されたメッセージのプロパティと内容は、現在のトランザクションに対してコミットまたはロールバックが実行されるまでの間、各種の ml_qa_get ストアドプロシージャーを使用して読み込むことができます。現在のトランザクションに対してコミットまたはロールバックが実行された時点で、そのメッセージデータは読み込めなくなります。コミットを実行する前に、メッセージ内の必要なデータを表形式のデータとして、または SQL 変数に保存してください。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_getmessage」687 ページ
- 「ml_qa_getmessagenowait」689 ページ

例

次の例では、アドレス myaddress に送信されたすべてのメッセージの内容をデータベースサーバーメッセージウィンドウに出力します。メッセージが受信されなかった場合は、データベースサーバーメッセージウィンドウを 10 秒ごとに更新します。

```
begin
   declare @msgid varchar(128);
   loop
      set @msgid = ml_qa_getmessagetimeout( 'myaddress', 10000 );
      if @msgid is null then
         message 'waiting for a message...';
      else
         message 'a message with content ' || ml_qa_gettextcontent( @msgid ) || ' has been received';
      commit;
      end if;
   end loop;
end
```

ml_qa_grant_messaging_permissions

QAnywhere ストアドプロシージャーを使用するパーミッションを他のユーザーに与えます。

パラメーター

項目	説明	備考
1	データベースユーザー ID	VARCHAR(128)

備考

QAnywhere ストアドプロシージャーを実行するパーミッションを自動的に与えられるのは、DBA 権限を持つユーザーだけです。それ以外のユーザーに対しては、DBA 権限を持つユーザーがこのストアドプロシージャーを実行して、パーミッションを付与する必要があります。

このプロシージャーは ml_qa_message_group というグループにユーザーを追加して、QAnywhere の全ストアドプロシージャーを実行するパーミッションを付与します。

参照

- 「SQL アプリケーションの設定」59 ページ

例

たとえば、user1 というデータベース ID を持つユーザーにメッセージングパーミッションを付与するには、次のコードを実行します。

```
call dbo.ml_qa_grant_messaging_permissions( 'user1' )
```

ml_qa_listener_queue

メッセージを非同期的に受信するために、**ml_qa_listener_*queue*** というストアドプロシージャー (*queue* はメッセージキューの名前) を作成します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は QAnywhere Listener から取得できます。

備考

注意
このストアドプロシージャーは他のすべての QAnywhere ストアドプロシージャーとは異なり、デフォルトでは提供されていません。ユーザーがストアドプロシージャー **ml_qa_listener_*queue*** (*queue* はメッセージキューの名前) を作成すると、そのストアドプロシージャーは QAnywhere によって使用されます。

メッセージは接続で同期的に受信できますが、多くの場合は非同期で受信する方が便利です。特定のアドレスを持つメッセージがキューに登録された場合に呼び出されるストアドプロシージャーを作成できます。このプロシージャーの名前は、**ml_qa_listener_*queue*** (*queue* はメッセージキューの名前) にする必要があります。このようなプロシージャーが存在する場合、そのプロシージャーは、指定されたアドレスを持つメッセージがキューに登録されるたびに呼び出されます。

このプロシージャーは別の接続から呼び出されます。このプロシージャーの実行中に SQL エラーが発生しないかぎり、メッセージの確認とコミットは自動的に行われます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

> **注意**
> このプロシージャー内では、コミットやロールバックを実行しないでください。

キューの名前は QAnywhere アドレスの一部です。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「QAnywhere メッセージアドレス」60 ページ
- 「非同期的なメッセージ受信」73 ページ
- 「同期的なメッセージ受信」71 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ

例

次の例では、アドレス executesql を持つメッセージがキューに登録されるたびに呼び出されるプロシージャーを作成します。この例のプロシージャーでは、メッセージの内容が、現在のデータベースに対して実行できる SQL 文であるものと想定しています。

```
CREATE PROCEDURE ml_qa_listener_executesql(IN @msgid VARCHAR(128))
begin
 DECLARE @execstr LONG VARCHAR;
 SET @execstr = ml_qa_gettextcontent( @msgid );
 EXECUTE IMMEDIATE @execstr;
end
```

ml_qa_putmessage

メッセージを送信します。

パラメーター

項目	説明	備考
1	メッセージ ID	VARCHAR(128)。メッセージ ID は、ml_qa_createmessage または ml_qa_getmessage から取得できます。
2	アドレス	VARCHAR(128)

備考

指定するメッセージ ID は、ml_qa_createmessage を使用して、すでに作成されていることが必要です。ml_qa_putmessage を呼び出す前にそのメッセージ ID に関連付けられていた内容、プロパティ、ヘッダーだけが、メッセージとともに送信されます。ml_qa_putmessage の呼び出し後に追加されたものは無視されます。

送信のためにメッセージが実際にキューに登録される前に、コミットを実行する必要があります。

参照

- 「SQL アプリケーションの設定」59 ページ
- 「ml_qa_createmessage」687 ページ
- 「ml_qa_getmessage」687 ページ

例

次の例では、「a simple message」という内容のメッセージを作成し、アドレス clientid¥queuename に送信します。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_settextcontent( @msgid, 'a simple message' );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   commit;
end
```

ml_qa_triggersendreceive

Mobile Link サーバーとのメッセージの同期をトリガーします。

備考

通常、メッセージ同期は QAnywhere Agent が処理します。ただし、同期ポリシーがオンデマンドの場合、メッセージ同期をトリガーするのはアプリケーション側の役割です。これを行うには、このストアドプロシージャーを使用します。メッセージ同期のトリガーは、現在のトランザクションがコミットされるまで適用されません。

参照

- 「SQL アプリケーションの設定」59 ページ

例

次の例では、メッセージの送信後すぐに、メッセージ転送が開始されます。

```
begin
   declare @msgid varchar(128);
   set @msgid = ml_qa_createmessage();
   call ml_qa_settextcontent( @msgid, 'my simple message' );
   call ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   call ml_qa_triggersendreceive();
   commit;
end
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

メッセージヘッダーとメッセージプロパティ

メッセージヘッダー

QAnywhere のすべてのメッセージで、同じヘッダーフィールドセットがサポートされます。ヘッダーフィールドには、メッセージを識別してルートを指定するために、クライアントとプロバイダーの両方で使用される値が設定されます。

次のメッセージヘッダーが事前に定義されています。メッセージヘッダーの使い方は、使用するクライアントアプリケーションの種類によって変わります。

- **メッセージ ID**　読み込み専用。新規メッセージのメッセージ ID。このヘッダーの値は、メッセージの送信後にのみ設定されます。 次の項を参照してください。

  - QAMessage.MessageID プロパティ [QAnywhere .NET]302 ページ
  - QAMessage.getMessageID メソッド [QAnywhere C++]464 ページ と QAMessage.setMessageID メソッド [QAnywhere C++]474 ページ
  - QAMessage.getMessageID メソッド [QAnywhere Java]590 ページ
  - SQL API：「ml_qa_createmessage」687 ページと「ml_qa_getmessage」687 ページ

- **メッセージの作成時刻のタイムスタンプ**　読み込み専用。メッセージの Timestamp ヘッダーフィールドには、メッセージが作成された時刻が格納されます。これは、協定世界時 (UTC: Coordinated Universal Time) です。この時刻は、メッセージが実際に送信された時刻ではないので注意してください。メッセージの実際の送信時刻は、トランザクションの進行状況やクライアント側のキューに登録されているその他のメッセージの影響で、作成時刻よりも遅れる可能性があります。このヘッダーは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。 次の項を参照してください。

  - QAMessage.Timestamp プロパティ [QAnywhere .NET]304 ページ
  - QAMessage.getTimestamp メソッド [QAnywhere C++]468 ページ と QAMessage.setTimestamp メソッド [QAnywhere C++]476 ページ
  - QAMessage.getTimestamp メソッド [QAnywhere Java]593 ページ
  - SQL API：「ml_qa_gettimestamp」661 ページ

- **返信先アドレス**　読み込み／書き込み。VARCHAR(128) 型の返信アドレス。返信アドレスが存在しない場合は NULL。このヘッダーは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。 次の項を参照してください。

  - QAMessage.ReplyToAddress プロパティ [QAnywhere .NET]303 ページ
  - QAMessage.getReplyToAddress メソッド [QAnywhere C++]466 ページ と QAMessage.setReplyToAddress メソッド [QAnywhere C++]475 ページ
  - QAMessage.getReplyToAddress メソッド [QAnywhere Java]592 ページ と QAMessage.setReplyToAddress メソッド [QAnywhere Java]598 ページ
  - SQL API：「ml_qa_getreplytoaddress」660 ページと「ml_qa_setreplytoaddress」664 ページ

- **メッセージアドレス**　読み込み専用。VARCHAR(128) 型の QAnywhere のメッセージアドレス。QAnywhere のメッセージアドレスは *id¥queue-name* の形式を取ります。このヘッダー

は、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。
次の項を参照してください。

○ QAMessage.Address プロパティ [QAnywhere .NET]301 ページ
○ QAMessage.getAddress メソッド [QAnywhere C++]459 ページ
○ QAMessage.getAddress メソッド [QAnywhere Java]585 ページ
○ SQL API：「ml_qa_getaddress」655 ページ

● **メッセージの再配信のステータス**　読み込み専用。再配信のステータスを示す BIT 型の値。1 はメッセージが再配信中であることを示し、0 は再配信中ではないことを示します。

受信されたが受信確認されていないメッセージは、再配信される場合があります。たとえば、メッセージは受信されたものの、メッセージを受信したアプリケーションがメッセージの内容の処理を完了する前にクラッシュした場合などです。このような場合には、メッセージは再配信とマーク付けされて、メッセージの処理が完了していない可能性があることを示す警告が受信側に送信されます。

たとえば、次のような手順を経てメッセージが受信されるとします。

1. 非トランザクション指向の QAnywhere Manager を使用するアプリケーションがメッセージを受信します。

2. このアプリケーションは、T1 というデータベーステーブルにメッセージの内容とメッセージ ID を書き込み、変更をコミットします。

3. アプリケーションがメッセージの受信を確認します。

手順 1 と 2、または手順 2 と 3 の間でアプリケーションに障害が発生した場合は、アプリケーションの再起動時にメッセージが再配信されます。

手順 1 と 2 の間で障害が発生した場合は、手順 2 と 3 を実行してメッセージを再配信してください。手順 2 と 3 の間で障害が発生した場合は、メッセージはすでに処理されているので、メッセージの確認のみ必要です。

アプリケーションの障害時に何が起きたかを特定するには、アプリケーションで ml_qa_getredelivered を呼び出して、メッセージがすでに再配信されていたかどうかをチェックします。テーブル T1 で検索する必要があるのは、再配信されるメッセージだけです。アプリケーションで障害が発生することはほとんどないので、この方法は、受信したメッセージのメッセージ ID にアプリケーションからアクセスして、メッセージがテーブル T1 に格納されているかどうかをチェックするよりも効率的です。

このヘッダーは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

次の項を参照してください。

- ○ QAMessage.Redelivered プロパティ [QAnywhere .NET]303 ページ
- ○ QAMessage.getRedelivered メソッド [QAnywhere C++]465 ページ と QAMessage.setRedelivered メソッド [QAnywhere C++]475 ページ
- ○ QAMessage.getRedelivered メソッド [QAnywhere Java]592 ページ
- ○ SQL API：「ml_qa_getredelivered」658 ページ

● **メッセージの有効期限**　読み取り専用。ただし SQL API の場合は読み取り／書き込み。TIMESTAMP 型の有効期限。有効期限がない場合は NULL が返されます。メッセージは、対象となる受信者によって指定の時間内に受信されないと期限切れになります。期限切れになったメッセージは、QAnywhere のデフォルトの削除ルールを使って削除できます。このヘッダーは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。 次の項を参照してください。

- ○ QAMessage.Expiration プロパティ [QAnywhere .NET]301 ページ
- ○ QAMessage.getExpiration メソッド [QAnywhere C++]461 ページ
- ○ QAMessage.getExpiration メソッド [QAnywhere Java]587 ページ
- ○ SQL API：「ml_qa_getexpiration」656 ページと「ml_qa_setexpiration」662 ページ

● **メッセージの優先度**　読み込み／書き込み。QAnywhere API では、10 レベルの優先度が定義されています。0 が最低の優先度、9 が最高の優先度を表します。クライアントは、優先度 0 ～ 4 を通常のメッセージ、優先度 5 ～ 9 を緊急度の高いメッセージとみなす必要があります。このヘッダーは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。 次の項を参照してください。

- ○ QAMessage.Priority プロパティ [QAnywhere .NET]303 ページ
- ○ QAMessage.getPriority メソッド [QAnywhere C++]464 ページ
- ○ QAMessage.getPriority メソッド [QAnywhere Java]590 ページ
- ○ SQL API：「ml_qa_getpriority」658 ページ

● **どのメッセージへの返信であるかを示すメッセージ ID**　読み込み／書き込み。VARCHAR(128) 型の in-reply-to ID。クライアントは、In-Reply-To ID ヘッダーフィールドを使用してメッセージ間リンクを設定できます。これは、応答メッセージをそれに対応する要求メッセージとリンクさせる場合によく使用されます。in-reply-to ID は、返信対象メッセージの ID です。このヘッダーは、メッセージの受信後からロールバックまたはコミットが行われるまでの間に読み込むことができます。ロールバックまたはコミットが行われた後で読み込むことはできません。 次の項を参照してください。

- ○ QAMessage.InReplyToID プロパティ [QAnywhere .NET]302 ページ
- ○ QAMessage.getInReplyToID メソッド [QAnywhere C++]462 ページ
- ○ QAMessage.getInReplyToID メソッド [QAnywhere Java]588 ページ
- ○ SQL API：「ml_qa_getinreplytoid」657 ページ

一部のメッセージヘッダーは転送ルールで使用できます。「ルールエンジンで定義される変数」780 ページを参照してください。

参照

- QAMessage インターフェイス [QAnywhere .NET]285 ページ
- QAMessage クラス [QAnywhere C++]454 ページ
- QAMessage インターフェイス [QAnywhere Java]582 ページ
- SQL API：「メッセージヘッダー」655 ページ

メッセージプロパティ

各メッセージには、アプリケーションで定義されたプロパティ値をサポートする組み込みの機能があります。これらのメッセージプロパティを使用すると、アプリケーションで定義されたメッセージフィルタリングを実行できます。

メッセージプロパティは名前と値の組み合わせです。オプションでメッセージに挿入して、構造を示すことができます。たとえば、.NET API では、定数 MessageProperties.ORIGINATOR で識別される、事前に定義されたメッセージプロパティ ias_Originator は、メッセージを送信したメッセージストアの ID を指定します。メッセージプロパティを転送ルールで使用して、メッセージを転送することが妥当であるかどうかを判別することができます。

メッセージプロパティには、次の 2 種類があります。

- **事前に定義されたメッセージプロパティ**　このメッセージプロパティには、必ず ias_ または IAS_ のプレフィクスが付けられています。
- **カスタムのメッセージプロパティ**　ユーザーが定義したメッセージプロパティです。このプロパティに ias_ または IAS_ のプレフィクスを付けることはできません。

事前定義またはカスタムのプロパティのどちらの場合も、get メソッドと set メソッドを使用してメッセージストアプロパティにアクセスし、プロパティの名前を最初のパラメーターとして渡します。

参照

- 「メッセージプロパティの管理」701 ページ

事前に定義されたメッセージプロパティ

利便性のためにいくつかのメッセージプロパティは事前に定義されています。事前に定義されたプロパティは読み込み可能ですが、設定はできません。事前に定義されたメッセージプロパティは次のとおりです。

- **ias_Adapters**　ネットワークステータス通知メッセージのプロパティ。Mobile Link サーバーへの接続で使用できるネットワークアダプターのリストです。これは文字列のリストで、各文字列はパイプ記号で区切られます。
- **ias_DeliveryCount**　整数。メッセージを配信するために、これまでに実行された試行回数を示します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- **ias_MessageType**　整数。メッセージのタイプを示します。次のいずれかのメッセージタイプを取ります。

値	メッセージタイプ	説明
0	REGULAR	メッセージに ias_MessageType プロパティが設定されていない場合は、標準メッセージになります。
13	PUSH_NOTIFICATION	Push 通知がサーバーから受信されると、PUSH_NOTIFICATION タイプのメッセージがシステムキューに送信されます。「Push 通知の通知」63 ページを参照してください。
14	NETWORK_STATUS_NOTIFICATION	ネットワークステータスが変化すると、このタイプのメッセージがシステムキューに送信されます。「ネットワークステータス通知」62 ページを参照してください。

- **ias_RASNames**　文字列。ネットワークステータス通知メッセージのプロパティ。Mobile Link サーバーへの接続で使用できる RAS エントリ名のリストです。各文字列はパイプ記号で区切られます。

- **ias_NetworkStatus**　整数。ネットワークステータス通知メッセージのプロパティ。ネットワーク接続のステータスです。値 1 は接続済み、0 はそれ以外を意味します。

- **ias_Originator**　文字列。メッセージの発信者のメッセージストア ID。

- **ias_Status**　整数。メッセージの現在のステータス。このプロパティは SQL API ではサポートされていません。次のいずれかの値を取ります。

ステータスコード	説明
1	保留 - メッセージは送信されましたが、まだ受信されていません。
10	受信中 - メッセージは受信中であるか、または受信されましたがまだ確認されていません。
20	最終 - メッセージは最終ステータスになりました。
30	失効 - 有効期限が切れる前にメッセージが受信されませんでした。

ステータスコード	説明
40	キャンセル済み - メッセージはキャンセルされました。
50	受信不可 - メッセージに間違った形式があるか、失敗した試行回数が多すぎて配信されませんでした。
60	受信済み - メッセージは受信され、確認されました。

ステータス値には定数があります。次の項を参照してください。

- StatusCodes 列挙体 [QAnywhere .NET]322 ページ
- StatusCodes クラス [QAnywhere C++]491 ページ
- StatusCodes インターフェイス [QAnywhere Java]616 ページ

- **ias_StatusTime**　メッセージが現在のステータスになった時刻。タイムスタンプには、プラットフォームに合わせたフォーマットが使用されます。これはローカル時間です。C++ API では、Windows と PocketPC プラットフォームのタイムスタンプは SYSTEMTIME フォーマットです。これが FILETIME フォーマットに変換されて、qa_long 値にコピーされます。このプロパティは SQL API ではサポートされていません。

API	このプロパティからの戻り値
.NET	DateTime
C++	string
Java	java.util.Date オブジェクト

メッセージプロパティの定数

.NET、C++、Java 用の QAnywhere API には、メッセージプロパティを指定するための定数があります。 次の項を参照してください。

- MessageProperties クラス [QAnywhere .NET]189 ページ
- MessageProperties クラス [QAnywhere C++]377 ページ
- MessageProperties インターフェイス [QAnywhere Java]497 ページ

カスタムのメッセージプロパティ

QAnywhere では、C++、Java、または .NET の API を使用してメッセージプロパティを定義できます。カスタムのメッセージプロパティでは、名前と値を組み合わせてオブジェクトに関連付けることができます。次に例を示します。

```
msg.SetStringProperty("Product", "widget");
msg.SetFloatProperty("Price",1.00);
msg.SetIntProperty("Quantity",10);
```


Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

メッセージプロパティ名では大文字と小文字を区別しません。文字、数字、アンダースコアを使用できますが、先頭は文字でなければなりません。次の名前は予約語なので、メッセージプロパティ名として使用することはできません。

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

メッセージプロパティの管理

メッセージプロパティの管理には、次の QAMessage メソッドを使用できます。

注意
カスタムプロパティは取得と設定はできますが、事前に定義されたプロパティは取得するだけで設定はできません。

メッセージプロパティ管理用メソッド (.NET)

- Object GetProperty(String name)
- void SetProperty(String name, Object value)
- boolean GetBooleanProperty(String name)
- void SetBooleanProperty(String name, boolean value)
- byte GetByteProperty(String name)
- void SetByteProperty(String name, byte value)
- short GetShortProperty(String name)
- void SetShortProperty(String name, short value)
- int GetIntProperty(String name)
- void SetIntProperty(String name, int value)
- long GetLongProperty(String name)
- void SetLongProperty(String name, long value)
- float GetFloatProperty(String name)
- void SetFloatProperty(String name, float value)
- double GetDoubleProperty(String name)
- void SetDoubleProperty(String name, double value)
- String GetStringProperty(String name)
- void SetStringProperty(String name, String value)
- IEnumerator GetPropertyNames()
- void ClearProperties()
- PropertyType GetPropertyType(string propName)
- bool PropertyExists(string propName)

QAMessage インターフェイス [QAnywhere .NET]285 ページを参照してください。

メッセージプロパティ管理用メソッド (C++)

- qa_bool getBooleanProperty(qa_const_string name, qa_bool * value)
- qa_bool setBooleanProperty(qa_const_string name, qa_bool value)
- qa_bool getByteProperty(qa_const_string name, qa_byte * value)
- qa_bool setByteProperty(qa_const_string name, qa_byte value)
- qa_bool getShortProperty(qa_const_string name, qa_short * value)
- qa_bool setShortProperty(qa_const_string name, qa_short value)
- qa_bool getIntProperty(qa_const_string name, qa_int * value)
- qa_bool setIntProperty(qa_const_string name, qa_int value)
- qa_bool getLongProperty(qa_const_string name, qa_long * value)
- qa_bool setLongProperty(qa_const_string name, qa_long value)
- qa_bool getFloatProperty(qa_const_string name, qa_float * value)
- qa_bool setFloatProperty(qa_const_string name, qa_float value)
- qa_bool getDoubleProperty(qa_const_string name, qa_double * value)
- qa_bool setDoubleProperty(qa_const_string name, qa_double value)
- qa_int getStringProperty(qa_const_string name, qa_string value, qa_int len)
- qa_bool setStringProperty(qa_const_string name, qa_const_string value)
- void QAMessage::clearProperties()
- qa_short QAMessage::getPropertyType(qa_const_string name)
- qa_bool QAMessage::propertyExists(qa_const_string name)

QAMessage クラス [QAnywhere C++]454 ページを参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

メッセージプロパティ管理用メソッド (Java)

- void clearProperties()
- boolean getBooleanProperty(String name)
- void setBooleanProperty(String name, boolean value)
- byte getByteProperty(String name)
- void setByteProperty(String name, byte value)
- double getDoubleProperty(String name)
- void setDoubleProperty(String name, double value)
- java.util.Date getExpiration() void setFloatProperty(String name, float value)
- float getFloatProperty(String name)
- int getIntProperty(String name)
- void setIntProperty(String name, int value)
- long getLongProperty(String name)
- void setLongProperty(String name, long value)
- Object getProperty(String name)
- void setProperty(String name, Object value)
- java.util.Enumeration getPropertyNames()
- short getPropertyType(String name)
- short getShortProperty(String name)
- void setShortProperty(String name, short value)
- String getStringProperty(String name)
- void setStringProperty(String name, String value)
- boolean propertyExists(String name)

QAMessage インターフェイス [QAnywhere Java]582 ページを参照してください。

メッセージプロパティ管理用 SQL ストアドプロシージャー

- ml_qa_getbooleanproperty
- ml_qa_getbyteproperty
- ml_qa_getdoubleproperty
- ml_qa_getfloatproperty
- ml_qa_getintproperty
- ml_qa_getlongproperty
- ml_qa_getpropertynames
- ml_qa_getshortproperty
- ml_qa_getstringproperty
- ml_qa_setbooleanproperty
- ml_qa_setbyteproperty
- ml_qa_setdoubleproperty
- ml_qa_setfloatproperty
- ml_qa_setfloatproperty
- ml_qa_setintproperty
- ml_qa_setlongproperty
- ml_qa_setshortproperty
- ml_qa_setstringproperty

「メッセージプロパティ」665 ページを参照してください。

例

```
// C++ example.
QAManagerFactory factory;
QAManager * mgr = factory->createQAManager();
mgr->open(AcknowledgementMode::EXPLICIT_ACKNOWLEDGEMENT);
QAMessage * msg = mgr->createTextMessage();
msg->setStringProperty( "tm_Subject", "Some message subject." );
mgr->putMessage( "myqueue", msg );

// C# example.
QAManager mgr = QAManagerFactory.Instance.CreateQAManager();
mgr.Open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
QAMessage msg = mgr.CreateTextMessage();
msg.SetStringProperty( "tm_Subject", "Some message subject." );
mgr.PutMessage( "myqueue", msg );

// Java example
QAManager mgr = QAManagerFactory.getInstance().createQAManager();
mgr.open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
QAMessage msg = mgr.createTextMessage();
msg.setStringProperty("tm_Subject", "Some message subject.");
mgr.putMessage("myqueue", msg);

-- SQL example
begin
   DECLARE @msgid VARCHAR(128);
   SET @msgid = ml_qa_createmessage();
   CALL ml_qa_setfloatproperty( @msgid, 'myfloatproperty1', -1.3e-5 );
   CALL ml_qa_setfloatproperty( @msgid, 'myfloatproperty2', 1.3e5 );
   CALL ml_qa_putmessage( @msgid, 'clientid¥queuename' );
   COMMIT;
end
```

サーバー管理要求のリファレンス

サーバー管理要求の親タグ

Condition タグ

MessageDetailsRequest に含めるメッセージをフィルタリングするには、次の <condition> タグのサブタグを使用します。次のサブタグは、<condition> タグ内でいくつでも指定できます。同じタグを複数使用した場合、指定された値は論理 OR 結合されます。これに対して、異なる 2 つのタグを使用した場合は、値は論理 AND 結合されます。

<condition> タグのサブタグ	**説明**
<address>	指定されたアドレスに送信されるメッセージを選択します。
<archived>	アーカイブメッセージストアにあるメッセージの詳細情報を返します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

<condition> タグのサブタグ	説明
<customRule>	ルールに基づいてメッセージを選択します。
<kind>	バイナリまたはテキストのメッセージをフィルタリングします。たとえば、<kind>*text*</kind> と指定するとテキストメッセージがフィルタリングされ、<kind>*binary*</kind> と指定するとバイナリメッセージがフィルタリングされます。
<messageId>	特定のメッセージ ID を持つメッセージを選択します。
<originator>	指定されたクライアントから発信されるメッセージを選択します。
<priority>	現在優先度が指定されているメッセージを選択します。
<property>	指定されたメッセージプロパティを持つメッセージを選択します。プロパティ名と値を確認するには、<property>*property-name=property-value*</property> 構文を使用します。プロパティが存在するかどうかを確認するには、<property>*property-name*</property> フォーマットを使用します。
<status>	現在ステータスが指定されているメッセージを選択します。

CustomRule タグ

より複雑な条件ステートメントを作成するには、<condition> タグ (およびその他のタグ) のサブタグとして <customRule> タグを使用します。このタグではデータをサーバールールとして使用します。このルールは、サーバー転送ルールで使用されるサーバールールに似ています。このクエリは、転送ルールの条件部分と同じ方法で作成できます。

例

次の条件では、優先度が 4 に設定され、発信者の名前が'%sender%' と一致し、ステータスが 20 以上であるという探索条件を満たすメッセージを選択します。

```
<condition>
 <priority>4</priority>
 <customRule>ias_Originator LIKE '%sender%' AND ias_Status &gt;= 20</customRule>
</condition>
```

参照

- 「条件構文」776 ページ

Schedule タグ

オプションで、スケジュールに従ってサーバー管理要求を実行するように設定できます。サーバー管理要求を実行するスケジュールを定義するには、次の <schedule> タグのサブタグを使用します。

<schedule> タグのサブタグ	**説明**
<starttime>	サーバーがレポート生成を開始する時刻を定義します。次に例を示します。 `<starttime>09:00:00</starttime>`

<between>	<starttime> と <endtime> の 2 つのサブタグを持ち、サーバーがレポートを生成する間隔を定義します。<starttime> と同じ <schedule> タグ内で使用することはできません。次に例を示します。
	
	`<between>`
	`  <starttime>Mon Jan 16 09:00:00 EST 2006</starttime>`
	`  <endtime>Mon Jan 17 09:00:00 EST 2006</endtime>`
	`</between>`

<everyhour>	後続のレポートとの間隔を時間単位で定義します。<everyminute> または <everysecond> と同じ <schedule> タグ内で使用することはできません。たとえば、次の要求では、午前 9 時から 2 時間ごとにレポートが生成されます。
	
	`<schedule>`
	`  <starttime>09:00:00</starttime>`
	`  <everyhour>2</everyhour>`
	`</schedule>`

<everyminute>	後続のレポートとの間隔を分単位で定義します。<everyhour> または <everysecond> と同じ <schedule> タグ内で使用することはできません。
	
	`<schedule>`
	`  <everyminute>10</everyminute>`
	`</schedule>`

<everysecond>	後続のレポートとの間隔を秒単位で定義します。<everyhour> または <everyminute> と同じ <schedule> タグ内で使用することはできません。
	
	`<schedule>`
	`  <everysecond>45</everysecond>`
	`</schedule>`

<ondayofweek>	タグごとに、スケジュールを実行する曜日を 1 つ指定します。たとえば、次のスケジュールは月曜日と火曜日に実行します。
	
	`<schedule>`
	`  <ondayofweek>Monday</ondayofweek>`
	`  <ondayofweek>Tuesday</ondayofweek>`
	`</schedule>`

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

<schedule> タグのサブタグ	説明
<ondayofmonth>	タグごとに、スケジュールを実行する日を 1 つ指定します。たとえば、次のスケジュールは 15 日に実行します。 <schedule> <ondayofmonth>15</ondayofmonth> </schedule>
<startdate>	スケジュールを実行する日付。次に例を示します。 <startdate>Mon Jan 16 2006</startdate>

スケジュールを変更するには、同じ要求 ID を指定して新しいサーバー管理要求を登録します。スケジュールを削除するには、同じ要求 ID を指定してサーバー管理要求を登録し、<schedule>none</schedule> というタグが含まれるようにします。

注意

- <ondayofweek> タグと <ondayofmonth> タグを除き、各タグは <schedule> タグ内で一度だけしか使用できません。
- <between> タグと個々の <starttime> タグは、同じ <schedule> タグ内で使用できません。
- 同じ <schedule> タグ内で使用できるのは、<everysecond>、<everyminute>、<everyhour> のいずれか 1 つだけです。

例

次の例では、各メッセージの ID とステータスも含め、サーバー上のすべてのメッセージについてレポートを作成する永続的なスケジュールを作成します。また、要求 ID dailyMessageStatus に割り当てられている永続的な要求の上書きも行います。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <MessageDetailsRequest>
   <request>
     <replyAddr>myclient¥messageStatusQueue</replyAddr>
     <requestId>dailyMessageStatus</requestId>
     <schedule>
       <everyhour>24</everyhour>
     </schedule>
     <persistent/>
     <messageId/>
     <status/>
   </request>
  </MessageDetailsRequest>
</actions>
```

このレポートは次のようになります。レポートはアドレス myclient¥messageStatusQueue に送信されます。サーバー上には、ステータスが 60 (受信済み) と 1 (保留中) の 2 つのメッセージがあることがわかります。

```
<?xml version="1.0" encoding="UTF-8"?>
<MessageDetailsReport>
```

```
    <requestId>dailyMessageStatus</requestId>
    <UTCDatetime>Mon Jan 16 15:03:04 EST 2007</UTCDatetime>
    <statusDescription>Scheduled report</statusDescription>
    <messageCount>2</messageCount>
    <message>
      <messageId>ID:26080b8927f83f9722357eab0a0628eb</messageId>
      <status>60</status>
    </message>
    <message>
      <messageId>ID:fe857fa8-a7d7-4266-985b-a1818a85d1a2</messageId>
      <status>1</status>
    </message>
  </MessageDetailsReport>
```

MessageDetailsReport タグ

メッセージの詳細レポートは、<MessageDetailsReport> タグを含む XML メッセージです。このレポートは、レポートヘッダーとそれに続くオプションの <message> タグから構成されます。各レポートのヘッダーは、次のタグから構成されます。

<MessageDetailsReport> タグのサブタグ	**説明**
<message>	レポートの本文は <message> タグのリストから構成されます。<message> タグのサブタグは、選択条件を満たすメッセージごとに特定の詳細情報を表示します。メッセージが選択されていない場合や、元の要求で詳細要素が指定されていない場合は、<message> タグはレポートに含まれません。それ以外の場合は、メッセージごとに独自の <message> タグがあります。
<messageCount>	要求の選択条件を満たすメッセージの数。
<requestId>	レポートを生成した要求の ID。
<statusDescription>	レポートが生成された理由の簡単な説明。
<UTCDateline>	レポートが生成された時刻と日付。

Message タグ

<message> タグのサブタグ	**説明**
<address>	メッセージのアドレス。myclient¥myqueue などです。
<contentSize>	メッセージのサイズ。テキストメッセージの場合は文字数、バイナリメッセージの場合はバイト数です。
<expires>	メッセージが配信されない場合に期限切れとなる日付と時刻。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

<message> タグのサブタグ	説明
<kind>	メッセージがバイナリ (1) であるか、テキスト (2) であるかを示します。
<messageId>	新規メッセージのメッセージ ID。
<originator>	メッセージの発信者のメッセージストア ID。
<priority>	0 ～ 9 の整数で指定したメッセージの優先度。0 は優先度が最も低く、9 は最も高いことを表します。
<property>	メッセージのプロパティ。
<status>	メッセージの現在のステータス。
<statusTime>	メッセージが現在のステータスになった時刻。これは現地時刻です。
<transmissionStatus>	メッセージの同期ステータス。次のいずれかの値を取ります。 ● 0 - メッセージは、目的の受信側メッセージストアに転送されていません。 ● 1 - メッセージは、目的の受信側メッセージストアに転送済みです。 ● 2 - 受信先および送信元のメッセージストアが同じであるため、転送は不要です。 ● 3 - メッセージは目的の受信先に転送されましたが、まだ受信確認されていません。メッセージ転送が中断された可能性があるため、QAnywhere がメッセージを再転送する可能性があります。

例

次に、メッセージの詳細レポートの例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<MessageDetailsReport>
 <requestId>testReport</requestId>
 <UTCDatetime>Mon Jan 16 15:03:04 EST 2006</UTCDatetime>
 <statusDescription>Scheduled report</statusDescription>
 <messageCount>1</messageCount>
 <message>
  <messageId>ID:26080b8927f83f9722357eab0a0628eb</messageId>
  <status>60</status>
  <property>
   <name>myPropName</name>
   <value>myPropVal</value>
  </property>
 </message>
</MessageDetailsReport>
```

次の <condition> タグは、(msgId=ID:144... OR msgId=ID225...) AND (status=pending) AND (kind=textmessage) AND (値が 'myVal' のプロパティ 'myProp' を含む) という探索条件を満たすメッセージを選択します。

```
<condition>
 <messageId>ID:144d7e44dc2d7e1d</messageId>
 <messageId>ID:22578sd5dsd99s8e</messageId>
 <status>1</status>
 <kind>text</kind>
 <property>myProp=myVal</property>
</condition>
```

ワンタイム要求は、<schedule> タグが指定されていない要求です。ワンタイム要求は、1 つのレポートを生成する場合に使用され、レポートが送信されると削除されます。次の要求では、現在サーバー上に置かれている優先度 9 のすべてのメッセージについて、メッセージ ID、ステータス、送信先アドレスを表示するレポートを 1 つ生成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <MessageDetailsRequest>
  <request>
   <requestId>testRequest</client>
   <condition>
    <priority>9</priority>
   </condition>
   <messageId/>
   <status/>
   <address/>
  </request>
 </MessageDetailsRequest>
</actions>
```

次のメッセージの詳細要求サンプルでは、メッセージ ID とメッセージステータスを含むレポートを生成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <MessageDetailsRequest>
    <!-- ... -->
    <messageId />
    <status />
  </MessageDetailsRequest>
</actions>
```

参照

- 「メッセージヘッダー」695 ページ
- 「メッセージプロパティ」698 ページ
- 「事前に定義されたメッセージプロパティ」698 ページ

サーバー管理要求 DTD

次に、サーバー管理要求の XML 文書の定義をすべて示します。この DTD は、この項で説明するサーバー管理要求タグの概要として示されています。

```
<!DOCTYPE actions [
<!ELEMENT actions (ActionsResponseId?,(CloseConnector|OpenConnector|RestartRules|SetProperty
```

```
   |ClientStatusRequest|MessageDetailsRequest|CancelMessageRequest|PauseConnector|
ResumeConnector
   |CancelMessages|GetClientList)+)>
<!ELEMENT ActionsResponseId (#PCDATA)>
<!ELEMENT GetClientList EMPTY>
<!ELEMENT CloseConnector (client+)>
<!ELEMENT PauseConnector (client+)>
<!ELEMENT ResumeConnector (client+)>
<!ELEMENT OpenConnector (client+)>
<!ELEMENT RestartRules (client+)>
<!ELEMENT CancelMessages (client+)>
<!ELEMENT SetProperty (prop+)>
<!ELEMENT prop (client,name,value?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT value (#PCDATA)>
<!ELEMENT GetProperties (client+)>
<!ELEMENT ClientStatusRequest (requestId?,replyAddr?,client+,(schedule+,persistent?)?,onEvent*)>
<!ELEMENT onEvent (#PCDATA)>
<!ELEMENT requestId (#PCDATA)>
<!ELEMENT client (#PCDATA)>
<!ELEMENT replyAddr (#PCDATA)>
<!ELEMENT persistent EMPTY>
<!ELEMENT MessageDetailsRequest (requestId?,replyAddr?,(schedule+,persistent?)?,condition?,
   getAddress?,getArchived?,getContentSize?,getExpires?,getKind?,getMessageId?,getOriginator?,
   getPriority?,getProperties?,getStatus?,getStatusTime?,getTransmissionStatus?)>
<!ELEMENT getAddress EMPTY>
<!ELEMENT getArchived EMPTY>
<!ELEMENT getContentSize EMPTY>
<!ELEMENT getExpires EMPTY>
<!ELEMENT getKind EMPTY>
<!ELEMENT getMessageId EMPTY>
<!ELEMENT getOriginator EMPTY>
<!ELEMENT getPriority EMPTY>
<!ELEMENT getProperties EMPTY>
<!ELEMENT getStatus EMPTY>
<!ELEMENT getStatusTime EMPTY>
<!ELEMENT getTransmissionStatus EMPTY>
<!ELEMENT CancelMessageRequest (requestId?,replyAddr?,report?,(schedule
+,persistent?)?,condition?,
   getAddress?,getArchived?,getContentSize?,getExpires?,getKind?,getMessageId?,getOriginator?,
   getPriority?,getProperties?,getStatus?,getStatusTime?,getTransmissionStatus?)>
<!ELEMENT report EMPTY>
<!ELEMENT schedule (((startTime|between)?,everyHour?,everyMinute?,everySecond?,onDayOfWeek*,
   onDayOfMonth*)?)>
<!ELEMENT between (startTime,endTime)>
<!ELEMENT startTime (#PCDATA)>
<!ELEMENT endTime (#PCDATA)>
<!ELEMENT everyHour (#PCDATA)>
<!ELEMENT everyMinute (#PCDATA)>
<!ELEMENT everySecond (#PCDATA)>
<!ELEMENT onDayOfWeek (#PCDATA)>
<!ELEMENT onDayOfMonth (#PCDATA)>
<!ELEMENT condition ((messageId|status|priority|address|originator|kind|archived|
   customRule|property)+)>
<!ELEMENT archived EMPTY>
<!ELEMENT messageId (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT priority (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT originator (#PCDATA)>
<!ELEMENT kind (#PCDATA)>
<!ELEMENT customRule (#PCDATA)>
<!ELEMENT property (#PCDATA)>
]>
```

QAnywhere Agent ユーティリティのリファレンス

qaagent ユーティリティ

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を送受信します。このユーティリティは、クライアントメッセージストアが SQL Anywhere データベースである場合のみ使用してください。

dbmlsync など、メッセージストアの同期に使用される外部ユーティリティは、QAnywhere ではサポートされていません。

構文

qaagent [*option* ...]

