オフラインアプリケーション
開発ガイド
Magic xpa

本マニュアルに記載の内容は、将来予告なしに変更することがあります。これらの情報について MSE（Magic Software Enterprises Ltd.）および MSJ（Magic Software Japan K.K.）は、いかなる責任も負いません。

本マニュアルの内容につきましては、万全を期して作成していますが、万一誤りや不正確な記述があったとしても、MSE および MSJ はいかなる責任、債務も負いません。

MSE および MSJ は、この製品の商業価値や特定の用途に対する適合性の保証を含め、この製品に関する明示的、あるいは黙示的な保証は一切していません。

本マニュアルに記載のソフトウェアは、製品の使用許諾契約書に記載の条件に同意をされたライセンス所有者に対してのみ供給されるものです。同ライセンスの許可する条件のもとでのみ、使用または複製することが許されます。

当該ライセンスが特に許可している場合を除いては、いかなる媒体へも複製することはできません。ライセンス所有者自身の個人使用目的で行う場合を除き、MSE または MSJ の書面による事前の許可なしでは、いかなる条件下でも、本マニュアルのいかなる部分も、電子的、機械的、撮影、録音、その他のいかなる手段によっても、コピー、検索システムへの記憶、電送を行うことはできません。

サードパーティ各社商標の引用は、MSE および MSJ の製品に対するコンパチビリティに関しての情報提供のみを目的としてなされるものです。

本マニュアルにおいて、説明のためにサンプルとして引用されている会社名、製品名、住所、人物は、特に断り書きのないかぎり、すべて架空のものであり、実在のものについて言及するものではありません。

Magic は Magic Software Japan K.K. の登録商標です。
Magic xpa は Magic Software Enterprises Ltd. のイスラエルその他の国での商標または登録商標です。
Magic xpa Enterprise Studio、Magic xpa Enterprise Client、Magic xpa Enterprise Server および Magic xpa RIA Server は Magic Software Japan K.K. の商標です。

一般に、会社名、製品名は各社の商標または登録商標です。
MSE および MSJ は、本製品の使用またはその使用によってもたらされる結果に関する保証や告知は一切していません。この製品のもたらす結果およびパフォーマンスに関する危険性は、すべてユーザが責任を負うものとします。

この製品を使用した結果、または使用不可能な結果生じた間接的、偶発的、副次的な損害（営利損失、業務中断、業務情報の損失などの損害も含む）に関し、事前に損害の可能性が勧告されていた場合であっても、MSE および MSJ、その管理者、役員、従業員、代理人は、いかなる場合にも一切責任を負いません。

2014 年 5 月 30 日

はじめに 1
オフラインの課題の対応 2
オフラインのアプリケーションパターン 2
用語と定義 2
リソースをローカルにキャッシュ 3
オフラインプログラムの定義 3
アプリケーションリソースのキャッシュ 3
クライアントでデータの保存／更新 3
ユーザ情報のキャッシュ 3
オフライン使用のためにイメージをダウンロード 3
サードパーティの .NET アセンブリ（Windows のみ）をダウンロード 3
オフラインアプリケーションのフロー 4
メインプログラムの実行 4
タスク前 4
オフラインタスクの制限 4
オフラインプログラムからサーバにアクセスする 4
サーバアクセスの失敗と復旧 5
サーバに接続することなくアプリケーションを開始する 5
ローカル（オフライン）ストレージ 6
ローカルデータベースの定義 6
ローカルデータベースとデータソースの制限 6
ローカルデータソースのデータベース構造を変更する 6
クライアント／サーバ間でのデータ同期 7
同期をサポートするデータソース構造 7
同期プログラム 7
パフォーマンスの改善 9
冗長なコンポーネントとメニューの削除 9
サーバへの冗長な呼び出しの回避 9
非インタラクティブクライアントの処理でレコードレベルのトランザクションを回避 9
アプリケーションリソースのキャッシュ 9
サーバとクライアントの間における大量のデータのコピー 9
データベースの事前準備 9
イメージやリソースのパッケージ化（モバイルデバイス） 10

はじめに

Magic xpa は、サーバに接続していない間もリッチインターネットのクライアントアプリケーションが動作できるようになり、クライアント側にローカルにデータを保存できるようになりました。Windows と iOS、Android デバイス用に、接続時とオフライン時の両方に対応したアプリケーションを開発できるようになりました。オフラインアプリケーションは、インターネット接続が時々切れたり、制限されたり、利用できない場所で、ユーザが継続して操作できるようにすることが可能になります。オフラインで動作している間、データは Local データベースに保存され、定期的にインターネットの接続を再開して、サーバとデータを同期させることができます。

この新しい機能の恩恵を得るには、このドキュメントによってこの機能を知っていただく事を推奨します。

オフラインの課題の対応

オフラインの実装を計画する際、解決しておく必要のある課題と制約を理解することが重要です。これによって、アプリケーションは、サーバと接続しないオフライン状態で完全に動作することを可能にします。接続されたアプリケーションとは異なり、サーバと接続していない場合や、断続的な状態の場合、使いやすさとデータの保全性を損なうことなく、アプリケーションが適切にこの手順を処理するように調整する必要があります。

