リッチクライアント
アプリケーションの開発
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。

2016 年 6 月 1 日

1 概要

リッチクライアントとは何でしょうか？ 1
何故リッチクライアントを使用するのでしょうか？ 1

2 リッチクライアントタスクの作成

リッチクライアントでの Hellow World 2
プログラムの実行 3
既存の知識を利用する 3
リッチクライアントの照会リスト 3
タスクフローはどのように動作するのでしょうか 3
色とフォントの使用 4

3 パフォーマンスについて

基本的な構成要素 5
データ項目 6
関数 6
ユーザ定義関数 6
式 7
データビュー 7
範囲と位置付式 7
［代入］特性 8
［リンク］コマンド 8
タスクフローの追跡 9
フォームとコントロール 9
サーバ側の式を使用する 10
コントロールによる他の問題 10
ロジックユニット 10
タスク前 - サーバ 11
［タスク後］ 11
項目変更 12
コントロールとレコードレベル 12
エラーレベル 12
システムとユーザイベント 12
ロジックユニット内の処理コマンド 14
［イベント実行］処理コマンド 14
［コール］処理コマンド 16
［コール OS］処理コマンド 16
［ブロック］処理コマンド 17
［エラー］処理コマンド 19
［項目更新］処理コマンド 19
［アクション］処理コマンド 19
グループ処理 21
リッチクライアントタスクの設定 22
クライアント関数 23

4 複数のタスクで動作

他のプログラムを呼び出す 25
コンテキスト管理 25
並行処理 26
非インタラクティブタスク 27
サブフォーム 27
サブフォームの初期設定 27
サブフォームの再表示 27
［タブサイクル］特性 28
［出力先］特性 28
サブフォームの終了 29
フレーム 29

フォームタイプ 30
フレームの接続先 30
フレーム名 31
トランザクション 31
トランザクション開始 32
トランザクションモード 32
オンラインとリッチクライアントでのトランザクションモードの比較 33
ローカルデータソース 33

5 リッチクライアントでの MDI

6 ブラウザコントロール

他のアプリケーションへのアクセス 35
［ブラウザ］コントロール 35
ブラウザコントロール特性 36
ブラウザコントロールイベント 36
ブラウザコントロール関数 36
PDF の表示 36

7 アプリケーションのモニタ

デバッガ 38
サーバ同期 38
クライアントの異常終了 38
ネットワーク情報の表示 39

8 アプリケーションの実行

9 ユーザ認証

アプリケーションの起動時に環境変数を取得する 41
アプリケーションにパラメータを渡す 41

10 オフラインアプリケーションの開発

11 モバイル用の RIA アプリケーションの開発

12 オンラインから RIA への変更

リッチクライアント準備ユーティリティ 44
RM コンバータ 44
オプティマイザ 44

第 1 章 概要

Magic の最初のリリースから今日まで、この開発ツールは、常に高い生産的と保守性を提供していました。これを実現している理由のうちの 1 つは、製品の設計（「1 つのコードを複数の方法で利用する」パラダイム）に組み入れられます。この方法は、モデルの動作で確認することができます。そこでは、1 つのモデルを作成することで、多くの異なる種類のデータソースとプログラムの作成時に使用することができます。データソースの動作で確認することもできます。データソースが動作する方法でそれを参照することもできます。それが ISAM や SQL、メモリテーブルまたは XML であるかどうかに関わらず、データソースはプログラムによって同じ方法で使用することができます。

開発者は、コードの再利用がどのよう行われているかを詳細を知ることなくこの機能をりようすることができます。そしてより短い期間に学習することができます。プログラムコードは、OS や DBMS のどちらにも依存していないため、プログラムはこのどちらに対して異なるバージョンに対応させることができます。

Web 向けの開発は、常にプログラミング言語（例えば、よく利用される JavaScript など）の習得が必要となり、さらに Web ページの設計に関する知識も必要となります。Magic では、すべて同じスタジオで行われ、アプリケーションのフロントエンドは、他のクライアント / サーバ用アプリケーションと同じ方法で設計することができます。

開発者が今まで使用していたのと同じ生産的な開発ツールを使用して、Web 向用の技術にアクセスすることができます。

リッチクライアント・プログラミングは、通常の Magic xpa のオンライン・プログラミングとよく似ています。そして、どれほど簡単にこの新技術を使用できるか分かると思います。

それでは、始めましょう！

リッチクライアントとは何でしょうか？

リッチクライアントは、Magic xpa の「リッチ」な実行手段の一つです。

Magic xpa を使用していれば、異なるデータベースなどと接続させる方法などは理解していると思います。プログラムで使用されるデータソースを作成することができます。しかし、物理データソース（ISAM や SQL、XML など）は、実行時や利用するコンポーネントをもとに定義することができます。