オプション	説明
@data	指定された環境変数または設定ファイルからオプションを読み込みます。「@data qaagent オプション」715 ページを参照してください。
-c *connection-string*	クライアントメッセージストアに接続するための接続文字列を指定します。 「-c qaagent オプション」715 ページを参照してください。
-cd *seconds*	データベースへの再試行を実行する間の遅延時間を指定します。 「-cd qaagent オプション」717 ページを参照してください。
-cr *number-of-retries*	接続がエラーになってから、データベースへの接続を再試行する回数を指定します。 「-cr qaagent オプション」717 ページを参照してください。
-fd *seconds*	プライマリ Mobile Link サーバーへの再試行を実行する間の遅延時間を指定します。 「-fd qaagent オプション」717 ページを参照してください。
-fr *number-of-retries*	接続がエラーになってから、プライマリ Mobile Link サーバーへの接続を再試行する回数を指定します。 「-fr qaagent オプション」718 ページを参照してください。
-id *id*	QAnywhere Agent が接続するクライアントメッセージストアの ID を指定します。 「-id qaagent オプション」719 ページを参照してください。
-idl *download-size*	メッセージ転送中に使用するダウンロードの最大サイズを指定します。 「-idl qaagent オプション」720 ページを参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

オプション	説明
-iu *upload-size*	メッセージ転送中に使用するアップロードの最大サイズを指定します。 「-iu qaagent オプション」720 ページを参照してください。
-lp *number*	Mobile Link Listener が Mobile Link サーバーからの通知を受信するポートを指定します。デフォルトは 5001 です。 「-lp qaagent オプション」721 ページを参照してください。
-mn *password*	Mobile Link ユーザーの新しいパスワードを指定します。 「-mn qaagent オプション」722 ページを参照してください。
-mp *password*	Mobile Link ユーザーのパスワードを指定します。「-mp qaagent オプション」722 ページを参照してください。
-mu *username*	Mobile Link ユーザーを指定します。「-mu qaagent オプション」723 ページを参照してください。
-o *logfile*	出力メッセージを保存するファイルを指定します。「-o qaagent オプション」723 ページを参照してください。
-on *size*	QAnywhere Agent のメッセージログファイルの最大サイズを指定します。ログファイルがこのサイズに達すると、現在のファイルが拡張子 .old の付いた名前に変更され、新しいファイルが作成されます。「-on qaagent オプション」724 ページを参照してください。
-os *size*	QAnywhere Agent のメッセージログファイルの最大サイズを指定します。ログファイルがこのサイズに達すると、新しい名前を持つ新しいログファイルが作成されて使用されます。 「-os qaagent オプション」725 ページを参照してください。
-ot *logfile*	出力メッセージを記録するファイルを指定します。 「-ot qaagent オプション」725 ページを参照してください。
-pc{+|-}	メッセージ転送用に永続的な接続を有効にします。「-pc qaagent オプション」726 ページを参照してください。
-policy *policy-type*	QAnywhere Agent によって使用される転送ポリシーを指定します。「-policy qaagent オプション」727 ページを参照してください。
-push *mode*	Push 通知を有効または無効にします。デフォルトは有効です。「-push qaagent オプション」729 ページを参照してください。

オプション	説明
-q	QAnywhere Agent をクワイエットモードで起動し、ウィンドウをシステムトレイ内に最小化します。「-q qaagent オプション」730 ページを参照してください。
-qi	QAnywhere Agent をクワイエットモードで起動し、ウィンドウを完全に非表示にします。 「-qi qaagent オプション」731 ページを参照してください。
-si	データベースを、クライアントメッセージストアとして使用できるように初期化します。 「-si qaagent オプション」731 ページを参照してください。
-su	クライアントメッセージストアを現在のバージョンにアップグレードします。dbunload/reload は実行しません。 「-su qaagent オプション」733 ページを参照してください。
-sur	クライアントメッセージストアを現在のバージョンにアップグレードします。クライアントメッセージストアの dbunload/reload を実行します。 「-sur qaagent オプション」734 ページを参照してください。
-sv	SQL Anywhere ネットワークデータベースサーバーをデータベースサーバーとして使用します。「-sv qaagent オプション」735 ページを参照してください。
-v [*levels*]	ログの冗長レベルを指定します。「-v qaagent オプション」735 ページを参照してください。
-wc *name*	QAnywhere Agent または QAnywhere Ultra Light Agent を起動するウィンドウクラス名を指定します。「qastop ユーティリティ」760 ページを参照してください。
-x { **http**|**tcpip**|**tls**|**https** } [(*keyword*=*value*;...)]	Mobile Link サーバーとの通信に使用するプロトコルのオプションを指定します。 「-x qaagent オプション」736 ページを参照してください。
-xd	QAnywhere Agent で Mobile Link サーバーの動的アドレス指定を使用することを指定します。「-xd qaagent オプション」737 ページを参照してください。

参照

- 「QAnywhere Agent の起動」47 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

@data qaagent オプション

指定された環境変数または設定ファイルからオプションを読み込みます。

構文

qaagent @{ *filename* | *environment-variable* } ...

備考

このオプションを指定すると、環境変数または設定ファイル内にコマンドラインオプションを記述できます。指定された名前の環境変数と指定された名前の設定ファイルの両方が存在する場合、環境変数が使用されます。

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

Windows Mobile ではショートカットのコマンドラインが 256 文字に制限されているため、このオプションが役に立ちます。

Sybase Central での同等機能

Sybase Central の QAnywhere プラグインには、**[エージェント設定ファイルの新規作成]** (1 つは SQL Anywhere 用でもう 1 つは Ultra Light 用) というタスクがあります。これらのタスクのいずれかを選択すると、ファイル名を入力するように要求されます。次に **[プロパティ]** ウィンドウが表示され、コマンド情報を入力できます。生成されるファイルの拡張子は *.qaa* です。ファイル拡張子 *.qaa* は、Sybase Central の規則に従っています。つまりこのファイルは、@data オプションで作成するファイルと同じです。Sybase Central で作成されたコマンドファイルは、@data の設定ファイルとして使用できます。

参照

- 「設定ファイル」『SQL Anywhere サーバー データベース管理』
- 「ファイル非表示ユーティリティ (dbfhide)」『SQL Anywhere サーバー データベース管理』

-c qaagent オプション

クライアントメッセージストアに接続するための文字列を指定します。

構文

qaagent -c *connection-string* ...

デフォルト

接続パラメーター	デフォルト値
uid	ml_qa_user
pwd	qanywhere

備考

接続文字列では、接続パラメーターを *keyword=value* の形式で指定します。パラメーターとパラメーター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。

ODBC データソースは、一般的にはクライアントデバイスでは使用されません。ODBC は qaagent で使用されません。

以下に、よく使用する接続パラメーターの一部を示します。

- **dbf=*filename*** 指定されたファイル名を持つメッセージストアに接続します。「DatabaseFile (DBF) 接続パラメーター」『SQL Anywhere サーバー データベース管理』を参照してください。

- **dbn=*database-name*** データベースファイルではなくデータベース名を指定して、すでに実行されているクライアントメッセージストアに接続します。「DatabaseName (DBN) 接続パラメーター」『SQL Anywhere サーバー データベース管理』を参照してください。

- **server=*server-name*** すでに実行されているデータベースサーバーの名前を指定します。デフォルト値はデータベース名です。「ServerName (Server) 接続パラメーター」『SQL Anywhere サーバー データベース管理』を参照してください。

- **uid=*user*** クライアントメッセージストアに接続するためのデータベースユーザー ID を指定します。このパラメーターは、デフォルトの UID または PWD 接続パラメーターを変更する場合に必要となります。 「Userid (UID) 接続パラメーター」『SQL Anywhere サーバー データベース管理』を参照してください。

- **pwd=*password*** データベースユーザー ID に対応するパスワードを指定します。このパラメーターは、デフォルトの UID または PWD 接続パラメーターを変更する場合に必要となります。 「Password (PWD) 接続パラメーター」『SQL Anywhere サーバー データベース管理』を参照してください。

- **dbkey=*key*** データベースにアクセスするための暗号化キーを指定します。 「DatabaseKey (DBKEY) 接続パラメーター」『SQL Anywhere サーバー データベース管理』を参照してください。

- **start=*startline*** データベースサーバーの開始行を指定します。開始行を指定しない場合、Windows Mobile の場合のデフォルトは start=dbsrv12 -m -gn 5、他の Windows プラットフォームの場合のデフォルトは start=dbsrv12 -m です。-m オプションを指定すると、トランザクションログの内容がチェックポイントで削除されます。このオプションを指定することをおすすめします。 次の項を参照してください。

  - 「StartLine (START) 接続パラメーター」『SQL Anywhere サーバー データベース管理』
  - 「-m dbeng12/dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』
  - 「-gn dbsrv12 サーバーオプション」『SQL Anywhere サーバー データベース管理』

参照

- 「接続パラメーター」『SQL Anywhere サーバー データベース管理』
- 「SQL Anywhere データベース接続」『SQL Anywhere サーバー データベース管理』

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

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

-cd qaagent オプション

このオプションが -cr オプションと一緒に指定されている場合は、データベースへの接続を試行する間隔を指定します。

構文

qaagent -cd *seconds* ...

備考

再試行間のデフォルトの遅延時間は 10 秒です。

再試行に失敗すると、QAnywhere Agent はエラーを表示し、停止を待機します。

操作中にデータベース接続に失敗すると、接続の完了およびスレッドと外部プロセスの終了を含む終了手順が完了してから、QAnywhere Agent が再起動します。

このオプションは、qaagent -cr オプションと一緒に使用する必要があります。-cr オプションはデータベースへの接続を再試行する回数を指定し、-cd オプションは再試行の間隔を指定します。

-cr qaagent オプション

QAnywhere Agent がデータベースへの接続を再試行する回数を指定します。

構文

qaagent -cr *number-of-retries* ...

備考

デフォルトの再試行回数は 3 回で、再試行間の遅延時間は 10 秒です。

-fd qaagent オプション

このオプションが -fr オプションと一緒に指定されている場合は、Mobile Link サーバーへの接続を試行する間隔を指定します。

構文

qaagent -fd *seconds* ...

デフォルト

- -fr だけを指定し -fd を指定しない場合は、遅延は 0 です (間隔をあけずに、すぐに再試行が行われます)。
- -fr を指定しないと、デフォルトで再試行が行われません。

備考

このオプションは、qaagent -fr オプションと一緒に使用する必要があります。-fr オプションはプライマリサーバーへの接続を再試行する回数を指定し、-fd オプションは再試行の間隔を指定します。

通常このオプションは、-x オプションを使用してフェールオーバー Mobile Link サーバーを指定する場合に使用します。フェールオーバー Mobile Link サーバーが設定されている場合、デフォルトでは、プライマリサーバーへの接続で障害が発生するとすぐに、QAnywhere Agent は代替サーバーへの接続を試行します。-fr オプションを指定すると、QAnywhere Agent は代替サーバーに接続する前に、プライマリサーバーへの接続を再試行します。-fd オプションを併せて使用することで、プライマリサーバーへの接続を試行する間隔も指定できます。

-fd オプションは 10 秒以下に設定することをおすすめします。

このオプションは、qaagent -xd オプションと一緒に使用することはできません。

参照

- 「-fr qaagent オプション」718 ページ
- 「-x qaagent オプション」736 ページ
- 「フェールオーバーメカニズムの設定」38 ページ

-fr qaagent オプション

QAnywhere Agent がプライマリ Mobile Link サーバーへの接続を再試行する回数を指定します。

構文

qaagent -fr *number-of-retries* ...

デフォルト

0 - QAnywhere Agent がプライマリ Mobile Link サーバーへの接続を再試行しないことを意味します。

備考

デフォルトでは、QAnywhere Agent が Mobile Link サーバーに接続できない場合、エラーは発生せずメッセージも送信されません。このオプションは、QAnywhere Agent が Mobile Link サーバーへの接続を再試行することを指定します。さらに、QAnywhere Agent が代替サーバーへの接続を試行するまで、またはエラーを発行するまで (代替サーバーが指定されていない場合) に、Mobile Link サーバーへの接続を再試行する回数も指定します。

通常このオプションは、-x オプションを使用してフェールオーバー Mobile Link サーバーを指定する場合に使用します。フェールオーバー Mobile Link サーバーが設定されている場合、デフォルトでは、プライマリサーバーへの接続で障害が発生するとすぐに、QAnywhere Agent は代替サーバーへの接続を試行します。このオプションを指定すると、QAnywhere Agent は代替サーバーに接続する前に、プライマリサーバーへの接続を再試行します。

-fd オプションを併せて使用することで、プライマリサーバーへの接続再試行の間隔も指定できます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

このオプションは、qaagent -xd オプションと一緒に使用することはできません。

参照

- 「-fd qaagent オプション」717 ページ
- 「-x qaagent オプション」736 ページ
- 「フェールオーバーメカニズムの設定」38 ページ

-id qaagent オプション

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

構文

qaagent -id *id* ...

デフォルト

ID のデフォルト値は、Agent が動作しているデバイスの名前です。デバイス名がユニークでない場合は、-id オプションを使用します。

備考

各クライアントメッセージストアは、メッセージストア ID と呼ばれるユニークな文字列で識別されます。メッセージストアに最初に接続するときに ID を設定しなかった場合、ID はデフォルトでデバイス名になります。次回以降の接続時には、設定済みのメッセージストア ID を -id オプションとともに指定する必要があります。

メッセージストア ID は Mobile Link リモート ID と対応しています。すべての Mobile Link アプリケーションではリモートデータベースごとにユニークな ID が必要であるため、メッセージストア ID が要求されます。

デバイス上で 2 つ目の qaagent インスタンスを起動する場合は、-id オプションを使用してユニークなメッセージストア ID を指定する必要があります。

次の文字は ID の構成文字として使用できません。

- " (二重引用符)
- 制御文字
- ¥¥ (二重円記号)

この他にも次の制約が適用されます。

- ID は 120 文字以内である必要がある。
- 単一の円記号 (¥) を使用できるのは、エスケープ文字として使用する場合に限られる。
- クライアントメッセージストアデータベースで quoted_identifier オプションが Off に設定されている場合は、英数字、アンダースコア (_)、アットマーク (@)、シャープ (#)、ドル記号 ($) のみを ID に含めることができる。

参照

- 「Mobile Link ユーザー」『Mobile Link クライアント管理』
- 「Mobile Link ユーザーの作成と登録」『Mobile Link クライアント管理』
- 「クライアントメッセージストアの設定」25 ページ

-idl qaagent オプション

インクリメンタルダウンロードのサイズを指定します。

構文

qaagent -idl *download-size* [**K** | **M**] ...

デフォルト

-1 - 最大ダウンロードサイズが指定されていないことを意味します。

備考

このオプションは、メッセージ転送のダウンロード部分のサイズをバイト単位で指定します。キロバイトまたはメガバイトの単位を指定するには、それぞれサフィックス K、M を使用します。

QAnywhere Agent の起動時に、このオプションで指定されている値が ias_MaxDownloadSize メッセージストアプロパティに割り当てられます。このメッセージストアプロパティによって、ダウンロードサイズの上限が定義されます。転送がトリガーされると、サーバーは、すべてのメッセージの合計サイズがこのオプションで設定されている上限に到達するまで、クライアントへの配信対象メッセージをタグ付けします。サーバーは、キューに登録されているすべてのメッセージが配信されるまで、メッセージをバッチ送信します。各バッチのメッセージが転送されるたびに転送ルールが再実行されるため、転送中に優先度の高いメッセージがキューに登録された場合は、そのメッセージがキューの先頭に移動されます。

キューに登録された配信対象メッセージのうちでダウンロードスレッショルドを超えたものは、複数の小さいメッセージパーツに分割されます。各メッセージパーツは別個にダウンロードできるため、何度か同期を行いながらメッセージを段階的にダウンロードできます。送信先ですべてのメッセージパーツが受信されると、完全なメッセージが受信されたことになります。

インクリメンタルダウンロードのサイズは概算値です。実際のダウンロードサイズは、メッセージのサイズ以外にも多くの要素によって異なります。

参照

- 「事前に定義されたクライアントメッセージストアプロパティ」761 ページの ias_MaxDownloadSize

-iu qaagent オプション

インクリメンタルアップロードのサイズを指定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

qaagent -iu *upload-size* [**K** | **M**] ...

デフォルト

256K

備考

このオプションは、メッセージ転送のアップロード部分のサイズをバイト単位で指定します。キロバイトまたはメガバイトの単位を指定するには、それぞれサフィックス K、M を使用します。

QAnywhere Agent の起動時に、このオプションで指定されている値が ias_MaxUploadSize メッセージストアプロパティに割り当てられます。このメッセージストアプロパティによって、アップロードサイズの上限が定義されます。転送がトリガーされると、QAnywhere Agent は、すべてのメッセージの合計サイズがこのオプションで設定されている上限に到達するまで、サーバーへの配信対象メッセージをタグ付けします。この上限に到達すると、タグ付けされたメッセージがサーバーに送信されます。メッセージがサーバーで受信され、受信確認がサーバーからクライアントに正常に送信された場合は、転送のダウンロードフェーズが失敗しても、メッセージは問題なく配信されたとみなされます。QAnywhere Agent は、キューに登録されているすべてのメッセージが配信されるまで、メッセージをサーバーにバッチ送信します。各バッチのメッセージが転送されるたびに転送ルールが再実行されるため、転送中に優先度の高いメッセージがキューに登録された場合は、そのメッセージがキューの先頭に移動されます。

アップロードスレッショルドを超えたメッセージは、複数の小さいメッセージパーツに分割されます。各メッセージパーツは別個にアップロードできるため、何度か同期を行いながらメッセージを段階的にアップロードできます。送信先ですべてのメッセージパーツが受信されると、完全なメッセージが受信されたことになります。

インクリメンタルアップロードのサイズは概算値です。実際のアップロードサイズは、メッセージのサイズ以外にも多くの要素によって異なります。

参照

- 「事前に定義されたクライアントメッセージストアプロパティ」761 ページの ias_MaxUploadSize

-lp qaagent オプション

Mobile Link Listener のポートを指定します。

構文

qaagent -lp *number* ...

デフォルト

5001

備考

Mobile Link Listener が Mobile Link サーバーからの UDP 通知を受信するポートの番号です。通知は、サーバー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。QAnywhere Agent では、Mobile Link Listener に制御コマンドを送信するために、UDP ポートも使用します。

参照

- 「Push 通知によるメッセージングのシナリオ」7 ページ
- 「-push qaagent オプション」729 ページ

-mn qaagent オプション

Mobile Link ユーザーの新しいパスワードを指定します。

構文

qaagent -mp *password* ...

デフォルト

なし

備考

パスワードを変更する場合に使用します。

参照

- 「Mobile Link ユーザー」『Mobile Link クライアント管理』
- 「-mp qaagent オプション」722 ページ
- 「-mu qaagent オプション」723 ページ

-mp qaagent オプション

Mobile Link ユーザーの Mobile Link パスワードを指定します。

構文

qaagent -mp *password* ...

デフォルト

なし。

備考

Mobile Link サーバーでユーザー認証が必要な場合は、-mp オプションを使用して Mobile Link パスワードを指定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「Mobile Link ユーザー」『Mobile Link クライアント管理』
- 「-mu qaagent オプション」723 ページ

-mu qaagent オプション

Mobile Link ユーザー名を指定します。

構文

qaagent -mu *username* ...

デフォルト

クライアントメッセージストア ID

備考

Mobile Link ユーザー名は、Mobile Link サーバーで認証を行う場合に使用します。

存在しないユーザー名を指定した場合は、自動的に作成されます。

すべての Mobile Link ユーザー名は、サーバーメッセージストアに登録する必要があります。「QAnywhere クライアントユーザー名の登録」32 ページを参照してください。

参照

- 「Mobile Link ユーザー」『Mobile Link クライアント管理』
- 「-id qaagent オプション」719 ページ
- 「-mp qaagent オプション」722 ページ
- 「リモート ID」『Mobile Link クライアント管理』

-o qaagent オプション

指定されたログファイルに出力を送信します。

構文

qaagent -o *logfile* ...

デフォルト

なし。

備考

指定された名前のファイルにログを出力します。ファイルが既に存在している場合は、新しいログ情報がそのファイルに追加されます。SQL Anywhere 同期クライアント (dbmlsync) は、指定された名前に *_sync* というサフィックスを付けたものを名前とするファイルに、ログを出力します。Mobile Link Listener ユーティリティ (dblsn) は、指定された名前に *_lsn* というサフィックスを付けたものを名前とするファイルに、ログを出力します。

たとえば、*c:¥tmp¥mylog.out* というログファイルを指定した場合、qaagent は *c:¥tmp¥mylog.out* に、dbmlsync は *c:¥tmp¥mylog_sync.out* に、dblsn は *c:¥tmp¥mylog_lsn.out* に、それぞれログを出力します。

参照

- 「-ot qaagent オプション」725 ページ
- 「-on qaagent オプション」724 ページ
- 「-os qaagent オプション」725 ページ
- 「-v qaagent オプション」735 ページ

-on qaagent オプション

QAnywhere Agent のメッセージログファイルの最大サイズを指定します。ログファイルがこのサイズに達すると、現在のファイルが拡張子 *.old* の付いた名前に変更され、新しいファイルが作成されます。

構文

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

デフォルト

なし。

備考

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

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

> **注意**
> *.old* ファイルがすでに存在する場合は、そのファイルが上書きされます。古いログファイルを削除しないようにするには、代わりに -os オプションを使用します。
>
> このオプションは、-os オプションと同時に使用できません。

参照

- 「-o qaagent オプション」723 ページ
- 「-ot qaagent オプション」725 ページ
- 「-os qaagent オプション」725 ページ
- 「-v qaagent オプション」735 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

-os qaagent オプション

QAnywhere Agent のメッセージログファイルの最大サイズを指定します。ログファイルがこのサイズに達すると、新しい名前を持つ新しいログファイルが作成されて使用されます。

構文

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

デフォルト

なし。

備考

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

QAnywhere Agent は、現在のファイルサイズを確認してから、ファイルにメッセージのログを出力します。ログメッセージによって、ファイルサイズが指定したサイズより大きくなると、QAnywhere Agent はメッセージログファイルの名前を *yymmddxx.mls* に変更します。ここで、*xx* は 00 ～ 99 の連続した数字で、*yymmdd* は現在の年月日を表します。

このオプションを使用すると、古いメッセージログファイルを削除して、ディスク容量を解放できます。常に最新の出力が -o または -ot で指定したファイルに追加されます。

> **注意**
> このオプションは、-on オプションとは一緒に使用できません。

参照

- 「-o qaagent オプション」723 ページ
- 「-ot qaagent オプション」725 ページ
- 「-on qaagent オプション」724 ページ
- 「-v qaagent オプション」735 ページ

-ot qaagent オプション

ログファイルをトランケートし、そのファイルに出力メッセージを追加します。

構文

qaagent -ot *logfile* ...

デフォルト

なし。

備考

指定された名前のファイルにログを出力します。ファイルが存在する場合は、まずファイルのサイズが 0 にトランケートされます。SQL Anywhere 同期クライアント (dbmlsync) は、指定された名前に *_sync* というサフィックスを付けたものを名前とするファイルに、ログを出力します。Mobile Link Listener ユーティリティ (dblsn) は、指定された名前に *_lsn* というサフィックスを付けたものを名前とするファイルに、ログを出力します。

たとえば、*c:¥tmp¥mylog.out* というログファイルを指定した場合、qaagent は *c:¥tmp¥mylog.out* に、dbmlsync は *c:¥tmp¥mylog_sync.out* に、dblsn は *c:¥tmp¥mylog_lsn.out* に、それぞれログを出力します。

参照

- 「-o qaagent オプション」723 ページ
- 「-on qaagent オプション」724 ページ
- 「-os qaagent オプション」725 ページ
- 「-v qaagent オプション」735 ページ

-pc qaagent オプション

同期と同期の間で、Mobile Link サーバーへの永続的な接続を維持します。

構文

qaagent -pc { **+** | **-** } ...

デフォルト

-pc-

備考

ネットワークカバレッジが良好で QAnywhere 上のメッセージトラフィックが多い場合は、永続的な接続を有効にする (-pc+) と便利です。この場合は、メッセージ転送が行われるたびに TCP/IP 接続の設定と切断を実行するというネットワーク負荷を軽減できます。

クライアントデバイスにパブリック IP アドレスが設定されており、UDP や SMS でアクセス可能であれば、次のような場合に永続的な接続を無効にする (-pc-) と便利です。

- クライアントデバイスでダイヤルアップネットワーキングを使用しており、接続時間にかかるコストが懸案である場合。
- QAnywhere 上のメッセージトラフィックが少ない場合。永続的な TCP/IP 接続ではネットワークサーバーのリソースが消費されるため、スケーラビリティに影響を与える可能性があります。
- クライアントデバイスのネットワークカバレッジが安定していない場合。接続可能なときに自動ポリシーを使用してメッセージを転送できます。このような環境で永続的な接続を維持しようとしても、CPU リソースが浪費されるだけで、何の役にも立ちません。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「-push qaagent オプション」729 ページ
- 「-pc+ dbmlsync オプション」『Mobile Link クライアント管理』

-policy qaagent オプション

メッセージ転送のタイミングを決定するポリシーを指定します。

構文

qaagent -policy *policy-type* ...

policy-type: **ondemand** | **scheduled**[*interval-in-seconds*] | **automatic** | *rules-file*

デフォルト

- デフォルトのポリシータイプは automatic です。
- デフォルトの scheduled ポリシーの間隔は 900 秒 (15 分) です。

備考

QAnywhere ではポリシーを使用して、メッセージ転送のタイミングを決定します。*policy-type* には、次のいずれかの値を指定できます。

- **ondemand**　QAnywhere クライアントアプリケーションによって適切なメソッドが呼び出されたときだけ、メッセージが転送されます。

  QAManager PutMessage() メソッドは、メッセージをローカルのキューに登録します。キューに登録されたメッセージは、QAManager TriggerSendReceive() メソッドが呼び出されるまでサーバーに転送されません。同様に、サーバー上で送信待機中のメッセージは、クライアントによって TriggerSendReceive() が呼び出されるまで、クライアントに送信されません。

  ondemand ポリシーが使用されている場合は、アプリケーションはサーバーから Push 通知を受信するとメッセージを転送します。Push 通知時にはシステムメッセージが QAnywhere クライアントに配信されます。クライアントアプリケーションでは、TriggerSendReceive() を呼び出すことによって、このシステムメッセージに応答できます。

  例については、「システムキュー」61 ページを参照してください。

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

  ○前回の時間間隔が経過した後、クライアントメッセージストアに新しいメッセージを受信した。

○前回の時間間隔が経過した後、メッセージステータスが変化した。この現象は、通常、アプリケーションがメッセージの受信を確認したときに起こります。 受信確認の詳細については、次の項を参照してください。

- AcknowledgementMode 列挙体 [QAnywhere .NET]317 ページ
- AcknowledgementMode クラス [QAnywhere C++]375 ページ
- AcknowledgementMode インターフェイス [QAnywhere Java]495 ページ

○前回の時間間隔が経過した後、Push 通知を受信した。

○前回の時間間隔が経過した後、ネットワークステータス変更通知を受信した。

○Push 通知が無効にされた。

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

○ QAManagerBase.TriggerSendReceive メソッド [QAnywhere .NET]278 ページ
○ QAManagerBase.triggerSendReceive メソッド [QAnywhere C++]450 ページ
○ QAManagerBase.triggerSendReceive メソッド [QAnywhere Java]577 ページ
○ SQL：「ml_qa_triggersendreceive」694 ページ

- **automatic**　次のいずれかのイベントが発生した場合に、メッセージを転送します。

QAnywhere Agent は、メッセージキューをできるかぎり最新の状態に維持しようとします。次のいずれかのイベントが発生すると、クライアントのキューに登録されているメッセージがサーバーに配信され、サーバー上のキューに登録されているメッセージがクライアントに配信されます。

○PutMessage() の呼び出し。

○TriggerSendReceive() の呼び出し。

○Push 通知。

通知の詳細については、「Push 通知によるメッセージングのシナリオ」7 ページを参照してください。

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

- ***rules-file***　クライアントの転送ルールファイルを指定します。転送ルールファイルには、メッセージを転送するタイミングを決定するための複雑なルールセットを記述できます。

「クライアント側の転送ルール」783 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」49 ページ
- 「Push 通知によるメッセージングのシナリオ」7 ページ

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

-push qaagent オプション

Push 通知が有効であるかどうかを指定します。

構文

qaagent -push *mode* ...

mode : **none** | **connected** | **disconnected** | **lwpoll**

デフォルト

connected

オプション

モード	説明
none	Push 通知は、このエージェントに対して無効です。Mobile Link Listener (dblsn) は起動されません。
connected	Push 通知は、このエージェントに対して有効です。永続的な TCP/IP 接続が使用されます。Mobile Link Listener (dblsn) が qaagent によって起動され、Mobile Link サーバーへの永続的な接続を維持しようとします。クライアントデバイスにパブリック IP アドレスが設定されていない場合や、UDP メッセージを許可しないファイアウォールの背後に Mobile Link サーバーが置かれている場合は、このモードを使用すると便利です。これはデフォルトです。
disconnected	Push 通知は、このエージェントに対して有効です。永続的でない UDP 接続が使用されます。Mobile Link Listener (dblsn) が qaagent によって起動されますが、Mobile Link サーバーへの永続的な接続は維持されません。代わりに、UDP リスナーが Mobile Link から Push 通知を受信します。クライアントデバイスにパブリック IP アドレスが設定されており、UDP や SMS でアクセス可能であれば、次のような場合にこのモードを使用すると便利です。 ● クライアントデバイスでダイヤルアップネットワーキングを使用しており、接続時間にかかるコストが懸案である場合。 ● QAnywhere 上のメッセージトラフィックが少ない場合。永続的な TCP/IP 接続ではネットワークサーバーのリソースが消費されるため、スケーラビリティに影響を与える可能性があります。 ● クライアントデバイスのネットワークカバレッジが安定していない場合。接続可能なときに自動ポリシーを使用してメッセージを転送できます。このような環境で永続的な接続を維持しようとしても、CPU リソースが浪費されるだけで、何の役にも立ちません。

モード	説明
lwpoll	Push 通知は、ライトウェイトポーリングを使用しているこのエージェントに対して有効です。ライトウェイトポールのポール期間を調整するには、**lwpoll** の後に希望の間隔 (秒単位) を角カッコで囲んで指定します。たとえば、-push lwpoll[5] は、ポールの間隔が 5 秒のライトウェイトポーリングを使用するように QAnywhere Agent を設定します。間隔が指定されていない場合、デフォルトの間隔は 60 秒になります。 セキュリティ保護された環境でライトウェイトポーリングを使用するクライアントでは、他の Push 通知モードと異なり、Mobile Link Listener 用の Mobile Link ユーザーを登録する必要はありません。

備考

通知を使用しない場合は、このオプションを none に設定します。通知を無効にした場合、*dblsn.exe* 実行プログラムをクライアントに配備する必要はありません。

通知を使用しない QAnywhere の詳細については、「簡単なメッセージングシナリオ」5 ページを参照してください。

UDP を使用している場合、disconnected モードの ActiveSync で Push 通知を使用することはできません。これは、ActiveSync の UDP 実装に制限があるためです。

参照

- 「Push 通知」34 ページ
- 「-pc qaagent オプション」726 ページ
- 「QAnywhere Agent の起動」47 ページ
- 「Push 通知の通知」63 ページ
- 「-lp qaagent オプション」721 ページ

-q qaagent オプション

QAnywhere Agent をクワイエットモードで起動し、ウィンドウをシステムトレイ内に最小化します。

構文

qaagent -q ...

デフォルト

なし。

備考

QAnywhere Agent をクワイエットモード (-q) で起動すると、メインウィンドウはシステムトレイ内に最小化されます。また、メッセージストア用のデータベースサーバーが、-qi オプションを指定した状態で起動されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

● 「-qi qaagent オプション」 731 ページ

-qi qaagent オプション

QAnywhere Agent をクワイエットモードで起動し、ウィンドウを完全に非表示にします。

構文

qaagent -qi ...

デフォルト

なし。

備考

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

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

-qi クワイエットモードで動作している QAnywhere Agent は、**qastop** と入力することによってのみ停止できます。

参照

● 「qastop ユーティリティ」 760 ページ
● 「-q qaagent オプション」 730 ページ

-si qaagent オプション

データベースを、クライアントメッセージストアとして使用できるように初期化します。

構文

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

デフォルト

なし。このオプションは、クライアントメッセージストアの初期化のために、一度だけ使用します。

備考

このオプションを使用する前に、SQL Anywhere データベースを作成する必要があります。-si オプションを指定すると、QAnywhere Agent は、データベースを初期化して、QAnywhere システムテーブルなどのデータベースオブジェクトを追加します。初期化後、QAnywhere Agent はすぐに終了します。

-si オプションを使用するときは、初期化の対象となるデータベースを、-c オプションで接続文字列として指定する必要があります。-c オプションに指定する接続文字列には、DBA 権限を持つユーザー ID も指定します。ユーザー ID とパスワードを指定しなければ、デフォルトユーザー 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ー ml_qa_user とパスワード qanywhere が作成されます。ユーザー 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ー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ー 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ー ID またはパスワード (あるいは両方) を指定する必要があります。

注意
デフォルトのパスワードは必ず変更してください。それには、GRANT 文を使用します。

-si オプションで、クライアントメッセージストアの ID を指定することはできません。ID を割り当てるには、-si を実行するとき、または次回 qaagent を実行するときに -id オプションを指定します。-id オプションを指定しなければ、デフォルトでデバイス名が ID として割り当てられます。

メッセージストアは作成されているが ID が設定されていないという状態の場合、そのメッセージストアに対してローカルな QAnywhere アプリケーション同士ではメッセージを送受信できますが、リモートの QAnywhere アプリケーションとのメッセージ交換はできません。ID が割り当てられた時点で、リモートメッセージングも可能になります。

qaagent の -si オプションと -sil オプションは、相互に排他的です。-sil を指定して初期化したデータベースでは、-si を指定した qaagent を実行できません。

参照

- 「クライアントメッセージストアの設定」25 ページ
- 「セキュリティ保護されたクライアントメッセージストアの作成」121 ページ
- 「パスワードの変更」『SQL Anywhere サーバー データベース管理』

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"
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

-sil qaagent オプション

ローカルのアプリケーション間メッセージングのために SQL Anywhere データベースを初期化します。

構文

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

デフォルト

なし。このオプションは、クライアントメッセージストアの初期化のために、一度だけ使用します。

備考

このオプションでは、ローカル専用メッセージング用に同期データベースを使用できます。

> **注意**
> QAnywhere リモートメッセージング用には同期データベースを使用できません。QAnywhere リモートメッセージングアプリケーションの場合は、メッセージストアデータベースが QAnywhere 専用である必要があります。

qaagent の -si オプションと -sil オプションは、相互に排他的です。-si を指定して初期化したデータベースでは、-sil を指定した qaagent を実行できません。

参照

- 「-si qaagent オプション」 731 ページ

例

次に、ローカル専用メッセージング用にデータベースを初期化するコマンドの例を示します。

```
qaagent -c "dbf=qanywhere.db" -sil
```

-su qaagent オプション

クライアントメッセージストアを現在のバージョンにアップグレードします。

構文

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

備考

アンロードと再ロードを実行した後で、カスタムアクションを実行してから qaagent をアップグレードする場合は、このオプションが便利です。10.0.0 よりも前のメッセージストアからアップグレードする場合に、アンロードと再ロードを自動的に実行するには、-sur オプションを使用します。

10.0.0 よりも前のメッセージストアからアップグレードする場合は、最初にメッセージストアを手動でアンロードしてから、再ロードする必要があります。

この操作はアップグレードが完了すると終了します。

この操作は取り消せません。

参照

- 「-sur qaagent オプション」734 ページ

例

バージョン 9 のデータベースからアップグレードするには、最初にデータベースのアンロードと再ロードを行います。

```
dbunload  -q  -c "UID=dba;PWD=sql;DBF=qanywhere.db" -ar
```

次に、-su オプションを指定して qaagent を実行します。

```
qaagent -q -su -c "UID=dba;PWD=sql;DBF=qanywhere.db"
```

-sur qaagent オプション

クライアントメッセージストアを現在のバージョンにアップグレードします。

構文

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

備考

アップグレードするデータベースを接続文字列に指定してください。-sur オプションを指定すると、メッセージストアのアンロード、再ロード、アップグレードが自動的に実行されます。

Windows Mobile では、自動アンロード／再ロードはサポートされていません。Windows Mobile の場合、まずメッセージストアを手動でアンロードおよび再ロードする必要があります。

アンロード／再ロードは、製品のメジャーバージョン間でメッセージストアをアップグレードするために必要です。アンロードと再ロードを手動で行う場合は、-su オプションを指定します。たとえば、再ロード後にカスタムアクションを実行してからアップグレードする必要がある場合は、-su オプションを使用します。

この操作はアップグレードが完了すると終了します。

この操作は取り消せません。

参照

- 「-su qaagent オプション」733 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

例

次の例では、qanywhere.db というバージョン 9.0.2 の SQL Anywhere データベースをアンロードと再ロードしてから、QAnywhere バージョン 12 にアップグレードします。

```
qaagent -q -sur -c "UID=dba;PWD=sql;DBF=qanywhere.db"
```

-sv qaagent オプション

パーソナルデータベースサーバーの代わりに SQL Anywhere ネットワークデータベースサーバーをデータベースサーバーとして使用するように、エージェントに通知します。

構文

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

備考

デフォルトでは、qaagent は dbeng12 アプリケーションを使用してパーソナルデータベースサーバーに接続します。-sv を指定すると、qaagent は dbsrv12 アプリケーションを使用してネットワークデータベースサーバーに接続します。

参照

- 「SQL Anywhere データベースサーバーの構文」『SQL Anywhere サーバー データベース管理』

-v qaagent オプション

QAnywhere Agent のメッセージログファイルにログを出力し、QAnywhere Agent のメッセージウィンドウに表示する情報を指定できます。

構文

qaagent -v *levels* ...