これらの課題には、以下のような切断時の技術的な面とデータ一貫性の側面に関するものがあります。

- クライアントにサーバ関連のデータのサブセットやクライアントのみのデータを保存する。
- ユーザ認証が必要となるシステム上でクライアントにユーザ証明書を確実に保存する。
- サーバデータの更新一貫性を維持している間、クライアントでのデータ入力を許可する。
- データに対する効果的な双方向性を持つ同期メカニズムを提供する。
- 連続的な処理とデータ一貫性を可能にする上で、断続的に途切れるネットワーク接続状態（ネットワークが切断したり、接続が遅かったりする）での動作を行う。
- 接続中に更新を可能にする上で、クライアント上にアプリケーションのリソース（メタデータやイメージなど）を保存する。

上記の課題は、ネットワークの接続性が保証される時に使用されるパターンとは様々な点で異なるオフラインのパターンを定義して、追加される使用シナリオを処理することを開発者に要求します。Magic xpa は、開発者がこれらの課題に取り組んで、完全なオフライン処理が提供できるためのツールと機能を提供します。

オフラインのアプリケーションパターン

接続されたアプリケーションとは異なり、オフラインのアプリケーションは、ネットワークが断続的であったり、接続されていなかったりした状態で動作するように設計されています。この制限のため、オフラインのアプリケーションでは異なる実行フローを定義する必要があります。代表的なオフラインのアプリケーションは、以下のように動作します。

- 最初に起動する際、オフラインアプリケーションは、オフライン処理で必要なすべてのリソースをダウンロードして同期しなければなりません。そのようなリソースには、アプリケーションのメタデータやイメージ、クライアント側のデータなどが含まれています。オフラインで動作できるようにする前に、オフラインのアプリケーションは、少なくとも一回はサーバに接続できるようにしなければなりません。開発者は、必要とされるすべてのリソースとデータが現段階で利用できることを確認しなければなりません。
- ユーザ認証を必要とするアプリケーションについては、ユーザ証明書はクライアントで確実に保存されなければなりません。そして、サーバ認証なしで処理を許可するようにします。有効性を確実にするために、そのような証明書は接続時に再確認する必要があります。
- 最初に起動した後のすべてのユーザ操作は、ローカルのリソース（ローカルデータやイメージなど）だけを使用して行わなければなりません。ローカルのリソースだけを使用することにより、アプリケーションは動作が保証され、インターネットの接続状態に依存せず、サーバアクセスを必要としなくなります。すべてのデータ更新は、ローカル上のデータベースに保存されます。
- アプリケーションに依存するタイミングで、アプリケーションは定期的にサーバに対して変更されたローカルデータを同期させる必要があります。そして、最後に同期された後に修正されたサーバ上のデータをダウンロードしなければなりません。データオブジェクトが複数のクライアントや同時にサーバでも更新されることができる場合、競合の適切な回避処理を実装する必要があります。Master-Master のパターンが、オフラインスタイルのデータ同期を処理するために提案されます。
- 接続中に、Magic xpa は自動的にメタデータ・オブジェクトと同期するため、オフラインで実行するアプリケーションは、そのメタデータ・オブジェクトに対する定期的な変更に対して同期させることができるようにしなければなりません。通常は、起動時にインターネット接続が利用可能ならば、メタデータ・オブジェクトは自動的に同期されます。アプリケーションが変更した場合、開発者はメタデータの更新を許可するようにしなければなりません。

以下のセクションでは、上記の内容や各機能を可能とするための Magic xpa でサポートされている機能について紹介しています。

用語と定義

このドキュメントでは、以下の用語が使用されています。

- オフラインプログラム …… クライアントだけで実行するプログラム。このプログラムはサーバにアクセスすることはなく、サーバのリソースを使用することができません。
- ローカルデータソース…… クライアント側に保存されるデータベース内のデータソース。
- Local …… Magic xpa が使用するローカルデータベース用のゲートウェイを Local ゲートウェイといいます。このゲートウェイがアクセスする DBMS を Local DBMS と表記しています。

リソースをローカルにキャッシュ

オフラインプログラムの定義

サーバへのアクセスを行うことなく動作するプログラムは、特別にオフラインプログラムとして定義しなければなりません。オフラインプログラムは最初の接続の際に自動的にクライアントにダウンロードされて、クライアントのキャッシュに格納されます。起動する度にサーバ確認が行われる非オフラインプログラムとは異なり、オフラインプログラムはクライアント側のキャッシュから常に読み込まれ、サーバと接続する必要がありません。

［タスク特性］ダイアログ、または［プログラム］リポジトリの［オフライン］のチェックボックスをチェックすることによって、リッチクライアントプログラムをオフラインとして定義することができます。