Magic xpa は、これ実現するために、.NET フレームワーク上で実行するシンクライアントを使用します。

何故リッチクライアントを使用するのでしょうか？

リッチクライアントの実行機能には以下のような便利な機能があります。

- **シンクライアント** …… リッチクライアントを使用することで、クライアント側に専用のソフトウェアをインストールする必要がなくなります。サーバでクライアントで必要なすべてのデータ、ロジック、およびフロー管理モジュールを提供します。アプリケーションのすべてのソフトウェア要素はサーバ側に存在します。これにより、管理や保守が容易となり、経費が削減され拡張性が向上します。
- **ブラウザを使用しない** …… リッチクライアントプログラムはインターネット環境で実行しますが Web ブラウザを必要としません（最初の起動時のみ、InternetExploler または FireFox が必要です）。
- **OS のネイティブなルック & フィール** …… リッチクライアントは、.NET ベースで作成されており、クライアント OS に基づいたルック & フィールで動作します。
- **複雑なプログラム**…… リッチクライアント・プログラムは、標準的なインターネット上のブラウザ・プログラムより複雑なものにすることができます。Magic xpa のオンライン・プログラムとほとんど同じように動作します。
- **プラットホームに依存しない** …… リッチクライアントは、.NET フレームワーク上で動作します。これは、1 つの OS またはバージョンに依存しないことを意味します。
- **堅固** …… リッチクライアント・プログラムは、高いボリューム・トランザクションと高いリクエスト量を処理することができます。
- **コンテキスト管理** …… Magic xpa エンジンは、コンテキスト管理機能を各ユーザに提供するため、各ユーザに対して独立したセッションを維持します。これは、Magic xpa のオンライン環境と同じように動作します。

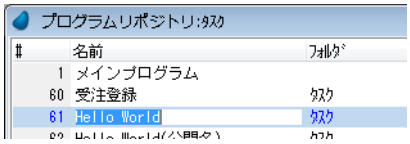
何よりも、大きな学習曲線を必要とせずこの機能を得ることができます。たとえ、背後で異なる機能を実現していても、リッチクライアント・プログラムは、通常の Magic xpa のオンライン・プログラムと同じように動作するように設計されています。

第 2 章 リッチクライアントタスクの作成

リッチクライアント・タスクの作成は、非常に簡単です。リッチクライアントの特性のため、いくつかの変更点がありますが、基本的には、Magic xpa のオンラインプログラムで使用する場合と同じ方法になります。

非常に単純なリッチクライアント・タスクを作成して、それがどのように動作するかを確認してみましょう。それから、より複雑なプログラムへ進めることで、今までとの違いを学ぶことができます。

リッチクライアントでの Hellow World

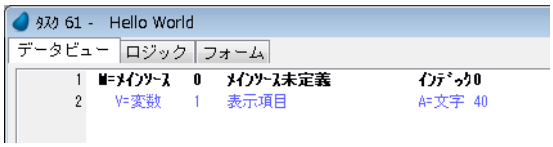


古典的な単純なプログラム（Hellow World ）から始めましょう。

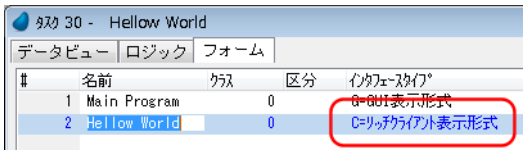
新しいプログラムを作成してください

次に、［タスクタイプ］特性で「リッチクライアント」を選択します。

これによって、プログラムはリッチクライアント・プログラムとして実行されます。

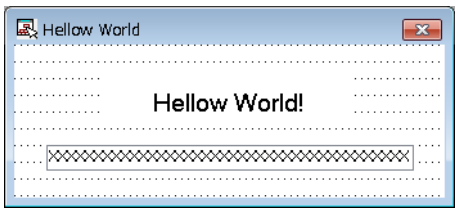


少なくとも一つのデータ項目を定義します。



これはリッチクライアント・タスクになるため、メインフォームのデフォルトはリッチクライアント表示になります。

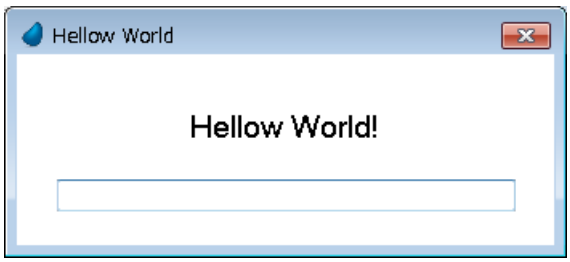
これは、リッチクライアント・プログラムで使用されるフォームのタイプです。



次に、［コントロール／項目］パレットからテキストとデータ項目をドラッグして、オンラインタスクと同じようにフォームを作成してください。

次に、オンラインタスクと同じようにこのプログラムを実行します。

プログラムの実行



実行すると、図のようなウィンドウが表示されます。これを見て以下のことに気づくと思います。

最初に、「Hellow World!」プログラムは、それ自身が独立したウインドウを持っています。MDI の要素として表示されません。これはスタンドアロンの .NET プログラムで、Magic xpa の実行エンジンによって直接実行している訳ではありません。現在、あなたの PC 上で実行していますが、インターネット経由で世界中のどこからでも実行させることができます。