デフォルト

最低の冗長レベル

備考

-v オプションはメッセージログファイルとメッセージウィンドウに影響します。qaagent コマンドラインで -o または -ot を指定すると、メッセージログファイルだけが作成されます。

冗長レベルが高すぎるとパフォーマンスに影響する可能性があるため、通常は、冗長レベルを高くするのは開発段階のみとしてください。

-v を単独で指定すると、最小限の情報のログが取られます。

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

- **+** すべてのロギングオプションを有効にします。

- **l** Mobile Link Listener のロギングをすべて表示します。このオプションを指定すると、Mobile Link Listener (dblsn) が冗長レベル -v3 で起動されます。 「Windows デバイス用の Mobile Link Listener ユーティリティ (dblsn)」『Mobile Link サーバー起動同期』の -v オプションを参照してください。

- **m** dbmlsync のロギングをすべて表示します。このオプションを指定すると、SQL Anywhere 同期サーバー (dbmlsync) が冗長レベル -v+ で起動されます。 dbmlsync に関する「-v dbmlsync オプション」『Mobile Link クライアント管理』を参照してください。

- **n** ネットワークステータス変化通知をすべて表示します。QAnywhere Agent は、これらの通知を Mobile Link Listener ユーティリティから受信します。

- **p** Push 通知をすべて表示します。QAnywhere Agent は、これらの通知を、Mobile Link サーバー上の Mobile Link Notifier を介して Mobile Link 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 qaagent オプション」723 ページ
- 「-ot qaagent オプション」725 ページ
- 「-on qaagent オプション」724 ページ
- 「-os qaagent オプション」725 ページ

-x qaagent オプション

Mobile Link サーバーとの通信に使用するネットワークプロトコルとプロトコルオプションを指定します。

構文

qaagent -x *protocol* [**(** *protocol-options*;... **)** ...]

protocol: **http**, **tcpip**, **https**, **tls**

protocol-options: *keyword*=*value*

備考

protocol-options の完全なリストについては、「Mobile Link クライアントネットワークプロトコルオプション」『Mobile Link クライアント管理』を参照してください。

-x オプションは、Mobile Link サーバーが QAnywhere Agent と同じデバイス上に存在しない場合、必須となります。

-x オプションは複数個指定できます。これによって、複数の Mobile Link サーバーへのフェールオーバーを設定できます。フェールオーバーを設定すると、QAnywhere Agent は、コマンドラインに入力された順番で各 Mobile Link サーバーへの接続を試みます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

QAnywhere Agent は Listener を使用する場合もあります。Listener は、クライアントへの転送を待機中のメッセージがサーバー上に存在することを知らせる通知を Mobile Link サーバーから受信します。この Listener は最初に指定されている Mobile Link サーバーだけを使用し、それ以外のサーバーへのフェールオーバーは行いません。

参照

- 「Mobile Link クライアントネットワークプロトコルオプション」『Mobile Link クライアント管理』
- 「通信ストリームの暗号化」123 ページ
- 「トランスポートレイヤーセキュリティ」『SQL Anywhere サーバー データベース管理』
- 「フェールオーバーメカニズムの設定」38 ページ
- 「-fd qaagent オプション」717 ページ
- 「-fr qaagent オプション」718 ページ

-xd qaagent オプション

QAnywhere Agent で Mobile Link サーバーの動的アドレス指定を使用することを指定します。

構文

qaagent -xd

備考

-xd を指定すると、QAnywhere Agent では、メッセージストアプロパティに基づいて Mobile Link サーバーのプロトコルとアドレスを特定できます。したがって、単一の Mobile Link サーバーのアドレスを動的に特定できます。Mobile Link サーバーのアドレスは、QAnywhere Agent が実行されているデバイスでアクティブな現在のネットワークによって異なります。

QAnywhere アプリケーションでは、Mobile Link サーバーの通信プロトコルとアドレスを示すメッセージストアプロパティを初期化し、現在アクティブなネットワークインターフェイスへの関係を確立する必要があります。モバイルデバイスのネットワークが切り替わると、QAnywhere Agent でアクティブなネットワークが検出され、Mobile Link サーバーの通信プロトコルとアドレスが自動的に調整されます。このとき再起動は必要ありません。

参照

- 「クライアントメッセージストアプロパティ」29 ページ

例

次の例では、デバイスが接続されているネットワークのタイプに基づいて、適切な Mobile Link のアドレスが使用されるようにプロパティを設定しています。たとえば、デバイスが LAN に接続されている場合、適切な LAN アドレスが使用されます。

```
QAManager mgr;
...
mgr.SetStringStoreProperty( "LAN.CommunicationAddress", "host=1.2.3.4;port=10997" );
mgr.SetStringStoreProperty( "LAN.CommunicationType", "tcpip" );
mgr.SetStringStoreProperty( "WAN.CommunicationAddress", "host=5.6.7.8;port=7777" );
mgr.SetStringStoreProperty( "WAN.CommunicationType", "tcpip" );
```

```
mgr.SetStringStoreProperty( "EL3C589 Ethernet Adapter.type", "LAN" );
mgr.SetStringStoreProperty( "Acme Wireless Adapter.type", "WAN" );
```

qauagent ユーティリティ

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を送受信します。このユーティリティは、クライアントメッセージストアが Ultra Light データベースである場合のみ使用してください。

注意
dbmlsync ユーティリティは、SQL Anywhere メッセージストアを同期するように設計されていますが、Ultra Light メッセージストアをサポートしていません。

構文

qauagent [*option* ...]

オプション	説明
@data	指定された環境変数または設定ファイルからオプションを読み込みます。「@data qauagent オプション」740 ページを参照してください。
-c *connection-string*	クライアントメッセージストアに接続するための接続文字列を指定します。 「-c qauagent オプション」741 ページを参照してください。
-cd *seconds*	データベースへの再試行を実行する間の遅延時間を指定します。 「-cd qaagent オプション」717 ページを参照してください。
-cr *number-of-retries*	接続がエラーになってから、データベースへの接続を再試行する回数を指定します。 「-cr qaagent オプション」717 ページを参照してください。
-fd *seconds*	プライマリサーバーへの再試行を実行する間の遅延時間を指定します。 「-fd qauagent オプション」743 ページを参照してください。
-fr *number-of-retries*	接続がエラーになってから、プライマリサーバーへの接続を再試行する回数を指定します。 「-fr qauagent オプション」744 ページを参照してください。
-id *id*	QAnywhere Ultra Light Agent が接続するクライアントメッセージストアの ID を指定します。 「-id qauagent オプション」744 ページを参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

オプション	説明
-idl *download-size*	メッセージ転送時に使用するダウンロードの最大サイズを指定します。 「-idl qauagent オプション」745 ページを参照してください。
-iu *upload-size*	メッセージ転送時に使用するアップロードの最大サイズを指定します。 「-iu qauagent オプション」746 ページを参照してください。
-lp *number*	Mobile Link Listener が Mobile Link サーバーからの通知を受信するポートを指定します。デフォルトは 5001 です。 「-lp qauagent オプション」747 ページを参照してください。
-mn *password*	Mobile Link ユーザーの新しいパスワードを指定します。「-mn qauagent オプション」747 ページを参照してください。
-mp *password*	Mobile Link ユーザーのパスワードを指定します。 「-mp qauagent オプション」748 ページを参照してください。
-mu *username*	Mobile Link ユーザーを指定します。 「-mp qauagent オプション」748 ページを参照してください。
-o *logfile*	出力メッセージを記録するファイルを指定します。 「-o qauagent オプション」749 ページを参照してください。
-on *size*	QAnywhere Ultra Light Agent のメッセージログファイルの最大サイズを指定します。ログファイルがこのサイズに達すると、現在のファイルが拡張子 .old の付いた名前に変更され、新しいファイルが作成されます。 「-on qauagent オプション」749 ページを参照してください。
-os *size*	QAnywhere Ultra Light Agent のメッセージログファイルの最大サイズを指定します。ログファイルがこのサイズに達すると、新しい名前を持つ新しいログファイルが作成されて使用されます。「-os qauagent オプション」750 ページを参照してください。
-ot *logfile*	出力メッセージを記録するファイルを指定します。 「-ot qauagent オプション」751 ページを参照してください。
-policy *policy-type*	QAnywhere Ultra Light Agent によって使用される転送ポリシーを指定します。「-policy qauagent オプション」751 ページを参照してください。
-push *mode*	Push 通知を有効または無効にします。デフォルトは有効です。「-push qauagent オプション」753 ページを参照してください。

オプション	説明
-q	QAnywhere Ultra Light Agent をクワイエットモードで起動し、ウィンドウをシステムトレイ内に最小化します。「-q qauagent オプション」755 ページを参照してください。
-qi	QAnywhere Ultra Light Agent をクワイエットモードで起動し、ウィンドウを完全に非表示にします。「-qi qauagent オプション」755 ページを参照してください。
-si	データベースを、クライアントメッセージストアとして使用できるように初期化します。「-si qauagent オプション」756 ページを参照してください。
-sil	ローカルのアプリケーション間メッセージングのために SQL Anywhere データベースを初期化します。「-sil qaagent オプション」733 ページを参照してください。
-su	クライアントメッセージストアを現在のバージョンにアップグレードします。dbunload/reload は実行しません。 「-su qauagent オプション」757 ページを参照してください。
-v [*levels*]	ログの冗長レベルを指定します。 「-v qauagent オプション」757 ページを参照してください。
-wc *name*	QAnywhere Ultra Light Agent を起動するウィンドウクラス名を指定します。 「qastop ユーティリティ」760 ページを参照してください。
-x { **http**|**tcpip**|**tls**|**https** } [(*keyword*=*value*;...)]	Mobile Link サーバーとの通信に使用するプロトコルのオプションを指定します。. 「-x qauagent オプション」758 ページを参照してください。
-xd	QAnywhere Ultra Light Agent で Mobile Link サーバーの動的アドレス指定を使用することを指定します。「-xd qauagent オプション」759 ページを参照してください。

参照

- 「QAnywhere Agent の起動」47 ページ

@data qauagent オプション

指定された環境変数または設定ファイルからオプションを読み込みます。

構文

qauagent @{ *filename* | *environment-variable* } ...

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

このオプションを指定すると、環境変数または設定ファイル内にコマンドラインオプションを記述できます。指定された名前の環境変数と指定された名前の設定ファイルの両方が存在する場合、環境変数が使用されます。

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

Windows Mobile ではショートカットのコマンドラインが 256 文字に制限されているため、このオプションが役に立ちます。

Sybase Central での同等機能

Sybase Central 用の QAnywhere 12 プラグインには、**[エージェントコマンドファイルの作成]** というタスクがあります。このタスクを選択すると、ファイル名を入力するように要求されます。次に **[プロパティ]** ウィンドウが表示され、コマンド情報を入力できます。生成されるファイルの拡張子は *.qaa* です。ファイル拡張子 *.qaa* は、Sybase Central の規則に従っています。つまりこのファイルは、@data オプションで作成するファイルと同じです。Sybase Central で作成されたコマンドファイルは、@data の設定ファイルとして使用できます。

参照

- 「設定ファイル」『SQL Anywhere サーバー データベース管理』
- 「ファイル非表示ユーティリティ (dbfhide)」『SQL Anywhere サーバー データベース管理』

-c qauagent オプション

クライアントメッセージストアに接続するための文字列を指定します。

構文

qauagent -c *connection-string* ...

デフォルト

接続パラメーター	デフォルト値
uid	ml_qa_user
pwd	qanywhere

備考

接続文字列では、接続パラメーターを *keyword=value* の形式で指定します。パラメーターとパラメーター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。

ODBC データソースは、一般的にはクライアントデバイスでは使用されません。ODBC は qauagent で使用されません。

以下に、よく使用する接続パラメーターの一部を示します。

- **dbf=*filename*** 指定されたファイル名を持つメッセージストアに接続します。
- **dbn=*database-name*** データベースファイルではなくデータベース名を指定して、すでに実行されているクライアントメッセージストアに接続します。
- **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ー ID を指定します。このパラメーターは、デフォルトの UID または PWD 接続パラメーターを変更する場合に必要となります。
- **pwd=*password*** データベースユーザー ID に対応するパスワードを指定します。このパラメーターは、デフォルトの UID または PWD 接続パラメーターを変更する場合に必要となります。
- **dbkey=*key*** データベースにアクセスするための暗号化キーを指定します。

参照

- 「接続パラメーター」『SQL Anywhere サーバー データベース管理』
- 「SQL Anywhere データベース接続」『SQL Anywhere サーバー データベース管理』
- 「Ultra Light DBF 接続パラメーター」『Ultra Light データベース管理とリファレンス』
- 「Ultra Light PWD 接続パラメーター」『Ultra Light データベース管理とリファレンス』
- 「Ultra Light DBKEY 接続パラメーター」『Ultra Light データベース管理とリファレンス』

例

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

-cd qauagent オプション

このオプションが -cr オプションと一緒に指定されている場合は、データベースへの接続を試行する間隔を指定します。

構文

qauagent -cd *seconds* ...

備考

再試行間のデフォルトの遅延時間は 10 秒です。

再試行に失敗すると、QAnywhere Agent はエラーを表示し、停止を待機します。

操作中にデータベース接続に失敗すると、接続の完了およびスレッドと外部プロセスの終了を含む終了手順が完了してから、QAnywhere Agent が再起動します。

このオプションは、qauagent -cr オプションと一緒に使用する必要があります。-cr オプションはデータベースへの接続を再試行する回数を指定し、-cd オプションは再試行の間隔を指定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

-cr qauagent オプション

QAnywhere Ultra Light Agent がデータベースへの接続を再試行する回数を指定します。

構文

qauagent -cr *number-of-retries* ...

備考

デフォルトの再試行回数は 3 回で、再試行間の遅延時間は 10 秒です。

-fd qauagent オプション

このオプションが -fr オプションと一緒に指定されている場合は、Mobile Link サーバーへの接続を試行する間隔を指定します。

構文

qauagent -fd *seconds* ...

デフォルト

- -fr だけを指定し -fd を指定しない場合は、遅延は 0 です (間隔をあけずに、すぐに再試行が行われます)。
- -fr を指定しないと、デフォルトで再試行が行われません。

備考

このオプションは、qauagent -fr オプションと一緒に使用する必要があります。-fr オプションはプライマリサーバーへの接続を再試行する回数を指定し、-fd オプションは再試行の間隔を指定します。

通常このオプションは、-x オプションを使用してフェールオーバー Mobile Link サーバーを指定する場合に使用します。フェールオーバー Mobile Link サーバーが設定されている場合、デフォルトでは、プライマリサーバーへの接続で障害が発生するとすぐに、QAnywhere Ultra Light Agent は代替サーバーへの接続を試行します。-fr オプションを指定すると、QAnywhere Ultra Light Agent は代替サーバーに接続する前に、プライマリサーバーへの接続を再試行します。-fd オプションを併せて使用することで、プライマリサーバーへの接続を試行する間隔も指定できます。

-fd オプションは 10 秒以下に設定することをおすすめします。

このオプションは、qauagent -xd オプションと一緒に使用することはできません。

参照

- 「-fr qauagent オプション」744 ページ
- 「-x qauagent オプション」758 ページ
- 「フェールオーバーメカニズムの設定」38 ページ

-fr qauagent オプション

QAnywhere Ultra Light Agent がプライマリ Mobile Link サーバーへの接続を再試行する回数を指定します。

構文

qauagent -fr *number-of-retries* ...

デフォルト

0 - QAnywhere Ultra Light Agent がプライマリ Mobile Link サーバーへの接続を再試行しないことを意味します。

備考

デフォルトでは、QAnywhere Ultra Light Agent が Mobile Link サーバーに接続できない場合、エラーは発生せずメッセージも送信されません。このオプションは、QAnywhere Ultra Light Agent が Mobile Link サーバーへの接続を再試行することを指定します。さらに、QAnywhere Ultra Light Agent が代替サーバーへの接続を試行するまで、またはエラーを発行するまで (代替サーバーが指定されていない場合) に、Mobile Link サーバーへの接続を再試行する回数も指定します。

通常このオプションは、-x オプションを使用してフェールオーバー Mobile Link サーバーを指定する場合に使用します。フェールオーバー Mobile Link サーバーが設定されている場合、デフォルトでは、プライマリサーバーへの接続で障害が発生するとすぐに、QAnywhere Ultra Light Agent は代替サーバーへの接続を試行します。このオプションを指定すると、QAnywhere Ultra Light Agent は代替サーバーに接続する前に、プライマリサーバーへの接続を再試行します。

-fd オプションを併せて使用することで、プライマリサーバーへの接続再試行の間隔も指定できます。

このオプションは、qauagent -xd オプションと一緒に使用することはできません。

参照

- 「-fd qauagent オプション」743 ページ
- 「-x qauagent オプション」758 ページ
- 「フェールオーバーメカニズムの設定」38 ページ

-id qauagent オプション

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

構文

qauagent -id *id* ...

デフォルト

ID のデフォルト値は、Agent が動作しているデバイスの名前です。デバイス名がユニークでない場合は、-id オプションを使用します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

備考

各クライアントメッセージストアは、メッセージストア ID と呼ばれるユニークな文字列で識別されます。メッセージストアに最初に接続するときに ID を設定しなかった場合、ID はデフォルトでデバイス名になります。次回以降の接続時には、設定済みのメッセージストア ID を -id オプションとともに指定する必要があります。

メッセージストア ID は Mobile Link リモート ID と対応しています。すべての Mobile Link アプリケーションではリモートデータベースごとにユニークな ID が必要であるため、メッセージストア ID が要求されます。

デバイス上で 2 つ目の qauagent インスタンスを起動する場合は、-id オプションを使用してユニークなメッセージストア ID を指定する必要があります。

次の文字は ID の構成文字として使用できません。

- " (二重引用符)
- 制御文字
- ¥¥ (二重円記号)

この他にも次の制約が適用されます。

- ID は 120 文字以内である必要がある。
- 単一の円記号 (¥) を使用できるのは、エスケープ文字として使用する場合に限られる。
- クライアントメッセージストアデータベースで quoted_identifier オプションが Off に設定されている場合は、英数字、アンダースコア (_)、アットマーク (@)、シャープ (#)、ドル記号 ($) のみを ID に含めることができる。

参照

- 「Mobile Link ユーザー」『Mobile Link クライアント管理』
- 「Mobile Link ユーザーの作成と登録」『Mobile Link クライアント管理』
- 「クライアントメッセージストアの設定」25 ページ

-idl qauagent オプション

インクリメンタルダウンロードのサイズを指定します。

構文

qauagent -idl *download-size* [**K** | **M**] ...

デフォルト

-1 - 最大ダウンロードサイズが指定されていないことを意味します。

備考

このオプションは、メッセージ転送のダウンロード部分のサイズをバイト単位で指定します。キロバイトまたはメガバイトの単位を指定するには、それぞれサフィックス K、M を使用します。

QAnywhere Ultra Light Agent の起動時に、このオプションで指定されている値が ias_MaxDownloadSize メッセージストアプロパティに割り当てられます。このメッセージストアプロパティによって、ダウンロードサイズの上限が定義されます。転送がトリガーされると、サーバーは、すべてのメッセージの合計サイズがこのオプションで設定されている上限に到達するまで、クライアントへの配信対象メッセージをタグ付けします。サーバーは、キューに登録されているすべてのメッセージが配信されるまで、メッセージをバッチ送信します。各バッチのメッセージが転送されるたびに転送ルールが再実行されるため、転送中に優先度の高いメッセージがキューに登録された場合は、そのメッセージがキューの先頭に移動されます。

キューに登録された配信対象メッセージのうちでダウンロードスレッショルドを超えたものは、複数の小さいメッセージパーツに分割されます。各メッセージパーツは別個にダウンロードできるため、何度か同期を行いながらメッセージを段階的にダウンロードできます。送信先ですべてのメッセージパーツが受信されると、完全なメッセージが受信されたことになります。

インクリメンタルダウンロードのサイズは概算値です。実際のダウンロードサイズは、メッセージのサイズ以外にも多くの要素によって異なります。

参照

- 「事前に定義されたクライアントメッセージストアプロパティ」761 ページの ias_MaxDownloadSize

-iu qauagent オプション

インクリメンタルアップロードのサイズを指定します。

構文

qauagent -iu *upload-size* [**K** | **M**] ...

デフォルト

256K

備考

このオプションは、メッセージ転送のアップロード部分のサイズをバイト単位で指定します。キロバイトまたはメガバイトの単位を指定するには、それぞれサフィックス K、M を使用します。

QAnywhere Ultra Light Agent の起動時に、このオプションで指定されている値が ias_MaxUploadSize メッセージストアプロパティに割り当てられます。このメッセージストアプロパティによって、アップロードサイズの上限が定義されます。転送がトリガーされると、QAnywhere Agent は、すべてのメッセージの合計サイズがこのオプションで設定されている上限に到達するまで、サーバーへの配信対象メッセージをタグ付けします。この上限に到達すると、タグ付けされたメッセージがサーバーに送信されます。メッセージがサーバーで受信され、受信確認がサーバーからクライアントに正常に送信された場合は、転送のダウンロードフェーズが失

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

敗しても、メッセージは問題なく配信されたとみなされます。QAnywhere Agent は、キューに登録されているすべてのメッセージが配信されるまで、メッセージをサーバーにバッチ送信します。各バッチのメッセージが転送されるたびに転送ルールが再実行されるため、転送中に優先度の高いメッセージがキューに登録された場合は、そのメッセージがキューの先頭に移動されます。

アップロードスレッショルドを超えたメッセージは、複数の小さいメッセージパーツに分割されます。各メッセージパーツは別個にアップロードできるため、何度か同期を行いながらメッセージを段階的にアップロードできます。送信先ですべてのメッセージパーツが受信されると、完全なメッセージが受信されたことになります。

インクリメンタルアップロードのサイズは概算値です。実際のアップロードサイズは、メッセージのサイズ以外にも多くの要素によって異なります。

参照

- 「事前に定義されたクライアントメッセージストアプロパティ」761 ページの ias_MaxUploadSize

-lp qauagent オプション

Mobile Link Listener のポートを指定します。

構文

qauagent -lp *number* ...

デフォルト

5001

備考

Mobile Link Listener が Mobile Link サーバーからの UDP 通知を受信するポートの番号です。通知は、サーバー上に待機中のメッセージが存在することを QAnywhere Ultra Light Agent に知らせるために使用されます。QAnywhere Ultra Light Agent では、Mobile Link Listener に制御コマンドを送信するために、UDP ポートも使用します。

参照

- 「Push 通知によるメッセージングのシナリオ」7 ページ
- 「-push qauagent オプション」753 ページ

-mn qauagent オプション

Mobile Link ユーザーの新しいパスワードを指定します。

構文

qauagent -mp *password* ...

デフォルト

なし。

備考

パスワードを変更する場合に使用します。

参照

- 「Mobile Link ユーザー」『Mobile Link クライアント管理』
- 「-mp qauagent オプション」748 ページ
- 「-mu qauagent オプション」748 ページ

-mp qauagent オプション

Mobile Link ユーザーの Mobile Link パスワードを指定します。

構文

qauagent -mp *password* ...

デフォルト

なし。

備考

Mobile Link サーバーでユーザー認証が必要な場合は、-mp オプションを使用して Mobile Link パスワードを指定します。

参照

- 「Mobile Link ユーザー」『Mobile Link クライアント管理』
- 「-mu qauagent オプション」748 ページ

-mu qauagent オプション

Mobile Link ユーザー名を指定します。

構文

qauagent -mu *username* ...

デフォルト

クライアントメッセージストア ID

備考

Mobile Link ユーザー名は、Mobile Link サーバーで認証を行う場合に使用します。

存在しないユーザー名を指定した場合は、自動的に作成されます。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

すべての Mobile Link ユーザー名は、サーバーメッセージストアに登録する必要があります。「QAnywhere クライアントユーザー名の登録」32 ページを参照してください。

参照

- 「Mobile Link ユーザー」『Mobile Link クライアント管理』
- 「-id qauagent オプション」744 ページ
- 「-mp qauagent オプション」748 ページ
- 「リモート ID」『Mobile Link クライアント管理』

-o qauagent オプション

指定されたログファイルに出力を送信します。

構文

qauagent -o *logfile* ...

デフォルト

なし。

備考

指定された名前のファイルにログを出力します。ファイルが既に存在している場合は、新しいログ情報がそのファイルに追加されます。Mobile Link Listener ユーティリティ (dblsn) は、指定された名前に *_lsn* というサフィックスを付けたものを名前とするファイルに、ログを出力します。

たとえば、*c:¥tmp¥mylog.out* というログファイルを指定した場合、qauagent は *c:¥tmp¥mylog.out* に、dblsn は *c:¥tmp¥mylog_lsn.out* に、それぞれログを出力します。

参照

- 「-ot qauagent オプション」751 ページ
- 「-on qauagent オプション」749 ページ
- 「-os qauagent オプション」750 ページ
- 「-v qauagent オプション」757 ページ

-on qauagent オプション

QAnywhere Ultra Light Agent のメッセージログファイルの最大サイズを指定します。ログファイルがこのサイズに達すると、現在のファイルが拡張子 *.old* の付いた名前に変更され、新しいファイルが作成されます。

構文

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

デフォルト

なし。

備考

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

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

> **注意**
> *.old* ファイルがすでに存在する場合は、そのファイルが上書きされます。古いログファイルを削除しないようにするには、代わりに -os オプションを使用します。
>
> このオプションは、-os オプションと同時に使用できません。

参照

- 「-o qauagent オプション」749 ページ
- 「-ot qauagent オプション」751 ページ
- 「-os qauagent オプション」750 ページ
- 「-v qauagent オプション」757 ページ

-os qauagent オプション

QAnywhere Ultra Light Agent のメッセージログファイルの最大サイズを指定します。ログファイルがこのサイズに達すると、新しい名前を持つ新しいログファイルが作成されて使用されます。

構文

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

デフォルト

なし。

備考

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

QAnywhere Ultra Light Agent は、現在のファイルサイズを確認してから、ファイルにメッセージのログを出力します。ログメッセージによって、ファイルサイズが指定したサイズより大きくなると、QAnywhere Ultra Light Agent はメッセージログファイルの名前を *yymmddxx.mls* に変更します。ここで、*xx* は 00 ～ 99 の連続した数字で、*yymmdd* は現在の年月日を表します。

このオプションを使用すると、古いメッセージログファイルを削除して、ディスク容量を解放できます。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。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

> **注意**
> このオプションは、-on オプションとは一緒に使用できません。

参照

- 「-o qauagent オプション」749 ページ
- 「-ot qauagent オプション」751 ページ
- 「-on qauagent オプション」749 ページ
- 「-v qauagent オプション」757 ページ

-ot qauagent オプション

ログファイルをトランケートし、そのファイルに出力メッセージを追加します。

構文

qauagent -ot *logfile* ...

デフォルト

なし。

備考

指定された名前のファイルにログを出力します。ファイルが存在する場合は、まずファイルのサイズが 0 にトランケートされます。Mobile Link Listener ユーティリティ (dblsn) は、指定された名前に *_lsn* というサフィックスを付けたものを名前とするファイルに、ログを出力します。

たとえば、*c:¥tmp¥mylog.out* というログファイルを指定した場合、qauagent は *c:¥tmp¥mylog.out* に、dblsn は *c:¥tmp¥mylog_lsn.out* に、それぞれログを出力します。

参照

- 「-o qauagent オプション」749 ページ
- 「-on qauagent オプション」749 ページ
- 「-os qauagent オプション」750 ページ
- 「-v qauagent オプション」757 ページ

-policy qauagent オプション

メッセージ転送のタイミングを決定するポリシーを指定します。

構文

qauagent -policy *policy-type* ...

policy-type: **ondemand** | **scheduled**[*interval-in-seconds*] | **automatic** | *rules-file*

デフォルト

- デフォルトのポリシータイプは automatic です。
- デフォルトの scheduled ポリシーの間隔は 900 秒 (15 分) です。

備考

QAnywhere ではポリシーを使用して、メッセージ転送のタイミングを決定します。*policy-type* には、次のいずれかの値を指定できます。

- **ondemand**　QAnywhere クライアントアプリケーションによって適切なメソッドが呼び出されたときだけ、メッセージが転送されます。

  QAManager PutMessage() メソッドは、メッセージをローカルのキューに登録します。キューに登録されたメッセージは、QAManager TriggerSendReceive() メソッドが呼び出されるまでサーバーに転送されません。同様に、サーバー上で送信待機中のメッセージは、クライアントによって TriggerSendReceive() が呼び出されるまで、クライアントに送信されません。

  ondemand ポリシーが使用されている場合は、アプリケーションはサーバーから Push 通知を受信するとメッセージを転送します。Push 通知時にはシステムメッセージが QAnywhere クライアントに配信されます。クライアントアプリケーションでは、TriggerSendReceive() を呼び出すことによって、このシステムメッセージに応答できます。

  例については、「システムキュー」61 ページを参照してください。

- **scheduled**　指定された間隔でメッセージを転送します。デフォルト値は 900 秒 (15 分) です。

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

  ○前回の時間間隔が経過した後、クライアントメッセージストアに新しいメッセージを受信した。

  ○前回の時間間隔が経過した後、メッセージステータスが変化した。この現象は、通常、アプリケーションがメッセージの受信を確認したときに起こります。

  ○前回の時間間隔が経過した後、Push 通知を受信した。

  ○前回の時間間隔が経過した後、ネットワークステータス変更通知を受信した。

  ○Push 通知が無効にされた。

  時間間隔を無視するには、トリガーの送受信メソッドを呼び出します。このメソッドを使用すると、時間間隔が経過する前にメッセージを転送できます。

- **automatic**　次のいずれかのイベントが発生した場合に、メッセージを転送します。

  QAnywhere Ultra Light Agent は、メッセージキューをできるかぎり最新の状態に維持しようとします。次のいずれかのイベントが発生すると、クライアントのキューに登録されているメッセージがサーバーに配信され、サーバー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。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

○PutMessage() の呼び出し。

○TriggerSendReceive() の呼び出し。

○Push 通知。

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

- ***rules-file*** クライアントの転送ルールファイルを指定します。転送ルールファイルには、メッセージを転送するタイミングを決定するための複雑なルールセットを記述できます。

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」49 ページ
- 「Push 通知によるメッセージングのシナリオ」7 ページ
- 「クライアント側の転送ルール」783 ページ

-push qauagent オプション

Push 通知が有効であるかどうかを指定します。

構文

qauagent -push *mode* ...

mode : **none** | **connected** | **disconnected** | **lwpoll**

デフォルト

connected

オプション

モード	説明
none	Push 通知は、このエージェントに対して無効です。Mobile Link Listener (dblsn) は起動されません。
connected	Push 通知は、このエージェントに対して有効です。永続的な TCP/IP 接続が使用されます。Mobile Link Listener (dblsn) が qauagent によって起動され、Mobile Link サーバーへの永続的な接続を維持しようとします。クライアントデバイスにパブリック IP アドレスが設定されていない場合や、UDP メッセージを許可しないファイアウォールの背後に Mobile Link サーバーが置かれている場合は、このモードを使用すると便利です。これはデフォルトです。

モード	説明
disconnected	Push 通知は、このエージェントに対して有効です。永続的でない UDP 接続が使用されます。Mobile Link Listener (dblsn) が qauagent によって起動されますが、Mobile Link サーバーへの永続的な接続は維持されません。代わりに、UDP リスナーが Mobile Link から Push 通知を受信します。クライアントデバイスにパブリック IP アドレスが設定されており、UDP や SMS でアクセス可能であれば、次のような場合にこのモードを使用すると便利です。 ● クライアントデバイスでダイヤルアップネットワーキングを使用しており、接続時間にかかるコストが懸案である場合。 ● QAnywhere 上のメッセージトラフィックが少ない場合。永続的な TCP/IP 接続ではネットワークサーバーのリソースが消費されるため、スケーラビリティに影響を与える可能性があります。 ● クライアントデバイスのネットワークカバレッジが安定していない場合。接続可能なときに自動ポリシーを使用してメッセージを転送できます。このような環境で永続的な接続を維持しようとしても、CPU リソースが浪費されるだけで、何の役にも立ちません。
lwpoll	Push 通知は、ライトウェイトポーリングを使用しているこのエージェントに対して有効です。ライトウェイトポールのポール期間を調整するには、**lwpoll** の後に希望の間隔 (秒単位) を角カッコで囲んで指定します。たとえば、-push lwpoll[5] は、ポールの間隔が 5 秒のライトウェイトポーリングを使用するように QAnywhere Agent を設定します。間隔が指定されていない場合、デフォルトの間隔は 60 秒になります。 セキュリティ保護された環境でライトウェイトポーリングを使用するクライアントでは、他の Push 通知モードと異なり、Mobile Link Listener 用の Mobile Link ユーザーを登録する必要はありません。

備考

通知を使用しない場合は、このオプションを none に設定します。通知を無効にした場合、*dblsn.exe* 実行プログラムをクライアントに配備する必要はありません。

通知を使用しない QAnywhere の詳細については、「簡単なメッセージングシナリオ」5 ページを参照してください。

UDP を使用している場合、disconnected モードの ActiveSync で Push 通知を使用することはできません。これは、ActiveSync の UDP 実装に制限があるためです。

参照

- 「Push 通知」34 ページ
- 「QAnywhere Agent の起動」47 ページ
- 「Push 通知の通知」63 ページ
- 「-lp qauagent オプション」747 ページ

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

-q qauagent オプション

QAnywhere Ultra Light Agent をクワイエットモードで起動し、ウィンドウをシステムトレイ内に最小化します。

構文

qauagent -q ...

デフォルト

なし。

備考

QAnywhere Ulra Light Agent をクワイエットモード (-q) で起動すると、メインウィンドウはシステムトレイ内に最小化されます。また、メッセージストア用のデータベースサーバーが、-qi オプションを指定した状態で起動されます。

参照

● 「-qi qauagent オプション」755 ページ

-qi qauagent オプション

QAnywhere Ultra Light Agent をクワイエットモードで起動し、ウィンドウを完全に非表示にします。

構文

qauagent -qi ...

デフォルト

なし。

備考

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

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

-qi クワイエットモードで動作している QAnywhere Ultra Light Agent は、**qastop** と入力することによってのみ停止できます。

参照

- 「qastop ユーティリティ」760 ページ
- 「-q qauagent オプション」755 ページ

-si qauagent オプション

データベースを、クライアントメッセージストアとして使用できるように初期化します。

構文

qauagent -c "*connection-string*" -si ...

デフォルト

なし。このオプションは、クライアントメッセージストアの初期化のために、一度だけ使用します。

備考

このオプションを使用する前に、Ultra Light データベースを作成する必要があります。-si オプションを指定すると、QAnywhere Ultra Light Agent は、データベースを初期化して、QAnywhere システムテーブルなどのデータベースオブジェクトを追加します。初期化後、QAnywhere Ultra Light Agent はすぐに終了します。

-si オプションを使用するときは、初期化の対象となるデータベースを、-c オプションで接続文字列として指定する必要があります。-c オプションに指定する接続文字列には、DBA 権限を持つユーザー ID も指定します。ユーザー ID とパスワードを指定しなければ、デフォルトユーザー 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ー ml_qa_user とパスワード qanywhere が作成されます。ユーザー 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ー名とパスワードを変更しないかぎり、qauagent の起動時にパスワードやユーザー ID を -c オプションに指定する必要はありません。変更した場合は、qauagent コマンドラインの -c オプションにユーザー ID またはパスワード (あるいは両方) を指定する必要があります。

> **注意**
> デフォルトのパスワードは必ず変更してください。それには、GRANT 文を使用します。

-si オプションで、クライアントメッセージストアの ID を指定することはできません。ID を割り当てるには、-si を実行するとき、または次回 qauagent を実行するときに -id オプションを指定します。-id オプションを指定しなければ、デフォルトでデバイス名が ID として割り当てられます。

メッセージストアは作成されているが ID が設定されていないという状態の場合、そのメッセージストアに対してローカルな QAnywhere アプリケーション同士ではメッセージを送受信できますが、リモートの QAnywhere アプリケーションとのメッセージ交換はできません。ID が割り当てられた時点で、リモートメッセージングも可能になります。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

参照

- 「クライアントメッセージストアの設定」25 ページ
- 「セキュリティ保護されたクライアントメッセージストアの作成」121 ページ
- 「パスワードの変更」『SQL Anywhere サーバー データベース管理』

例

次のコマンドを実行すると、*qaclient.db* データベースへの接続が実行され、このデータベースが QAnywhere クライアントメッセージストアとして初期化されます。QAnywhere Ultra Light Agent は、初期化が完了するとすぐに終了します。

```
qauagent -si -c "DBF=qaclient.udb"
```

-su qauagent オプション

クライアントメッセージストアを現在のバージョンにアップグレードします。10.0.0 よりも前のメッセージストアからアップグレードする場合は、最初にメッセージストアを手動でアンロードしてから、再ロードする必要があります。

構文

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

備考

アンロードと再ロードを実行した後で、カスタムアクションを実行してから qauagent をアップグレードする場合は、このオプションが便利です。10.0.0 よりも前のメッセージストアからアップグレードする場合に、アンロードと再ロードを自動的に実行するには、-sur オプションを使用します。

この操作はアップグレードが完了すると終了します。

この操作は取り消せません。

例

バージョン 9 のデータベースからアップグレードするには、最初にデータベースのアンロードと再ロードを行います。

```
dbunload  -q  -c "UID=dba;PWD=sql;DBF=qanywhere.db" -ar
```

次に、-su オプションを指定して qauagent を実行します。

```
qauagent -q -su -c "UID=dba;PWD=sql;DBF=qanywhere.db"
```

-v qauagent オプション

このオプションを使用すると、メッセージログファイルに出力する情報と、QAnywhere Ultra Light Agent コンソールに表示する情報を指定できます。

構文

qauagent -v *levels* ...