コンポーネントで定義されたオフラインプログラムの場合、［即時有効］特性をオンにする必要があります。これによって、コンポーネントのオフラインプログラムは、最初の接続の際にクライアントのキャッシュにダウンロードされます。

アプリケーションリソースのキャッシュ

サーバへの最初の接続の際、すべてのアプリケーション関連の環境メタデータ（例えば Magic.ini ファイルや基本色定義ファイル、フォントファイルなど）とアプリケーションメタデータ（例えばメインプログラムやメニュー）は、サーバから読み込まれ、クライアントのキャッシュに自動的に保存されます。これらのリソースは、オフラインプログラムで自動的に利用できます。

クライアントでデータの保存／更新

Magic xpa は、RIA クライアントでローカルデータを保存したり、アクセスしたりするための新しいデータベース・ゲートウェイ（Local ゲートウェイ）を提供します。このゲートウェイは SQLite データベースに基づいており、ローカルデータベースに対する SQL アクセスを可能にします。メインまたはリンク用のデータソースとして使用することができ、必要なデータソースの処理をサポートしているため、Magic xpa は他のデータソースを使用した場合と同じように動作させることができます。

ローカルデータソースを定義して動作するための詳細な説明は、第 4 章「ローカル（オフライン）ストレージ」で説明しています。

ユーザ情報のキャッシュ

Windows 版 RIA の場合、Magic xpa の認証機能を利用すると、ユーザログオン情報やセキュリティ情報が自動的にクライアントのキャッシュに暗号化して保存されます。サーバに接続することなくアプリケーションを実行すると、最後にログオンした時の（権利を含む）詳細情報が使用されます。サーバに接続しないでアプリケーションを実行する場合、［ログオン］ダイアログは表示されません。ログオン情報は、最初の接続時や、それ以降で接続したアプリケーション開始時に自動的に同期されます。

オフライン使用のためにイメージをダウンロード

非オフラインプログラムと同じように、イメージは、サーバまたはクライアントから読み込まれるように定義することができます。いくつかのイメージは、サーバからのみ読み込まれるように事前に定義されています。

サーバに接続すると、サーバイメージとして定義されたものは自動的にダウンロードされ、サーバからクライアントのキャッシュに転送され、クライアントのキャッシュから読み込まれます。イメージがクライアントのキャッシュに存在しない場合は、無視され表示されません。

オフラインプログラムで使用されるすべてのイメージがネットワーク切断時に利用できることを確実にするために、最初の起動時に ServerFileToClient() 関数を使用して、オフラインプログラムで使用されるすべてのサーバイメージをクライアントにコピーしなければなりません。この関数はサーバとクライアントのファイルのタイムスタンプを比較して、クライアントのキャッシュに新規または更新されたサーバファイルをダウンロードします。変更されないファイルは、ダウンロードされません。

イメージファイルをコピーする場合のパスは、プログラム側でのパス指定と同じ方法で指定してください。すなわち、相対パスで指定されている場合は、ServerFileToClient() 関数で相対パスを指定し、サーバ上の絶対パスで指定されている場合は、ServerFileToClient() 関数でも絶対パスを指定してください。

サードパーティの .NET アセンブリ（Windows のみ）をダウンロード

［コンポーネント］リポジトリで定義されるサードパーティの .NET アセンブリは、オフラインプログラムで使用される場合、自動的にクライアントにダウンロードされません。

オフラインプログラムがサードパーティの .NET アセンブリを使用する場合、これらをクライアント上にインストールするか、最初の起動時に ServerFileToClient() 関数を使用してアセンブリをクライアントへコピーするようにする必要があります。

オフラインアプリケーションのフロー

オフラインプログラムを持つアプリケーションは、接続状態での起動をサポートする一方、オフラインで起動できるように起動時の処理と実行シーケンスを少し変更する必要があります。このセクションでは、実行フローの変更内容について説明します。

メインプログラムの実行

起動時は、クライアントは常にサーバに接続しようとします、そして、インターネットへの接続が可能であれば、メインプログラムはそれに応じて実行されます。

- クライアントがサーバに接続できる場合、メインプログラムはサーバとクライアントの両方で実行するため、将来のリクエストに提供する用意のためコンテキストがサーバでオープンされます。
- クライアントがサーバに接続できない場合、サーバでコンテキストをオープンすることなく、メインプログラムはクライアントでのみ実行します。その後で、サーバへのアクセスが発生した場合、以下で説明するように、クライアントは再度サーバに接続しようとします。

アプリケーションを閉じる場合は、以下のように動作します。

- アプリケーションがサーバに接続している場合は、コンテキストを閉じるためにサーバにアクセスします。
- サーバへの接続がなく、メインプログラムの［タスク後］にサーバ側のロジックが定義されていない場合、アプリケーションはエラーを表示することなくクライアント側で終了します。この場合、コンテキスト非稼働タイムアウトに達するまで、サーバ側のコンテキストはオープンされたままになります。