しかし、それは Web ブラウザのウインドウで動作している訳ではありません。プログラムは、HTML で記述されてはいません。

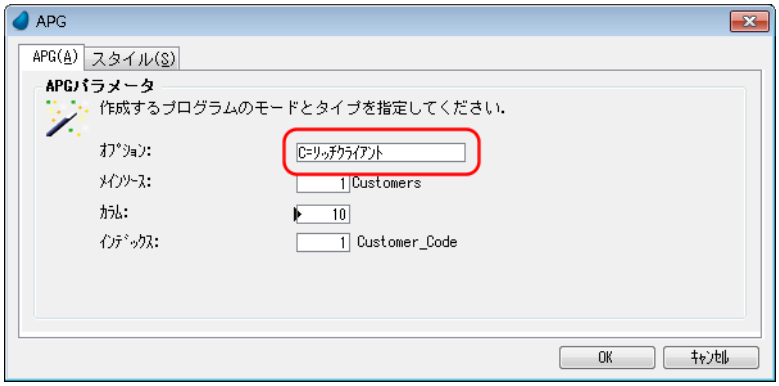
これは .NET で記述されています。そして、実行するプラットホームに対応した標準的なユーザインタフェースを利用していることを意味します。Hello World プログラムは、.NET フレームワークをサポートする様々なプラットホーム上で実行させることができます。

これは非常に異なる実行環境です。そして、これを正しく動作させるために必要なことががいくつかあります。しかし、プログラミングのほとんどは、既にご存知の Magic xpa のプログラミングになります。

既存の知識を利用する

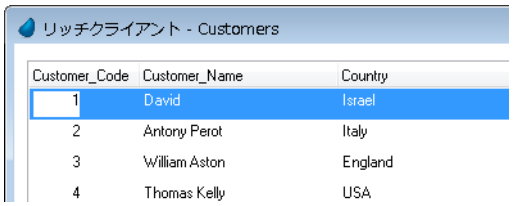
開発の観点から、リッチクライアントの利点は、プログラムがオンライン・タスクに似ていることです。新しいパラダイムを学ぶ必要はなく、すでにある技術が非常に役に立ちます。

リッチクライアントの照会リスト



次に、もう少し難しくいリッチクライアント・プログラムを作成してみましょう。このセクションでは、レコード表示するための、ラインモードのプログラムを作成します。

Ctrl+G を押下して APG を実行し、［オプション］で「リッチクライアント」を選択します。



これで、プログラムを実行すると、全ての顧客リストが表示されます。これは、リッチクライアントの .NET 環境で動作します。

タスクフローはどのように動作するのでしょうか

このタスクが動作する場合、何が起きているかを確認してみましょう。

実際は、「顧客リスト」のクライアントプログラムはあなたのサーバ PC ではなく、どこか別の場所で実行するということを心にとめておいてください。それは、プログラムが直接データベースにアクセスしないことを意味します。また、必ずしも、Windows 環境で実行している訳ではありません。しかし、リモートデスクトップのような「スクリーンスクラッパー」でもありません。その代わりに、いくつかの処理はクライアント環境で実行されて、サーバ環境も実行されます。これが全て自動的に実行されるため、必要に応じてプログラムを最適化することができるように、何が行われているかを知っておくことが重要です。

	実行箇所	説明
初期設定	サーバ	新規のコンテキストこのタスクに対応して作成されます。
タスク前	サーバ	クライアント・タスクが実行される前に、サーバ側の Magic xpa エンジンがデータベーステーブルをオープンし、変数項目を初期化し、範囲値と位置付値が使用できるようにし、最初のデータビューを作成します。 クライアント・タスクは、まだ開始されていません。全ての処理コマンドはサーバで実行されます。
ユーザ操作	クライアント／サーバ	ユーザがタスクにアクセスする間、処理を実行して、データを表示するために、.NET エンジンは送信されたローカル XML を使用して動作しています。 しかし、必要に応じて、いくつかの処理はサーバでも実行されます。 ユーザが操作中に、［レコード前］と［レコード後］は、オンライン・タスクで使用される場合と同じ基準で実行されます。 レコード群がタスク初期化中に送られるため、クライアントはサーバから毎回レコードを取得する必要はありません。
タスク後	クライアント／サーバ	リッチクライアント・タスクは、すでに終了しています。定義された処理内容に従って、クライアントかサーバで処理が実行されます。［タスク後］が終了した後、そこで、タスクをクローズするためサーバへのアクセスが行われます。

いくつかのロジックユニットの扱いについては、第 3 章「パフォーマンスについて」で説明します。

色とフォントの使用

Magic xpa で通常に行うように、色とフォントを使用することができます。色とフォントは、クライアント・プログラムに渡されます。

このプログラムが指定したフォントにアクセスされない場合、OS が表示するデフォルトフォントを使用します。

異なる OS 間で汎用的に使用される（または、少なくとも将来的にサポートされる）フォントを選ぶことによって、このような問題を回避することができます。

第 3 章 パフォーマンスについて

リッチクライアントのパラダイムでは、処理はサーバとクライアントで分担されます。クライアント／サーバ間の通信は、圧縮された XML データでやり取りされます。この通信バスはボトルネックとなる場合があり、これによってプログラムのパフォーマンスに影響を及ぼすことになります。このため、効率的なプログラムを作成するには、やり取りする通信量を最小にする必要があります。

たとえば、DBMS アクセスはサーバ側で発生しますが、ユーザ操作はクライアント側で発生します。理想的には、一旦データがクライアントに送信された場合、レコードがコミットされる準備ができるまで、ユーザはサーバーにアクセスすることなくフィールドからフィールドにタブ移動できるようにすることできます。しかし、［コントロール後］で評価される条件内で DBExist() 関数が使用されている場合、ユーザによるタブ移動が行われる度に、サーバへのアクセスが発生します。これによってユーザ操作が遅くなります。