デフォルト

最低の冗長レベル

備考

-v オプションは、ログファイルとコンソールに影響します。qauagent コマンドラインで -o または -ot を指定すると、メッセージログだけが記録されます。

冗長レベルが高すぎるとパフォーマンスに影響する可能性があるため、通常は、冗長レベルを高くするのは開発段階のみとしてください。

-v を単独で指定すると、最小限の情報のログが取られます。

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

- **+** すべてのロギングオプションを有効にします。
- **l** Mobile Link Listener のロギングをすべて表示します。このオプションを指定すると、Mobile Link Listener (dblsn) が冗長レベル -v3 で起動されます。 「Windows デバイス用の Mobile Link Listener ユーティリティ (dblsn)」『Mobile Link サーバー起動同期』の -v オプションを参照してください。
- **m** 同期のロギングをすべて表示します。
- **n** ネットワークステータス変化通知をすべて表示します。QAnywhere Ultra Light Agent は、これらの通知を Mobile Link Listener ユーティリティから受信します。
- **p** Push 通知をすべて表示します。QAnywhere Ultra Light Agent は、これらの通知を、Mobile Link サーバー上の Mobile Link Notifier を介して Mobile Link Listener ユーティリティから受信します。
- **q** 転送ルールを表すために使用される SQL を表示します。
- **s** QAnywhere Ultra Light Agent によって開始されたメッセージ同期をすべて表示します。

参照

- 「-o qauagent オプション」749 ページ
- 「-ot qauagent オプション」751 ページ
- 「-on qauagent オプション」749 ページ
- 「-os qauagent オプション」750 ページ

-x qauagent オプション

Mobile Link サーバー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。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

構文

qauagent -x *protocol* [**(** *protocol-options*;... **)** ...]

protocol: **http**, **tcpip**, **https**, **tls**

protocol-options: *keyword*=*value*

備考

protocol-options の完全なリストについては、「Mobile Link クライアントネットワークプロトコルオプション」『Mobile Link クライアント管理』を参照してください。

-x オプションは、Mobile Link サーバーが QAnywhere Ultra Light Agent と同じデバイス上に存在しない場合、必須となります。

-x オプションは複数個指定できます。これによって、複数の Mobile Link サーバーへのフェールオーバーを設定できます。フェールオーバーを設定すると、QAnywhere Ultra Light Agent は、コマンドラインに入力された順番で各 Mobile Link サーバーへの接続を試みます。

QAnywhere Ultra Light Agent は Listener を使用する場合もあります。Listener は、クライアントへの転送を待機中のメッセージがサーバー上に存在することを知らせる通知を Mobile Link サーバーから受信します。この Listener は最初に指定されている Mobile Link サーバーだけを使用し、それ以外のサーバーへのフェールオーバーは行いません。

参照

- 「Mobile Link クライアントネットワークプロトコルオプション」『Mobile Link クライアント管理』
- 「通信ストリームの暗号化」 123 ページ
- 「トランスポートレイヤーセキュリティ」『SQL Anywhere サーバー データベース管理』
- 「フェールオーバーメカニズムの設定」 38 ページ
- 「-fd qauagent オプション」 743 ページ
- 「-fr qauagent オプション」 744 ページ

-xd qauagent オプション

QAnywhere Ultra Light Agent で Mobile Link サーバーの動的アドレス指定を使用することを指定します。

構文

qauagent -xd

備考

-xd を指定すると、QAnywhere Ultra Light Agent では、メッセージストアプロパティに基づいて Mobile Link サーバーのプロトコルとアドレスを特定できます。したがって、単一の Mobile Link サーバーのアドレスを動的に特定できます。Mobile Link サーバーのアドレスは、QAnywhere Ultra Light Agent が実行されているデバイスでアクティブな現在のネットワークによって異なります。

QAnywhere アプリケーションでは、Mobile Link サーバーの通信プロトコルとアドレスを示すメッセージストアプロパティを初期化し、現在アクティブなネットワークインターフェイスへの関係を確立する必要があります。モバイルデバイスのネットワークが切り替わると、QAnywhere Ultra Light Agent でアクティブなネットワークが検出され、Mobile Link サーバーの通信プロトコルとアドレスが自動的に調整されます。このとき再起動は必要ありません。

参照

- 「クライアントメッセージストアプロパティ」29 ページ

例

次の例では、デバイスが接続されているネットワークのタイプに基づいて、適切な Mobile Link のアドレスが使用されるようにプロパティを設定しています。たとえば、デバイスが LAN に接続されている場合、適切な LAN アドレスが使用されます。

```
QAManager mgr;
...
mgr.SetStringStoreProperty( "LAN.CommunicationAddress", "host=1.2.3.4;port=10997" );
mgr.SetStringStoreProperty( "LAN.CommunicationType", "tcpip" );
mgr.SetStringStoreProperty( "WAN.CommunicationAddress", "host=5.6.7.8;port=7777" );
mgr.SetStringStoreProperty( "WAN.CommunicationType", "tcpip" );
mgr.SetStringStoreProperty( "EL3C589 Ethernet Adapter.type", "LAN" );
mgr.SetStringStoreProperty( "Acme Wireless Adapter.type", "WAN" );
```

qastop ユーティリティ

エージェントがクワイエットモードで動作しているときに QAnywhere Agent または QAnywhere Ultra Light Agent を停止します。

構文

qastop [*option* ...]

オプション	説明
-id *id*	QAnywhere Agent または QAnywhere Ultra Light Agent を停止するクライアントメッセージストアの ID を指定します。
-wc *name*	QAnywhere Agent または QAnywhere Ultra Light Agent を停止するウィンドウクラス名を指定します。

参照

- 「-q qaagent オプション」730 ページ
- 「-q qauagent オプション」755 ページ
- 「QAnywhere Agent の停止」48 ページ

QAnywhere のプロパティ

クライアントメッセージストアプロパティ

以下の項では、クライアントメッセージストアプロパティについて説明します。

事前に定義されたクライアントメッセージストアプロパティ

利便性のために、いくつかのクライアントメッセージストアプロパティは事前に定義されています。事前に定義されたメッセージストアプロパティは次のとおりです。

- **ias_Adapters**　Mobile Link サーバーへの接続で使用できるネットワークアダプターのリスト。これは文字列のリストで、各文字列はパイプ記号で区切られます。

- **ias_MaxDeliveryAttempts**　このプロパティが定義されている場合は、メッセージのステータスが「受信不可」に設定される前に、受信確認されていないメッセージを配信できる最大回数を示します。デフォルトでは、このプロパティは定義されていません。これは値 -1 と同等で、クライアントライブラリは受信確認されていないメッセージの配信を無制限に試行します。

- **ias_MaxDownloadSize**　ダウンロードのインクリメントサイズ。デフォルトでは、QAnywhere が使用する最大ダウンロードサイズは -1 です。これは最大サイズの指定がないことを意味します。サーバーコネクターまたは送信先エイリアスから送信されたメッセージが、指定されたダウンロードのインクリメントサイズを超えた場合、メッセージは小さなメッセージパーツに分割され、別個のダウンロードとして送信されます。このプロパティは、qaagent -idl オプションで設定します。 SQL Anywhere のメッセージストアの場合は「-idl qaagent オプション」720 ページ、Ultra Light のメッセージストアの場合は「-idl qauagent オプション」745 ページを参照してください。

- **ias_MaxUploadSize**　アップロードのインクリメントサイズ。デフォルトでは、QAnywhere は 256 K のインクリメントサイズでメッセージをアップロードします。メッセージが指定されたアップロードのインクリメントサイズを超えた場合、メッセージは小さなメッセージパーツに分割され、別個のアップロードとして送信されます。このプロパティは、qaagent -iu オプションで設定します。 SQL Anywhere のメッセージストアの場合は 「-iu qaagent オプション」720 ページ、Ultra Light のメッセージストアの場合は 「-iu qauagent オプション」746 ページ を参照してください。.

- **ias_Network**　現在使用されているネットワークに関する情報。このプロパティは読み込み可能ですが、設定はできません。ias_Network は特殊なプロパティです。このプロパティには、デバイスで使用されている現在のネットワークに関する情報を提供するいくつかの組み込み属性が定義されています。次の属性は、QAnywhere によって自動的に設定されます。

  - **ias_Network.Adapter**　ネットワークカードを使用している場合は、現在のネットワークカード名 (Adapter 属性に割り当てられたネットワークカードの名前は、そのネットワークとの接続が確立されたとき、Agent ウィンドウに表示されます)。

  - **ias_Network.RAS**　現在の RAS エントリ名。

- ○ **ias_Network.IP**　デバイスに割り当てられた現在の IP アドレス。
- ○ **ias_Network.MAC**　現在使用しているネットワークカードの MAC アドレス。

- **ias_RASNames**　文字列。Mobile Link サーバーへの接続で使用できる RAS エントリ名のリスト。各文字列はパイプ記号で区切られます。
- **ias_StoreID**　メッセージストア ID。
- **ias_StoreInitialized**　このメッセージストアが QAnywhere メッセージング用に正常に初期化された場合は True。それ以外の場合は False。

  SQL Anywhere のメッセージストアの場合は「-si qaagent オプション」731 ページ、Ultra Light のメッセージストアの場合は「-si qauagent オプション」756 ページを参照してください。
- **ias_StoreVersion**　QAnywhere で定義されている、このメッセージストアのバージョン番号。

事前に定義されたメッセージプロパティの詳細については、次の項を参照してください。

- MessageStoreProperties クラス [QAnywhere C++]385 ページ
- MessageProperties クラス [QAnywhere .NET]189 ページ
- MessageStoreProperties インターフェイス [QAnywhere Java]504 ページ
- SQL API：「メッセージストアプロパティ」685 ページ

カスタムのクライアントメッセージストアプロパティ

QAnywhere では、QAnywhere C++、Java、SQL、または .NET API を使用して、ユーザー独自のクライアントメッセージストアプロパティを定義できます。これらのプロパティは、同じメッセージストアに接続されたアプリケーション間で共有されます。また、サーバーメッセージストアと同期されるため、このクライアント用のサーバー側の転送ルールでも使用できます。

クライアントメッセージストアプロパティ名では大文字と小文字を区別しません。文字、数字、アンダースコアを使用できますが、先頭は文字でなければなりません。次の名前は予約語なので、メッセージストアプロパティ名として使用することはできません。

- NULL
- TRUE
- FALSE
- NOT
- AND
- OR
- BETWEEN
- LIKE
- IN
- IS
- ESCAPE (SQL Anywhere メッセージストアのみ)
- **ias_** で始まるすべての名前

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

カスタムのクライアントメッセージストアプロパティ属性の使用

クライアントメッセージストアプロパティには、ユーザーが属性を定義できます。属性を定義するには、プロパティ名の後にドットを付け、その後に属性名を追加します。この機能の主な目的は、ユーザーのネットワーク情報を転送ルールで使用できるようにすることです。

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

例 (SQL Anywhere のみ)

次に、カスタムのクライアントメッセージストアプロパティ属性を設定する簡単な例を示します。この例の Object プロパティは、Shape と Color の 2 つの属性を持ちます。Sharp 属性の値は Round、Color 属性の値は Blue です。

```
// C++ example.
mgr->setStringStoreProperty( "Object.Shape", "Round" );
mgr->setStringStoreProperty( "Object.Color", "Blue" );

// C# example.
mgr.SetStoreStringProperty( "Object.Shape", "Round" );
mgr.SetStringStoreProperty( "Object.Color", "Blue" );

// Java example
mgr.setStringStoreProperty( "Object.Shape", "Round" );
mgr.setStringStoreProperty( "Object.Color", "Blue" );

-- SQL example
BEGIN
   CALL ml_qa_setstoreproperty( 'Object.Shape', 'Round' );
   CALL ml_qa_setstoreproperty( 'Object.Color', 'Blue' );
   COMMIT;
END
```

すべてのクライアントメッセージストアプロパティには、最初は値を持たない Type 属性があります。Type 属性の値は、別のプロパティの名前でなければなりません。プロパティの Type 属性を設定すると、そのプロパティは Type 属性の値に割り当てられているプロパティの属性を継承します。次の例の Object プロパティは、Circle プロパティの属性を継承しています。したがって、Object.Shape の値は Round、Object.Color の値は Blue になります。

```
// C++ example
QAManager qa_manager;
qa_manager->setStoreStringProperty( "Circle.Shape", "Round" );
qa_manager->setStoreStringProperty( "Circle.Color", "Blue" );
qa_manager->setStoreStringProperty( "Object.Type", "Circle" );

// C# example
QAManager qa_manager;
qa_manager.SetStringStoreProperty( "Circle.Shape", "Round" );
qa_manager.SetStringStoreProperty( "Circle.Color", "Blue" );
qa_manager.SetStringStoreProperty( "Object.Type", "Circle" );

// Java example
QAManager qa_manager;
qa_manager.setStringStoreProperty( "Circle.Shape", "Round" );
qa_manager.setStringStoreProperty( "Circle.Color", "Blue" );
qa_manager.setStringStoreProperty( "Object.Type", "Circle" );
```

```
-- SQL example
BEGIN
   CALL ml_qa_setstoreproperty( 'Circle.Shape', 'Round' );
   CALL ml_qa_setstoreproperty( 'Circle.Color', 'Blue' );
   CALL ml_qa_setstoreproperty( 'Object.Type', 'Circle');
   COMMIT;
END
```

例

次の C# の例は、メッセージストアプロパティを使用して、ユーザーのネットワーク情報を転送ルールに指定する方法を示しています。

LAN、無線 LAN、無線 WAN の 3 つの接続オプションが用意されている Windows ラップトップがあるとします。LAN 経由でネットワークにアクセスするには、My LAN Card という名前のネットワークカードを使用します。無線 LAN 経由でネットワークにアクセスするには、My Wireless LAN Card という名前のネットワークカードを使用します。無線 WAN 経由でネットワークにアクセスするには、My Wireless WAN Card という名前のネットワークカードを使用します。

ここでは、LAN または無線 LAN を使用して接続している場合は、サーバーにすべてのメッセージを送信し、無線 WAN を使用して接続している場合は、優先度の高いメッセージだけを送信するアプリケーションを開発します。優先度の高いメッセージとは、優先度が 7 以上のメッセージであると定義します。

まず、使用しているネットワークアダプターの名前を検索します。ネットワークアダプターの名前は、カードを装着し、ドライバー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、そのアダプター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ー名が表示されます。

```
"Listener thread received message '[netstat] network-adapter-name !...'
```

次に、LAN、WLAN、WWAN の 3 種類のネットワークタイプに対応するクライアントメッセージストアプロパティを定義し、各プロパティに Cost 属性を割り当てます。Cost 属性は 1 ～ 3 の値をとり、そのネットワークを使用したときに発生するコストを表します。1 が最も低いコストを表します。

```
QAManager qa_manager;
qa_manager.SetStoreProperty( "LAN.Cost", "1" );
qa_manager.SetStoreProperty( "WLAN.Cost", "2" );
qa_manager.SetStoreProperty( "WWAN.Cost", "3" );
```

次に、使用するネットワークカードごとに 1 つずつ、全部で 3 つのクライアントメッセージストアプロパティを定義します。このプロパティ名はネットワークカード名と一致させます。ネットワークの種類を Type 属性に割り当てることで、各プロパティに適切なネットワーク種別を割り当てます。したがって、各プロパティは、それぞれのネットワークの種類の属性を継承します。

```
QAManager qa_manager;
qa_manager.SetStoreProperty( "My LAN Card.Type", "LAN" );
qa_manager.SetStoreProperty( "My Wireless LAN Card.Type", "WLAN" );
qa_manager.SetStoreProperty( "My Wireless WAN Card.Type", "WWAN" );
```

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ネットワーク接続が確立されると、QAnywhere は、ias_Network プロパティの Adapter 属性に、My LAN Card、My Wireless LAN Card、または My Wireless WAN Card のいずれかを、使用しているネットワークに応じて自動的に設定します。同様に、ias_Network プロパティの Type 属性に、My LAN Card、My Wireless LAN Card、または My Wireless WAN Card のいずれかを自動的に設定して、使用しているネットワークの属性を ias_Network プロパティが継承するようにします。

最後に、次の転送ルールを作成します。

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

転送ルールの詳細については、「QAnywhere 転送ルールと削除ルール」774 ページを参照してください。

クライアントメッセージストアプロパティの列挙

QAnywhere .NET、C++、Java API では、事前定義とカスタムのメッセージストアプロパティを列挙できます。

.NET の例

QAManagerBase.GetStorePropertyNames メソッド [QAnywhere .NET]を参照してください。

```
// qaManager is a QAManager instance.
IEnumerator propertyNames = qaManager.GetStorePropertyNames();
```

C++ の例

QAManagerBase.beginEnumStorePropertyNames メソッド [QAnywhere C++]を参照してください。

```
// qaManager is a QAManager instance.
qa_store_property_enum_handle handle = qaManager->beginEnumStorePropertyNames();
qa_char propertyName[256];

if( handle != qa_null ) {
   while(qaManager->nextStorePropertyName(handle, propertyName, 255) != -1) {
      // Do something with the message store property name.
   }
   // Message store properties cannot be set after
   // the beginEnumStorePropertyNames call
   // and before the endEnumStorePropertyNames call.
   qaManager->endEnumStorePropertyNames(handle);
}
```

Java の例

QAManagerBase.getStorePropertyNames メソッド [QAnywhere Java]564 ページを参照してください。

```
// qaManager is a QAManager instance.
Enumeration propertyNames = qaManager.getStorePropertyNames();
```

アプリケーションでのクライアントメッセージストアプロパティの管理

次の QAManagerBase メソッドを使用すると、クライアントメッセージストアプロパティを取得および設定できます。

クライアントメッセージストアプロパティ管理用メソッド (C++)

- qa_bool getBooleanStoreProperty(qa_const_string name, qa_bool * value)
- qa_bool setBooleanStoreProperty(qa_const_string name, qa_bool value)
- qa_bool getByteStoreProperty(qa_const_string name, qa_byte * value)
- qa_bool setByteStoreProperty(qa_const_string name, qa_byte value)
- qa_bool getShortStoreProperty(qa_const_string name, qa_short * value)
- qa_bool setShortStoreProperty(qa_const_string name, qa_short value)
- qa_bool getIntStoreProperty(qa_const_string name, qa_int * value)
- qa_bool setIntStoreProperty(qa_const_string name, qa_int value)
- qa_bool getLongStoreProperty(qa_const_string name, qa_long * value)
- qa_bool setLongStoreProperty(qa_const_string name, qa_long value)
- qa_bool getFloatStoreProperty(qa_const_string name, qa_float * value)
- qa_bool setFloatStoreProperty(qa_const_string name, qa_float value)
- qa_bool getDoubleStoreProperty(qa_const_string name, qa_double * value)
- qa_bool setDoubleStoreProperty(qa_const_string name, qa_double value)
- qa_int getStringStoreProperty(qa_const_string name, qa_string value, qa_int len)
- qa_bool setStringStoreProperty(qa_const_string name, qa_const_string value)
- qa_store_property_enum_handle QAManagerBase::beginEnumStorePropertyNames()
- virtual qa_int QAManagerBase::nextStorePropertyName(qa_store_property_enum_handle h, qa_string buffer, qa_int bufferLen)
- virtual void QAManagerBase::endEnumStorePropertyNames(qa_store_property_enum_handle h)

QAManagerBase クラス [QAnywhere C++]418 ページを参照してください。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

クライアントメッセージストアプロパティ管理用メソッド (C#)

- Object GetStoreProperty(String name)
- void SetStoreProperty(String name, Object value)
- boolean GetBooleanStoreProperty(String name)
- void SetBooleanStoreProperty(String name, boolean value)
- byte GetByteStoreProperty(String name)
- void SetByteStoreProperty(String name, byte value)
- short GetShortStoreProperty(String name)
- void SetShortStoreProperty(String name, short value)
- int GetIntStoreProperty(String name)
- void SetIntStoreProperty(String name, int value)
- long GetLongStoreProperty(String name)
- void SetLongStoreProperty(String name, long value)
- float GetFloatStoreProperty(String name)
- void SetFloatStoreProperty(String name, float value)
- double GetDoubleStoreProperty(String name)
- void SetDoubleStoreProperty(String name, double value)
- String GetStringStoreProperty(String name)
- void SetStringStoreProperty(String name, String value)
- IEnumerator GetStorePropertyNames()

QAManagerBase インターフェイス [QAnywhere .NET]239 ページを参照してください。

クライアントメッセージストアプロパティ管理用メソッド (Java)

- boolean getBooleanStoreProperty(String name)
- void setBooleanStoreProperty(String name, boolean value)
- byte getByteStoreProperty(String name)
- void setByteStoreProperty(String name, byte value)
- double getDoubleStoreProperty(String name)
- void setDoubleStoreProperty(String name, double value)
- float getFloatStoreProperty(String name)
- void setFloatStoreProperty(String name, float value)
- int getIntStoreProperty(String name)
- void setIntStoreProperty(String name, int value)
- long getLongStoreProperty(String name)
- void setLongStoreProperty(String name, long value)
- short getShortStoreProperty(String name)
- void setShortStoreProperty(String name, short value)
- void setStringStoreProperty(String name, String value)
- String getStringStoreProperty(String name)
- java.util.Enumeration getStorePropertyNames()

QAManagerBase インターフェイス [QAnywhere Java]544 ページを参照してください。

クライアントメッセージストアプロパティ管理用 SQL ストアドプロシージャー

- ml_qa_getstoreproperty
- ml_qa_setstoreproperty

「メッセージストアプロパティ」685 ページを参照してください。

サーバープロパティ

サーバープロパティは、Sybase Central またはサーバー管理要求を使用して設定できます。サーバープロパティは、常にデータベースには保存されます。 次の項を参照してください。

- 「サーバー管理要求でのサーバープロパティの設定」168 ページ
- 「Sybase Central でのサーバープロパティの設定」770 ページ

サーバープロパティ

- **ianywhere.qa.server.autoRulesEvaluationPeriod**　メッセージの転送ルールと持続性ルールを含む、ルールの評価時間 (ミリ秒単位)。一般に、ルールは、サーバーストアへのメッセージの転送中に動的に評価されるため、ルール評価時間は、時間の影響を大きく受けるルールのみが対象です。デフォルト値は **60000** (1 分) です。

- **ianywhere.qa.server.compressionLevel**　QAnywhere コネクターが受信する各メッセージに適用される、デフォルトの圧縮量。圧縮は 0 ～ 9 の範囲の整数で表されます。0 は圧縮がない状態で、9 は最も圧縮された状態です。デフォルトは **0** です。

  コネクターの圧縮レベルをコネクタープロパティファイルにも設定した場合は、その設定が、該当のコネクターに対して優先されます。 「JMS コネクタープロパティの設定」138 ページを参照してください。

- **ianywhere.qa.server.connectorPropertiesFiles**

  > **注意**
  > Sybase Central に置き換えられています。

  QAnywhere コネクターの設定を、JMS などの外部のメッセージシステムに対して指定する、1 つ以上のファイルのリスト。デフォルトは、コネクターなしです。

  「コネクター」135 ページを参照してください。

- **ianywhere.qa.server.disableNotifications**　このプロパティを true に設定すると、保留中のメッセージのメッセージに関するサーバーからの通知が無効になります。これにより、クライアントを待機するメッセージがサーバー上にある場合、クライアントへの通知を開始するように要求するサーバー上の処理は無効になります。ファイアウォールの制限によって通知が不可能である場合など、サーバーから通知を送信できないときに、このプロパティを true に設定します。デフォルトは false です。

- **ianywhere.qa.server.logLevel**　メッセージング機能のログレベル。プロパティ値は、1、2、3、4 のいずれかです。1 は、メッセージエラーのみを記録することを指定します。2 は、さらに警告も記録することを指定します。3 は、さらに情報メッセージも記録することを指定します。4 は、Mobile Link サーバーで転送される各 QAnywhere メッセージの詳細を含めて、さらに詳細な情報メッセージも記録することを指定します。デフォルトは **2** です。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

これらのログメッセージは、Mobile Link サーバーメッセージウィンドウに出力されます。mlsrv12 の -o オプションまたは -ot オプションが指定されている場合、メッセージは Mobile Link サーバーのメッセージログファイルに出力されます。

- **ianywhere.qa.server.id**　サーバー管理要求の送信先アドレスのエージェント部分を指定します。このプロパティを設定しなかった場合、この値は ianywhere.server になります。

- **ianywhere.qa.server.password.e**　サーバー管理要求の認証用パスワードを指定します。このプロパティが設定されていない場合、パスワードは QAnywhere です。

  「サーバー管理要求」153 ページを参照してください。

- **ianywhere.qa.server.scheduleDateFormat**　サーバー側の転送ルールで使用する日付形式を指定します。デフォルトの日付形式は yyyy-MM-dd です。

文字	日付のコンポーネント	例
y	年	1996
M	月	July
d	日	10

- **ianywhere.qa.server.scheduleTimeFormat**　サーバー側の転送ルールで使用する時間形式を指定します。デフォルトの時間形式は HH:mm:ss です。

文字	日付のコンポーネント	例
a	AM/PM マーカー	PM
H	1 日の特定の時刻 (0 ～ 23 の値)	0
k	1 日の特定の時刻 (1 ～ 24 の値)	24
K	午前／午後の特定の時刻 (0 ～ 11 の値)	0
h	午前／午後の特定の時刻 (1 ～ 12 の値)	12
m	分	30
s	秒	55

- **ianywhere.qa.server.transmissionRulesFile**

> **注意**
> Sybase Central に置き換えられています。

メッセージの転送と持続性に関するルールを指定するファイル。デフォルトでは、メッセージのフィルターはなく、メッセージの最終ステータスがメッセージの発信者に転送されると、メッセージは削除されます。

Sybase Central でのサーバープロパティの設定

♦ Sybase Central でのサーバープロパティの設定

1. Sybase Central を起動します。

   [スタート] » **[プログラム]** » **[SQL Anywhere 12]** » **[管理ツール]** » **[Sybase Central]** をクリックします。

2. **[接続]** » **[QAnywhere 12 に接続]** をクリックします。

3. **[ODBC データソース名]** または **[ODBC データソースファイル]** を指定し、必要に応じて **[ユーザー ID]** と **[パスワード]** を指定します。**[OK]** をクリックします。

4. 左ウィンドウ枠の **[サーバーメッセージストア]** の下で、データソースの名前をクリックします。

5. **[ファイル]** » **[プロパティ]** をクリックします。

JMS コネクタープロパティ

JMS コネクターを設定するには、次のプロパティを使用します。

- **ianywhere.connector.nativeConnection**　JMS コネクターを実装する Java クラスを指定します。QAnywhere の内部使用のためのものである場合は、削除や変更はできません。

- **ianywhere.connector.id (旧式)**　JMS コネクターをユニークに識別する識別子を指定します。デフォルトは、コネクタープロパティ ianywhere.connector.address の値です。

- **ianywhere.connector.address**　コネクター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、コネクターを指定するために使用されます。指定したアドレスは、コネクターの Mobile Link サーバーメッセージウィンドウに表示されるすべてのログ出力エラー、警告、情報メッセージの先頭に付けられます。

  「JMS コネクターへの QAnywhere メッセージの送信」139 ページを参照してください。

  Sybase Central では、このプロパティは**コネクターウィザード**の **[コネクター名]** ページにある **[コネクター名]** フィールドで設定します。

- **ianywhere.connector.incoming.priority**　すべての受信メッセージに割り当てられている優先度を、整数で指定します。この値が指定されていないか負の数である場合、そのコネクター

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

のタイプでデフォルトの値が指定されます。JMS では、JMS メッセージの優先度がデフォルトで使用されます。Web サービスでは、デフォルトは 4 です。

- **ianywhere.connector.incoming.retry.max**　コネクターで QAnywhere のメッセージストアへの JMS メッセージの転送をリトライする最大回数を指定します。この回数に達すると、JMS メッセージは ianywhere.connector.jms.deadMessageDestination プロパティの値に再転送されます。デフォルトは -1 であり、コネクターのリトライ回数は無制限であることを意味します。

- **ianywhere.connector.incoming.ttl**　すべての受信メッセージに割り当てられている存続期間 (ミリ秒単位) を、整数で指定します。この値が指定されていないか負の数である場合、そのコネクターのタイプでデフォルトの値が指定されます。値が 0 の場合、メッセージは期限切れになりません。JMS では、JMS メッセージの有効期限を使用してデフォルト値が計算されます。Web サービスでは、デフォルトは 0 です。

- **ianywhere.connector.outgoing.deadMessageAddress**　処理できないメッセージの送信先アドレスを指定します。たとえば、間違った形式の JMS アドレスや不明な JMS アドレスがメッセージに含まれている場合、そのメッセージは受信不可とマーク付けされて、メッセージのコピーがこのアドレスに送信されます。

  このアドレスが指定されていない場合は、メッセージは受信不可とマーク付けされますが、メッセージのコピーは送信されません。

  Sybase Central では、このプロパティは **[コネクタープロパティ]** ウィンドウの **[プロパティ]** タブで、**[新規]** をクリックして設定します。

- **ianywhere.connector.logLevel**　Mobile Link サーバーメッセージウィンドウとメッセージログファイルに出力されるコネクター情報の詳細度を指定します。次の詳細度レベルがあります。

  - **1**　エラーメッセージを出力します。
  - **2**　エラーメッセージと警告メッセージを出力します。
  - **3**　エラー、警告、情報の各メッセージを出力します。
  - **4**　エラー、警告、情報、デバッグのメッセージを出力します。

  Sybase Central では、このプロパティは **[コネクタープロパティ]** ウィンドウの **[一般]** タブにある **[ロギングレベル]** セクションで設定します。

  すべてのコネクターに対して、このプロパティを設定することもできます。Sybase Central でこれを行うには、サーバーメッセージストアに接続して、メッセージストアを右クリックし、**[プロパティ]** をクリックします。**[プロパティ]** ウィンドウを使用して、次の項目を設定できます。

- **ianywhere.connector.compressionLevel**　JMS から受信したメッセージのデフォルトのメッセージ圧縮率を指定します。0 ～ 9 の整数値で指定します。0 は圧縮なし、9 は最大の圧縮率を表します。

Sybase Central では、このプロパティは **[コネクタープロパティ]** ウィンドウの **[一般]** タブにある **[圧縮レベル]** セクションで設定します。

すべてのコネクターに対して、このプロパティを設定することもできます。Sybase Central でこれを行うには、サーバーメッセージストアに接続し、**[このメッセージストアのプロパティの変更]** をクリックして、**[サーバーのプロパティ]** タブを開きます。

- **ianywhere.connector.jms.deadMessageDestination** JMS メッセージを QAnywhere メッセージに変換できない場合に、JMS メッセージが送信されるアドレスを指定します。JMS メッセージがサポートされないクラスのインスタンスである場合、JMS メッセージに QAnywhere のアドレスが指定されていない場合、予期しない JMS プロバイダー例外が発生した場合、または予期しない QAnywhere 例外が発生した場合には、JMS メッセージを QAnywhere メッセージに変更できません。

  Sybase Central では、このプロパティは **[コネクタープロパティ]** ウィンドウの **[JMS]** タブにある **[その他]** セクションで、**[Dead メッセージの送信先]** フィールドを使用して設定します。

- **ianywhere.connector.outgoing.retry.max** QAnywhere から外部メッセージングシステムへの出力メッセージについての、デフォルトのリトライ回数です。デフォルト値は 5 です。コネクターのリトライ回数を無制限にするには 0 を指定します。

  Sybase Central では、このプロパティは **[コネクタープロパティ]** ウィンドウの **[プロパティ]** タブで、**[新規]** をクリックして設定します。

- **ianywhere.connector.runtimeError.retry.max** RuntimeException を引き起こすメッセージの処理をコネクターがリトライする回数を指定します。dead になっているメッセージキューが指定されている場合は、メッセージはそのキューに登録されます。それ以外の場合は、メッセージは受信不可とマーク付けされてスキップされます。サーバーのリトライ回数を無制限にするには、この値に 0 を指定します。

- **ianywhere.connector.startupType** 起動のタイプには、automatic、manual、disabled のいずれかを指定できます。

- **xjms.jndi.authName** 外部の JMS JNDI ネームサービスに接続するときに使用する認証名を指定します。

  Sybase Central では、このプロパティは**コネクターウィザード**の **[JNDI の設定]** ページにある **[ユーザー名]** フィールド、または **[コネクタープロパティ]** ウィンドウの **[JMS]** タブにある **[JNDI]** セクションで **[ユーザー名]** フィールドを使用して設定します。

- **xjms.jndi.factory** 外部の JMS JNDI ネームサービスにアクセスするときに使用するファクトリ名を指定します。Sybase Central では、このプロパティは**コネクターウィザード**の **[JNDI の設定]** ページにある **[JNDI ファクトリ]** フィールド、または **[コネクタープロパティ]** ウィンドウの **[JMS]** タブにある **[JNDI]** セクションで **[JNDI ファクトリ]** フィールドを使用して設定します。

- **xjms.jndi.password.e** 外部の JMS JNDI ネームサービスに接続するときに使用する認証パスワードを指定します。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

Sybase Central では、このプロパティは**コネクターウィザード**の **[JNDI の設定]** ページにある **[パスワード]** フィールド、または **[コネクタープロパティ]** ウィンドウの **[JMS]** タブにある **[JNDI]** セクションで **[パスワード]** フィールドを使用して設定します。

- **xjms.jndi.url** JMS JNDI ネームサービスにアクセスするときに使用する URL を指定します。

  Sybase Central では、このプロパティは**コネクターウィザード**の **[JNDI の設定]** ページにある **[ネームサービス URL]** フィールド、または **[コネクタープロパティ]** ウィンドウの **[JMS]** タブにある **[JNDI]** セクションで **[URL]** フィールドを使用して設定します。

- **xjms.password.e** 外部の JMS プロバイダーに接続するときに使用する認証パスワードを指定します。

- **xjms.queueConnectionAuthName** 外部 JMS キュー接続に接続するときに使用するユーザー ID を指定します。

  Sybase Central では、このプロパティは**コネクターウィザード**の **[JMS キューの設定]** ページにある **[ユーザー名]** フィールド、または **[コネクタープロパティ]** ウィンドウの **[JMS]** タブにある **[キュー]** セクションで **[ユーザー名]** フィールドを使用して設定します。

- **xjms.queueConnectionPassword.e** 外部 JMS キュー接続に接続するときに使用するパスワードを指定します。

  Sybase Central では、このプロパティは**コネクターウィザード**の **[JMS キューの設定]** ページにある **[パスワード]** フィールド、または **[コネクタープロパティ]** ウィンドウの **[JMS]** タブにある **[キュー]** セクションで **[パスワード]** フィールドを使用して設定します。

- **xjms.queueFactory** 外部 JMS プロバイダーのキューファクトリ名を指定します。

  Sybase Central では、このプロパティは**コネクターウィザード**の **[JMS キューの設定]** ページにある **[キューファクトリ]** フィールド、または **[コネクタープロパティ]** ウィンドウの **[JMS]** タブにある **[キュー]** セクションで **[キューファクトリ]** フィールドを使用して設定します。

- **xjms.receiveDestination** JMS からの QAnywhere クライアント宛てのメッセージを受信するコネクターが使用するキュー名を指定します。

  Sybase Central では、このプロパティは**コネクターウィザード**の **[コネクター名]** ページにある **[受信側]** フィールドで設定します。

- **xjms.topicFactory** 外部 JMS プロバイダーのトピックファクトリ名を指定します。

  Sybase Central では、このプロパティは**コネクターウィザード**の **[JMS トピックの設定]** ページにある **[トピックファクトリ]** フィールド、または **[コネクタープロパティ]** ウィンドウの **[JMS]** タブにある **[トピック]** セクションで **[トピックファクトリ]** フィールドを使用して設定します。

- **xjms.topicConnectionAuthName** 外部 JMS トピック接続に接続するときに使用するユーザー ID を指定します。

Sybase Central では、このプロパティは**コネクターウィザード**の **[JMS トピックの設定]** ページにある **[ユーザー名]** フィールド、または **[コネクタープロパティ]** ウィンドウの **[JMS]** タブにある **[トピック]** セクションで **[ユーザー名]** フィールドを使用して設定します。

- **xjms.topicConnectionPassword.e**　外部 JMS トピック接続に接続するときに使用するパスワードを指定します。

  Sybase Central では、このプロパティは**コネクターウィザード**の **[JMS トピックの設定]** ページにある **[パスワード]** フィールド、または **[コネクタープロパティ]** ウィンドウの **[JMS]** タブにある **[トピック]** セクションで **[パスワード]** フィールドを使用して設定します。

QAnywhere 転送ルールと削除ルール

ルールの構文

ルールの構文

各ルールの形式は次のとおりです。

schedules=*condition*

スケジュール構文

スケジュール構文

schedules : { **AUTOMATIC** | *schedule-spec* ,... }

schedule-spec :
{ **START TIME** *start-time* | **BETWEEN** *start-time* **AND** *end-time* }
[**EVERY** *period* { **HOURS** | **MINUTES** | **SECONDS** }]
[**ON** { (*day-of-week*, ...) | (*day-of-month*, ...) }]
[**START DATE** *start-date*]

パラメーター

- **AUTOMATIC**　転送ルールでは、メッセージのステータスが変更されたり、ネットワークステータスに変更があった場合に、ルールが評価されます。削除ルールでは、メッセージ転送が開始されたときに、削除ルール条件を満たすメッセージが削除されます。

- **schedule-spec**　AUTOMATIC 以外のスケジュール仕様には、条件が評価される時刻を指定します。指定された時刻になると、対応する条件が評価されます。

- **START TIME**　イベントがスケジュールされた各日の最初のスケジュール時刻。START DATE を指定した場合、START TIME はその日付を参照します。START DATE を指定しない場合、START TIME の対象となる日付は、現在の日付 (指定された時刻を過ぎていない場合) とそれ以降の各日 (スケジュールに EVERY または ON が含まれる場合) となります。

- **BETWEEN ...AND ...**　1 日のうち、スケジュールされた時刻が発生する範囲。START DATE を指定した場合、スケジュールされた時刻はその日が来るまで発生しません。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