メインプログラムで［ウインドウ表示］特性を「Yes」に設定している場合、たとえサーバへの接続がなくても MDI ウインドウが開きます。MDI ウィンドウを利用することで、オフラインプログラムを実行したり、サーバへの再接続するまで待って非オフラインプログラムを開始させることができます。サーバへの接続ができない場合にアプリケーションを終了させる場合は、メインプログラムで［無効サーバ］イベントに対するロジックユニットを作成し、ここで［終了（X）］イベントを発行することで可能になります。

タスク前

オフラインタスクでは、非オフラインタスクとは対照的に、［タスク前］ロジックユニットはクライアントで実行されます。したがって、ここにはクライアント側のロジックを定義することができます。

オフラインタスクの制限

オフラインタスクには、以下の制限事項があります。

- ［ツリー］コントロールは、サポートされません。
- ［フレーム形式］フォームは、サポートされません。
- サーバ側のデータソースは、使用できません。
- サーバ側の処理コマンドと式は、使用できません。
- オフラインプログラムのサブフォームにはオフラインプログラムのみ使用できます。
- 非オフラインプログラムのサブフォームには、オフラインプログラムを使用することができません。
- オフラインタスクには、オフラインのサブタスクのみ定義できます。
- Studio 環境でオフライン機能を試す場合、バックグラウンドモードで実行させる必要があります。アプリケーションがサーバにアクセスしないでクライアント上で起動した場合、サーバへのアクセスを再開しようとする時にサーバがオンライン（フォアグラウンド）モードで動作している場合、アプリケーションはサーバにアクセスすることができません。

オフラインプログラムからサーバにアクセスする

オフラインプログラムは未接続でも動作することができるクライアントプログラムであるため、それらは直接サーバにアクセスすることができません。これは、サーバプログラムを呼び出すことができないことや、サーバ関数を使用することができないことを意味します。しかし、オフラインプログラムがサーバプログラム（たとえば、定期的にデータを同期させる）の呼び出しを開始することを許可することが必要な場合があります。

オフラインプログラムからサーバプログラムを呼び出すには、メインプログラム内に定義された［イベント］ロジックユニットを実行させるイベントをオフラインプログラムで発行することで実現できます。このメインプログラムの［イベント］ロジックユニットでは、必要なサーバ側の処理を実行させることができます。

上記の技術を利用することで、クライアントはサーバにアクセスすることなくアプリケーションを開始し、後でサーバにアクセスすることができるようになります。この場合、クライアントはサーバに接続しようとします。そして、接続できると、コンテキストがサーバでオープンされ、サーバ側のロジックが実行されます。

オフラインプログラムを閉じるとき、呼び出された非オフラインプログラムは自動的に終了されない場合があります。

サーバ呼び出しの際にネットワークエラーが発生した場合や、サーバ側でリクエストを終了させることができなかった場合、以下のセクションで説明しているように動作します。

最初の起動時に従ってサーバアクセスを実行すると、クライアントは更新されたメタデータ（例えば、環境ファイルやオフラインプログラム）をサーバから再読み込みしようとしません。変更がメタデータで行われ、結果としてクライアントのメタデータがサーバより古いと、サーバアクセスは失敗します。そして、以下のセクションで説明するように動作します。

サーバアクセスの失敗と復旧

サーバアクセスが失敗すると、クライアントは以下のエラー回復手順を実行します。

1. 処理を再実行するか、アボートするかをユーザに確認します。クライアントは、ユーザリクエストと同じくらいの回数分処理を再実行します。このダイアログボックスは、ClientSessionSet() 関数で EnableCommunicationDialogs キーパラメータと 'FALSE'LOG の値を指定することで無効にできます。このダイアログボックスが無効の場合や、アボートを選択した場合、ステップ 2 に移ります。
2. クライアントで実行しているすべての非オフラインプログラムは、終了します。
3. ［無効サーバ］イベントが発生します。サーバアクセスのエラーが発生すると、このイベントは実行ログを書き込みます。
4. ServerLastAccessStatus() 関数は、エラーコードを返します。サーバへの最後のアクセスが正常に終了した場合は "0" が、エラーが発生した場合は "0" 以外が返ります。ネットワークエラーが発生した場合の対応に関する詳細な情報を表示させるたり、エラーに対して異なるロジックを実行したりする場合、この関数を使用することができます。

   リソースへのアクセスを含めたサーバへのアクセス条件として ”ServerLastAccessStatus=0” を設定することが最善の方法です。この条件を設定することで、クライアントが直近でサーバにアクセスできた場合だけ、サーバに確実にアクセスできるようになります。クライアントがサーバへのアクセスに失敗すると、この条件によって、サーバへの再接続を試みなくなります。
5. フローは最後に実行したオフラインプログラムから継続されます。そして、失敗したサーバアクションにもとづいて次のアクションから開始されます。これによって、ユーザが動作を継続し後でサーバを呼び出すことができます。

サーバに接続することなくアプリケーションを開始する

デフォルトでは、アプリケーションが開始されるとサーバに接続します。リッチクライアントの実行特性ファイルの ”ConnectOnStartup”パラメータを”N” に設定することで、サーバにアクセスすることなくアプリケーションを実行させることができます。この設定は、クライアントが起動時にサーバにアクセスする必要ない場合に役立ちます。