ただし、何ができて、何ができないか、または何が効率的かを覚えておく必要はありません。このようなことを明らかにするための Magic xpa では明示することができるます。このセクションでは、それらのツールの使用方法について説明しています。

ノート

ログが使用されると、各リクエスト毎にログが書き込まれるため、多くのリクエスト（バッチ処理のような）を実行すると、パフォーマンスが遅くなるかもしれません。

以下の内容で説明しています。

- **フォームとコントロール**
- **データビュー** …… データビューは、タスクが開始される前に確立されます。そして、これはデータ項目を処理する方法に影響を及ぼすことができます。
- **関数**…… 各関数は、クライアント側またはサーバ側、ニュートラルに分かれます。
- **式** …… データビューと関数の実行箇所は、式の実行箇所に影響します。
- **ロジックユニット** …… ロジックユニットが実行される場所は、それがどこから呼び出されるか、または使用される処理コマンドや式に依存します。

基本的な構成要素

Magic xpa のタスクは、3 つの基本的なセクションで構築されています。リッチクライアントタスクでは：

- データビューは、サーバ上で構築されます。
- フォームは、ユーザ操作で利用されます。
- ロジックユニットによって、サーバ側で実行されたり、クライアントで実行されたり、混在して実行したりします。

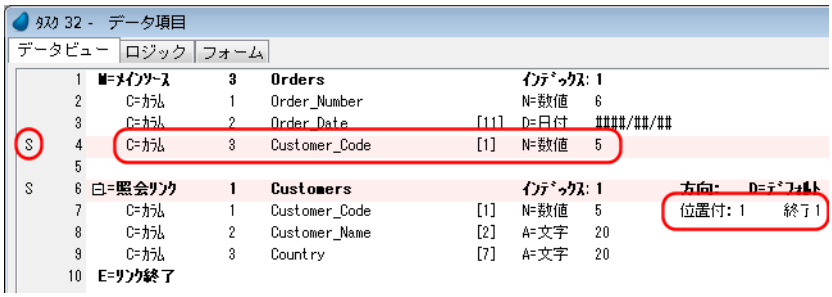
これらについて順に説明していきます。

しかし、データ項目や関数、ユーザ関数、式は、これらのセクションの全てで使用されています。

- データ項目は、データビューで宣言されます。通常、これらはニュートラルですが、サーバ側で処理されるものもあります。
- 関数は、Magic xpa に組み込まれているもので、これらの機能によってクライアント側かサーバ側、またはニュートラルになります。
- ユーザ関数は開発者によって作成されます。実行する場所は、使用する処理コマンドと変数に依存します。
- 式は、上記の項目を使用して構築されます。タスク特性や、範囲、代入、条件、コントロールとフォームで使用されるため、これらは特に重要です。

このセクションでは、これらの基本的な構築ブロックについて説明しています。

データ項目



データ項目は、[データビュー] エディタで定義されます。これらは通常ニュートラルです。データビューはサーバ上で構築されますが、ユーザ操作によって値は変更されます。トランザクションがクローズされると、これらの変更内容はクライアントで保存され、適切な時間が経過すると保存のために送り返されます。

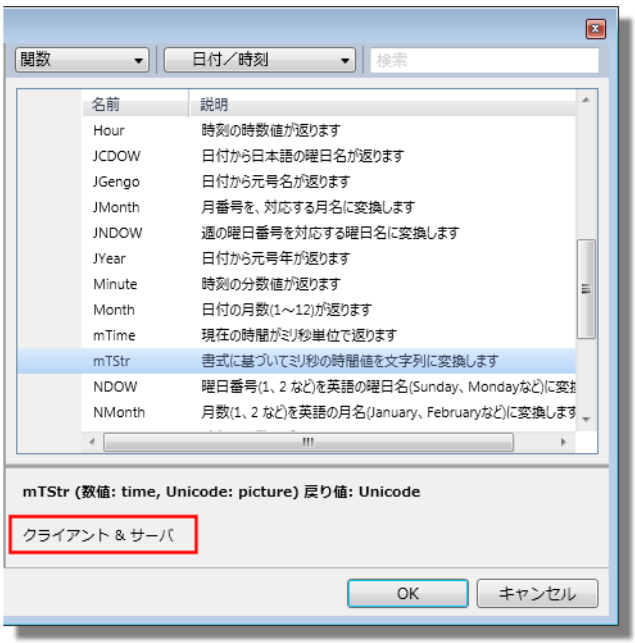
しかし、データ項目によっては、クライアントで変更される度に、サーバアクセスがすぐに必要となります。このようなデータ項目は、サーバ側項目と呼ばれています。上の画像で示すように、これらのデータ項目には、上図のように番号カラムで「S」が表示され、色分けされます。

データ項目がリンクの範囲 / 位置付で使用されている場合、このようなことが発生します。変項目の値が変更される度に、再リンクが実行されます。このため、サーバへのアクセスが発生します。上記の例では、Customers のレコードに対するリンクで使用される Customer Code があります。この Customer Code が式で使用される場合、その式はサーバ側で実行されます。

リッチクライアント・プログラムで、データ・ビューにたくさんの内容を含む BLOB 項目が定義されている場合、BLOB はサーバとクライアントの間で（他の全ての変数と同じように）渡されます。これによってプログラムが遅くなるかもしれません。