- **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、サーバー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。AND で指定された範囲の始まりを確認します。そうであれば、それが次のスケジュールされた時刻となります。

2. 次のスケジュールされた時刻が現在の日付にない場合は、イベントが実行される次の日付を調べます。

3. その日付の START TIME、または BETWEEN ...AND で指定された範囲の始まりを確認します。

QAnywhere スケジュール構文は、SQL Anywhere CREATE EVENT スケジュール構文に由来します。

キーワードは、大文字と小文字を区別しません。

参照

- 「CREATE EVENT 文」『SQL Anywhere サーバー SQL リファレンス』

例

次の例に示すサーバー側の転送ルールファイルは、sample_store_id というクライアントメッセージストア ID で識別されるクライアントに適用されます。この転送ルールファイルでは、二重のスケジュールを作成します。まず、優先度の高いメッセージが 1 時間に 1 回送信されます。スケジュールは every 1 hours (1 時間ごと) に設定されており、条件は ias_priority=9 です。さらに、午前 8 時から 9 時までの間は、優先度の高いメッセージが 1 分ごとに送信されます。

```
[sample_store_id]
; This rule governs when messages are transmitted to the client
; store with id sample_store_id.
;
  START TIME '06:00:00' EVERY 1 hours = ias_Priority = 9
  BETWEEN '08:00:00' AND '09:00:00' EVERY 1 minutes = ias_Priority = 9
```

条件構文

QAnywhere 条件は、SQL の構文に似ています。条件の評価対象は、メッセージストア内のメッセージです。条件は、true、false、unknown のいずれかに評価されます。条件が空の場合は、すべてのメッセージが条件を満たすものと判定されます。条件は、転送ルール、削除ルール、QAnywhere プログラミング API で使用できます。

キーワードと文字列比較では、大文字と小文字を区別しません。

構文

condition :
expression **IS** [**NOT**] **NULL**
| *expression compare expression*
| *expression* [**NOT**] **BETWEEN** *expression* **AND** *expression*
| *expression* [**NOT**] **LIKE** *pattern* [**ESCAPE** *character*]
| *expression* [**NOT**] **IN** (*string*, ...)
| **NOT** *condition*
| *condition* **AND** *condition*
| *condition* **OR** *condition*
| (*condition*)

compare: **=** |**>** | **<** | **>=** | **<=** | **<>**

expression:
constant
| *arithmetic-expression*
| *rule-variable*
| (*expression*)
| | *rule-function* (*expression*, ...)

constant: *integer* | *floating-point number* | *string* | *boolean*

arithmetic-expression:
numeric-constant
| *rule-variable*

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

| - *arithmetic-expression*
| *arithmetic-expression operator arithmetic-expression*
| *(arithmetic-expression)*
| *rule-function (expression, ...)*

numeric-constant: *integer* | *floating-point number*

整数: -2**63 ～ 2**63-1 の範囲内の整数です。

浮動小数点数: 2.2250738585072e-308 ～ 1.79769313486231e+308 の範囲内の指数表記の数です。

文字列: 一重引用符で囲まれた一連の文字です。文字列内の一重引用符を表すには、一重引用符を 2 つ続けて使用します。

ブール値: 文が **TRUE** または **FALSE**、**T** または **F**、**Y** または **N**、**1** または **0** であるもの。

operator: **+** | **-** | ***** | **/**

rule-variable:

「ルール変数」780 ページを参照してください。

rule-function:

「ルール関数」779 ページを参照してください。

パラメーター

- **BETWEEN**　BETWEEN 条件は、true、false、または unknown として評価できます。NOT キーワードがない場合、*expression* が start expression と end expression の間にあれば、条件は true と評価されます。

  NOT キーワードを使用すると条件の意味が逆になりますが、UNKNOWN は変わりません。

  BETWEEN 条件は、次の 2 つの不等式の組み合わせと等価です。

  [NOT] (*expression* **>=** *start-expression* **AND** *arithmetic-expression* **<=** *end-expr* **)**

  次に例を示します。

  ○age BETWEEN 15 AND 19 この文は次の文と同義です。 age >=15 AND age <= 19

  ○age NOT BETWEEN 15 AND 19 は age < 15 OR age > 19 と同義です。

- **IN**　IN 条件は、次の規則に従って評価されます。

  ○*expression* が NULL でなく、リスト内の少なくとも 1 つと等しい場合、true です。

  ○*expression* が NULL で値リストが空でない場合、または少なくとも 1 つの値が NULL で、expression が他の値のいずれとも等しくない場合、unknown です。

  ○いずれの値も NULL でなく、*expression* がリスト内のいずれの値とも等しくない場合、false です。

NOT キーワードを指定すると、評価結果の true と false が逆になります。

次に例を示します。

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

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

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

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

● **LIKE**　LIKE 条件は、true、false、または unknown として評価できます。

NOT キーワードがない場合、*expression* が *like expression* に一致すれば、条件は true と評価されます。*expression* または *like expression* が NULL 値の場合、この条件は unknown です。

NOT キーワードを使用すると条件の意味が逆になりますが、unknown は変わりません。

like expression には、任意の数のワイルドカードを指定できます。次のワイルドカードを使用できます。

ワイルドカード	一致するもの
_ (アンダースコア)	任意の 1 文字
% (パーセント記号)	0 個以上の文字からなる任意の文字列

次に例を示します。

○phone LIKE 12%3 は、'123' または '12993' の場合は true、'1234' の場合は false になります。

○word LIKE 's_d' は、'sad' の場合は true、'said' の場合は false になります。

○phone NOT LIKE '12%3' は、'123' または '12993' の場合は false、'1234' の場合は true になります。

● **エスケープ文字**　エスケープ文字は、*pattern* 内のワイルドカード文字 (_、**%**) の特殊な意味をなくすために使用される 1 文字リテラルです。次に例を示します。

○underscored LIKE '¥_%' ESCAPE '¥' は、'_myvar' の場合は true、'myvar' の場合は false になります。

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

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

ルール関数

転送ルールでは次の関数を使用できます。

構文	説明
DATEADD(*datepart*, *count*, *datetime* **)**	日時にいくつかの日時の単位を加算した日時を返します。*datepart* は、year、quarter、month、week、day、hour、minute、second のいずれかです。たとえば、次の例では 2 か月が追加されて、値 2006-07-02 00:00:00.0 が返されます。 DATEADD(month, 2, '2006/05/02')
DATEPART(*datepart*, *date* **)**	日時の値の一部の値を返します。datepart は、year、quarter、month、week、day、dayofyear、weekday、hour、minute、second のいずれかです。たとえば、次の例では 5 月を数値で取得するため、値 5 が返されます。 DATEPART(month, '2006/05/02')
DATETIME(*string* **)**	文字列値を日時に変換します。文字列のフォーマットは 'yyyy-mm-dd hh:nn:ss' であることが必要です。
LENGTH(*string* **)**	文字列の文字数を返します。
SUBSTRING(*string*, *start*, *length* **)**	文字列の部分文字列を返します。*start* は、返される部分文字列が開始する位置を文字単位で指定します。*length* は、返される部分文字列の長さを文字単位で指定します。

例

次の削除ルールは、最終ステータスになってから 11 日以上経過したメッセージをすべて削除します。

```
START TIME '06:00:00' every 1 hours = ias_Status >= ias_FinalState ¥
  AND ias_StatusTime < DATEADD( day, -10, ias_CurrentTimestamp) ¥
  AND ias_TransmissionStatus = ias_Transmitted
```

ルール変数

QAnywhere ルール変数は、ルールの条件部分で使用できます。ルール変数として使用できるのは次のとおりです。

- 「メッセージプロパティ」
- 「クライアントメッセージストアプロパティ」
- 「ルールエンジンで定義される変数」

ルール変数としてのプロパティの使用

メッセージプロパティとメッセージストアプロパティを転送ルール変数として使用できます。どちらの場合も、事前に定義されたプロパティを使用したり、カスタムプロパティを作成したりできます。同じ名前のメッセージプロパティとメッセージストアプロパティがある場合は、メッセージプロパティが使用されます。この優先順位を無効にするには、次のようにしてプロパティを明示的に参照します。

- メッセージストアプロパティ名に ias_Store というプレフィクスを付けます。
- メッセージプロパティ名に ias_Message というプレフィクスを付けます。

たとえば、次の自動転送ルールでは、**urgent** というカスタムメッセージプロパティが **yes** に設定されているすべてのメッセージが選択されます。

```
automatic = ias_Message.urgent = 'yes'
```

次の自動転送ルールでは、カスタムメッセージストアプロパティ **transmitNow** が **yes** に設定されたときにメッセージが選択されます。

```
automatic = ias_Store.transmitNow = 'yes'
```

ルールエンジンで定義される変数

次の変数はルールエンジンで定義されています。

- **ias_Address**　メッセージのアドレス。myclient¥myqueue など。
- **ias_ContentSize**　メッセージのサイズ。テキストメッセージの場合は文字数、バイナリメッセージの場合はバイト数。
- **ias_ContentType**　次のメッセージタイプ。

IAS_TEXT_CONTENT	メッセージの内容は、Unicode 文字で構成されます。
IAS_BINARY_CONTENT	メッセージの内容は、未解釈のバイトシーケンスとして扱われます。

- **ias_CurrentDate**　現在の日付。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

次のどちらかの方法で指定された文字列を、ias_currentDate と比較できます。

○明確に解釈される形式の文字列。

○クライアントメッセージストアデータベース用に設定された date_format オプションに従った文字列。

「データベースオプション」『SQL Anywhere サーバー データベース管理』と「date_format オプション」『SQL Anywhere サーバー データベース管理』を参照してください。

- **ias_CurrentTime**　現在の時刻。

  ias_CurrentTime と比較できるのは、時、分、秒がコロンとピリオドで区切られた形式 (hh:mm:ss.sss) で指定された文字列です。**am** または **pm** を指定しないと、24 時間表記が使用されます。「time_format オプション」『SQL Anywhere サーバー データベース管理』を参照してください。

- **ias_CurrentTimestamp**　現在のタイムスタンプ (現在の日付と時刻)。「time_format オプション」『SQL Anywhere サーバー データベース管理』を参照してください。

- **ias_Expires**　メッセージが配信されない場合に期限切れとなる日付と時刻。

- **ias_Network**　現在使用中のネットワークに関する情報。ias_Network は特殊な転送ルール変数です。このプロパティには、デバイスで使用されている現在のネットワークに関する情報を提供する多数の組み込み属性が定義されています。

- **ias_Priority**　0 ～ 9 の整数で指定したメッセージの優先度。0 は優先度が最も低く、9 は優先度が高いことを表します。

- **ias_Status**　メッセージの現在のステータス。次のいずれかの値を取ります。

IAS_CANCELLED_STATE	メッセージはキャンセルされました。
IAS_EXPIRED_STATE	メッセージが宛先に受信される前に期限切れになりました。
IAS_FINAL_STATE	メッセージが受信されたか、期限切れになりました。したがって、>=IAS_FINAL_STATE であれば、メッセージが受信されたか期限切れであることを意味し、<IAS_FINAL_STATE であれば、メッセージが受信もされず、期限切れでもないことを意味します。
IAS_PENDING_STATE	メッセージが宛先に受信されていません。
IAS_RECEIVED_STATE	メッセージが宛先に受信されました。

IAS_UNRECEIVABLE_STATE	メッセージに間違った形式があるか、失敗した試行回数が多すぎたため、受信不可のマークが付けられています。

参照：

- MessageProperties.STATUS フィールド [QAnywhere .NET]197 ページ
- MessageProperties.STATUS 変数 [QAnywhere C++]384 ページ
- MessageProperties.STATUS 変数 [QAnywhere Java]503 ページ

● **ias_StatusTime**　メッセージが現在のステータスになった時刻。

参照：

- MessageProperties.STATUS_TIME フィールド [QAnywhere .NET]197 ページ
- MessageProperties.STATUS_TIME 変数 [QAnywhere C++]384 ページ
- MessageProperties.STATUS_TIME 変数 [QAnywhere Java]503 ページ

● **ias_TransmissionStatus**　メッセージの同期ステータス。次のいずれかの値を取ります。

IAS_UNTRANSMITTED	メッセージは、目的の受信側メッセージストアに転送されていません。
IAS_TRANSMITTED	メッセージは、目的の受信側メッセージストアに転送済みです。
IAS_DO_NOT_TRANSMIT	受信先と送信元のメッセージストアが同じであるため、転送は不要です。
IAS_TRANSMITTING	メッセージは目的の受信先に転送されましたが、まだ受信確認されていません。メッセージ転送が中断された可能性があるため、QAnywhere がメッセージを再転送する可能性があります。

参照：

- MessageProperties.TRANSMISSION_STATUS フィールド [QAnywhere .NET]198 ページ
- MessageProperties.TRANSMISSION_STATUS 変数 [QAnywhere C++]385 ページ
- MessageProperties.TRANSMISSION_STATUS 変数 [QAnywhere Java]504 ページ

参照

● クライアントストアプロパティの作成方法、転送ルールでの使用方法の詳細については、「カスタムのクライアントメッセージストアプロパティ属性の使用」763 ページを参照してください。

Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

メッセージ転送ルール

転送ルールは、サーバー側とクライアント側に作成できます。

参照

- 「クライアント側の転送ルール」783 ページ
- 「サーバー側の転送ルール」784 ページ

クライアント側の転送ルール

クライアント側の転送ルールは、クライアントからサーバーに転送されるメッセージの動作を管理します。クライアント側の転送ルールは QAnywhere Agent によって処理されます。

デフォルトでは、QAnywhere Agent は自動ポリシーを使用します。QAnywhere Agent の転送ポリシーとして転送ルールファイルを指定することによって、このデフォルトの動作を変更、カスタマイズできます。

次の qaagent コマンドライン (一部) は、QAnywhere Agent 用のルールファイルを指定する方法を表しています。

```
qaagent -policy myrules.txt ...
```

転送ルールファイルには、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。

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。

Sybase Central QAnywhere 12 プラグインを使用して、QAnywhere Agent ルールファイルを作成することもできます。

参照

- 「ルールの構文」774 ページ
- 「条件構文」776 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」49 ページ
- 「-policy qaagent オプション」727 ページ
- 「クライアント側の削除ルール」786 ページ
- 「メッセージの削除ルール」786 ページ

例

たとえば、次に示すクライアント側の転送ルールファイルでは、営業時間中はサイズが小さく、優先度が高いメッセージだけを転送し、営業時間外はどのメッセージでも転送するように指定しています。このルールは自動的 (automatic) に実行されます。つまり、条件が満たされると、即座にメッセージが転送されます。この例から分かるように、条件には、メッセージからの情報とそれ以外の情報 (現在時刻など) を指定できます。

```
automatic=(ias_ContentSize < 100000 AND ias_Priority > 7 ) ¥
 OR datepart(Weekday,ias_CurrentDate) in ( 1, 7 ) ¥
 OR ias_CurrentTime < datetime('8:00:00') ¥
 OR ias_CurrentTime > datetime('18:00:00')
```

サーバー側の転送ルール

デフォルトルールの設定

特定のメッセージストアや送信先エイリアスに対してサーバー側の転送ルールを指定することも、すべてのクライアントに対してデフォルトルールを指定することもできます。明示的な転送ルールを使用するユーザーを除き、すべてのユーザーがデフォルトルールを使用します。

デフォルトルールを設定するには、ianywhere.server.defaultClient という特殊なクライアント名を使用します。

スケジュールされたサーバー側の転送ルール

スケジュールされたサーバー側の転送ルールを指定するときは、次の点に注意してください。

- クライアントに対する自動ルールの評価は、そのクライアントが同期されるたび、および自動ルールの評価時に行われる。
- クライアントに対するスケジュールされたルールの評価は、指定されたスケジュールで行われる。
- ルールの評価により、現在ルール条件を満たしているメッセージを持つクライアントに対して、Push 通知が送信される。
- クライアントが同期されるたびに、そのクライアントの自動ルールの条件を満たすメッセージがクライアントに転送される。
- クライアントが前回同期された後でスケジュールされたルールによってそのクライアントに Push 通知が送信された場合に限り、次回の同期時にスケジュールされたルールの条件を満たすすべてのメッセージがクライアントに転送される。

転送ルールファイルによるサーバー側の転送ルールの指定 (旧式)

サーバー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。

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

特定のクライアントに対して転送ルールを指定するには、対象クライアントのクライアントメッセージストア ID を角カッコで囲んで各セクションの先頭に指定します。

サーバー側のデフォルトの転送ルールを作成して、すべてのユーザーに適用することができます。

デフォルトの転送ルールを指定するには、次の行を含むセクションを開始します。

```
[ianywhere.server.defaultClient]
```

新しい転送ルールを有効にするには、Mobile Link サーバーを再起動する必要があります。再起動が必要なのは、転送ルールファイルで指定された転送ルールを有効にする場合だけです。Sybase Central またはサーバー管理要求を使用して指定されたサーバー側の転送ルールは、指定後すぐに有効になります。

例

次に示したサーバー側の転送ルールファイルのセクションでは、優先度の高いメッセージだけを送信するデフォルトルールが作成されます。

```
[ianywhere.server.defaultClient]
auto = ias_Priority > 6
```

次の例で示すサーバー側の転送ルールファイルでは、sample_store_id というクライアントメッセージストア ID で識別されるクライアントだけにルールが適用されます。

```
[sample_store_id]
; This rule governs when messages are transmitted to the client
; store with id sample_store_id.
;
;   ias_Priority >= 7
;
; Messages with priority 7 or greater should always be
; transmitted.
;
;   ias_ContentSize < 100
;
; Small messages (messages less than 100 characters or
; bytes in size) should always be transmitted.
;
;   ias_CurrentTime < '8:00am' OR ias_CurrentTime > '6:00pm'
;
; Messages outside business hours should always be
; transmitted

auto = ias_Priority >= 7 OR ias_ContentSize < 100 ¥
   OR ias_CurrentTime < datetime('8:00:00') ¥
   OR ias_CurrentTime > datetime('18:00:00')
```

次の例では、qanywhere というクライアントメッセージストア ID で識別されるクライアントだけにルールが適用されます。

```
[qanywhere]
; This rule governs when messages are transmitted to the client
; store with id qanywhere.
;
;   tm_Subject not like '%non-business%'
;
```

```
; Messages with the property tm_Subject set to a value that
; includes the phrase 'non-business' should not be transmitted
;
;    ias_CurrentTime < '8:00:00' OR ias_CurrentTime > '18:00:00'
;
; Messages outside business hours should always be
; transmitted

auto = tm_Subject NOT LIKE '%non-business%' ¥
   OR ias_CurrentTime < datetime('8:00am') OR ias_CurrentTime > datetime('6:00pm')
```

参照

- メッセージングプロパティファイルの詳細については、「-m mlsrv12 オプション」『Mobile Link サーバー管理』を参照してください。
- サーバー側の削除ルールの詳細については、「サーバー側の削除ルール」 787 ページを参照してください。

メッセージの削除ルール

削除ルールは、クライアントメッセージストアとサーバーメッセージストアに格納されているメッセージの持続性を決定します。

デフォルトの動作

QAnywhere のメッセージは、有効期限を過ぎてもメッセージが送受信されていない場合に期限切れになります。メッセージが期限切れになると、デフォルトの削除ルールで削除されます。メッセージが 1 回以上受信されていても受信確認されなかった場合は、有効期限を過ぎても再受信できます。

クライアント側の削除ルール

デフォルトでは、メッセージのステータスが受信済み、失効、キャンセル済み、配信不可のいずれかで、最終ステータスがサーバーメッセージストアに転送済みである場合に、クライアントメッセージストアからメッセージが削除されます。しかし、デフォルトよりも早くメッセージを削除したい場合もあれば、デフォルトよりも長くメッセージを残したい場合もあります。それには、クライアント側の転送ルールファイル内に削除セクションを記述します。削除セクションの先頭に **[system:delete]** を指定する必要があります。

次に、クライアント側の転送ルールファイル内の削除ルールセクションの例を示します。

```
[system:delete]

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

 Copyright © 2012, iAnywhere Solutions, Inc. - SQL Anywhere 12.0.1