たとえば、同期プロセスをユーザによって手動で行ったり、毎週行ったりする場合があります。起動時に常に同期プログラムを呼び出す必要がある場合は、起動時にサーバへの接続を省略しないようにしてください。

“ConnectOnStartup” を ”N” に設定してクライアントの起動時にサーバに接続していない場合、ServerLastAccessStatus() 関数は 0 以外の値が返されます。これによって、サーバへのアクセスロジックを条件付けることができます。メタデータが変更された時のみ、クライアントは起動時にサーバに接続し、この場合、サーバへのアクセスロジックも実行されます。

ローカル（オフライン）ストレージ

RIA クライアントは、ローカルデータソースをサポートします。ローカルデータソースを使用することで、サーバにほとんどアクセスしないオフラインプログラムでクライアントにデータを保存したり使用することができます。

ローカルストレージを使用することで、データをローカルのキャッシュに格納してパフォーマンスを向上させることができます。サーバデータのサブセットをローカルに保存して、後でサーバと同期することで、オフラインの処理を完結させることができます。

ローカルデータベースの定義

ローカルデータベースは、たとえば、DBMS が MS-SQL Server または Memory に設定される場合と同じように［データベース］テーブルの［DBMS］カラムで「Local」と指定することで定義することができます。

Local DBMS のデータベースを使用したデータソースは、クライアント側のデータソースで、クライアントのキャッシュに保存されます。

ローカルデータベースとデータソースの制限

ローカルデータベースとデータソースには以下の制限があります。

- ローカルデータベースは、ダイレクト SQL ステートメントで使用することができません。
- ローカルデータソースは、リッチクライアントタスクでのみ使用できます（オンラインやバッチの各タスクで使用することはできません）。
- ローカルデータソースのトランザクションは、物理トランザクションとして動作します。ローカルデータソースを含むすべてのタスクは、同じトランザクション内にあります。
- ローカルデータのコミットは、トランザクションをオープンしたタスクに従って実行されます。トランザクションが並行タスクでも開始されている場合、これらのタスクのデータもコミットされます。
- ローカルデータソースを持つタスクをロールバックすると、以下もロールバックされます
  - ローカルデータソースを持つすべてのタスク。
  - 親タスクの遅延トランザクション。
- ローカルデータソースは、［ツリー］コントロールが定義されたタスクで使用することができません。
- タスクには、サーバとローカルのデータソースを混在させることができません。

以下の機能は、ローカルデータソースではサポートされません。

- ユニークインデックスまたは RowID SQL ポジションの値
- ローカルデータソース間の［結合リンク］コマンド
- ［コール］処理コマンドの［同期］特性
- 位置の範囲
- SQL 範囲
- インデックスの最適化
- ユーザソートと Magic ソート
- 遅延トランザクション
- 定義取得
- ［データベース］特性の［開発時の定義変更］特性
- ［エラー］ロジックユニット …… ローカルデータソースのエラーに対するエラー処理を定義することはできません。
- チャンクサイズ

ローカルデータソースのデータベース構造を変更する

ローカルデータソースの構造（例えば、フィールドの削除や、書式の修正、フィールドの追加）が変更された場合、プロジェクトのメタデータがクライアントと同期した後で、変更内容がクライアントのローカルデータソースに自動的に反映されます。

既存のローカルデータは、可能な限り新しい構造のデータソースに自動的に変換されます。

ローカルデータソースの構造の変更が、幾つかのデータ（例えば、ユニークなインデックスの追加や文字列フィールドを数値に変更した場合）を無視した場合、このデータは失われます。したがって、ローカルデータを必要とする場合、そのような変更は避けることを推奨します。

古い構造から新しいものへの変換が、データソースの ISN に基づいて行われることに注意してください。これは、ローカルデータソースが Studio 環境で名前を変更した場合、それがローカルデータベースにも反映されることを意味します。最初のローカルデータソースが変更されないままにしたい場合、新しい構造に対応した新しいデータソースを作成する必要があります。

クライアント／サーバ間でのデータ同期

ローカルデータはサーバデータのサブセットの場合があるため、サーバの新しいデータによってローカルデータを更新したり、更新されたローカルデータでサーバ側を更新したりする必要があります。さまざまなパターンが、サーバ側のデータソースと（クライアント側の）ローカルデータソース間でデータを同期させるためにあります。以下で説明する情報は、サーバと複数のクライアントの間で双方向での更新処理に関連して、Master ― Master のレコードレベルの同期パターンに従っています。そして、楽観的な並行コントロールとみなします。

一方向性の更新のみを必要とする場合、以下で説明する操作のサブセットだけを必要とします。

このパターンに関するより詳細な説明（英語）は、http://msdn.microsoft.com/en-us/library/ff650702.aspx を参照してください。