BLOB データを使用しない場合は、データビューに BLOB 項目を定義しないことを推奨します。または、それらがサーバ側で使用されるのであれば、これらを処理する新しいバッチタスクを作成しなければなりません。

関数



まず、関数を見てみましょう。これらは式で使用される最も小さな構成要素です。そして、式がどちらで実行されるかは使用される関数（とデータ項目）に依存します。各関数はクライアント側かサーバ側または、ニュートラルになります。

関数の一覧は、Magic xpa のリファレンスヘルプで参照することができます。しかし、関数を選択する際、関数一覧の左下で実行される場所を確認することができます。

ここには、以下のように表示されます。

- **サーバ**
- **クライアント**
- **クライアント & サーバ（すなわちニュートラル）**

ユーザ定義関数

開発者に定義される関数は、使用される場所と関数内に含まれる処理によって評価されます。関数がそれを使用する式と同じタスク内で定義されている場合は、式は関数と同じ場所で実行されます。

関数がメインプログラムまたはコンポーネントで定義されている場合、それを使用する式は、実行場所が未定として設定されます。

ユーザ定義関数には、クライアント側とサーバ側の両方の処理コマンドを含めることはできません。

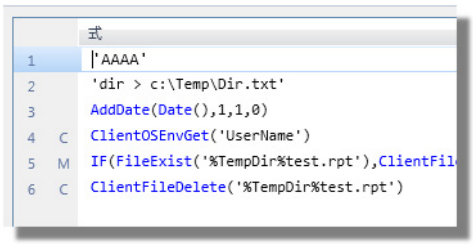
式

式は、一つ以上の関数とデータ項目で構成されています。式の実行場所は、関数とデータ項目の実行場所に依存します。

式はクライアント側で実行されるか、サーバ側か、ニュートラルかまたは混在かで分類されます。

- クライアント側の式は、クライアントでのみ実行できます。
- サーバ側の式は、サーバ上でのみ実行できます。
- しかし、ニュートラルな式は、どちらでも実行することができます。これらは、処理コマンドがサーバで実行される場合は、サーバ側で処理され、処理コマンドがクライアントで実行される場合は、クライアント側で処理されます。
- 混在式は、サーバ側とクライアント側の要因が含まれています。

どのような式が使用されているかを把握しておくことが重要です。式は、タスク内の多くの異なる場所で使用されています。式がサーバ側の式でクライアントで評価される場所で使用されている場合、処理を実行しようとすると、クライアントが停止され、リクエストはサーバ側に送り返されます。場合によっては、構文チェッカーによってエラーが表示されます。動作する場合もありますが、遅くなります。



式は、実行される場所に応じて番号カラムにマークが表示されます。

- S = **サーバ側**
- C = **クライアント側**
- M = **混在**
- **空白 = ニュートラル**

式は、処理コマンドや特性、代入値で使用されるため、知っておく必要があります。式が実行される場所に合った使い方がされるように確認してください。

データビュー

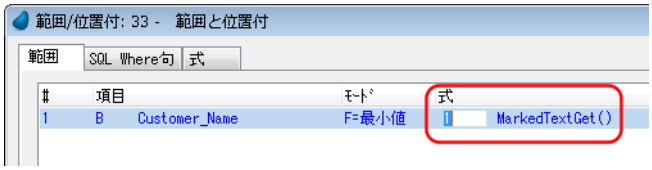
データビューは、サーバ上で構成されます。使用される実際のレコードとフィールドは、クライアントに送信される XML に含まれます。

しかし、例えば、リッチクライアントのデータビューに影響を及ぼす設定として以下のものがあります。

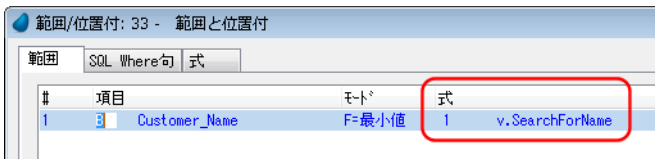
- データ項目
- 範囲と位置付
- ［代入］特性
- ［リンク］コマンド

範囲と位置付式

範囲と位置付式は（代入式と同じように）、クライアントで評価されたり、サーバ側で評価されたりします。したがって、範囲と位置付で使用される式は、ニュートラルである必要があります。



たとえば、この例では、範囲で MarkedTextGet() 関数を使用しています。サーバ側の特性でクライアント側関数を使用しているので、この場合は構文エラーになります。



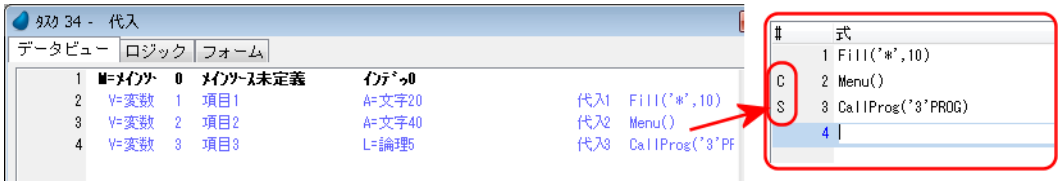
この問題を解決するには、適当な関数を使用して変数項目を更新する親タスクを作成します。次に、直接その変数を参照するようにするか、範囲に使用されるパラメータとして渡すことができます。