```
START TIME '1:00:00' ON ( 'Sunday' ) = ias_Status >= ias_FinalState
```

サーバー側の削除ルール

デフォルトでは、メッセージのステータスが受信済み、失効、キャンセル済み、配信不可のいずれかで、最終ステータスがメッセージの発信者に返送済みである場合に、サーバーメッセージストアからメッセージが削除されます。しかし、監査などのためにメッセージを通常よりも長く残したい場合があります。

サーバー側の削除ルールは、サーバーメッセージストアのすべてのメッセージに適用されます。

参照

- 「クライアント側の転送ルール」783 ページ
- 「サーバー側の転送ルール」784 ページ
- AcknowledgementMode 列挙体 [QAnywhere .NET]317 ページ
- AcknowledgementMode クラス [QAnywhere C++]375 ページ
- AcknowledgementMode インターフェイス [QAnywhere Java]495 ページ
- ias_Status の詳細については、「ルール変数」780 ページを参照してください。

索引

記号

.NET モバイル Web サービスアプリケーションの設定
説明, 98
@data オプション
QAnywhere Agent ユーティリティ (qaagent), 715
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 740
-cd オプション
QAnywhere Agent ユーティリティ (qaagent), 717
QAnywhere Agent ユーティリティ (qauagent), 742
-cr オプション
QAnywhere Agent ユーティリティ (qaagent), 717
QAnywhere Agent ユーティリティ (qauagent), 743
-c オプション
QAnywhere Agent ユーティリティ (qaagent), 715
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 741
-fd オプション
QAnywhere Agent ユーティリティ (qaagent), 717
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 743
-fr オプション
QAnywhere Agent ユーティリティ (qaagent), 718
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 744
-idl オプション
QAnywhere Agent ユーティリティ (qaagent), 720
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 745
-id オプション
QAnywhere Agent ユーティリティ (qaagent), 719
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 744
QAnywhere のエージェント停止ユーティリティ (qastop), 760
-iu オプション
QAnywhere Agent ユーティリティ (qaagent), 720
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 746
-lp オプション
QAnywhere Agent ユーティリティ (qaagent), 721
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 747
-mn オプション
QAnywhere Agent ユーティリティ (qaagent), 722
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 747
-mp オプション
QAnywhere Agent ユーティリティ (qaagent), 722
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 748
-mu オプション
QAnywhere Agent ユーティリティ (qaagent), 723
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 748
-on オプション
QAnywhere Agent ユーティリティ (qaagent), 724
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 749
-os オプション
QAnywhere Agent ユーティリティ (qaagent), 725
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 750
-ot オプション
QAnywhere Agent ユーティリティ (qaagent), 725
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 751
-o オプション
QAnywhere Agent ユーティリティ (qaagent), 723
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 749
-pc オプション

QAnywhere Agent ユーティリティ (qaagent), 726
-policy オプション
QAnywhere Agent ユーティリティ (qaagent), 727
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 751
-push オプション
QAnywhere Agent ユーティリティ (qaagent), 729
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 753
-qi オプション
QAnywhere Agent ユーティリティ (qaagent), 731
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 755
-q オプション
QAnywhere Agent ユーティリティ (qaagent), 730
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 755
-sil オプション
QAnywhere Agent ユーティリティ (qaagent), 733
-si オプション
QAnywhere Agent ユーティリティ (qaagent), 731
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 756
-sur オプション
QAnywhere Agent ユーティリティ (qaagent), 734
-su オプション
QAnywhere Agent ユーティリティ (qaagent), 733
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 757
-v オプション
QAnywhere Agent ユーティリティ (qaagent), 735
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 757
-wc オプション
QAnywhere のエージェント停止ユーティリティ (qastop), 760
-xd オプション
QAnywhere Agent ユーティリティ (qaagent), 737
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 759
-x オプション
QAnywhere Agent ユーティリティ (qaagent), 736
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 758

A

acknowledgeAll メソッド
QAManager インターフェイス [QAnywhere Java API], 541
QAManager クラス [QAnywhere C++ API], 416
AcknowledgeAll メソッド
QAManager インターフェイス [QAnywhere .NET API], 236
AcknowledgementMode インターフェイス [QAnywhere Java API]
EXPLICIT_ACKNOWLEDGEMENT 変数, 496
IMPLICIT_ACKNOWLEDGEMENT 変数, 496
TRANSACTIONAL 変数, 496
説明, 495
AcknowledgementMode クラス [QAnywhere C++ API]
EXPLICIT_ACKNOWLEDGEMENT 変数, 376
IMPLICIT_ACKNOWLEDGEMENT 変数, 376
TRANSACTIONAL 変数, 377
説明, 375
AcknowledgementMode 列挙体 [QAnywhere .NET API]
説明, 317
acknowledgeUntil メソッド
QAManager インターフェイス [QAnywhere Java API], 542
QAManager クラス [QAnywhere C++ API], 416
AcknowledgeUntil メソッド
QAManager インターフェイス [QAnywhere .NET API], 237
acknowledge メソッド
QAManager インターフェイス [QAnywhere Java API], 541
QAManager クラス [QAnywhere C++ API], 415
WSResult クラス [QAnywhere Java API], 635
Acknowledge メソッド

QAManager インターフェイス [QAnywhere .NET API], 235
WSResult クラス [QAnywhere .NET API], 344
ADAPTERS フィールド
MessageProperties クラス [QAnywhere .NET API], 192
ADAPTERS 変数
MessageProperties インターフェイス [QAnywhere Java API], 499
MessageProperties クラス [QAnywhere C++ API], 380
ADAPTER フィールド
MessageProperties クラス [QAnywhere .NET API], 191
ADAPTER 変数
MessageProperties インターフェイス [QAnywhere Java API], 499
MessageProperties クラス [QAnywhere C++ API], 379
Address プロパティ
QAMessage インターフェイス [QAnywhere .NET API], 301
Address メッセージヘッダー
QAnywhere のメッセージヘッダー, 695
ALL 変数
QueueDepthFilter インターフェイス [QAnywhere Java API], 615
QueueDepthFilter クラス [QAnywhere C++ API], 490
API
QAnywhere SQL API, 655
archived タグ
QAnywhere サーバー管理要求, 155
automatic ポリシー
QAnywhere Agent, 728
QAnywhere Ultra Light Agent, 752

B

beginEnumPropertyNames メソッド
QAMessage クラス [QAnywhere C++ API], 457
beginEnumStorePropertyNames メソッド
QAManagerBase クラス [QAnywhere C++ API], 423
BodyLength プロパティ
QABinaryMessage インターフェイス [QAnywhere .NET API], 218
browseClose メソッド
QAManagerBase クラス [QAnywhere C++ API], 423
browseMessagesByID メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 548
QAManagerBase クラス [QAnywhere C++ API], 424
BrowseMessagesByID メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 245
browseMessagesByQueue メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 549
QAManagerBase クラス [QAnywhere C++ API], 425
BrowseMessagesByQueue メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 246
browseMessagesBySelector メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 550
QAManagerBase クラス [QAnywhere C++ API], 425
BrowseMessagesBySelector メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 246
browseMessages メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 548
QAManagerBase クラス [QAnywhere C++ API], 423
BrowseMessages メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 243
browseNextMessage メソッド
QAManagerBase クラス [QAnywhere C++ API], 426

C

CANCELED 変数
StatusCodes インターフェイス [QAnywhere Java API], 617
StatusCodes クラス [QAnywhere C++ API], 492
CancelMessageRequest タグ
QAnywhere サーバー管理要求, 157
cancelMessage メソッド

QAManagerBase インターフェイス [QAnywhere Java API], 550
QAManagerBase クラス [QAnywhere C++ API], 427
CancelMessage メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 247
castToBinaryMessage メソッド
QAMessage クラス [QAnywhere C++ API], 458
castToTextMessage メソッド
QAMessage クラス [QAnywhere C++ API], 458
ClearBody メソッド
QAMessage インターフェイス [QAnywhere .NET API], 288
clearProperties メソッド
QAMessage インターフェイス [QAnywhere Java API], 585
QAMessage クラス [QAnywhere C++ API], 459
ClearProperties メソッド
QAMessage インターフェイス [QAnywhere .NET API], 288
clearRequestProperties メソッド
WSBase クラス [QAnywhere Java API], 623
ClearRequestProperties メソッド
WSBase クラス [QAnywhere .NET API], 327
ClientStatusRequest タグ
QAnywhere サーバー管理要求, 162
CloseConnector タグ
QAnywhere サーバー管理要求, 162
close メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 551
QAManagerBase クラス [QAnywhere C++ API], 427
Close メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 248
commit メソッド
QATransactionalManager インターフェイス [QAnywhere Java API], 613
QATransactionalManager クラス [QAnywhere C++ API], 488
Commit メソッド
QATransactionalManager インターフェイス [QAnywhere .NET API], 313
COMMON_ALREADY_OPEN_ERROR フィールド
QAException クラス [QAnywhere .NET API], 223
COMMON_ALREADY_OPEN_ERROR 変数
QAError クラス [QAnywhere C++ API], 405
QAException クラス [QAnywhere Java API], 530
COMMON_GET_INIT_FILE_ERROR フィールド
QAException クラス [QAnywhere .NET API], 223
COMMON_GET_INIT_FILE_ERROR 変数
QAError クラス [QAnywhere C++ API], 405
QAException クラス [QAnywhere Java API], 531
COMMON_GET_PROPERTY_ERROR フィールド
QAException クラス [QAnywhere .NET API], 223
COMMON_GET_PROPERTY_ERROR 変数
QAError クラス [QAnywhere C++ API], 405
QAException クラス [QAnywhere Java API], 531
COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG フィールド
QAException クラス [QAnywhere .NET API], 224
COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG 変数
QAError クラス [QAnywhere C++ API], 406
QAException クラス [QAnywhere Java API], 531
COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID フィールド
QAException クラス [QAnywhere .NET API], 224
COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID 変数
QAError クラス [QAnywhere C++ API], 406
QAException クラス [QAnywhere Java API], 532
COMMON_GETQUEUEDEPTH_ERROR フィールド
QAException クラス [QAnywhere .NET API], 224
COMMON_GETQUEUEDEPTH_ERROR 変数
QAError クラス [QAnywhere C++ API], 406
QAException クラス [QAnywhere Java API], 531
COMMON_INIT_ERROR フィールド
QAException クラス [QAnywhere .NET API], 225
COMMON_INIT_ERROR 変数
QAError クラス [QAnywhere C++ API], 406
QAException クラス [QAnywhere Java API], 532
COMMON_INIT_THREAD_ERROR フィールド

QAException クラス [QAnywhere .NET API], 225
COMMON_INIT_THREAD_ERROR 変数
QAError クラス [QAnywhere C++ API], 406
QAException クラス [QAnywhere Java API], 532
COMMON_INVALID_PROPERTY フィールド
QAException クラス [QAnywhere .NET API], 225
COMMON_INVALID_PROPERTY 変数
QAError クラス [QAnywhere C++ API], 407
QAException クラス [QAnywhere Java API], 532
COMMON_MSG_ACKNOWLEDGE_ERROR フィールド
QAException クラス [QAnywhere .NET API], 225
COMMON_MSG_ACKNOWLEDGE_ERROR 変数
QAError クラス [QAnywhere C++ API], 407
QAException クラス [QAnywhere Java API], 532
COMMON_MSG_CANCEL_ERROR_SENT フィールド
QAException クラス [QAnywhere .NET API], 226
COMMON_MSG_CANCEL_ERROR_SENT 変数
QAError クラス [QAnywhere C++ API], 407
QAException クラス [QAnywhere Java API], 533
COMMON_MSG_CANCEL_ERROR フィールド
QAException クラス [QAnywhere .NET API], 226
COMMON_MSG_CANCEL_ERROR 変数
QAError クラス [QAnywhere C++ API], 407
QAException クラス [QAnywhere Java API], 533
COMMON_MSG_NOT_WRITEABLE_ERROR フィールド
QAException クラス [QAnywhere .NET API], 226
COMMON_MSG_NOT_WRITEABLE_ERROR 変数
QAError クラス [QAnywhere C++ API], 407
QAException クラス [QAnywhere Java API], 533
COMMON_MSG_RETRIEVE_ERROR フィールド
QAException クラス [QAnywhere .NET API], 226
COMMON_MSG_RETRIEVE_ERROR 変数
QAError クラス [QAnywhere C++ API], 408
QAException クラス [QAnywhere Java API], 533
COMMON_MSG_STORE_ERROR フィールド
QAException クラス [QAnywhere .NET API], 227
COMMON_MSG_STORE_ERROR 変数
QAError クラス [QAnywhere C++ API], 408
QAException クラス [QAnywhere Java API], 533
COMMON_MSG_STORE_NOT_INITIALIZED フィールド
QAException クラス [QAnywhere .NET API], 227
COMMON_MSG_STORE_NOT_INITIALIZED 変数
QAError クラス [QAnywhere C++ API], 408
QAException クラス [QAnywhere Java API], 534
COMMON_MSG_STORE_TOO_LARGE フィールド
QAException クラス [QAnywhere .NET API], 227
COMMON_MSG_STORE_TOO_LARGE 変数
QAError クラス [QAnywhere C++ API], 408
QAException クラス [QAnywhere Java API], 534
COMMON_NO_DEST_ERROR フィールド
QAException クラス [QAnywhere .NET API], 227
COMMON_NO_DEST_ERROR 変数
QAError クラス [QAnywhere C++ API], 408
QAException クラス [QAnywhere Java API], 534
COMMON_NO_IMPLEMENTATION フィールド
QAException クラス [QAnywhere .NET API], 228
COMMON_NO_IMPLEMENTATION 変数
QAError クラス [QAnywhere C++ API], 408
QAException クラス [QAnywhere Java API], 534
COMMON_NOT_OPEN_ERROR フィールド
QAException クラス [QAnywhere .NET API], 228
COMMON_NOT_OPEN_ERROR 変数
QAError クラス [QAnywhere C++ API], 409
QAException クラス [QAnywhere Java API], 534
COMMON_OPEN_ERROR フィールド
QAException クラス [QAnywhere .NET API], 228
COMMON_OPEN_ERROR 変数
QAError クラス [QAnywhere C++ API], 409
QAException クラス [QAnywhere Java API], 535
COMMON_OPEN_LOG_FILE_ERROR フィールド
QAException クラス [QAnywhere .NET API], 228
COMMON_OPEN_LOG_FILE_ERROR 変数
QAError クラス [QAnywhere C++ API], 409

QAException クラス [QAnywhere Java API], 535
COMMON_OPEN_MAXTHREADS_ERROR フィールド
QAException クラス [QAnywhere .NET API], 229
COMMON_OPEN_MAXTHREADS_ERROR 変数
QAError クラス [QAnywhere C++ API], 409
QAException クラス [QAnywhere Java API], 535
COMMON_REOPEN_ERROR フィールド
QAException クラス [QAnywhere .NET API], 229
COMMON_REOPEN_ERROR 変数
QAException クラス [QAnywhere Java API], 535
COMMON_SELECTOR_SYNTAX_ERROR フィールド
QAException クラス [QAnywhere .NET API], 229
COMMON_SELECTOR_SYNTAX_ERROR 変数
QAError クラス [QAnywhere C++ API], 409
QAException クラス [QAnywhere Java API], 535
COMMON_SET_PROPERTY_ERROR フィールド
QAException クラス [QAnywhere .NET API], 229
COMMON_SET_PROPERTY_ERROR 変数
QAError クラス [QAnywhere C++ API], 410
QAException クラス [QAnywhere Java API], 535
COMMON_TERMINATE_ERROR フィールド
QAException クラス [QAnywhere .NET API], 230
COMMON_TERMINATE_ERROR 変数
QAError クラス [QAnywhere C++ API], 410
QAException クラス [QAnywhere Java API], 536
COMMON_UNEXPECTED_EOM_ERROR フィールド
QAException クラス [QAnywhere .NET API], 230
COMMON_UNEXPECTED_EOM_ERROR 変数
QAError クラス [QAnywhere C++ API], 410
QAException クラス [QAnywhere Java API], 536
COMMON_UNREPRESENTABLE_TIMESTAMP フィールド
QAException クラス [QAnywhere .NET API], 230
COMMON_UNREPRESENTABLE_TIMESTAMP 変数
QAError クラス [QAnywhere C++ API], 410
QAException クラス [QAnywhere Java API], 536
COMPRESSION_LEVEL プロパティ
QAnywhere Manager の設定プロパティ, 85
condition タグ
QAnywhere サーバー管理要求, 704
CONNECT_PARAMS プロパティ
QAnywhere Manager の設定プロパティ, 85
createBinaryMessage メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 551
QAManagerBase クラス [QAnywhere C++ API], 428
CreateBinaryMessage メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 248
createQAManager メソッド
QAManagerFactory クラス [QAnywhere C++ API], 451
QAManagerFactory クラス [QAnywhere Java API], 578
CreateQAManager メソッド
QAManagerFactory クラス [QAnywhere .NET API], 280
createQATransactionalManager メソッド
QAManagerFactory クラス [QAnywhere C++ API], 452
QAManagerFactory クラス [QAnywhere Java API], 580
CreateQATransactionalManager メソッド
QAManagerFactory クラス [QAnywhere .NET API], 282
createTextMessage メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 552
QAManagerBase クラス [QAnywhere C++ API], 428
CreateTextMessage メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 249
customrule タグ
QAnywhere サーバー管理要求, 705

D

DATABASE_TYPE プロパティ
QAnywhere Manager の設定プロパティ, 85
DATEADD 関数
QAnywhere 構文, 779
DATEPART 関数
QAnywhere 構文, 779

DATETIME 関数
QAnywhere 構文, 779
dbeng12
QAnywhere Agent, 48
dblsn ユーティリティ
QAnywhere Agent, 48
QAnywhere アーキテクチャー, 8
QAnywhere の設定, 37
dbmlsync ユーティリティ
QAnywhere Agent, 48
DEFAULT_PRIORITY 変数
QAMessage インターフェイス [QAnywhere Java API], 599
QAMessage クラス [QAnywhere C++ API], 477
DEFAULT_TIME_TO_LIVE 変数
QAMessage インターフェイス [QAnywhere Java API], 599
QAMessage クラス [QAnywhere C++ API], 477
deleteMessage メソッド
QAManagerBase クラス [QAnywhere C++ API], 429
deleteQAManager メソッド
QAManagerFactory クラス [QAnywhere C++ API], 452
deleteQATransactionalManager メソッド
QAManagerFactory クラス [QAnywhere C++ API], 453
DELIVERY_COUNT フィールド
MessageProperties クラス [QAnywhere .NET API], 193
DELIVERY_COUNT 変数
MessageProperties インターフェイス [QAnywhere Java API], 500
MessageProperties クラス [QAnywhere C++ API], 380
DetailedMessage プロパティ
QAException クラス [QAnywhere .NET API], 222
DTD
QAnywhere サーバー管理要求, 710

E

EAServer
QAnywhere, 9
endEnumPropertyNames メソッド
QAMessage クラス [QAnywhere C++ API], 459
endEnumStorePropertyNames メソッド
QAManagerBase クラス [QAnywhere C++ API], 429
ErrorCode プロパティ
QAException クラス [QAnywhere .NET API], 222
WSException クラス [QAnywhere .NET API], 335
ExceptionListener2 デリゲート [QAnywhere .NET API]
説明, 315
ExceptionListener デリゲート [QAnywhere .NET API]
説明, 315
Expiration プロパティ
QAMessage インターフェイス [QAnywhere .NET API], 301
Expiration メッセージヘッダー
QAnywhere のメッセージヘッダー, 695
EXPIRED 変数
StatusCodes インターフェイス [QAnywhere Java API], 617
StatusCodes クラス [QAnywhere C++ API], 492
EXPLICIT_ACKNOWLEDGEMENT 変数
AcknowledgementMode インターフェイス [QAnywhere Java API], 496
AcknowledgementMode クラス [QAnywhere C++ API], 376

F

FINAL 変数
StatusCodes インターフェイス [QAnywhere Java API], 617
StatusCodes クラス [QAnywhere C++ API], 492

G

getAddress メソッド
QAMessage インターフェイス [QAnywhere Java API], 585
QAMessage クラス [QAnywhere C++ API], 459
getAllQueueDepth メソッド
QAManagerBase クラス [QAnywhere C++ API], 429
getArrayValue メソッド
WSResult クラス [QAnywhere Java API], 635
GetArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 344
getBigDecimalArrayValue メソッド

WSResult クラス [QAnywhere Java API], 636
getBigDecimalValue メソッド
WSResult クラス [QAnywhere Java API], 636
getBigIntegerArrayValue メソッド
WSResult クラス [QAnywhere Java API], 636
getBigIntegerValue メソッド
WSResult クラス [QAnywhere Java API], 637
getBodyLength メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 514
QABinaryMessage クラス [QAnywhere C++ API], 394
GetBoolArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 345
getBooleanArrayValue メソッド
WSResult クラス [QAnywhere Java API], 637
GetBooleanArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 345
getBooleanProperty メソッド
QAMessage インターフェイス [QAnywhere Java API], 586
QAMessage クラス [QAnywhere C++ API], 460
GetBooleanProperty メソッド
QAMessage インターフェイス [QAnywhere .NET API], 288
getBooleanStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 552
QAManagerBase クラス [QAnywhere C++ API], 430
GetBooleanStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 249
getBooleanValue メソッド
WSResult クラス [QAnywhere Java API], 638
GetBooleanValue メソッド
WSResult クラス [QAnywhere .NET API], 346
GetBoolValue メソッド
WSResult クラス [QAnywhere .NET API], 346
getByteArrayValue メソッド
WSResult クラス [QAnywhere Java API], 638
GetByteArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 347
getByteProperty メソッド
QAMessage インターフェイス [QAnywhere Java API], 586
QAMessage クラス [QAnywhere C++ API], 460
GetByteProperty メソッド
QAMessage インターフェイス [QAnywhere .NET API], 289
getByteStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 553
QAManagerBase クラス [QAnywhere C++ API], 431
getByteValue メソッド
WSResult クラス [QAnywhere Java API], 638
GetByteValue メソッド
WSResult クラス [QAnywhere .NET API], 347
getCharacterArrayValue メソッド
WSResult クラス [QAnywhere Java API], 639
getCharacterValue メソッド
WSResult クラス [QAnywhere Java API], 639
GetCharArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 348
GetCharValue メソッド
WSResult クラス [QAnywhere .NET API], 348
GetDecimalArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 349
GetDecimalValue メソッド
WSResult クラス [QAnywhere .NET API], 349
getDetailedMessage メソッド
QAException クラス [QAnywhere Java API], 530
getDoubleArrayValue メソッド
WSResult クラス [QAnywhere Java API], 640
GetDoubleArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 350
getDoubleProperty メソッド
QAMessage インターフェイス [QAnywhere Java API], 587
QAMessage クラス [QAnywhere C++ API], 461
GetDoubleProperty メソッド
QAMessage インターフェイス [QAnywhere .NET API], 289
getDoubleStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 554
QAManagerBase クラス [QAnywhere C++ API], 431
GetDoubleStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 250
getDoubleValue メソッド
WSResult クラス [QAnywhere Java API], 640
GetDoubleValue メソッド
WSResult クラス [QAnywhere .NET API], 350

getErrorCode メソッド
QAException クラス [QAnywhere Java API], 530
WSException クラス [QAnywhere Java API], 629
getErrorMessage メソッド
WSResult クラス [QAnywhere Java API], 641
GetErrorMessage メソッド
WSResult クラス [QAnywhere .NET API], 351
getExpiration メソッド
QAMessage インターフェイス [QAnywhere Java API], 587
QAMessage クラス [QAnywhere C++ API], 461
getFloatArrayValue メソッド
WSResult クラス [QAnywhere Java API], 641
GetFloatArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 351
getFloatProperty メソッド
QAMessage インターフェイス [QAnywhere Java API], 588
QAMessage クラス [QAnywhere C++ API], 462
GetFloatProperty メソッド
QAMessage インターフェイス [QAnywhere .NET API], 290
getFloatStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 554
QAManagerBase クラス [QAnywhere C++ API], 432
GetFloatStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 251
getFloatValue メソッド
WSResult クラス [QAnywhere Java API], 641
GetFloatValue メソッド
WSResult クラス [QAnywhere .NET API], 351
getInReplyToID メソッド
QAMessage インターフェイス [QAnywhere Java API], 588
QAMessage クラス [QAnywhere C++ API], 462
getInstance メソッド
QAManagerFactory クラス [QAnywhere Java API], 582
GetInt16ArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 352
GetInt16Value メソッド
WSResult クラス [QAnywhere .NET API], 352
GetInt32ArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 353
GetInt32Value メソッド
WSResult クラス [QAnywhere .NET API], 353
GetInt64ArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 354
GetInt64Value メソッド
WSResult クラス [QAnywhere .NET API], 354
GetIntArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 355
getIntegerArrayValue メソッド
WSResult クラス [QAnywhere Java API], 642
getIntegerValue メソッド
WSResult クラス [QAnywhere Java API], 642
getIntProperty メソッド
QAMessage インターフェイス [QAnywhere Java API], 589
QAMessage クラス [QAnywhere C++ API], 463
GetIntProperty メソッド
QAMessage インターフェイス [QAnywhere .NET API], 291
getIntStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 555
QAManagerBase クラス [QAnywhere C++ API], 432
GetIntStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 252
GetIntValue メソッド
WSResult クラス [QAnywhere .NET API], 355
getLastErrorMsg メソッド
QAManagerBase クラス [QAnywhere C++ API], 433
QAManagerFactory クラス [QAnywhere C++ API], 453
getLastError メソッド
QAManagerBase クラス [QAnywhere C++ API], 433
QAManagerFactory クラス [QAnywhere C++ API], 453
getLastNativeError メソッド
QAManagerBase クラス [QAnywhere C++ API], 434
QAManagerFactory クラス [QAnywhere C++ API], 454
getLongArrayValue メソッド
WSResult クラス [QAnywhere Java API], 642
GetLongArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 356
getLongProperty メソッド

QAMessage インターフェイス [QAnywhere Java API], 589
QAMessage クラス [QAnywhere C++ API], 463
GetLongProperty メソッド
QAMessage インターフェイス [QAnywhere .NET API], 291
getLongStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 555
QAManagerBase クラス [QAnywhere C++ API], 434
GetLongStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 252
getLongValue メソッド
WSResult クラス [QAnywhere Java API], 643
GetLongValue メソッド
WSResult クラス [QAnywhere .NET API], 356
getMessageBySelectorNoWait メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 557
QAManagerBase クラス [QAnywhere C++ API], 436
GetMessageBySelectorNoWait メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 255
getMessageBySelectorTimeout メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 558
QAManagerBase クラス [QAnywhere C++ API], 436
GetMessageBySelectorTimeout メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 255
getMessageBySelector メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 557
QAManagerBase クラス [QAnywhere C++ API], 435
GetMessageBySelector メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 254
getMessageID メソッド
QAMessage インターフェイス [QAnywhere Java API], 590
QAMessage クラス [QAnywhere C++ API], 464
getMessageNoWait メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 559
QAManagerBase クラス [QAnywhere C++ API], 437
GetMessageNoWait メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 256
getMessageTimeout メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 560
QAManagerBase クラス [QAnywhere C++ API], 438
GetMessageTimeout メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 257
getMessage メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 556
QAManagerBase クラス [QAnywhere C++ API], 435
GetMessage メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 253
getMode メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 560
QAManagerBase クラス [QAnywhere C++ API], 438
getNativeErrorCode メソッド
QAException クラス [QAnywhere Java API], 530
GetNullableBoolArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 357
GetNullableBoolValue メソッド
WSResult クラス [QAnywhere .NET API], 357
GetNullableDecimalArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 358
GetNullableDecimalValue メソッド
WSResult クラス [QAnywhere .NET API], 358
GetNullableDoubleArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 359
GetNullableDoubleValue メソッド
WSResult クラス [QAnywhere .NET API], 359
GetNullableFloatArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 360
GetNullableFloatValue メソッド
WSResult クラス [QAnywhere .NET API], 360
GetNullableIntArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 361

GetNullableIntValue メソッド
WSResult クラス [QAnywhere .NET API], 361
GetNullableLongArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 362
GetNullableLongValue メソッド
WSResult クラス [QAnywhere .NET API], 362
GetNullableSByteArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 363
GetNullableSByteValue メソッド
WSResult クラス [QAnywhere .NET API], 363
GetNullableShortArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 364
GetNullableShortValue メソッド
WSResult クラス [QAnywhere .NET API], 364
getObjectArrayValue メソッド
WSResult クラス [QAnywhere Java API], 643
GetObjectArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 365
getObjectValue メソッド
WSResult クラス [QAnywhere Java API], 644
GetObjectValue メソッド
WSResult クラス [QAnywhere .NET API], 365
getPrimitiveBooleanArrayValue メソッド
WSResult クラス [QAnywhere Java API], 644
getPrimitiveBooleanValue メソッド
WSResult クラス [QAnywhere Java API], 644
getPrimitiveByteArrayValue メソッド
WSResult クラス [QAnywhere Java API], 645
getPrimitiveByteValue メソッド
WSResult クラス [QAnywhere Java API], 645
getPrimitiveCharArrayValue メソッド
WSResult クラス [QAnywhere Java API], 646
getPrimitiveCharValue メソッド
WSResult クラス [QAnywhere Java API], 646
getPrimitiveDoubleArrayValue メソッド
WSResult クラス [QAnywhere Java API], 646
getPrimitiveDoubleValue メソッド
WSResult クラス [QAnywhere Java API], 647
getPrimitiveFloatArrayValue メソッド
WSResult クラス [QAnywhere Java API], 647
getPrimitiveFloatValue メソッド
WSResult クラス [QAnywhere Java API], 648
getPrimitiveIntArrayValue メソッド
WSResult クラス [QAnywhere Java API], 648
getPrimitiveIntValue メソッド
WSResult クラス [QAnywhere Java API], 649
getPrimitiveLongArrayValue メソッド
WSResult クラス [QAnywhere Java API], 649
getPrimitiveLongValue メソッド
WSResult クラス [QAnywhere Java API], 649
getPrimitiveShortArrayValue メソッド
WSResult クラス [QAnywhere Java API], 650
getPrimitiveShortValue メソッド
WSResult クラス [QAnywhere Java API], 650
getPriority メソッド
QAMessage インターフェイス [QAnywhere Java API], 590
QAMessage クラス [QAnywhere C++ API], 464
getPropertyNames メソッド
QAMessage インターフェイス [QAnywhere Java API], 591
GetPropertyNames メソッド
QAMessage インターフェイス [QAnywhere .NET API], 292
getPropertyType メソッド
QAMessage インターフェイス [QAnywhere Java API], 591
QAMessage クラス [QAnywhere C++ API], 465
GetPropertyType メソッド
QAMessage インターフェイス [QAnywhere .NET API], 293
getProperty メソッド
QAMessage インターフェイス [QAnywhere Java API], 590
GetProperty メソッド
QAMessage インターフェイス [QAnywhere .NET API], 292
getQueueDepth メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 561
QAManagerBase クラス [QAnywhere C++ API], 439
GetQueueDepth メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 258
getRedelivered メソッド
QAMessage インターフェイス [QAnywhere Java API], 592
QAMessage クラス [QAnywhere C++ API], 465
getReplyToAddress メソッド
QAMessage インターフェイス [QAnywhere Java API], 592
QAMessage クラス [QAnywhere C++ API], 466
getRequestID メソッド
WSResult クラス [QAnywhere Java API], 651
GetRequestID メソッド

WSResult クラス [QAnywhere .NET API], 366
getResult メソッド
WSBase クラス [QAnywhere Java API], 623
GetResult メソッド
WSBase クラス [QAnywhere .NET API], 328
GetSByteArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 366
GetSbyteProperty メソッド
QAMessage インターフェイス [QAnywhere .NET API], 293
GetSbyteStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 260
GetSByteValue メソッド
WSResult クラス [QAnywhere .NET API], 367
getServiceID メソッド
WSBase クラス [QAnywhere Java API], 623
GetServiceID メソッド
WSBase クラス [QAnywhere .NET API], 328
getShortArrayValue メソッド
WSResult クラス [QAnywhere Java API], 651
GetShortArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 367
getShortProperty メソッド
QAMessage インターフェイス [QAnywhere Java API], 592
QAMessage クラス [QAnywhere C++ API], 466
GetShortProperty メソッド
QAMessage インターフェイス [QAnywhere .NET API], 294
getShortStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 562
QAManagerBase クラス [QAnywhere C++ API], 439
GetShortStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 260
getShortValue メソッド
WSResult クラス [QAnywhere Java API], 651
GetShortValue メソッド
WSResult クラス [QAnywhere .NET API], 368
GetSingleArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 368
GetSingleValue メソッド
WSResult クラス [QAnywhere .NET API], 369
getStatus メソッド
WSResult クラス [QAnywhere Java API], 652
GetStatus メソッド
WSResult クラス [QAnywhere .NET API], 369
getStorePropertyNames メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 564
GetStorePropertyNames メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 262
getStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 563
GetStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 261
getStringArrayValue メソッド
WSResult クラス [QAnywhere Java API], 652
GetStringArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 370
getStringProperty メソッド
QAMessage インターフェイス [QAnywhere Java API], 593
QAMessage クラス [QAnywhere C++ API], 467
GetStringProperty メソッド
QAMessage インターフェイス [QAnywhere .NET API], 294
getStringStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 564
QAManagerBase クラス [QAnywhere C++ API], 440
GetStringStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 262
getStringValue メソッド
WSResult クラス [QAnywhere Java API], 653
GetStringValue メソッド
WSResult クラス [QAnywhere .NET API], 370
getTextLength メソッド
QATextMessage インターフェイス [QAnywhere Java API], 606
QATextMessage クラス [QAnywhere C++ API], 482
getText メソッド
QATextMessage インターフェイス [QAnywhere Java API], 605
QATextMessage クラス [QAnywhere C++ API], 482
getTimestampAsString メソッド

QAMessage クラス [QAnywhere C++ API], 469
getTimestamp メソッド
QAMessage インターフェイス [QAnywhere Java API], 593
QAMessage クラス [QAnywhere C++ API], 468
GetUIntArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 371
GetUIntValue メソッド
WSResult クラス [QAnywhere .NET API], 371
GetULongArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 372
GetULongValue メソッド
WSResult クラス [QAnywhere .NET API], 372
GetUShortArrayValue メソッド
WSResult クラス [QAnywhere .NET API], 373
GetUShortValue メソッド
WSResult クラス [QAnywhere .NET API], 373
getValue メソッド
WSResult クラス [QAnywhere Java API], 653
GetValue メソッド
WSResult クラス [QAnywhere .NET API], 374

I

ianywhere.connector.address プロパティ
QAnywhere JMS コネクター, 770
QAnywhere Web サービスコネクター, 146
ianywhere.connector.compressionLevel プロパティ
QAnywhere JMS コネクター, 771
QAnywhere Web サービスコネクター, 146
ianywhere.connector.id プロパティ
QAnywhere JMS コネクター (旧式), 770
QAnywhere Web サービスコネクター (旧式), 146
ianywhere.connector.incoming.retry.max プロパティ
QAnywhere JMS コネクター, 771
ianywhere.connector.jms.deadMessageDestination プロパティ
QAnywhere JMS コネクター, 772
ianywhere.connector.logLevel プロパティ
QAnywhere JMS コネクター, 771
QAnywhere Web サービスコネクター, 146
ianywhere.connector.NativeConnection プロパティ
QAnywhere JMS コネクター, 770
QAnywhere Web サービスコネクター, 146
ianywhere.connector.outgoing.deadMessageAddress プロパティ
QAnywhere JMS コネクター, 771
ianywhere.connector.outgoing.retry.max プロパティ
QAnywhere JMS コネクター, 772
QAnywhere Web サービスコネクター, 147
ianywhere.connector.runtimeError.retry.max プロパティ
QAnywhere JMS コネクター, 772
ianywhere.connector.startupType プロパティ
QAnywhere JMS コネクター, 772
QAnywhere Web サービスコネクター, 147
ianywhere.qa.server.autoRulesEvaluationPeriod プロパティ
QAnywhere のサーバープロパティ, 768
ianywhere.qa.server.compressionLevel プロパティ
QAnywhere のサーバープロパティ, 768
ianywhere.qa.server.connectorPropertiesFile プロパティ
QAnywhere のサーバープロパティ, 768
ianywhere.qa.server.disableNotifications プロパティ
QAnywhere のサーバープロパティ, 768
ianywhere.qa.server.id プロパティ
QAnywhere のサーバープロパティ, 769
ianywhere.qa.server.logLevel プロパティ
QAnywhere のサーバープロパティ, 768
ianywhere.qa.server.password.e プロパティ
QAnywhere のサーバープロパティ, 769
ianywhere.qa.server.rules プロパティ
QAnywhere 転送ルール, 169
ianywhere.qa.server.scheduleDateFormat プロパティ
QAnywhere のサーバープロパティ, 769
ianywhere.qa.server.scheduleTimeFormat プロパティ
QAnywhere のサーバープロパティ, 769
ianywhere.qa.server.transmissionRulesFile プロパティ
QAnywhere のサーバープロパティ, 769
iAnywhere.QAnywhere.Client ネームスペース
QAnywhere .NET API, 189
ianywhere.qanywhere.client パッケージ
QAnywhere Java API, 495
iAnywhere.QAnywhere.StandAloneClient ネームスペース
QAnywhere .NET API, 189
ianywhere.qanywhere.standaloneclient パッケージ
QAnywhere Java API, 495
iAnywhere.QAnywhere.WS ネームスペース
QAnywhere .NET API, 325
ianywhere.qanywhere.ws パッケージ
QAnywhere Java API, 620

ianywhere.server.defaultClient
QAnywhere 転送ルール, 784
ias_Adapters
QAnywhere 事前に定義されたメッセージプロパティ, 698
QAnywhere ネットワークステータス通知, 62
QAnywhere のメッセージストアプロパティ, 761
ias_Address
QAnywhere 転送ルール変数, 780
ias_ContentSize
QAnywhere 転送ルール変数, 780
ias_ContentType
QAnywhere 転送ルール変数, 780
ias_CurrentDate
QAnywhere 転送ルール変数, 780
ias_CurrentTime
QAnywhere 転送ルール変数, 780
ias_CurrentTimestamp
QAnywhere 転送ルール変数, 780
ias_DeliveryCount
QAnywhere 事前に定義されたメッセージプロパティ, 698
ias_Expires
QAnywhere 転送ルール変数, 780
ias_ExpireState
QAnywhere 転送ルール変数, 780
ias_FinalState
QAnywhere 転送ルール変数, 780
ias_MaxDeliveryAttempts
QAnywhere 転送ルール変数, 780
QAnywhere のメッセージストアプロパティ, 761
ias_MaxDownloadSize