クライアントとサーバが異なるタイムゾーンで動作している場合、UTC（ローカルでない）の時間を使用して、すべてのタイムスタンプを保存することが必要です。

同期をサポートするデータソース構造

サーバ側のデータソースと（クライアント側の）ローカルデータソースとのデータ同期は、変更されたレコード（両方のデータソース上で）と最後の同期時間を確認することで行われます。効果的で適切に変更の経過を追跡するには、両方のデータ構造とデータを表示して更新するロジックに変更することが必要になります。

最終更新時間と同期時間の経過を記録するには：

1. 同期する必要がある各データソース（サーバ、または、クライアントで）には、以下の追加フィールドが必要です。
   - 最後に変更されたタイムスタンプ …… YYYYMMDDHHMMSS フォーマットの文字列
     このフィールドは、データソースで内に新しく更新されたレコードを見つけるために使用されます。
   - 削除フラグ …… 論理値
     このフィールドは、削除された行を確認するために使用されます。
   - このフィールドでインデックスを定義することを推奨します。
2. 新しいローカルデータソースには、最後の同期時間の経過を記録することが要求されます。このデータソースには、以下のフィールドを含めてください。
   - 最後に同期したタイムスタンプ…… YYYYMMDDHHMMSS フォーマットの文字列
     このフィールドは、新しく更新されたレコードを見つける範囲条件として使用されます。

2 つの同期用タイムスタンプのフィールドを使用することを考慮してください。1 つは、サーバ上の検索用、もう 1 つはクライアント上の検索用です。以下のようなシナリオで使用します。

- **クライアントとサーバのクロックが同期していない場合。**
- **サーバ側データの同期タイミングがクライアントデータのタイミングと異なる場合。たとえば、サーバ側データがクライアントに同期する処理は、開始時に行われ、ローカル側のデータがサーバに同期する処理はタスクの終了時に行われます。**

同期プログラム

以下のプログラムは、同期させたい各データソースに対して定義する必要があります。

- サーバからローカルへの同期 …… サーバのデータでローカル側のデータソースを更新します。
- ローカルからサーバへの同期 …… ローカルのデータでサーバ側のデータソースを更新します。

- **更新を双方向で行う場合、これらの両方のプログラが必要になります。これによってデータはサーバ側のデータソースとローカル側のデータソースの両方で更新されます。更新が一カ所でのみ行われる場合は、1 つのプログラムのみ必要です。**
- **双方向で更新する必要がある場合、クライアントからサーバへの同期から開始し、次にサーバからクライアントへの同期を行う必要があります。このように、クライアント更新が優先となります。サーバ更新を優先させたい場合は、サーバからクライアントへの同期プログラムを最初に実行しなければなりません。**
- **双方向で更新する必要がある場合、データを一時的なテーブルにコピーして、次に、（更新時刻に基づいて最後に更新されたレコードを維持するような）一定のルールを使用して実テーブルにデータをコピーしたい場合があるかもしれません。**
- **時々、フルデータコピー（通常はサーバからクライアントへ）を行う必要があります。この場合、DBDel() か ClientDBDel() 関数を使用して最初に転送先のデータソースを削除することで可能になります。データを新しいテーブルへコピーする方が、既存のテーブルにコピーするより速くなります。**

サーバデータでクライアントのデータソースを更新する

サーバのデータによるローカルデータソースの更新は、サーバで最後に行われた同期処理以降に更新されたレコードの選択と、サーバデータによってローカルデータソースを更新することで行われます。これは、メインソースとしてサーバ側のデータソースを使用した非インタラクティブで非オフラインのリッチクライアントプログラムを使用することよって行うことができます。

このプログラムには、以下のように定義します。

- レコードの範囲指定では、最後に同期したタイムスタンプから開始するようにします。
- ［タスク終了条件］特性を「Yes」に、［チェック時期］特性を「前置」に設定します。
- ［タスク後］に［アクション］処理コマンドを定義し、DataViewToDataSource() 関数を実行するように設定します。メインソースの必要な項目を（ローカルの）出力先データソースにコピーするようにします。

クライアントデータでサーバのデータソースを更新する

クライアントからサーバへの更新は、まったく同じように行います。違いは、メインソースにローカル側のデータソースを定義し、関数の転送先にサーバ側のデータソースを指定する点です。

同期管理プログラム

上記のプログラムを呼び出すために同期管理プログラムを使用することを推奨します。

管理プログラムは、非インタラクティブな非オフラインのリッチクライアントプログラムです。

このプログラムは以下のように動作します。

1. ローカルデータソースから最後に同期したタイムスタンプを読み込みます。
2. 変数項目に現在のタイムスタンプを保存します。
3. 最後の同期タイムスタンプから開始して、データを更新するように、2 つの同期プログラムを呼び出します。
4. 同期処理が正常に終了した場合、ローカルデータソースに一時的なタイムスタンプ（同期処理の開始時刻）を保存し、次の同期処理で使用します。