［代入］特性

クライアント側の式を［代入］特性で使用することはできません。［代入］特性にクライアント側の式を使用する必要がある場合、［レコード前］でこの式を使用して更新した項目を定義するようにしてください。

ノート

（例えば：タスクが登録モードの場合の実項目の［代入］特性のように）［代入］特性が実行された場合に、常にサーバへのアクセスが発生するようであれば、サーバ側の式を使用することは可能ですが推奨しません。



たとえば、この例では、3 つの変数項目が定義あり［代入］特性が定義されています。

- 最初の変数はニュートラルな式を使用しています。このため、左側にマークが表示されていません。これは、正しい使い方です。
- 2 番目の［代入］特性には Menu() 関数が使用されています。これはクライアント側の式になります。この場合、構文エラーになります。
- 3 番目の［代入］式には CallProg() 関数が使用されています。これはサーバ側の式になります。この場合も、構文チェックで警告が表示されます。

［代入］特性にクライアント側またはサーバ側の式を使用する必要がある場合、変数項目を使用する必要があります。

- サーバ側の式については、サーバへの余分なアクセスを避けるために［タスク前］で変数を更新をすることを推奨します。
- クライアント側の式については、［レコード前］で更新してください。

新しく作成されたレコードの初期値を設定する場合は、［レコード前］で更新しなければなりません。その際、［条件］特性で Stat(0,'C'MODE) を設定してください。

［リンク］コマンド

［リンク］コマンドは手続き的なコマンドではありませんが、サーバ側のコマンドです。これは、リンクが再計算されるたびに、リッチクライアントが新しいリンクを実行するためにサーバにアクセスすることを意味します。

ノート

［リンク］コマンドによる再計算によって、リッチクライアント・モジュールは、サーバエンジンを呼び出し、新しいレコードを取得します。これは、［リンク］コマンドを多用するとサーバ側へのアクセス負荷が高くなる場合があることを意味します。したがって、［リンク］コマンドの使用を制限することを推奨します。代わりになります。

代わりに［リンク結合］コマンドを使用することもできます。この場合、コマンドはメインソースに定義されたデータベースからデータを取得されて、データのかたまりとして、リッチクライアント・モジュールに送られます。

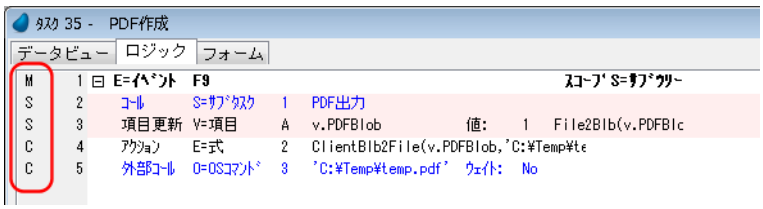
タスクフローの追跡

以上のように、リッチクライアントの処理コマンドは、ややシームレスになっています。ロジックが実行している箇所については、ユーザの立場からは簡単には説明できません。

しかし、開発者としては、プログラムを最適化することができるように、これを知っておく必要があります。

普通のオンライン・プログラミングで使用する場合とほぼ同じような処理コマンドと関数を使用することができます。大部分のロジックは、クライアントで実行されます。しかし、一部はサーバで実行されるものもあります。

クライアントで何が実行しているかを確認するには



ロジックユニットの左側には、1 文字のコードが表示されます。このコードは、処理がどこで実行するかについて示しています。

- S = サーバ
- C = クライアント
- M = 混在
- U= 不明
- E= エラー
- 空白 = ニュートラル

ノート

コマンド行が実行される場所を視覚的に表示するために色分けされます。サーバで実行される行は、開発基本色の #10 で表示され、デフォルトではピンクになっています。ニュートラルな行は、現在実行している場所でそのまま実行されるため、直前の行と同じ色になります。

処理コマンドの実行条件が False に設定されている場合は、実行されないため、実際のフローは異なるかもしれません。

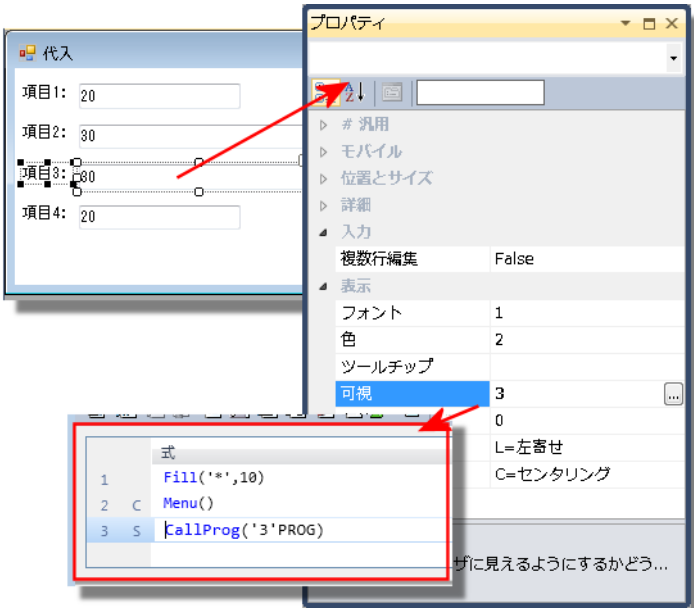
フォームとコントロール

フォームとコントロールは、ユーザとの対話用で、クライアント側の式を使用します。

ここでの問題は、[コントロール特性] に関係します。たとえば、コントロールが表示されるかどうか、あるいは、使用可能かどうかを制御したり、コントロールのフォントや色を変更する特性があります。これらの特性を式を使用して設定する場合、式は絶えず再評価されます。