QAnywhere のメッセージストアプロパティ, 761
ias_MaxUploadSize
QAnywhere のメッセージストアプロパティ, 761
ias_MessageType
QAnywhere 事前に定義されたメッセージプロパティ, 699
ias_Network
QAnywhere 事前に定義されたメッセージプロパティ, 699
QAnywhere 転送ルール変数, 780
QAnywhere のプロパティ, 764
QAnywhere のメッセージストアプロパティ, 762
ias_Network.Adapter
QAnywhere 転送ルール変数, 780
QAnywhere のメッセージストアプロパティ, 761
ias_Network.IP
QAnywhere 転送ルール変数, 780
QAnywhere のメッセージストアプロパティ, 762
ias_Network.MAC
QAnywhere 転送ルール変数, 780
QAnywhere のメッセージストアプロパティ, 762
ias_Network.RAS
QAnywhere 転送ルール変数, 780
QAnywhere のメッセージストアプロパティ, 761
ias_NetworkStatus
QAnywhere 事前に定義されたメッセージプロパティ, 699
QAnywhere ネットワークステータス通知, 62
ias_Originator
QAnywhere 事前に定義されたメッセージプロパティ, 699
QAnywhere 転送ルール変数, 780
ias_PendingState
QAnywhere 転送ルール変数, 780
ias_Priority
QAnywhere 転送ルール変数, 780
ias_RASNames
QAnywhere ネットワークステータス通知, 62
ias_Received
QAnywhere 転送ルール変数, 780
ias_Status
QAnywhere 事前に定義されたメッセージプロパティ, 699
QAnywhere 転送ルール変数, 780
ias_StatusTime
QAnywhere 事前に定義されたメッセージプロパティ, 700
QAnywhere 転送ルール変数, 780
ias_StoreID
QAnywhere のメッセージストアプロパティ, 762
ias_StoreInitialized
QAnywhere のメッセージストアプロパティ, 762

ias_StoreVersion
QAnywhere のメッセージストアプロパティ, 762
ias_TransmissionStatus
QAnywhere 転送ルール変数, 780
ID
QAnywhere アドレスについて, 60
IMPLICIT_ACKNOWLEDGEMENT 変数
AcknowledgementMode インターフェイス [QAnywhere Java API], 496
AcknowledgementMode クラス [QAnywhere C++ API], 376
INCOMING 変数
QueueDepthFilter インターフェイス [QAnywhere Java API], 615
QueueDepthFilter クラス [QAnywhere C++ API], 490
InReplyToID プロパティ
QAMessage インターフェイス [QAnywhere .NET API], 302
InReplyToID メッセージヘッダー
QAnywhere のメッセージヘッダー, 695
Instance プロパティ
QAManagerFactory クラス [QAnywhere .NET API], 285
IP フィールド
MessageProperties クラス [QAnywhere .NET API], 193
IP 変数
MessageProperties インターフェイス [QAnywhere Java API], 500
MessageProperties クラス [QAnywhere C++ API], 380

J

Java EE
QAnywhere, 1
JMS
メッセージング対応 Mobile Link の実行と JMS コネクター, 135
JMSDestination
QAnywhere メッセージの JMS メッセージへのマッピング, 140
JMSExpiration
QAnywhere メッセージの JMS メッセージへのマッピング, 140
JMSPriority
QAnywhere メッセージの JMS メッセージへのマッピング, 140
JMSReplyTo
QAnywhere メッセージの JMS メッセージへのマッピング, 140
JMSTimestamp
QAnywhere メッセージの JMS メッセージへのマッピング, 140
JMS コネクター
QAnywhere, 135
QAnywhere アドレス, 61
QAnywhere アーキテクチャー, 9
チュートリアル, 148
JMS コネクタープロパティ
設定, 138
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, 144
JMS プロバイダー
QAnywhere アーキテクチャー, 9
JMS メッセージの QAnywhere メッセージへのマッピング
説明, 143
JMS メッセージのアドレス指定
QAnywhere, 142

L

LENGTH 関数
QAnywhere 構文, 779
Listener の設定
QAnywhere, 37
Listener ユーティリティ (dblsn)
QAnywhere Agent, 48
QAnywhere アーキテクチャー, 8
QAnywhere の設定, 37
LOCAL 変数
QueueDepthFilter インターフェイス [QAnywhere Java API], 615
QueueDepthFilter クラス [QAnywhere C++ API], 490
StatusCodes インターフェイス [QAnywhere Java API], 618
StatusCodes クラス [QAnywhere C++ API], 493
LOG_FILE プロパティ
QAnywhere Manager の設定プロパティ, 85

M

MAC フィールド
MessageProperties クラス [QAnywhere .NET API], 194
MAC 変数
MessageProperties インターフェイス [QAnywhere Java API], 501
MessageProperties クラス [QAnywhere C++ API], 381
MAX_DELIVERY_ATTEMPTS フィールド
MessageStoreProperties クラス [QAnywhere .NET API], 199
MAX_DELIVERY_ATTEMPTS 変数
MessageStoreProperties インターフェイス [QAnywhere Java API], 504
MessageStoreProperties クラス [QAnywhere C++ API], 386
MAX_IN_MEMORY_MESSAGE_SIZE プロパティ
QAnywhere Manager の設定プロパティ, 85
messagedetailsreport タグ
QAnywhere サーバー管理要求, 708
MessageDetailsRequest タグ
QAnywhere サーバー管理要求, 174
MessageID プロパティ
QAMessage インターフェイス [QAnywhere .NET API], 302
MessageID メッセージヘッダー
QAnywhere のメッセージヘッダー, 695
MessageListener2 デリゲート [QAnywhere .NET API]
説明, 316
MessageListener クラス
QAnywhere (.NET), 73
QAnywhere (Java), 74
QAnywhere システムメッセージ, 61
MessageListener デリゲート [QAnywhere .NET API]
説明, 316
MessageProperties インターフェイス [QAnywhere Java API]
ADAPTERS 変数, 499
ADAPTER 変数, 499
DELIVERY_COUNT 変数, 500
IP 変数, 500
MAC 変数, 501
MSG_TYPE 変数, 501
NETWORK_STATUS 変数, 501
ORIGINATOR 変数, 502
RASNAMES 変数, 503
RAS 変数, 502
STATUS_TIME 変数, 503
STATUS 変数, 503
TRANSMISSION_STATUS 変数, 504
説明, 497
MessageProperties クラス [QAnywhere .NET API]
ADAPTERS フィールド, 192
ADAPTER フィールド, 191
DELIVERY_COUNT フィールド, 193
IP フィールド, 193
MAC フィールド, 194
MSG_TYPE フィールド, 194
NETWORK_STATUS フィールド, 195
ORIGINATOR フィールド, 195
RASNAMES フィールド, 196
RAS フィールド, 196
STATUS_TIME フィールド, 197
STATUS フィールド, 197
TRANSMISSION_STATUS フィールド, 198
説明, 189
MessageProperties クラス [QAnywhere C++ API]
ADAPTERS 変数, 380
ADAPTER 変数, 379
DELIVERY_COUNT 変数, 380
IP 変数, 380
MAC 変数, 381
MSG_TYPE 変数, 381
NETWORK_STATUS 変数, 382
ORIGINATOR 変数, 382
RASNAMES 変数, 383
RAS 変数, 383
STATUS_TIME 変数, 384
STATUS 変数, 384
TRANSMISSION_STATUS 変数, 385
説明, 377
MessageStoreProperties インターフェイス [QAnywhere Java API]
MAX_DELIVERY_ATTEMPTS 変数, 504
説明, 504
MessageStoreProperties クラス [QAnywhere .NET API]
MAX_DELIVERY_ATTEMPTS フィールド, 199
説明, 198

MessageStoreProperties クラス [QAnywhere C++ API]
MAX_DELIVERY_ATTEMPTS 変数, 386
説明, 385
MessageType インターフェイス [QAnywhere Java API]
NETWORK_STATUS_NOTIFICATION 変数, 506
PUSH_NOTIFICATION 変数, 506
REGULAR 変数, 506
説明, 505
MessageType クラス [QAnywhere C++ API]
NETWORK_STATUS_NOTIFICATION 変数, 387
PUSH_NOTIFICATION 変数, 388
REGULAR 変数, 388
説明, 386
MessageType 列挙体 [QAnywhere .NET API]
説明, 318
ml_qa_createmessage
QAnywhere ストアドプロシージャー, 687
ml_qa_getaddress
QAnywhere ストアドプロシージャー, 655
ml_qa_getbinarycontent
QAnywhere ストアドプロシージャー, 681
ml_qa_getbooleanproperty
QAnywhere ストアドプロシージャー, 665
ml_qa_getbyteproperty
QAnywhere ストアドプロシージャー, 666
ml_qa_getcontentclass
QAnywhere ストアドプロシージャー, 682
ml_qa_getdoubleproperty
QAnywhere ストアドプロシージャー, 667
ml_qa_getexpiration
QAnywhere ストアドプロシージャー, 656
ml_qa_getfloatproperty
QAnywhere ストアドプロシージャー, 668
ml_qa_getinreplytoid
QAnywhere ストアドプロシージャー, 657
ml_qa_getintproperty
QAnywhere ストアドプロシージャー, 669
ml_qa_getlongproperty
QAnywhere ストアドプロシージャー, 670
ml_qa_getmessage
QAnywhere ストアドプロシージャー, 687
ml_qa_getmessagenowait
QAnywhere ストアドプロシージャー, 689
ml_qa_getmessagetimeout
QAnywhere ストアドプロシージャー, 690
ml_qa_getpriority
QAnywhere ストアドプロシージャー, 658
ml_qa_getpropertynames
QAnywhere ストアドプロシージャー, 671
ml_qa_getredelivered
QAnywhere ストアドプロシージャー, 658
ml_qa_getreplytoaddress
QAnywhere ストアドプロシージャー, 660
ml_qa_getshortproperty
QAnywhere ストアドプロシージャー, 672
ml_qa_getstoreproperty
QAnywhere ストアドプロシージャー, 685
ml_qa_getstringproperty
QAnywhere ストアドプロシージャー, 673
ml_qa_gettextcontent
QAnywhere ストアドプロシージャー, 683
ml_qa_gettimestamp
QAnywhere ストアドプロシージャー, 661
ml_qa_grant_messaging_permissions
QAnywhere ストアドプロシージャー, 691
ml_qa_listener_<queue>
QAnywhere ストアドプロシージャー, 692
ml_qa_listener_queue ストアドプロシージャー
QAnywhere SQL, 692
ml_qa_putmessage
QAnywhere ストアドプロシージャー, 693
ml_qa_setbinarycontent
QAnywhere ストアドプロシージャー, 684
ml_qa_setbooleanproperty
QAnywhere ストアドプロシージャー, 674
ml_qa_setbyteproperty
QAnywhere ストアドプロシージャー, 675
ml_qa_setdoubleproperty
QAnywhere ストアドプロシージャー, 676
ml_qa_setexpiration
QAnywhere ストアドプロシージャー, 662
ml_qa_setfloatproperty
QAnywhere ストアドプロシージャー, 677
ml_qa_setinreplytoid
QAnywhere ストアドプロシージャー, 663
ml_qa_setintproperty
QAnywhere ストアドプロシージャー, 677
ml_qa_setlongproperty
QAnywhere ストアドプロシージャー, 678
ml_qa_setpriority
QAnywhere ストアドプロシージャー, 664
ml_qa_setreplytoaddress

QAnywhere ストアドプロシージャー, 664
ml_qa_setshortproperty
QAnywhere ストアドプロシージャー, 679
ml_qa_setstoreproperty
QAnywhere ストアドプロシージャー, 686
ml_qa_setstringproperty
QAnywhere ストアドプロシージャー, 680
ml_qa_settextcontent
QAnywhere ストアドプロシージャー, 684
ml_qa_triggersendreceive
QAnywhere ストアドプロシージャー, 694
Mobile Link Listener ユーティリティ (dblsn)
QAnywhere アーキテクチャー, 8
QAnywhere の設定, 37
Mobile Link サーバー
QAnywhere, 31
Mobile Link サーバーの起動
QAnywhere JMS 統合, 138
QAnywhere メッセージング, 31
Mobile Link サーバーのログファイルビューアー
QAnywhere サーバーのログ, 33
QAnywhere のログ, 33
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, 123
Mobile Link の実行
QAnywhere、簡単なメッセージング, 31
QAnywhere メッセージングと JMS コネクター, 138
Mobile Link のログファイルビューアー
QAnywhere サーバーのログ, 33
Mobile Link ユーザー名
QAnywhere のサーバーメッセージストアへの追加, 32
Mobile Link ユーティリティ (dblsn)
QAnywhere Agent, 48
Mode プロパティ
QAManagerBase インターフェイス [QAnywhere .NET API], 279
MSG_TYPE フィールド
MessageProperties クラス [QAnywhere .NET API], 194
MSG_TYPE 変数
MessageProperties インターフェイス [QAnywhere Java API], 501
MessageProperties クラス [QAnywhere C++ API], 381

N

NativeErrorCode プロパティ
QAException クラス [QAnywhere .NET API], 222
network_status_notification
QAnywhere の ias_MessageType, 699
NETWORK_STATUS_NOTIFICATION 変数
MessageType インターフェイス [QAnywhere Java API], 506
MessageType クラス [QAnywhere C++ API], 387
NETWORK_STATUS_NOTIFICATION メッセージタイプ
QAnywhere システムキュー, 62
NETWORK_STATUS フィールド
MessageProperties クラス [QAnywhere .NET API], 195
NETWORK_STATUS 変数
MessageProperties インターフェイス [QAnywhere Java API], 501
MessageProperties クラス [QAnywhere C++ API], 382
nextPropertyName メソッド
QAMessage クラス [QAnywhere C++ API], 470
nextStorePropertyName メソッド
QAManagerBase クラス [QAnywhere C++ API], 441
Notifier の設定
QAnywhere, 36

O

ODBC データソース
QAnywhere Demo 12, 24
ondemand ポリシー
QAnywhere Agent, 727
QAnywhere Ultra Light Agent, 752
onException メソッド
QAMessageListener2 インターフェイス [QAnywhere Java API], 601
QAMessageListener インターフェイス [QAnywhere Java API], 600
WSListener インターフェイス [QAnywhere Java API], 631
OnException メソッド
WSListener インターフェイス [QAnywhere .NET API], 339
onMessage メソッド
QAManager クラス (C++), 73

QAMessageListener2 インターフェイス [QAnywhere Java API], 602
QAMessageListener インターフェイス [QAnywhere Java API], 601
QAMessageListener クラス [QAnywhere C++ API], 478
onResult メソッド
WSListener インターフェイス [QAnywhere Java API], 632
OnResult メソッド
WSListener インターフェイス [QAnywhere .NET API], 340
OpenConnector タグ
QAnywhere サーバー管理要求, 161
open メソッド
QAManager インターフェイス [QAnywhere Java API], 543
QAManager クラス [QAnywhere C++ API], 417
QATransactionalManager インターフェイス [QAnywhere Java API], 613
QATransactionalManager クラス [QAnywhere C++ API], 489
Open メソッド
QAManager インターフェイス [QAnywhere .NET API], 237
QATransactionalManager インターフェイス [QAnywhere .NET API], 314
ORIGINATOR フィールド
MessageProperties クラス [QAnywhere .NET API], 195
ORIGINATOR 変数
MessageProperties インターフェイス [QAnywhere Java API], 502
MessageProperties クラス [QAnywhere C++ API], 382
OUTGOING 変数
QueueDepthFilter インターフェイス [QAnywhere Java API], 616
QueueDepthFilter クラス [QAnywhere C++ API], 491

P

PENDING 変数
StatusCodes インターフェイス [QAnywhere Java API], 618
StatusCodes クラス [QAnywhere C++ API], 493
Priority プロパティ
QAMessage インターフェイス [QAnywhere .NET API], 303
Priority メッセージヘッダー
QAnywhere のメッセージヘッダー, 695
PROPERTY_TYPE_BOOLEAN 変数
PropertyType インターフェイス [QAnywhere Java API], 507
PROPERTY_TYPE_BYTE 変数
PropertyType インターフェイス [QAnywhere Java API], 508
PROPERTY_TYPE_DOUBLE 変数
PropertyType インターフェイス [QAnywhere Java API], 508
PROPERTY_TYPE_FLOAT 変数
PropertyType インターフェイス [QAnywhere Java API], 508
PROPERTY_TYPE_INT 変数
PropertyType インターフェイス [QAnywhere Java API], 508
PROPERTY_TYPE_LONG 変数
PropertyType インターフェイス [QAnywhere Java API], 508
PROPERTY_TYPE_SHORT 変数
PropertyType インターフェイス [QAnywhere Java API], 508
PROPERTY_TYPE_STRING 変数
PropertyType インターフェイス [QAnywhere Java API], 509
PROPERTY_TYPE_UNKNOWN 変数
PropertyType インターフェイス [QAnywhere Java API], 509
propertyExists メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 565
QAMessage インターフェイス [QAnywhere Java API], 594
QAMessage クラス [QAnywhere C++ API], 470
PropertyExists メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 263
QAMessage インターフェイス [QAnywhere .NET API], 295
PropertyType インターフェイス [QAnywhere Java API]
PROPERTY_TYPE_BOOLEAN 変数, 507
PROPERTY_TYPE_BYTE 変数, 508
PROPERTY_TYPE_DOUBLE 変数, 508
PROPERTY_TYPE_FLOAT 変数, 508

PROPERTY_TYPE_INT 変数, 508
PROPERTY_TYPE_LONG 変数, 508
PROPERTY_TYPE_SHORT 変数, 508
PROPERTY_TYPE_STRING 変数, 509
PROPERTY_TYPE_UNKNOWN 変数, 509
説明, 507
PropertyType 列挙体 [QAnywhere .NET API]
説明, 320
prop タグ
QAnywhere サーバー管理要求, 168
push_notification
QAnywhere の ias_MessageType, 699
PUSH_NOTIFICATION 変数
MessageType インターフェイス [QAnywhere Java API], 506
MessageType クラス [QAnywhere C++ API], 388
PUSH_NOTIFICATION メッセージタイプ
QAnywhere システムキュー, 63
Push 通知
qaagent -push オプション, 729
QAnywhere での処理, 61
QAnywhere の設定, 35
QAnywhere の例, 7
qauagent -push オプション, 753
説明, 34
Push 通知によるメッセージング
QAnywhere アーキテクチャー, 7
Push 通知の設定
QAnywhere, 35
putMessageTimeToLive メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 566
QAManagerBase クラス [QAnywhere C++ API], 442
PutMessageTimeToLive メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 264
putMessage メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 565
QAManagerBase クラス [QAnywhere C++ API], 441
PutMessage メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 263

Q

QA_NO_ERROR フィールド
QAException クラス [QAnywhere .NET API], 230
QA_NO_ERROR 変数
QAError クラス [QAnywhere C++ API], 410
QAException クラス [QAnywhere Java API], 536
qa.hpp
QAnywhere ヘッダーファイル, 56
qa.hpp ヘッダーファイル
クライアント用 QAnywhere C++ API リファレンス, 375
qaagent ユーティリティ
Windows Mobile 上での起動, 47
構文, 712
説明, 47
停止, 48
QAA ファイル
QAnywhere, 49
QABinaryMessage インターフェイス [QAnywhere .NET API]
BodyLength プロパティ, 218
ReadBinary メソッド, 204
ReadBoolean メソッド, 206
ReadChar メソッド, 207
ReadDouble メソッド, 207
ReadFloat メソッド, 208
ReadInt メソッド, 208
ReadLong メソッド, 209
ReadSbyte メソッド, 209
ReadShort メソッド, 210
ReadString メソッド, 210
Reset メソッド, 211
WriteBinary メソッド, 211
WriteBoolean メソッド, 214
WriteChar メソッド, 214
WriteDouble メソッド, 215
WriteFloat メソッド, 215
WriteInt メソッド, 215
WriteLong メソッド, 216
WriteSbyte メソッド, 216
WriteShort メソッド, 217
WriteString メソッド, 217
説明, 199
QABinaryMessage インターフェイス [QAnywhere Java API]
getBodyLength メソッド, 514

readBinary メソッド, 514
readBoolean メソッド, 516
readByte メソッド, 517
readChar メソッド, 517
readDouble メソッド, 518
readFloat メソッド, 518
readInt メソッド, 518
readLong メソッド, 519
readShort メソッド, 519
readString メソッド, 520
reset メソッド, 520
writeBinary メソッド, 520
writeBoolean メソッド, 523
writeByte メソッド, 523
writeChar メソッド, 524
writeDouble メソッド, 524
writeFloat メソッド, 525
writeInt メソッド, 525
writeLong メソッド, 526
writeShort メソッド, 526
writeString メソッド, 527
説明, 509
QABinaryMessage クラス
インスタンス化 (.NET), 63
インスタンス化 (C++), 64
QABinaryMessage クラス [QAnywhere C++ API]
getBodyLength メソッド, 394
QABinaryMessage デコンストラクター, 394
readBinary メソッド, 394
readBoolean メソッド, 395
readByte メソッド, 395
readChar メソッド, 395
readDouble メソッド, 396
readFloat メソッド, 396
readInt メソッド, 397
readLong メソッド, 397
readShort メソッド, 397
readString メソッド, 398
reset メソッド, 398
writeBinary メソッド, 399
writeBoolean メソッド, 399
writeByte メソッド, 399
writeChar メソッド, 400
writeDouble メソッド, 400
writeFloat メソッド, 401
writeInt メソッド, 401
writeLong メソッド, 401
writeShort メソッド, 402
writeString メソッド, 402
説明, 388
QABinaryMessage デコンストラクター
QABinaryMessage クラス [QAnywhere C++ API], 394
QAError クラス [QAnywhere C++ API]
COMMON_ALREADY_OPEN_ERROR 変数, 405
COMMON_GET_INIT_FILE_ERROR 変数, 405
COMMON_GET_PROPERTY_ERROR 変数, 405
COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG 変数, 406
COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID 変数, 406
COMMON_GETQUEUEDEPTH_ERROR 変数, 406
COMMON_INIT_ERROR 変数, 406
COMMON_INIT_THREAD_ERROR 変数, 406
COMMON_INVALID_PROPERTY 変数, 407
COMMON_MSG_ACKNOWLEDGE_ERROR 変数, 407
COMMON_MSG_CANCEL_ERROR_SENT 変数, 407
COMMON_MSG_CANCEL_ERROR 変数, 407
COMMON_MSG_NOT_WRITEABLE_ERROR 変数, 407
COMMON_MSG_RETRIEVE_ERROR 変数, 408
COMMON_MSG_STORE_ERROR 変数, 408
COMMON_MSG_STORE_NOT_INITIALIZED 変数, 408
COMMON_MSG_STORE_TOO_LARGE 変数, 408
COMMON_NO_DEST_ERROR 変数, 408
COMMON_NO_IMPLEMENTATION 変数, 408
COMMON_NOT_OPEN_ERROR 変数, 409
COMMON_OPEN_ERROR 変数, 409
COMMON_OPEN_LOG_FILE_ERROR 変数, 409
COMMON_OPEN_MAXTHREADS_ERROR 変数, 409
COMMON_SELECTOR_SYNTAX_ERROR 変数, 409
COMMON_SET_PROPERTY_ERROR 変数, 410
COMMON_TERMINATE_ERROR 変数, 410
COMMON_UNEXPECTED_EOM_ERROR 変数, 410

COMMON_UNREPRESENTABLE_TIMESTAMP 変数, 410
QA_NO_ERROR 変数, 410
説明, 403
QAException クラス [QAnywhere .NET API]
COMMON_ALREADY_OPEN_ERROR フィールド, 223
COMMON_GET_INIT_FILE_ERROR フィールド, 223
COMMON_GET_PROPERTY_ERROR フィールド, 223
COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG フィールド, 224
COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID フィールド, 224
COMMON_GETQUEUEDEPTH_ERROR フィールド, 224
COMMON_INIT_ERROR フィールド, 225
COMMON_INIT_THREAD_ERROR フィールド, 225
COMMON_INVALID_PROPERTY フィールド, 225
COMMON_MSG_ACKNOWLEDGE_ERROR フィールド, 225
COMMON_MSG_CANCEL_ERROR_SENT フィールド, 226
COMMON_MSG_CANCEL_ERROR フィールド, 226
COMMON_MSG_NOT_WRITEABLE_ERROR フィールド, 226
COMMON_MSG_RETRIEVE_ERROR フィールド, 226
COMMON_MSG_STORE_ERROR フィールド, 227
COMMON_MSG_STORE_NOT_INITIALIZED フィールド, 227
COMMON_MSG_STORE_TOO_LARGE フィールド, 227
COMMON_NO_DEST_ERROR フィールド, 227
COMMON_NO_IMPLEMENTATION フィールド, 228
COMMON_NOT_OPEN_ERROR フィールド, 228
COMMON_OPEN_ERROR フィールド, 228
COMMON_OPEN_LOG_FILE_ERROR フィールド, 228
COMMON_OPEN_MAXTHREADS_ERROR フィールド, 229
COMMON_REOPEN_ERROR フィールド, 229
COMMON_SELECTOR_SYNTAX_ERROR フィールド, 229
COMMON_SET_PROPERTY_ERROR フィールド, 229
COMMON_TERMINATE_ERROR フィールド, 230
COMMON_UNEXPECTED_EOM_ERROR フィールド, 230
COMMON_UNREPRESENTABLE_TIMESTAMP フィールド, 230
DetailedMessage プロパティ, 222
ErrorCode プロパティ, 222
NativeErrorCode プロパティ, 222
QA_NO_ERROR フィールド, 230
説明, 218
QAException クラス [QAnywhere Java API]
COMMON_ALREADY_OPEN_ERROR 変数, 530
COMMON_GET_INIT_FILE_ERROR 変数, 531
COMMON_GET_PROPERTY_ERROR 変数, 531
COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG 変数, 531
COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID 変数, 532
COMMON_GETQUEUEDEPTH_ERROR 変数, 531
COMMON_INIT_ERROR 変数, 532
COMMON_INIT_THREAD_ERROR 変数, 532
COMMON_INVALID_PROPERTY 変数, 532
COMMON_MSG_ACKNOWLEDGE_ERROR 変数, 532
COMMON_MSG_CANCEL_ERROR_SENT 変数, 533
COMMON_MSG_CANCEL_ERROR 変数, 533
COMMON_MSG_NOT_WRITEABLE_ERROR 変数, 533
COMMON_MSG_RETRIEVE_ERROR 変数, 533
COMMON_MSG_STORE_ERROR 変数, 533
COMMON_MSG_STORE_NOT_INITIALIZED 変数, 534
COMMON_MSG_STORE_TOO_LARGE 変数, 534
COMMON_NO_DEST_ERROR 変数, 534
COMMON_NO_IMPLEMENTATION 変数, 534
COMMON_NOT_OPEN_ERROR 変数, 534
COMMON_OPEN_ERROR 変数, 535

COMMON_OPEN_LOG_FILE_ERROR 変数, 535
COMMON_OPEN_MAXTHREADS_ERROR 変数, 535
COMMON_REOPEN_ERROR 変数, 535
COMMON_SELECTOR_SYNTAX_ERROR 変数, 535
COMMON_SET_PROPERTY_ERROR 変数, 535
COMMON_TERMINATE_ERROR 変数, 536
COMMON_UNEXPECTED_EOM_ERROR 変数, 536
COMMON_UNREPRESENTABLE_TIMESTAMP 変数, 536
getDetailedMessage メソッド, 530
getErrorCode メソッド, 530
getNativeErrorCode メソッド, 530
QA_NO_ERROR 変数, 536
説明, 527
QAManager
C++ アプリケーション設定, 56
Java アプリケーション設定, 58
設定プロパティ, 85
マルチスレッド, 84
QAManagerBase インターフェイス [QAnywhere .NET API]
BrowseMessagesByID メソッド, 245
BrowseMessagesByQueue メソッド, 246
BrowseMessagesBySelector メソッド, 246
BrowseMessages メソッド, 243
CancelMessage メソッド, 247
Close メソッド, 248
CreateBinaryMessage メソッド, 248
CreateTextMessage メソッド, 249
GetBooleanStoreProperty メソッド, 249
GetDoubleStoreProperty メソッド, 250
GetFloatStoreProperty メソッド, 251
GetIntStoreProperty メソッド, 252
GetLongStoreProperty メソッド, 252
GetMessageBySelectorNoWait メソッド, 255
GetMessageBySelectorTimeout メソッド, 255
GetMessageBySelector メソッド, 254
GetMessageNoWait メソッド, 256
GetMessageTimeout メソッド, 257
GetMessage メソッド, 253
GetQueueDepth メソッド, 258
GetSbyteStoreProperty メソッド, 260
GetShortStoreProperty メソッド, 260
GetStorePropertyNames メソッド, 262
GetStoreProperty メソッド, 261
GetStringStoreProperty メソッド, 262
Mode プロパティ, 279
PropertyExists メソッド, 263
PutMessageTimeToLive メソッド, 264
PutMessage メソッド, 263
ReOpen メソッド, 265
SetBooleanStoreProperty メソッド, 266
SetDoubleStoreProperty メソッド, 266
SetExceptionListener2 メソッド, 267
SetExceptionListener メソッド, 267
SetFloatStoreProperty メソッド, 268
SetIntStoreProperty メソッド, 269
SetLongStoreProperty メソッド, 269
SetMessageListener2 メソッド, 271
SetMessageListenerBySelector2 メソッド, 273
SetMessageListenerBySelector メソッド, 272
SetMessageListener メソッド, 270
SetProperty メソッド, 274
SetSbyteStoreProperty メソッド, 274
SetShortStoreProperty メソッド, 275
SetStoreProperty メソッド, 276
SetStringStoreProperty メソッド, 276
Start メソッド, 277
Stop メソッド, 277
TriggerSendReceive メソッド, 278
説明, 239
QAManagerBase インターフェイス [QAnywhere Java API]
browseMessagesByID メソッド, 548
browseMessagesByQueue メソッド, 549
browseMessagesBySelector メソッド, 550
browseMessages メソッド, 548
cancelMessage メソッド, 550
close メソッド, 551
createBinaryMessage メソッド, 551
createTextMessage メソッド, 552
getBooleanStoreProperty メソッド, 552
getByteStoreProperty メソッド, 553
getDoubleStoreProperty メソッド, 554
getFloatStoreProperty メソッド, 554
getIntStoreProperty メソッド, 555
getLongStoreProperty メソッド, 555
getMessageBySelectorNoWait メソッド, 557
getMessageBySelectorTimeout メソッド, 558
getMessageBySelector メソッド, 557
getMessageNoWait メソッド, 559
getMessageTimeout メソッド, 560

getMessage メソッド, 556
getMode メソッド, 560
getQueueDepth メソッド, 561
getShortStoreProperty メソッド, 562
getStorePropertyNames メソッド, 564
getStoreProperty メソッド, 563
getStringStoreProperty メソッド, 564
propertyExists メソッド, 565
putMessageTimeToLive メソッド, 566
putMessage メソッド, 565
reOpen メソッド, 566
setBooleanStoreProperty メソッド, 567
setByteStoreProperty メソッド, 568
setDoubleStoreProperty メソッド, 568
setFloatStoreProperty メソッド, 569
setIntStoreProperty メソッド, 569
setLongStoreProperty メソッド, 570
setMessageListener2 メソッド, 571
setMessageListenerBySelector2 メソッド, 573
setMessageListenerBySelector メソッド, 572
setMessageListener メソッド, 571
setProperty メソッド, 573
setShortStoreProperty メソッド, 574
setStoreProperty メソッド, 575
setStringStoreProperty メソッド, 575
start メソッド, 576
stop メソッド, 577
triggerSendReceive メソッド, 577
説明, 544
QAManagerBase クラス [QAnywhere C++ API]
beginEnumStorePropertyNames メソッド, 423
browseClose メソッド, 423
browseMessagesByID メソッド, 424
browseMessagesByQueue メソッド, 425
browseMessagesBySelector メソッド, 425
browseMessages メソッド, 423
browseNextMessage メソッド, 426
cancelMessage メソッド, 427
close メソッド, 427
createBinaryMessage メソッド, 428
createTextMessage メソッド, 428
deleteMessage メソッド, 429
endEnumStorePropertyNames メソッド, 429
getAllQueueDepth メソッド, 429
getBooleanStoreProperty メソッド, 430
getByteStoreProperty メソッド, 431
getDoubleStoreProperty メソッド, 431
getFloatStoreProperty メソッド, 432
getIntStoreProperty メソッド, 432
getLastErrorMsg メソッド, 433
getLastError メソッド, 433
getLastNativeError メソッド, 434
getLongStoreProperty メソッド, 434
getMessageBySelectorNoWait メソッド, 436
getMessageBySelectorTimeout メソッド, 436
getMessageBySelector メソッド, 435
getMessageNoWait メソッド, 437
getMessageTimeout メソッド, 438
getMessage メソッド, 435
getMode メソッド, 438
getQueueDepth メソッド, 439
getShortStoreProperty メソッド, 439
getStringStoreProperty メソッド, 440
nextStorePropertyName メソッド, 441
putMessageTimeToLive メソッド, 442
putMessage メソッド, 441
setBooleanStoreProperty メソッド, 442
setByteStoreProperty メソッド, 443
setDoubleStoreProperty メソッド, 443
setFloatStoreProperty メソッド, 444
setIntStoreProperty メソッド, 444
setLongStoreProperty メソッド, 445
setMessageListenerBySelector メソッド, 446
setMessageListener メソッド, 445
setProperty メソッド, 447
setShortStoreProperty メソッド, 447
setStringStoreProperty メソッド, 448
start メソッド, 449
stop メソッド, 449
triggerSendReceive メソッド, 450
説明, 418
QAManagerFactory クラス
Web サービス用にインスタンス化 (.NET), 99
Web サービス用にインスタンス化 (Java), 102
インスタンス化 (C++), 56
初期化 (.NET), 55
初期化 (Java), 58
トランザクション指向メッセージングの実装 (Java), 69
トランザクション指向メッセージングのための初期化 (.NET), 66
QAManagerFactory クラス [QAnywhere .NET API]
CreateQAManager メソッド, 280
CreateQATransactionalManager メソッド, 282
Instance プロパティ, 285
説明, 279

QAManagerFactory クラス [QAnywhere C++ API]
createQAManager メソッド, 451
createQATransactionalManager メソッド, 452
deleteQAManager メソッド, 452
deleteQATransactionalManager メソッド, 453
getLastErrorMsg メソッド, 453
getLastError メソッド, 453
getLastNativeError メソッド, 454
説明, 450
QAManagerFactory クラス [QAnywhere Java API]
createQAManager メソッド, 578
createQATransactionalManager メソッド, 580
getInstance メソッド, 582
説明, 578
QAManager インターフェイス [QAnywhere .NET API]
AcknowledgeAll メソッド, 236
AcknowledgeUntil メソッド, 237
Acknowledge メソッド, 235
Open メソッド, 237
Recover メソッド, 238
説明, 231
QAManager インターフェイス [QAnywhere Java API]
acknowledgeAll メソッド, 541
acknowledgeUntil メソッド, 542
acknowledge メソッド, 541
open メソッド, 543
recover メソッド, 543
説明, 536
QAManager クラス
Web サービス用にインスタンス化 (.NET), 99
Web サービス用にインスタンス化 (Java), 102
Web サービス用に初期化 (.NET), 99
Web サービス用に初期化 (Java), 102
Web サービス用の受信確認モード (.NET), 99
Web サービス用の受信確認モード (Java), 102
インスタンス化 (.NET), 55
インスタンス化 (C++), 56
インスタンス化 (Java), 58
受信確認モード (.NET), 55
受信確認モード (C++), 56
受信確認モード (Java), 58
初期化 (.NET), 55
初期化 (C++), 56
初期化 (Java), 58
QAManager クラス [QAnywhere C++ API]
acknowledgeAll メソッド, 416
acknowledgeUntil メソッド, 416
acknowledge メソッド, 415
open メソッド, 417
recover メソッド, 418
説明, 411
QAManager のプロパティ
プロパティファイル, 55
QAMessageListener2 インターフェイス [QAnywhere Java API]
onException メソッド, 601
onMessage メソッド, 602
説明, 601
QAMessageListener インターフェイス [QAnywhere Java API]
onException メソッド, 600
onMessage メソッド, 601
説明, 600
QAMessageListener クラス [QAnywhere C++ API]
onMessage メソッド, 478
QAMessageListener デコンストラクター, 478
説明, 477
QAMessageListener デコンストラクター
QAMessageListener クラス [QAnywhere C++ API], 478
QAMessage インターフェイス [QAnywhere .NET API]
Address プロパティ, 301
ClearBody メソッド, 288
ClearProperties メソッド, 288
Expiration プロパティ, 301
GetBooleanProperty メソッド, 288
GetByteProperty メソッド, 289
GetDoubleProperty メソッド, 289
GetFloatProperty メソッド, 290
GetIntProperty メソッド, 291
GetLongProperty メソッド, 291
GetPropertyNames メソッド, 292
GetPropertyType メソッド, 293
GetProperty メソッド, 292
GetSbyteProperty メソッド, 293
GetShortProperty メソッド, 294
GetStringProperty メソッド, 294
InReplyToID プロパティ, 302
MessageID プロパティ, 302
Priority プロパティ, 303
PropertyExists メソッド, 295
Redelivered プロパティ, 303
ReplyToAddress プロパティ, 303

SetBooleanProperty メソッド, 295
SetByteProperty メソッド, 296
SetDoubleProperty メソッド, 296
SetFloatProperty メソッド, 297
SetIntProperty メソッド, 297
SetLongProperty メソッド, 298
SetProperty メソッド, 299
SetSbyteProperty メソッド, 299
SetShortProperty メソッド, 300
SetStringProperty メソッド, 300
Timestamp プロパティ, 304
説明, 285
QAMessage インターフェイス [QAnywhere Java API]
clearProperties メソッド, 585
DEFAULT_PRIORITY 変数, 599
DEFAULT_TIME_TO_LIVE 変数, 599
getAddress メソッド, 585
getBooleanProperty メソッド, 586
getByteProperty メソッド, 586
getDoubleProperty メソッド, 587
getExpiration メソッド, 587
getFloatProperty メソッド, 588
getInReplyToID メソッド, 588
getIntProperty メソッド, 589
getLongProperty メソッド, 589
getMessageID メソッド, 590
getPriority メソッド, 590
getPropertyNames メソッド, 591
getPropertyType メソッド, 591
getProperty メソッド, 590
getRedelivered メソッド, 592
getReplyToAddress メソッド, 592
getShortProperty メソッド, 592
getStringProperty メソッド, 593
getTimestamp メソッド, 593
propertyExists メソッド, 594
setBooleanProperty メソッド, 594
setByteProperty メソッド, 595
setDoubleProperty メソッド, 595
setFloatProperty メソッド, 595
setInReplyToID メソッド, 596
setIntProperty メソッド, 596
setLongProperty メソッド, 597
setPriority メソッド, 597
setProperty メソッド, 597
setReplyToAddress メソッド, 598
setShortProperty メソッド, 598
setStringProperty メソッド, 599
説明, 582
QAMessage クラス
QAnywhere のメッセージプロパティの管理, 701
QAMessage クラス [QAnywhere C++ API]
beginEnumPropertyNames メソッド, 457
castToBinaryMessage メソッド, 458
castToTextMessage メソッド, 458
clearProperties メソッド, 459
DEFAULT_PRIORITY 変数, 477
DEFAULT_TIME_TO_LIVE 変数, 477
endEnumPropertyNames メソッド, 459
getAddress メソッド, 459
getBooleanProperty メソッド, 460
getByteProperty メソッド, 460
getDoubleProperty メソッド, 461
getExpiration メソッド, 461
getFloatProperty メソッド, 462
getInReplyToID メソッド, 462
getIntProperty メソッド, 463
getLongProperty メソッド, 463
getMessageID メソッド, 464
getPriority メソッド, 464
getPropertyType メソッド, 465
getRedelivered メソッド, 465
getReplyToAddress メソッド, 466
getShortProperty メソッド, 466
getStringProperty メソッド, 467
getTimestampAsString メソッド, 469
getTimestamp メソッド, 468
nextPropertyName メソッド, 470
propertyExists メソッド, 470
setAddress メソッド, 470
setBooleanProperty メソッド, 471
setByteProperty メソッド, 471
setDoubleProperty メソッド, 472
setFloatProperty メソッド, 472
setInReplyToID メソッド, 473
setIntProperty メソッド, 473
setLongProperty メソッド, 474
setMessageID メソッド, 474
setPriority メソッド, 474
setRedelivered メソッド, 475
setReplyToAddress メソッド, 475
setShortProperty メソッド, 475
setStringProperty メソッド, 476
setTimestamp メソッド, 476

説明, 454
QAnyNotifier_client
QAnywhere Notifier, 36
QAnywhere
JMS コネクターの使用方法, 135
SQL Anywhere クライアントメッセージストアへの接続, 715
WSDL コンパイラー, 97
アドレス, 60
アーキテクチャー, 3
機能, 2
クイックスタート, 12
クライアントサイドコンポーネントの設定, 34
クライアントメッセージストア, 25
クライアントメッセージストアへの接続, 741
コネクター, 135
削除ルール, 786
サーバーサイドコンポーネントの設定, 31
サーバーメッセージストア, 23
セキュリティ, 121
説明, 1
チュートリアル, 177
通知の受信, 73
転送ルール, 46
転送ルール変数, 780
配備, 117
フェールオーバー, 38
プログラミングインターフェイス, 51
メッセージアーカイブ, 24
メッセージ転送ルール, 783
モバイル Web サービス, 95
ローカルメッセージストア, 19
QAnywhere .NET API
AcknowledgementMode 列挙体, 317
ExceptionListener2 デリゲート, 315
ExceptionListener デリゲート, 315
iAnywhere.QAnywhere.Client ネームスペース, 189
iAnywhere.QAnywhere.StandAloneClient ネームスペース, 189
iAnywhere.QAnywhere.WS ネームスペース, 325
MessageListener2 デリゲート, 316
MessageListener デリゲート, 316
MessageProperties クラス, 189
MessageStoreProperties クラス, 198
MessageType 列挙体, 318
PropertyType 列挙体, 320
QABinaryMessage インターフェイス, 199
QAException クラス, 218
QAManagerBase インターフェイス, 239
QAManagerFactory クラス, 279
QAManager インターフェイス, 231
QAMessage インターフェイス, 285
QATextMessage インターフェイス, 304
QATransactionalManager インターフェイス, 309
QueueDepthFilter 列挙体, 321
StatusCodes 列挙体, 322
WSBase クラス, 325
WSException クラス, 332
WSFaultException クラス, 336
WSListener インターフェイス, 339
WSResult クラス, 340
WSStatus 列挙体, 374
概要, 51
初期化, 54
モバイル Web サービスの初期化, 98
QAnywhere 12 プラグイン
Sybase Central, 11
QAnywhere Agent
簡単なメッセージングアーキテクチャー, 7
説明, 41
QAnywhere Agent (qaagent) ユーティリティ
構文, 712
QAnywhere Agent の起動
説明, 47
QAnywhere Agent ユーティリティ (qaagent)
-cd オプション, 717
-cr オプション, 717
QAnywhere Agent ユーティリティ (qauagent)
-cd オプション, 742
-cr オプション, 743
QAnywhere API の初期化
説明, 54
QAnywhere C++ API
AcknowledgementMode クラス, 375
MessageProperties クラス, 377
MessageStoreProperties クラス, 385
MessageType クラス, 386
QABinaryMessage クラス, 388
QAError クラス, 403
QAManagerBase クラス, 418
QAManagerFactory クラス, 450
QAManager クラス, 411
QAMessageListener クラス, 477
QAMessage クラス, 454
QATextMessage クラス, 478

QATransactionalManager クラス, 484
QueueDepthFilter クラス, 489
StatusCodes クラス, 491
概要, 52
初期化, 56
QAnywhere Java API
AcknowledgementMode インターフェイス, 495
ianywhere.qanywhere.client パッケージ, 495
ianywhere.qanywhere.standaloneclient パッケージ, 495
ianywhere.qanywhere.ws パッケージ, 620
MessageProperties インターフェイス, 497
MessageStoreProperties インターフェイス, 504
MessageType インターフェイス, 505
PropertyType インターフェイス, 507
QABinaryMessage インターフェイス, 509
QAException クラス, 527
QAManagerBase インターフェイス, 544
QAManagerFactory クラス, 578
QAManager インターフェイス, 536
QAMessageListener2 インターフェイス, 601
QAMessageListener インターフェイス, 600
QAMessage インターフェイス, 582
QATextMessage インターフェイス, 603
QATransactionalManager インターフェイス, 609
QueueDepthFilter インターフェイス, 614
StatusCodes インターフェイス, 616
WSBase クラス, 621
WSException クラス, 627
WSFaultException クラス, 630
WSListener インターフェイス, 631
WSResult クラス, 632
WSStatus クラス, 653
概要, 52
初期化, 58
モバイル Web サービスの初期化, 101
QAnywhere Manager の設定プロパティ
.NET アプリケーション設定, 54
説明, 85
ファイル内で設定する, 86
プログラムによる設定, 88
プロパティファイル, 56
QAnywhere Notifier
アーキテクチャー, 8
QAnywhere Notifier の設定
説明, 36
QAnywhere SQL
説明, 59
QAnywhere SQL API
概要, 52
初期化, 59
説明, 59
リファレンス, 655
QAnywhere SQL API リファレンス
説明, 655
QAnywhere Ultra Light Agent ユーティリティ (qauagent)
構文, 738
QAnywhere アーキテクチャー
説明, 3
QAnywhere エージェント
アプリケーション間メッセージング, 4
QAnywhere から Web サービスに送信されるメッセージのアドレス指定
説明, 144
QAnywhere クライアント
概要, 5
停止, 84
配備, 117
QAnywhere クライアントアプリケーション
作成, 51
QAnywhere クライアントの配備
説明, 117
QAnywhere ゲートウェイの設定
説明, 38
QAnywhere 削除ルール
説明, 786
QAnywhere サーバー
簡単なメッセージングアーキテクチャー, 6
説明, 31
QAnywhere サーバーの起動
説明, 31
QAnywhere サーバーのログ
説明, 33
QAnywhere サーバーメッセージストア要求の管理
QAnywhere サーバー管理, 156
QAnywhere ストアドプロシージャー
ml_qa_createmessage, 687
ml_qa_getaddress, 655
ml_qa_getbinarycontent, 681
ml_qa_getbooleanproperty, 665
ml_qa_getbyteproperty, 666
ml_qa_getcontentclass, 682
ml_qa_getdoubleproperty, 667
ml_qa_getexpiration, 656

ml_qa_getfloatproperty, 668
ml_qa_getinreplytoid, 657
ml_qa_getintproperty, 669
ml_qa_getlongproperty, 670
ml_qa_getmessage, 687
ml_qa_getmessagenowait, 689
ml_qa_getmessagetimeout, 690
ml_qa_getpriority, 658
ml_qa_getpropertynames, 671
ml_qa_getredelivered, 658
ml_qa_getreplytoaddress, 660
ml_qa_getshortproperty, 672
ml_qa_getstoreproperty, 685
ml_qa_getstringproperty, 673
ml_qa_gettextcontent, 683
ml_qa_gettimestamp, 661
ml_qa_grant_messaging_permissions, 691
ml_qa_listener_queue, 692
ml_qa_putmessage, 693
ml_qa_setbinarycontent, 684
ml_qa_setbooleanproperty, 674
ml_qa_setbyteproperty, 675
ml_qa_setdoubleproperty, 676
ml_qa_setexpiration, 662
ml_qa_setfloatproperty, 677
ml_qa_setinreplytoid, 663
ml_qa_setintproperty, 677
ml_qa_setlongproperty, 678
ml_qa_setpriority, 664
ml_qa_setreplytoaddress, 664
ml_qa_setstoreproperty, 686
ml_qa_setstringproperty, 680
ml_qa_settextcontent, 684
ml_qa_triggersendreceive, 694
説明, 59
QAnywhere ストアドプロシージャー
ml_qa_setshortproperty, 679
QAnywhere ネームスペース
Web サービス用にインクルード, 99
インクルード, 55
QAnywhere のエージェント停止ユーティリティ (qastop)
構文, 760
QAnywhere の管理
説明, 153
QAnywhere の停止
説明, 84
QAnywhere のプロパティ
QAnywhere メッセージの JMS メッセージへのマッピング, 142
QAnywhere のメッセージプロパティ
説明, 698
QAnywhere のログファイルビューアー
説明, 33
QAnywhere パッケージ
Web サービス用にインクルード, 102
インクルード, 58
QAnywhere ヘッダーファイル
qa.hpp, 56
QAnywhere メッセージのアドレス指定
JMS, 139
QAnywhere メッセージの送信
JMS, 139
説明, 63
QAnywhere メッセージのマッピング
JMS メッセージ, 140
QAR ファイル
QAnywhere, 49
qastop ユーティリティ
qaagent -qi (クワイエットモード) での使用, 731
qauagent -qi (クワイエットモード) での使用, 755
構文, 760
QATextMessage インターフェイス [QAnywhere .NET API]
ReadText メソッド, 307
Reset メソッド, 308
TextLength プロパティ, 309
Text プロパティ, 308
WriteText メソッド, 308
説明, 304