同期するデータ量に応じて、（管理プログラムか、各同期プログラム上で）進捗状況を表示させる必要があります。

同期管理プログラムは、同期する必要があるデータ量と型にもとづいて、自動的またはユーザによって手動で起動されます。

プログラムが手動で実行された場合、同期プロセスが失敗すると、ユーザにメッセージを表示することを考慮してください。これは、［無効サーバ］イベントを使用したり、同期プログラムを呼び出した後で ServerLastAccessStatus() 関数を用いたりすることで実現できます。

削除レコードの処理

上記のように、各データソースには、レコードが削除されるかどうかを示すカラムがあります。

レコードは、データソースから物理的には削除されません。その代わりに、削除が必要なレコードをマークします。このように、削除レコードの同期は、他の更新カラムと同じように処理されます。

アプリケーション内で削除レコード以外を表示するようにします。

代わりに、削除されたレコードの経過を追跡するために、別のデータソースを使用することができます。このデータソースには、削除レコードを識別するカラムと、削除日付と時間のカラムを含める必要があります。この方法では、各削除処理の後、新しいレコードをデータソースに追加する必要があります。

ユーザがレコードを物理的に削除することを防止するには、以下のどれかを実行してください。

- **［タスク特性］の［削除］特性を「No」に設定する。**
- **［行削除］イベントに対する［イベント］ロジックユニットを追加し、［伝播］特性を「No」に設定する。**

このようにすることで、ユーザはアプリケーションの機能を使用する場合にだけレコードを削除することができるようになります。

パフォーマンスの改善

冗長なコンポーネントとメニューの削除

アプリケーションの性能を向上させる上での第一歩は、冗長なメニュー定義や使用していないコンポーネントを削除することです。デフォルトでは、新しいアプリケーションにデフォルトプルダウンメニューとユーザ機能コンポーネントへの参照定義が作成されるため、モバイルデバイスにとってこの点は特に重要です。両方のオブジェクトはモバイルアプリケーションで使用されることがないため、これらは削除しなければなりません。

サーバへの冗長な呼び出しの回避

大部分のアプリケーションでは、データを同期させ、オフラインで使用するリソースをダウンロードするための初期化プロセスを実行する開始プログラムが定義されています。

このプロセスの実行に条件付けを行ったり、少なくとも以下の条件でサーバにアクセスする処理を定義することを推奨します。

ServerLastAccessStatus()=0

この条件は、クライアントが最後にサーバへのアクセスが成功した場合だけ、サーバへのアクセスを確実にします。

したがって、クライアントが最後に行ったサーバへの接続が失敗した場合、それから、この条件によって、クライアントは処理を実行するためにサーバへの再接続を行うことがなくなります。

非インタラクティブクライアントの処理でレコードレベルのトランザクションを回避

多くのレコードを更新する非インタラクティブ処理がある場合、可能な限りタスクレベルのトランザクションを使用することが推奨されます。

レコードレベルのトランザクションを使用することは、各レコード毎にトランザクションを開始するため、パフォーマンスを低下させます。さらに、タスクがサーバ側のデータソースを更新するとき、レコードレベルのトランザクションの場合、トランザクションをコミットするために各レコード処理の後にサーバにアクセスするため、よりパフォーマンスが低下します。

アプリケーションリソースのキャッシュ

第 2 章「リソースをローカルにキャッシュ」（3 ページ）で説明したように、切断時にオフラインプログラムで使用されるすべてのイメージを利用できるようにするために、最初の起動時に、ServerFileToClient() 関数を使用してオフラインプログラムで使用されるすべてのサーバイメージをクライアントにコピーしなければなりません。

ファイルをクライアントへコピーする場合のパフォーマンスを向上させるために、ファイルを（個別ではなく）フォルダごとにダウンロードする必要があります。

ServerFileToClient() 関数は、フォルダとワイルドカードをサポートします。

複数のファイルが存在するフォルダを指定して関数を実行すると、1 つのリクエストによって全てのファイルの変更されたタイムスタンプをもとに変更ファイルを連続的に処理するようになります。これは、ファイルが変更されていなければ、アプリケーションの 2 回目の起動時に、1 つのリクエストだけが送られるだけになります。

各ファイル毎に関数を実行すると、各々のファイルに対して、変更によるタイムスタンプを取得する必要があり、変更されたファイルに対して連続的にリクエストを送ることになります。これは、アプリケーションが最初に起動された場合、フォルダを指定する場合と比べて、より多くのタイムスタンプのチェック要求が必要になります。また、2 回目以降にアプリケーションが起動されファイルが変更されていない場合、フォルダを指定する場合と比較して、各ファイル毎にタイムスタンプを取得する必要があります。

サーバとクライアントの間における大量のデータのコピー

複数のレコードをサーバからクライアントへ、または、クライアントからサーバへコピーする必要がある場合、DataviewToDataSource() 関数を使用する方が最適です。

この関数は（範囲条件に従って）一連のレコードをまとめてコピーします。これによってパフォーマンスが向上します。DataviewToDataSource 関数の詳細については Magic xpa のリファレンスヘルプを参照してください。