サーバ側の式が特性のどれか 1 つでも使用されると、コントロールが評価されたたびに、サーバへのアクセスが発生してしまいます。したがって、コントロールにサーバ側の式を使用することができません。このような場合、構文エラーが発生します。サーバ側の式は、"S" が表示されます。

サーバ側の式を使用する



たとえば、この例では、CallProg() はサーバ側の関数のため、クライアント側のコントロールの特性では使用できません。この場合、構文エラーが発生します。

この問題を解決するには、式の値を保持するために変数項目を使用して、[タスク前] でそれを更新するようにしてください。[タスク前] はサーバで実行されるため、この処理コマンドによってサーバへの追加されたアクセスは発生しません。

コントロールによる他の問題

サブフォーム

サブフォームを使用している場合、フォームに関する別の問題があります。サブフォームを使用すると、第 4 章「複数のタスクで動作」で説明しているように、サブフォームは特定の規則に従って再表示されます。

[テーブル] コントロール

表データがある場合、テーブルの全ての行がタスク開始前に読み込まれるように、タスクの [ビュー事前読込] 特性を使用することができます。ユーザがテーブルを表示している場合、この設定によってサーバへのアクセス回数が最小になります。この機能の利便性は、テーブルのサイズに依存します。テーブルが大きすぎると、最初の読み込み時間が大きく、不便になるかもしれません。

[ツリー] コントロール

タスクが開始される前に、サーバからツリーのすべてのレコードを取得するために、[ツリー] コントロールの [ノード事前読込] 特性を使用することができます。これによって、サーバへのアクセスを最小にすることができます。しかし [テーブル] コントロールと同様に、データセットが大き過ぎる場合、最初の読み込み時間が、より速いツリー操作の利便性を上回るかもしれません。

ロジックユニット

すでにお分かりのように、ロジックユニットの多くは、どこで実行されるかは、処理コマンドに依存します。しかし、ロジックユニットのいくつかは、実行場所が固定されています。

タスク前 - サーバ

タスクの初期設定を行う時に、[タスク前]が実行されます。ここは、純粋なサーバ側のロジックユニットです。そして、クライアント側の処理コマンドは、実行されません。

クライアント・タスクが開始される前に、サーバ側の Magic xpa エンジンは、データベーステーブルをオープンし、変数項目を初期化して、範囲と位置付値を使用して、最初のデータビューを作成します。

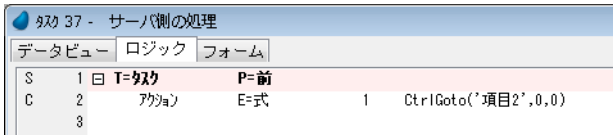
クライアント・タスクは、まだ開始されていません。処理コマンドは、サーバで実行されます。

クライアント側のプログラムとデータビューのデータは、小さく、暗号化された XML データにパッケージ化されます。そして、それはクライアント側にインターネット経由で送られます。

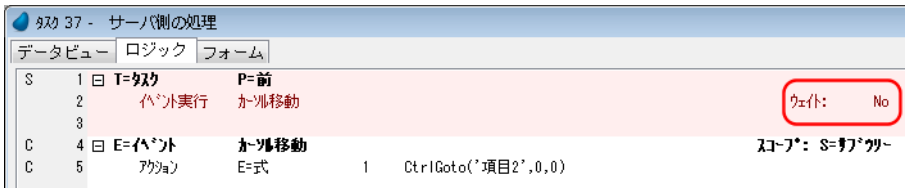
[タスク前]（タスクの初期化段階）の実行が、ほとんどがサーバ側での実行されることを知っておいてください。これは、初期化段階では、リッチクライアントを呼び出す[コール]処理コマンドが以下のルールに従っていることを意味しています。

すべての[コール]処理コマンドは指定された時間に実行されます、しかし、呼び出されたタスクのウィンドウは[タスク前]の完了後にクライアント側で開きます。他の処理コマンドが実行されたあとに、すべての[コール]処理コマンドが実行されることを意味します。

クライアント側の処理コマンドを使用する



「タスク前]は、常にサーバ側で実行されるため、ここでクライアント側の関数を使用することができません。この例では、[タスク前]でクライアント側の関数（CtrlGoTo()）を使用しています。これは、利用できません。



この問題に対応するには、サーバ側のロジックユニットで、[ウェイト]特性を「No」に設定した[イベント実行]処理コマンドを使用することです。この場合、サーバ側のロジックユニットは、サーバで実行されます。しかし、イベントそのものはクライアントで実行されます。

[タスク前]で実行箇所不明のイベント

[タスク前]で[イベント実行]処理コマンドを使用するには注意する必要があります。イベントが発行された場合、構文チェッカーはイベントがどこで処理されるかを必ずしも把握できません。「[タスク前]で実行箇所不明のイベント」（ページ 11）でこの問題に対応しています。