QATextMessage インターフェイス [QAnywhere Java API]
getTextLength メソッド, 606
getText メソッド, 605
readText メソッド, 606
reset メソッド, 607
setText メソッド, 607
writeText メソッド, 607
説明, 603
QATextMessage クラス
インスタンス化 (.NET), 63
インスタンス化 (C++), 64
QATextMessage クラス [QAnywhere C++ API]
getTextLength メソッド, 482
getText メソッド, 482

readText メソッド, 482
reset メソッド, 483
setText メソッド, 483
writeText メソッド, 483
説明, 478
QATransactionalManager インターフェイス [QAnywhere .NET API]
Commit メソッド, 313
Open メソッド, 314
Rollback メソッド, 314
説明, 309
QATransactionalManager インターフェイス [QAnywhere Java API]
commit メソッド, 613
open メソッド, 613
rollback メソッド, 614
説明, 609
QATransactionalManager クラス
インスタンス化 (Java), 69
初期化 (.NET), 66
トランザクション指向メッセージングの実装 (C++), 67
トランザクション指向メッセージングの実装 (Java), 69
トランザクション指向メッセージングのためのインスタンス化 (.NET), 66
QATransactionalManager クラス [QAnywhere C++ API]
commit メソッド, 488
open メソッド, 489
rollback メソッド, 489
説明, 484
qauagent ユーティリティ
構文, 738
QueueDepthFilter インターフェイス [QAnywhere Java API]
ALL 変数, 615
INCOMING 変数, 615
LOCAL 変数, 615
OUTGOING 変数, 616
説明, 614
QueueDepthFilter クラス [QAnywhere C++ API]
ALL 変数, 490
INCOMING 変数, 490
LOCAL 変数, 490
OUTGOING 変数, 491
説明, 489
QueueDepthFilter 列挙体 [QAnywhere .NET API]
説明, 321

R

RASNames
QAnywhere のメッセージプロパティ, 62
RASNAMES フィールド
MessageProperties クラス [QAnywhere .NET API], 196
RASNAMES 変数
MessageProperties インターフェイス [QAnywhere Java API], 503
MessageProperties クラス [QAnywhere C++ API], 383
RAS フィールド
MessageProperties クラス [QAnywhere .NET API], 196
RAS 変数
MessageProperties インターフェイス [QAnywhere Java API], 502
MessageProperties クラス [QAnywhere C++ API], 383
readBinary メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 514
QABinaryMessage クラス [QAnywhere C++ API], 394
ReadBinary メソッド
QABinaryMessage インターフェイス [QAnywhere .NET API], 204
readBoolean メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 516
QABinaryMessage クラス [QAnywhere C++ API], 395
ReadBoolean メソッド
QABinaryMessage インターフェイス [QAnywhere .NET API], 206
readByte メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 517
QABinaryMessage クラス [QAnywhere C++ API], 395
readChar メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 517
QABinaryMessage クラス [QAnywhere C++ API], 395

ReadChar メソッド
QABinaryMessage インターフェイス [QAnywhere .NET API], 207
readDouble メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 518
QABinaryMessage クラス [QAnywhere C++ API], 396
ReadDouble メソッド
QABinaryMessage インターフェイス [QAnywhere .NET API], 207
readFloat メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 518
QABinaryMessage クラス [QAnywhere C++ API], 396
ReadFloat メソッド
QABinaryMessage インターフェイス [QAnywhere .NET API], 208
readInt メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 518
QABinaryMessage クラス [QAnywhere C++ API], 397
ReadInt メソッド
QABinaryMessage インターフェイス [QAnywhere .NET API], 208
readLong メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 519
QABinaryMessage クラス [QAnywhere C++ API], 397
ReadLong メソッド
QABinaryMessage インターフェイス [QAnywhere .NET API], 209
ReadSbyte メソッド
QABinaryMessage インターフェイス [QAnywhere .NET API], 209
readShort メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 519
QABinaryMessage クラス [QAnywhere C++ API], 397
ReadShort メソッド
QABinaryMessage インターフェイス [QAnywhere .NET API], 210
readString メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 520
QABinaryMessage クラス [QAnywhere C++ API], 398
ReadString メソッド
QABinaryMessage インターフェイス [QAnywhere .NET API], 210
readText メソッド
QATextMessage インターフェイス [QAnywhere Java API], 606
QATextMessage クラス [QAnywhere C++ API], 482
ReadText メソッド
QATextMessage インターフェイス [QAnywhere .NET API], 307
RECEIVED 変数
StatusCodes インターフェイス [QAnywhere Java API], 618
StatusCodes クラス [QAnywhere C++ API], 493
RECEIVING 変数
StatusCodes インターフェイス [QAnywhere Java API], 619
StatusCodes クラス [QAnywhere C++ API], 493
recover メソッド
QAManager インターフェイス [QAnywhere Java API], 543
QAManager クラス [QAnywhere C++ API], 418
Recover メソッド
QAManager インターフェイス [QAnywhere .NET API], 238
Redelivered プロパティ
QAMessage インターフェイス [QAnywhere .NET API], 303
Redelivered メッセージヘッダー
QAnywhere のメッセージヘッダー, 695
regular
QAnywhere の ias_MessageType, 699
REGULAR 変数
MessageType インターフェイス [QAnywhere Java API], 506
MessageType クラス [QAnywhere C++ API], 388
reOpen メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 566
ReOpen メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 265
ReplyToAddress プロパティ

QAMessage インターフェイス [QAnywhere .NET API], 303
ReplyToAddress メッセージヘッダー
QAnywhere のメッセージヘッダー, 695
reset メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 520
QABinaryMessage クラス [QAnywhere C++ API], 398
QATextMessage インターフェイス [QAnywhere Java API], 607
QATextMessage クラス [QAnywhere C++ API], 483
Reset メソッド
QABinaryMessage インターフェイス [QAnywhere .NET API], 211
QATextMessage インターフェイス [QAnywhere .NET API], 308
RestartRules タグ
QAnywhere サーバー管理要求, 156
rollback メソッド
QATransactionalManager インターフェイス [QAnywhere Java API], 614
QATransactionalManager クラス [QAnywhere C++ API], 489
Rollback メソッド
QATransactionalManager インターフェイス [QAnywhere .NET API], 314

S

scheduled ポリシー
QAnywhere Agent, 727
QAnywhere Ultra Light Agent, 752
schedule タグ
QAnywhere サーバー管理要求, 706
setAddress メソッド
QAMessage クラス [QAnywhere C++ API], 470
setBooleanProperty メソッド
QAMessage インターフェイス [QAnywhere Java API], 594
QAMessage クラス [QAnywhere C++ API], 471
SetBooleanProperty メソッド
QAMessage インターフェイス [QAnywhere .NET API], 295
setBooleanStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 567
QAManagerBase クラス [QAnywhere C++ API], 442
SetBooleanStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 266
setByteProperty メソッド
QAMessage インターフェイス [QAnywhere Java API], 595
QAMessage クラス [QAnywhere C++ API], 471
SetByteProperty メソッド
QAMessage インターフェイス [QAnywhere .NET API], 296
setByteStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 568
QAManagerBase クラス [QAnywhere C++ API], 443
setDoubleProperty メソッド
QAMessage インターフェイス [QAnywhere Java API], 595
QAMessage クラス [QAnywhere C++ API], 472
SetDoubleProperty メソッド
QAMessage インターフェイス [QAnywhere .NET API], 296
setDoubleStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 568
QAManagerBase クラス [QAnywhere C++ API], 443
SetDoubleStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 266
SetExceptionListener2 メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 267
SetExceptionListener メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 267
setFloatProperty メソッド
QAMessage インターフェイス [QAnywhere Java API], 595
QAMessage クラス [QAnywhere C++ API], 472
SetFloatProperty メソッド
QAMessage インターフェイス [QAnywhere .NET API], 297
setFloatStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 569

QAManagerBase クラス [QAnywhere C++ API], 444
SetFloatStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 268
setInReplyToID メソッド
QAMessage インターフェイス [QAnywhere Java API], 596
QAMessage クラス [QAnywhere C++ API], 473
setIntProperty メソッド
QAMessage インターフェイス [QAnywhere Java API], 596
QAMessage クラス [QAnywhere C++ API], 473
SetIntProperty メソッド
QAMessage インターフェイス [QAnywhere .NET API], 297
setIntStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 569
QAManagerBase クラス [QAnywhere C++ API], 444
SetIntStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 269
setListener メソッド
WSBase クラス [QAnywhere Java API], 623
SetListener メソッド
WSBase クラス [QAnywhere .NET API], 328
SetLogger メソッド
WSResult クラス [QAnywhere .NET API], 374
setLongProperty メソッド
QAMessage インターフェイス [QAnywhere Java API], 597
QAMessage クラス [QAnywhere C++ API], 474
SetLongProperty メソッド
QAMessage インターフェイス [QAnywhere .NET API], 298
setLongStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 570
QAManagerBase クラス [QAnywhere C++ API], 445
SetLongStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 269
setMessageID メソッド
QAMessage クラス [QAnywhere C++ API], 474
setMessageListener2 メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 571
SetMessageListener2 メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 271
setMessageListenerBySelector2 メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 573
SetMessageListenerBySelector2 メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 273
setMessageListenerBySelector メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 572
QAManagerBase クラス [QAnywhere C++ API], 446
SetMessageListenerBySelector メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 272
setMessageListener メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 571
QAManagerBase クラス [QAnywhere C++ API], 445
SetMessageListener メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 270
setPriority メソッド
QAMessage インターフェイス [QAnywhere Java API], 597
QAMessage クラス [QAnywhere C++ API], 474
SetProperty タグ
QAnywhere サーバー管理要求, 168
setProperty メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 573
QAManagerBase クラス [QAnywhere C++ API], 447
QAMessage インターフェイス [QAnywhere Java API], 597
WSBase クラス [QAnywhere Java API], 625
SetProperty メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 274
QAMessage インターフェイス [QAnywhere .NET API], 299
WSBase クラス [QAnywhere .NET API], 330
setQAManager メソッド

WSBase クラス [QAnywhere Java API], 625
SetQAManager メソッド
WSBase クラス [QAnywhere .NET API], 330
setRedelivered メソッド
QAMessage クラス [QAnywhere C++ API], 475
setReplyToAddress メソッド
QAMessage インターフェイス [QAnywhere Java API], 598
QAMessage クラス [QAnywhere C++ API], 475
setRequestProperty メソッド
WSBase クラス [QAnywhere Java API], 626
SetRequestProperty メソッド
WSBase クラス [QAnywhere .NET API], 331
SetSbyteProperty メソッド
QAMessage インターフェイス [QAnywhere .NET API], 299
SetSbyteStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 274
setServiceID メソッド
WSBase クラス [QAnywhere Java API], 626
SetServiceID メソッド
WSBase クラス [QAnywhere .NET API], 331
setShortProperty メソッド
QAMessage インターフェイス [QAnywhere Java API], 598
QAMessage クラス [QAnywhere C++ API], 475
SetShortProperty メソッド
QAMessage インターフェイス [QAnywhere .NET API], 300
setShortStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 574
QAManagerBase クラス [QAnywhere C++ API], 447
SetShortStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 275
setStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 575
SetStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 276
setStringProperty メソッド
QAMessage インターフェイス [QAnywhere Java API], 599
QAMessage クラス [QAnywhere C++ API], 476
SetStringProperty メソッド
QAMessage インターフェイス [QAnywhere .NET API], 300
setStringStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 575
QAManagerBase クラス [QAnywhere C++ API], 448
SetStringStoreProperty メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 276
setText メソッド
QATextMessage インターフェイス [QAnywhere Java API], 607
QATextMessage クラス [QAnywhere C++ API], 483
setTimestamp メソッド
QAMessage クラス [QAnywhere C++ API], 476
SQL ストアドプロシージャー
QAnywhere, 59
start メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 576
QAManagerBase クラス [QAnywhere C++ API], 449
Start メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 277
STATUS_ERROR 変数
WSStatus クラス [QAnywhere Java API], 654
STATUS_QUEUED 変数
WSStatus クラス [QAnywhere Java API], 654
STATUS_RESULT_AVAILABLE 変数
WSStatus クラス [QAnywhere Java API], 654
STATUS_SUCCESS 変数
WSStatus クラス [QAnywhere Java API], 654
STATUS_TIME フィールド
MessageProperties クラス [QAnywhere .NET API], 197
STATUS_TIME 変数
MessageProperties インターフェイス [QAnywhere Java API], 503
MessageProperties クラス [QAnywhere C++ API], 384
StatusCodes インターフェイス [QAnywhere Java API]
CANCELED 変数, 617
EXPIRED 変数, 617

FINAL 変数, 617
LOCAL 変数, 618
PENDING 変数, 618
RECEIVED 変数, 618
RECEIVING 変数, 619
TRANSMITTED 変数, 619
TRANSMITTING 変数, 619
UNRECEIVABLE 変数, 620
UNTRANSMITTED 変数, 620
説明, 616
StatusCodes クラス [QAnywhere C++ API]
CANCELED 変数, 492
EXPIRED 変数, 492
FINAL 変数, 492
LOCAL 変数, 493
PENDING 変数, 493
RECEIVED 変数, 493
RECEIVING 変数, 493
TRANSMITTED 変数, 494
TRANSMITTING 変数, 494
UNRECEIVABLE 変数, 494
UNTRANSMITTED 変数, 494
説明, 491
StatusCodes 列挙体 [QAnywhere .NET API]
説明, 322
STATUS フィールド
MessageProperties クラス [QAnywhere .NET API], 197
STATUS 変数
MessageProperties インターフェイス [QAnywhere Java API], 503
MessageProperties クラス [QAnywhere C++ API], 384
stop メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 577
QAManagerBase クラス [QAnywhere C++ API], 449
Stop メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 277
SUBSTRING 関数
QAnywhere 構文, 779

T

TestMessage アプリケーション
QAnywhere チュートリアル, 177
ソースコード, 183
TextLength プロパティ
QATextMessage インターフェイス [QAnywhere .NET API], 309
Text プロパティ
QATextMessage インターフェイス [QAnywhere .NET API], 308
Timestamp プロパティ
QAMessage インターフェイス [QAnywhere .NET API], 304
Timestamp メッセージヘッダー
QAnywhere のメッセージヘッダー, 695
TRANSACTIONAL 変数
AcknowledgementMode インターフェイス [QAnywhere Java API], 496
AcknowledgementMode クラス [QAnywhere C++ API], 377
TRANSMISSION_STATUS フィールド
MessageProperties クラス [QAnywhere .NET API], 198
TRANSMISSION_STATUS 変数
MessageProperties インターフェイス [QAnywhere Java API], 504
MessageProperties クラス [QAnywhere C++ API], 385
TRANSMITTED 変数
StatusCodes インターフェイス [QAnywhere Java API], 619
StatusCodes クラス [QAnywhere C++ API], 494
TRANSMITTING 変数
StatusCodes インターフェイス [QAnywhere Java API], 619
StatusCodes クラス [QAnywhere C++ API], 494
triggerSendReceive メソッド
QAManagerBase インターフェイス [QAnywhere Java API], 577
QAManagerBase クラス [QAnywhere C++ API], 450
TriggerSendReceive メソッド
QAManagerBase インターフェイス [QAnywhere .NET API], 278

U

UNRECEIVABLE 変数
StatusCodes インターフェイス [QAnywhere Java API], 620
StatusCodes クラス [QAnywhere C++ API], 494

UNTRANSMITTED 変数
StatusCodes インターフェイス [QAnywhere Java API], 620
StatusCodes クラス [QAnywhere C++ API], 494

W

WebLogic
QAnywhere, 9
webservice.http.authName プロパティ
モバイル Web サービスコネクター, 147
webservice.http.password.e プロパティ
モバイル Web サービスコネクター, 147
webservice.http.proxy.authName プロパティ
モバイル Web サービスコネクター, 147
webservice.http.proxy.host プロパティ
モバイル Web サービスコネクター, 147
webservice.http.proxy.password.e プロパティ
モバイル Web サービスコネクター, 147
webservice.http.proxy.port プロパティ
モバイル Web サービスコネクター, 147
webservice.url プロパティ
モバイル Web サービスコネクター, 144
Web サービス
QAnywhere：Web サービスコネクター, 144
QAnywhere：Web サービスコネクターの作成, 144
QAnywhere：Web サービスコネクタープロパティの設定, 146
QAnywhere、説明, 95
Web サービスコネクターの設定
モバイル Web サービス, 144
Web サービス要求の作成
モバイル Web サービス, 105
writeBinary メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 520
QABinaryMessage クラス [QAnywhere C++ API], 399
WriteBinary メソッド
QABinaryMessage インターフェイス [QAnywhere .NET API], 211
writeBoolean メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 523
QABinaryMessage クラス [QAnywhere C++ API], 399
WriteBoolean メソッド
QABinaryMessage インターフェイス [QAnywhere .NET API], 214
writeByte メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 523
QABinaryMessage クラス [QAnywhere C++ API], 399
writeChar メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 524
QABinaryMessage クラス [QAnywhere C++ API], 400
WriteChar メソッド
QABinaryMessage インターフェイス [QAnywhere .NET API], 214
writeDouble メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 524
QABinaryMessage クラス [QAnywhere C++ API], 400
WriteDouble メソッド
QABinaryMessage インターフェイス [QAnywhere .NET API], 215
writeFloat メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 525
QABinaryMessage クラス [QAnywhere C++ API], 401
WriteFloat メソッド
QABinaryMessage インターフェイス [QAnywhere .NET API], 215
writeInt メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 525
QABinaryMessage クラス [QAnywhere C++ API], 401
WriteInt メソッド
QABinaryMessage インターフェイス [QAnywhere .NET API], 215
writeLong メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 526
QABinaryMessage クラス [QAnywhere C++ API], 401
WriteLong メソッド
QABinaryMessage インターフェイス [QAnywhere .NET API], 216
WriteSbyte メソッド

QABinaryMessage インターフェイス [QAnywhere .NET API], 216
writeShort メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 526
QABinaryMessage クラス [QAnywhere C++ API], 402
WriteShort メソッド
QABinaryMessage インターフェイス [QAnywhere .NET API], 217
writeString メソッド
QABinaryMessage インターフェイス [QAnywhere Java API], 527
QABinaryMessage クラス [QAnywhere C++ API], 402
WriteString メソッド
QABinaryMessage インターフェイス [QAnywhere .NET API], 217
writeText メソッド
QATextMessage インターフェイス [QAnywhere Java API], 607
QATextMessage クラス [QAnywhere C++ API], 483
WriteText メソッド
QATextMessage インターフェイス [QAnywhere .NET API], 308
WS_STATUS_HTTP_ERROR フィールド
WSException クラス [QAnywhere .NET API], 335
WS_STATUS_HTTP_ERROR 変数
WSException クラス [QAnywhere Java API], 629
WS_STATUS_HTTP_OK フィールド
WSException クラス [QAnywhere .NET API], 336
WS_STATUS_HTTP_OK 変数
WSException クラス [QAnywhere Java API], 629
WS_STATUS_HTTP_RETRIES_EXCEEDED フィールド
WSException クラス [QAnywhere .NET API], 336
WS_STATUS_HTTP_RETRIES_EXCEEDED 変数
WSException クラス [QAnywhere Java API], 629
WS_STATUS_SOAP_PARSE_ERROR フィールド
WSException クラス [QAnywhere .NET API], 336
WS_STATUS_SOAP_PARSE_ERROR 変数
WSException クラス [QAnywhere Java API], 629
WSBase クラス [QAnywhere .NET API]
ClearRequestProperties メソッド, 327
GetResult メソッド, 328
GetServiceID メソッド, 328
SetListener メソッド, 328
SetProperty メソッド, 330
SetQAManager メソッド, 330
SetRequestProperty メソッド, 331
SetServiceID メソッド, 331
WSBase コンストラクター, 326
説明, 325
WSBase クラス [QAnywhere Java API]
clearRequestProperties メソッド, 623
getResult メソッド, 623
getServiceID メソッド, 623
setListener メソッド, 623
setProperty メソッド, 625
setQAManager メソッド, 625
setRequestProperty メソッド, 626
setServiceID メソッド, 626
WSBase コンストラクター, 621
説明, 621
WSBase コンストラクター
WSBase クラス [QAnywhere .NET API], 326
WSBase クラス [QAnywhere Java API], 621
WSDLC
QAnywhere、説明, 97
WSDL コンパイラー
QAnywhere, 97
QAnywhere、説明, 97
WSException クラス [QAnywhere .NET API]
ErrorCode プロパティ, 335
WS_STATUS_HTTP_ERROR フィールド, 335
WS_STATUS_HTTP_OK フィールド, 336
WS_STATUS_HTTP_RETRIES_EXCEEDED フィールド, 336
WS_STATUS_SOAP_PARSE_ERROR フィールド, 336
WSException コンストラクター, 334
説明, 332
WSException クラス [QAnywhere Java API]
getErrorCode メソッド, 629
WS_STATUS_HTTP_ERROR 変数, 629
WS_STATUS_HTTP_OK 変数, 629
WS_STATUS_HTTP_RETRIES_EXCEEDED 変数, 629
WS_STATUS_SOAP_PARSE_ERROR 変数, 629
WSException コンストラクター, 627
説明, 627

WSException コンストラクター
WSException クラス [QAnywhere .NET API], 334
WSException クラス [QAnywhere Java API], 627
WSFaultException クラス [QAnywhere .NET API]
WSFaultException コンストラクター, 338
説明, 336
WSFaultException クラス [QAnywhere Java API]
WSFaultException コンストラクター, 630
説明, 630
WSFaultException コンストラクター
WSFaultException クラス [QAnywhere .NET API], 338
WSFaultException クラス [QAnywhere Java API], 630
WSListener インターフェイス [QAnywhere .NET API]
OnException メソッド, 339
OnResult メソッド, 340
説明, 339
WSListener インターフェイス [QAnywhere Java API]
onException メソッド, 631
onResult メソッド, 632
説明, 631
WSResult クラス [QAnywhere .NET API]
Acknowledge メソッド, 344
GetArrayValue メソッド, 344
GetBoolArrayValue メソッド, 345
GetBooleanArrayValue メソッド, 345
GetBooleanValue メソッド, 346
GetBoolValue メソッド, 346
GetByteArrayValue メソッド, 347
GetByteValue メソッド, 347
GetCharArrayValue メソッド, 348
GetCharValue メソッド, 348
GetDecimalArrayValue メソッド, 349
GetDecimalValue メソッド, 349
GetDoubleArrayValue メソッド, 350
GetDoubleValue メソッド, 350
GetErrorMessage メソッド, 351
GetFloatArrayValue メソッド, 351
GetFloatValue メソッド, 351
GetInt16ArrayValue メソッド, 352
GetInt16Value メソッド, 352
GetInt32ArrayValue メソッド, 353
GetInt32Value メソッド, 353
GetInt64ArrayValue メソッド, 354
GetInt64Value メソッド, 354
GetIntArrayValue メソッド, 355
GetIntValue メソッド, 355
GetLongArrayValue メソッド, 356
GetLongValue メソッド, 356
GetNullableBoolArrayValue メソッド, 357
GetNullableBoolValue メソッド, 357
GetNullableDecimalArrayValue メソッド, 358
GetNullableDecimalValue メソッド, 358
GetNullableDoubleArrayValue メソッド, 359
GetNullableDoubleValue メソッド, 359
GetNullableFloatArrayValue メソッド, 360
GetNullableFloatValue メソッド, 360
GetNullableIntArrayValue メソッド, 361
GetNullableIntValue メソッド, 361
GetNullableLongArrayValue メソッド, 362
GetNullableLongValue メソッド, 362
GetNullableSByteArrayValue メソッド, 363
GetNullableSByteValue メソッド, 363
GetNullableShortArrayValue メソッド, 364
GetNullableShortValue メソッド, 364
GetObjectArrayValue メソッド, 365
GetObjectValue メソッド, 365
GetRequestID メソッド, 366
GetSByteArrayValue メソッド, 366
GetSByteValue メソッド, 367
GetShortArrayValue メソッド, 367
GetShortValue メソッド, 368
GetSingleArrayValue メソッド, 368
GetSingleValue メソッド, 369
GetStatus メソッド, 369
GetStringArrayValue メソッド, 370
GetStringValue メソッド, 370
GetUIntArrayValue メソッド, 371
GetUIntValue メソッド, 371
GetULongArrayValue メソッド, 372
GetULongValue メソッド, 372
GetUShortArrayValue メソッド, 373
GetUShortValue メソッド, 373
GetValue メソッド, 374
SetLogger メソッド, 374
説明, 340
WSResult クラス [QAnywhere Java API]
acknowledge メソッド, 635
getArrayValue メソッド, 635
getBigDecimalArrayValue メソッド, 636
getBigDecimalValue メソッド, 636
getBigIntegerArrayValue メソッド, 636

getBigIntegerValue メソッド, 637
getBooleanArrayValue メソッド, 637
getBooleanValue メソッド, 638
getByteArrayValue メソッド, 638
getByteValue メソッド, 638
getCharacterArrayValue メソッド, 639
getCharacterValue メソッド, 639
getDoubleArrayValue メソッド, 640
getDoubleValue メソッド, 640
getErrorMessage メソッド, 641
getFloatArrayValue メソッド, 641
getFloatValue メソッド, 641
getIntegerArrayValue メソッド, 642
getIntegerValue メソッド, 642
getLongArrayValue メソッド, 642
getLongValue メソッド, 643
getObjectArrayValue メソッド, 643
getObjectValue メソッド, 644
getPrimitiveBooleanArrayValue メソッド, 644
getPrimitiveBooleanValue メソッド, 644
getPrimitiveByteArrayValue メソッド, 645
getPrimitiveByteValue メソッド, 645
getPrimitiveCharArrayValue メソッド, 646
getPrimitiveCharValue メソッド, 646
getPrimitiveDoubleArrayValue メソッド, 646
getPrimitiveDoubleValue メソッド, 647
getPrimitiveFloatArrayValue メソッド, 647
getPrimitiveFloatValue メソッド, 648
getPrimitiveIntArrayValue メソッド, 648
getPrimitiveIntValue メソッド, 649
getPrimitiveLongArrayValue メソッド, 649
getPrimitiveLongValue メソッド, 649
getPrimitiveShortArrayValue メソッド, 650
getPrimitiveShortValue メソッド, 650
getRequestID メソッド, 651
getShortArrayValue メソッド, 651
getShortValue メソッド, 651
getStatus メソッド, 652
getStringArrayValue メソッド, 652
getStringValue メソッド, 653
getValue メソッド, 653
説明, 632
WSStatus クラス [QAnywhere Java API]
STATUS_ERROR 変数, 654
STATUS_QUEUED 変数, 654
STATUS_RESULT_AVAILABLE 変数, 654
STATUS_SUCCESS 変数, 654
説明, 653
WSStatus 列挙体 [QAnywhere .NET API]
説明, 374

X

xjms.jndi.authName プロパティ
QAnywhere JMS コネクター, 772
xjms.jndi.factory プロパティ
QAnywhere JMS コネクター, 772
xjms.jndi.password.e プロパティ
QAnywhere JMS コネクター, 772
xjms.jndi.url プロパティ
QAnywhere JMS コネクター, 773
xjms.password.e プロパティ
QAnywhere JMS コネクター, 773
xjms.queueConnectionAuthName プロパティ
QAnywhere JMS コネクター, 773
xjms.queueConnectionPassword.e プロパティ
QAnywhere JMS コネクター, 773
xjms.queueFactory プロパティ
QAnywhere JMS コネクター, 773
xjms.receiveDestination プロパティ
QAnywhere JMS コネクター, 773
xjms.topicConnectionAuthName プロパティ
QAnywhere JMS コネクター, 773
xjms.topicConnectionPassword.e プロパティ
QAnywhere JMS コネクター, 774
xjms.topicFactory プロパティ
QAnywhere JMS コネクター, 773

あ

アダプター
QAnywhere のメッセージプロパティ, 62
圧縮
QAnywhere JMS コネクター, 771
QAnywhere Web サービスコネクター, 146
アップグレード
QAnywhere [qaagent] -sur オプション, 734
QAnywhere [qaagent] -su オプション, 733
QAnywhere [qauagent] -su オプション, 757
アドレス
QAnywhere, 60
QAnywhere JMS コネクター, 61
QAnywhere メッセージの設定 (.NET), 63
QAnywhere メッセージの設定 (C++), 64
QAnywhere メッセージの設定 (Java), 65
アプリケーション間メッセージング
QAnywhere, 1

QAnywhere -sil オプション, 733
QAnywhere アーキテクチャー, 4
QAnywhere の例, 4
アプリケーションでのクライアントメッセージストアプロパティの管理
説明, 766
暗号化
QAnywhere クライアントメッセージストア, 122
QAnywhere 通信ストリーム, 123
アーカイブメッセージストア
QAnywhere, 24
アーカイブメッセージストア要求
QAnywhere サーバー管理要求, 155
アーキテクチャー
QAnywhere, 3

い

インクリメンタルアップロード
qaagent, 49
SQL Anywhere クライアントメッセージストアの QAnywhere メッセージ送信, 720
Ultra Light クライアントメッセージストアの QAnywhere メッセージ送信, 746
インクリメンタルダウンロード
qaagent, 49

え

永続的接続
qaagent -pc オプション, 726
エラー処理
QAnywhere, 80
エージェント設定ファイル
説明, 49
エージェント設定ファイルの新規作成
Sybase Central タスク, 86
エージェントルールファイル
説明, 49

お

親タグ
QAnywhere, 704

か

カスタムのメッセージストアプロパティ
QAnywhere, 762
カスタムのメッセージストアプロパティ属性
QAnywhere, 763
カスタムのメッセージプロパティ
QAnywhere, 700
関数
QAnywhere ストアドプロシージャー, 59
QAnywhere ルール, 779
簡単なメッセージング
QAnywhere アーキテクチャー, 5
QAnywhere の例, 5
管理
QAnywhere サーバーメッセージストア, 153

き

キュー
QAnywhere アドレスについて, 60

く

クイックスタート
QAnywhere, 12
モバイル Web サービス, 95
クライアント側の転送ルール
削除ルール, 786
クライアントステータスレポート
QAnywhere コネクターのサーバー管理要求, 167
クライアント転送ルールの更新
QAnywhere サーバー管理要求, 156
クライアントメッセージストア
ID の作成, 26
QAnywhere アーキテクチャー, 6
QAnywhere の暗号化, 122
QAnywhere のセキュリティ, 121
QAnywhere のプロパティ, 761
QAnywhere のプロパティの違い, 29
-si オプションによる QAnywhere SQL Anywhere クライアントメッセージストアの初期化, 731
-si オプションによる QAnywhere Ultra Light クライアントメッセージストアの初期化, 756
カスタムのメッセージストアプロパティ, 762
作成, 25
事前に定義されたメッセージストアプロパティ, 761
説明, 25
通信ストリームの暗号化, 123
パスワード, 122
クライアントメッセージストア ID

QAnywhere の説明, 26
クライアントメッセージストア ID の管理
パスワード, 122
クライアントメッセージストア ID の作成
QAnywhere, 26
クライアントメッセージストアの作成
QAnywhere, 121
クライアントメッセージストアプロパティ
QAnywhere の管理, 131
QAnywhere の属性, 763
クライアントメッセージストアプロパティの管理
QAnywhere, 131
クライアントメッセージストアを使用
Sybase Central タスク, 34
クライアントユーザー名
QAnywhere のサーバーメッセージストアへの追加, 32
クライアントユーザー名の登録
QAnywhere, 32
クライアント用 QAnywhere C++ API リファレンス
qa.hpp ヘッダーファイル, 375
クワイエットモード
QAnywhere Agent ユーティリティ (qaagent) -q, 730
QAnywhere Agent ユーティリティ (qaagent) -qi, 731
QAnywhere Ultra Light Agent ユーティリティ (qauagent) -q, 755
QAnywhere Ultra Light Agent ユーティリティ (qauagent) -qi, 755

け

ゲートウェイの設定
QAnywhere, 38

こ

コネクター
JMS 用の QAnywhere アドレス, 61
QAnywhere, 135
QAnywhere JMS コネクタープロパティ, 138
QAnywhere JMS での複数設定, 139
QAnywhere Web サービスコネクターのプロパティ, 146
QAnywhere サーバー管理要求, 159
QAnywhere モバイル Web サービス, 144
QAnywhere を閉じる, 162
QAnywhere を開く, 161
コネクターの管理
QAnywhere サーバー管理要求, 159
コネクターの削除
QAnywhere サーバー管理要求, 161
コネクターの作成と設定
QAnywhere サーバー管理要求, 159
コネクターの変更
QAnywhere サーバー管理要求, 160
コネクターのモニター
QAnywhere サーバー管理要求, 162
コネクターを閉じる
QAnywhere サーバー管理要求, 162
コネクターを開く
QAnywhere サーバー管理要求, 161
コマンドラインユーティリティ
QAnywhere Agent (qaagent) の構文, 712
QAnywhere Ultra Light Agent (qauagent) の構文, 738
コンパイル
QAnywhere、モバイル Web サービスアプリケーションの実行, 104

さ

削除
QAnywhere メッセージ, 786
削除ルール
QAnywhere, 786
作成
ml_qa_createmessage を使用した QAnywhere メッセージの作成, 687
QAnywhere サーバーメッセージストア, 24
参照
QAnywhere メッセージ, 76
サーバー管理要求
QAnywhere, 153
QAnywhere でのアドレス指定, 153
QAnywhere での認証, 124
QAnywhere でのフォーマット, 154
サーバー管理要求 DTD
QAnywhere, 710
サーバープロパティ
Sybase Central での QAnywhere の設定, 770
サーバー管理要求での QAnywhere の設定, 168
サーバーメッセージストア
QAnywhere, 23

QAnywhere アーキテクチャー, 6
QAnywhere クライアントプロパティ, 768
QAnywhere での設定, 24
QAnywhere のプロパティ, 768
Sybase Central でのプロパティの設定, 770
サーバー管理要求での QAnywhere の管理, 156
サーバー管理要求でのプロパティの設定, 168
説明, 23

し

システム関数
QAnywhere SQL API, 655
システムキュー
QAnywhere の説明, 61
システムキューメッセージ
QAnywhere, 61
システムメッセージ
QAnywhere, 61
事前に定義されたメッセージストアプロパティ
QAnywhere, 761
事前に定義されたメッセージプロパティ
QAnywhere, 698
持続性
QAnywhere メッセージ, 786
実行
QAnywhere モバイル Web サービス, 104
受信確認モード
QAManager クラス (.NET), 55
QAManager クラス (C++), 56
QAManager クラス (Java), 58
QAnywhere SQL API, 59
Web サービス用の QAManager クラス (.NET), 99
Web サービス用の QAManager クラス (Java), 102
条件
QAnywhere スケジュール構文, 776
条件構文
QAnywhere, 776
冗長性
QAnywhere [qaagent] -v オプション, 735
QAnywhere [qauagent] -v オプション, 757
初期化
QAnywhere SQL Anywhere クライアントメッセージストア, 731
QAnywhere Ultra Light クライアントメッセージストア, 756
処理
QAnywhere Push 通知とネットワークステータスの変化, 61
QAnywhere 例外の説明, 80

す

スケジュール
QAnywhere 転送ルール, 774
スケジュール構文
QAnywhere 転送ルール, 774
ストア ID
QAnywhere の説明, 26
ストアドプロシージャー
ml_qa_createmessage, 687
ml_qa_getaddress, 655
ml_qa_getbinarycontent, 681
ml_qa_getbooleanproperty, 665
ml_qa_getbyteproperty, 666
ml_qa_getcontentclass, 682
ml_qa_getdoubleproperty, 667
ml_qa_getexpiration, 656
ml_qa_getfloatproperty, 668
ml_qa_getinreplytoid, 657
ml_qa_getintproperty, 669
ml_qa_getlongproperty, 670
ml_qa_getmessage, 687
ml_qa_getmessagenowait, 689
ml_qa_getmessagetimeout, 690
ml_qa_getpriority, 658
ml_qa_getpropertynames, 671
ml_qa_getredelivered, 658
ml_qa_getreplytoaddress, 660
ml_qa_getshortproperty, 672
ml_qa_getstoreproperty, 685
ml_qa_getstringproperty, 673
ml_qa_gettextcontent, 683
ml_qa_gettimestamp, 661
ml_qa_grant_messaging_permissions, 691
ml_qa_listener_queue, 692
ml_qa_putmessage, 693
ml_qa_setbinarycontent, 684
ml_qa_setbooleanproperty, 674
ml_qa_setbyteproperty, 675
ml_qa_setdoubleproperty, 676
ml_qa_setexpiration, 662
ml_qa_setfloatproperty, 677
ml_qa_setinreplytoid, 663

ml_qa_setintproperty, 677
ml_qa_setlongproperty, 678
ml_qa_setpriority, 664
ml_qa_setreplytoaddress, 664
ml_qa_setstoreproperty, 686
ml_qa_setstringproperty, 680
ml_qa_settextcontent, 684
ml_qa_triggersendreceive, 694
QAnywhere, 59
ストアドプロシージャー
ml_qa_setshortproperty, 679

せ

セキュリティ
QAnywhere, 121
接続
QAnywhere SQL Anywhere クライアントメッセージストア, 715
QAnywhere Ultra Light クライアントメッセージストア, 741
接続文字列
QAnywhere SQL Anywhere クライアントメッセージストア, 715
QAnywhere Ultra Light クライアントメッセージストア, 741
設定
QAnywhere, 12
QAnywhere Java モバイル Web サービスアプリケーション, 101
QAnywhere JMS コネクタープロパティ, 138
QAnywhere Web サービスコネクターのプロパティ, 146
QAnywhere クライアントサイドコンポーネント, 34
QAnywhere クライアントメッセージストア, 25
QAnywhere サーバーサイドコンポーネント, 31
QAnywhere サーバーメッセージストア, 24
QAnywhere の Push 通知, 35
QAnywhere フェールオーバー, 38
QAnywhere メッセージング、説明, 31
QAnywhere モバイル Web サービス, 95

そ

送信先エイリアス
QAnywhere, 133
作成, 133
サーバー管理要求を使用した QAnywhere での作成, 156
サーバー管理要求を使用した作成, 170
送信先エイリアスの作成
QAnywhere, 133

ち

チュートリアル
QAnywhere JMS コネクター, 148
QAnywhere TestMessage, 177

つ

通信ストリーム
QAnywhere での暗号化, 123
通知
QAnywhere SQL Anywhere クライアントメッセージストア, 729
QAnywhere Ultra Light クライアントメッセージストア, 753
QAnywhere での処理, 61
QAnywhere の概要, 34

て

停止
QAnywhere, 84
QAnywhere モバイル Web サービス, 104
転送ルール
QAnywhere クライアントの説明, 783
QAnywhere サーバー管理要求での更新, 156
QAnywhere サーバーの説明, 784
QAnywhere ルールの構文, 774
概要, 46
クライアント管理要求での指定, 169
削除ルール, 786
デフォルトルール, 784
転送ルールファイルでの指定, 784
変数, 780
メッセージストアプロパティ, 764
転送ルール関数
QAnywhere, 779
転送ルール変数
QAnywhere, 780

と

同期的なメッセージ受信
QAnywhere, 71
同期の Web サービス要求

モバイル Web サービス, 105
動的アドレス指定
QAnywhere Agent ユーティリティ (qaagent), 737
QAnywhere Ultra Light Agent ユーティリティ (qauagent), 759
トランザクション
QAnywhere メッセージ, 66
トランザクション指向メッセージング
QAnywhere, 66

に

認証
QAnywhere, 123

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, 61
QAnywhere のメッセージプロパティ, 62
ネットワークステータス通知メッセージタイプ
QAnywhere システムキュー, 62
ネットワークの可用性
QAnywhere システムキューメッセージ, 62
QAnywhere のカスタムメッセージストアプロパティ, 763
ネットワークの可用性のモニタリング
QAnywhere システムキューメッセージ, 62
ネットワークプロパティ属性
QAnywhere クライアント, 763

は

配信条件構文
QAnywhere, 776
配備
QAnywhere アプリケーション, 117
はじめに
QAnywhere, 12

ひ

非同期的なメッセージ受信
QAnywhere, 73
非同期の Web サービス要求
モバイル Web サービス, 105

ふ

フェールオーバー
QAnywhere Agent -fd オプション, 717
QAnywhere Agent -fr オプション, 718
QAnywhere Ultra Light Agent -fd オプション, 743
QAnywhere Ultra Light Agent -fr オプション, 744
フェールオーバー
QAnywhere, 38
複数のコネクターの設定
QAnywhere, 139
プラグイン
QAnywhere, 11
プログラミングインターフェイス
QAnywhere, 51
プロパティ
QAnywhere Manager の設定, 85
QAnywhere のクライアントメッセージストアプロパティ, 761
QAnywhere のクライアントメッセージストアプロパティの違い, 29
QAnywhere のサーバーメッセージストアプロパティ, 768
QAnywhere のメッセージプロパティ, 698
プロパティの設定
QAnywhere QAManager、ファイル内, 86
QAnywhere QAManager、プログラム, 88

へ

ヘッダー
QAnywhere のメッセージヘッダー, 695
変数
QAnywhere 転送ルール, 780

ほ

ポリシー
QAnywhere, 49
QAnywhere アーキテクチャー, 5
QAnywhere チュートリアル, 181

ま

マルチスレッド
QAnywhere QAManager, 84

め

メッセージ
QAnywhere メッセージの送信, 63
メッセージアドレス
QAnywhere, 60

メッセージストア
QAnywhere クライアントアーキテクチャー, 6
QAnywhere クライアントプロパティ, 761
QAnywhere クライアントプロパティの違い, 29
QAnywhere クライアントメッセージストアの暗号化, 122
QAnywhere クライアントメッセージストアの作成, 25
QAnywhere サーバーアーキテクチャー, 6
QAnywhere サーバーメッセージストアの作成, 24
ローカルの QAnywhere アーキテクチャー, 4
メッセージストア ID
QAnywhere の説明, 26
QAnywhere のメッセージストアプロパティ, 762
メッセージストアプロパティ
QAnywhere カスタムのクライアント, 762
QAnywhere クライアント, 29
QAnywhere クライアントの管理, 131
QAnywhere クライアントの説明, 761
QAnywhere サーバーの説明, 768
QAnywhere 事前に定義された, 761
メッセージセレクター
QAnywhere, 76
メッセージタイプ
QAnywhere, 699
メッセージ転送ルール
説明, 783
メッセージのアドレス指定
JMS, 139
JMS から QAnywhere に送信, 142
メッセージのキャンセル
QAnywhere (.NET), 70
QAnywhere (C++), 70
QAnywhere (Java), 71
QAnywhere サーバー管理要求, 157
QAnywhere の説明, 70
メッセージの削除
QAnywhere サーバー管理要求, 158
メッセージの参照
QAnywhere, 76
メッセージの受信
QAnywhere、同期的, 71
QAnywhere の説明, 71
QAnywhere、非同期的, 73
メッセージの詳細要求
QAnywhere 説明, 173
メッセージの送信
QAnywhere, 63
QAnywhere JMS コネクター, 140
QAnywhere のトランザクション指向メッセージングの実装 (.NET), 67
QAnywhere のトランザクション指向メッセージングの実装 (C++), 68
QAnywhere のトランザクション指向メッセージングの実装 (Java), 69
メッセージの同期的受信
QAnywhere, 71
メッセージの非同期的受信
QAnywhere, 73
メッセージのマッピング
QAnywhere JMS, 143
メッセージの読み込み
QAnywhere, 75
メッセージプロパティ
QAnywhere の管理, 701
QAnywhere の説明, 698
メッセージプロパティの管理
QAnywhere, 701
メッセージヘッダー
QAnywhere の説明, 695
メッセージリスナー
QAnywhere, 73
メッセージング
QAnywhere アドレス, 60
QAnywhere クイックスタート, 12
QAnywhere の機能, 2
アプリケーション間, 1
外部メッセージングシステムを使用した QAnywhere, 9
メッセージングシステム
JMS と QAnywhere の統合, 135
メッセージング対応 Mobile Link
QAnywhere チュートリアル, 177
QAnywhere の設定, 31
簡単なメッセージングアーキテクチャー, 6
起動, 31

も

モバイル Web サービス
QAnywhere、サービスアプリケーションの記述, 98
QAnywhere、説明, 95
例, 107

モバイル Web サービスコネクター
QAnywhere, 144
モバイル Web サービスの停止
説明, 104

ゆ

ユーティリティ
QAnywhere Agent (qaagent) の構文, 712
QAnywhere Ultra Light Agent (qauagent) の構文, 738
QAnywhere のエージェント停止ユーティリティ (qastop), 760

よ

読み込み
サイズの大きな QAnywhere メッセージ, 75

ら

ランタイムライブラリ
QAnywhere モバイル Web サービス, 104

る

ルール関数
QAnywhere, 779
ルールの構文
QAnywhere 転送ルール, 774
ルールファイル
QAnywhere Agent -policy オプション, 728
QAnywhere Ultra Light Agent -policy オプション, 753
QAnywhere クライアント側の転送ルール, 783
QAnywhere サーバー側の転送ルール, 784
ルール変数
QAnywhere 転送ルール, 780

れ

例外
QAnywhere, 80

ろ

ロギング
QAnywhere Agent, 723
QAnywhere Ultra Light Agent, 749
QAnywhere サーバー, 33
ログファイル
QAnywhere サーバーの表示, 33
ログファイルビューアー
QAnywhere サーバーのログ, 33
ローカルメッセージストア
QAnywhere, 19
QAnywhere アーキテクチャー, 4
説明, 19