データベースの事前準備

アプリがオフラインで使用する必要のある大きなデータベース（国や都市名のマスタなど）を含んでいる場合、上記のように各データソースをクライアントへコピーするには時間がかかるかもしれません。

パフォーマンスを向上させるためには、オフラインで使用したいデータをサーバ上の SQLite データベースとして作成し、最初の起動時にこのデータベースファイルを ServerFileToClient() 関数を使用してクライアントへコピーすることができます。

このようにするには、以下の手順を実行します。

1. RIA のキャッシュフォルダ（Windows の場合は、%TEMP%¥MgxpaRIACache¥< サーバアドレス >）内のキャッシュファイルを削除してください。

2. サーバ側のデータソースからローカルのデータソースへデータをコピーするために、データコピープログラムを実行してください。このプログラムでは ServerFileToClient() 関数を使用することを推奨します。
3. アプリケーションの実行時の RIA キャッシュフォルダを開きます。このフォルダには、データベースファイル（ファイル名は、ローカルデータベース特性（例えば local.sqlite）で定義されています）が含まれています。
4. ローカルデータベースフォルダと同じ名前のフォルダをサーバに作成しファイルを置いてください。

   例：
   - **ローカルデータベースファイルが local.sqlite という名前の場合。サーバ上にプロジェクトの（EDP/ECF ファイルがある）作業フォルダにファイルを置いてください。**
   - **ローカルデータベースファイルが db¥local.sqlite という名前の場合、サーバ上のプロジェクトの作業フォルダ内の db サブフォルダにファイルを置いてください。**
   - **ローカルデータベースファイルが c:¥temp¥local.sqlite という名前の場合、サーバ上の c:¥temp¥local.sqlite にファイルを置いてください。**
5. 起動プログラムでデータベースファイルをサーバからクライアントへコピーしてください。

   ローカルデータベースが存在しない場合、ファイルをコピーするだけです。
   ローカルデータベース名で定義している値と同じパスを使用して DB ファイルをコピーしてください。
   例：
   - **ローカルデータベースファイルが local.sqlite という名前の場合：**
     ServerFileToClient('local.sqlite') を実行します。
   - **ローカルデータベースファイルが db¥local.sqlite という名前の場合：**
     ServerFileToClient('db¥local.sqlite') を実行します。
   - **ローカルデータベースファイルが c:¥temp¥local.sqlite という名前の場合：**
     ServerFileToClient('c:¥temp¥local.sqlite') 実行します。

- **（たとえば、レコードデータがキャッシュにあるかどうかを調べて）現在のセッション内で、以前、ローカルデータベースを使用していた場合、ローカルデータベースに対する接続を閉じるために ClientDBDiscnct() 関数を使用する必要があります。**
- **（たとえば、ユーザ名を保存するなどして）サーバからデータを取得する前にローカルデータベースにデータを保存する必要がある場合、異なる名前の 2 つのローカルデータベースを使用しなければなりません。この方法は、初期データを設定するだけです。後で、より多くのレコードを追加する場合は、第 5 章「クライアント／サーバ間でのデータ同期」（7 ページ）を参照してください。**

イメージやリソースのパッケージ化（モバイルデバイス）

モバイルパッケージの作成時、リソースファイルをパッケージ化することができ、インストール処理やアップグレード処理によってクライアント側のアプリケーション・キャッシュフォルダに展開することができます。

これらのファイルは、実行時にサーバ側から読み出すことなく Magic プログラムで利用することができます。これによって、サーバアクセスやダウンロードにかかる時間を節約することができます。

この機能の主要な使用法は、最初の実行にサーバからデータを読み込む代わりに、パッケージのインストールの際に、イメージやローカルデータベースを展開することです。

『モバイルアプリケーションの開発ガイド』の「リソースファイルのパッケージ化」の説明を参照してください。

イメージがパッケージ内に包まれていても、以下の場合はイメージフォルダの上で ServerFileToClient() を使用することを推奨します。

- イメージが変更されている場合、このイメージは再パッケージすることなくサーバからダウンロードして更新します。
- リソースファイルのパッケージ化

ServerFileToClient() の動作は、ローカルファイルが現在存在しており、メモリ内に情報を保存しているかどうかを確認します。イメージを使用するプログラムを実行すると、イメージは現在存在しているため、クライアントは再度サーバからを取得しようとしません。関数を使用しない場合は、クライアントはイメージの修正時刻をチェックするために、サーバにアクセスしなければなりません。

一旦イメージが 1 つのプログラムで使用されると、クライアントは全てのセッションで再度イメージを取得するためにサーバにアクセスしません。このため、この処理の利用は最初のプログラムだけに定義する点に注意してください。したがって、この関数はイメージを最初に使用するプログラムのパフォーマンスを向上させるだけのものです。

リソースファイルをパッケージに含めた場合、モバイルアプリケーションの汎用性は失われます。専用のモバイルアプリケーションを配布する場合のみこの方法を使用してください。


magic
OUTPERFORM THE FUTURE™