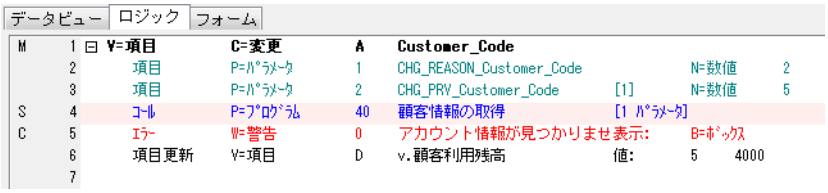
[タスク後]

タスクが終了する際、[タスク後]ロジックユニットが実行されます。これは、定義された処理コマンドによってサーバ側かクライアント側で実行されます。

タスクの呼び出しはすでに終了しているため、[タスク後]から呼び出された非モーダルのフォームで定義されたリッチクライアント・タスクは表示されません。[タスク後]ロジックユニットから呼び出されたリッチクライアントタスクは、サーバ側で実行され、すぐに終了します。つまり、タスクの[タスク前]と[タスク後]のロジックユニットは、サーバ側で自動的に実行されます。

しかし、呼ばれたタスクが並行起動の場合、この制限はありません。

項目変更



ユーザがレコードを参照する度に、[項目変更] ロジックユニットが実行されます。ロジックユニットはニュートラルです。しかし、この中にはサーバ側の処理コマンドを使用しないことを推奨します。これによって、データ項目が変更されるたびに、サーバにアクセスされることを回避できます。

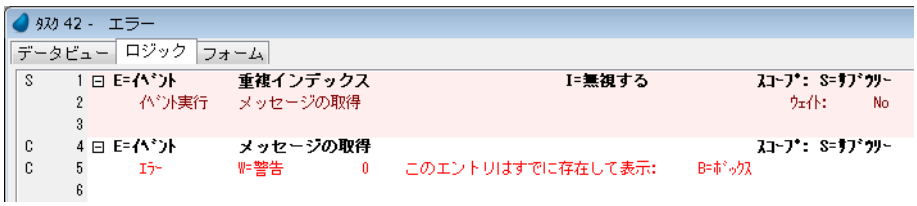
項目変更には、2 つの段階があります。再計算とコントロールの変更です。このため、サーバへのアクセスが行われると、この例のように、アクセスが 2 回発生します。したがって、クライアント側またはニュートラルな処理コマンドだけを使用するようにしなければなりません。

コントロールとレコードレベル

[コントロール前／後] と [レコード前／後] は、オンライン・プログラムと同じ規則に従って動作します。ユーザがレコードの操作をしている場合、[コントロール前／後] と [レコード前／後] はコントロールとレコードの間でユーザが操作する度に実行されます。この種の操作は、クライアント側で実行され、サーバ側での操作を必要としません。

したがって、[項目変更] ロジックユニットのように、これらのロジックユニットは、クライアント側またはニュートラルな処理コマンドのみを定義するようにしてください。この例では、[レコード前] と [コントロール前] でサーバ側の処理コマンドが定義されています。これらは、できれば、[タスク前] に移動してください。

エラーレベル



対応する DBMS エラーが発生した場合、エラーレベルのロジックユニットが実行されます。このロジックユニットは、サーバで実行されます。

確認ダイアログボックスを表示するロールバックオプションは、リッチクライアントでは実行されません。これは、ロールバック関数の最初のパラメータが無視されることを意味します。また、エラーイベントのメッセージ特性も、無視されます。

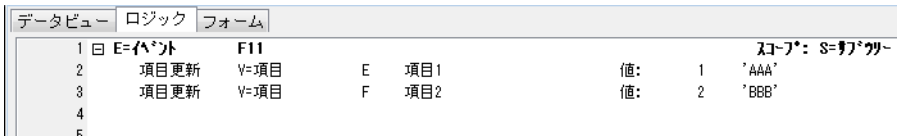
このため、ユーザにエラーを表示させたい場合は、非同期イベント（ウェイト =No）を使用してください。[エラー] ロジックユニットの終了後、イベントが実行される要因となります。

エラー処理の詳細については、Magic xpa のリファレンスヘルプのエラー処理のセクションを参照してください。

システムとユーザイベント

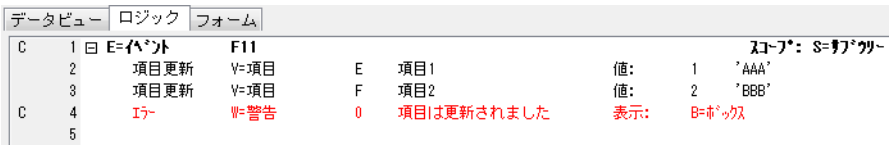
システムイベントとユーザイベントに対するイベントロジックユニットがどこで実行されるかは、定義された処理コマンドに依存します。簡単な例を見てみましょう。

ニュートラル側

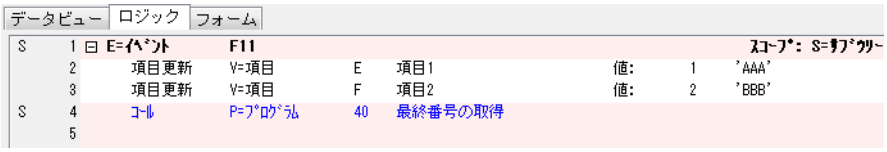


どちらからでも実行することができます。このようなロジックユニットは、ニュートラルとして扱われます。呼び出された場所で実行され、相手側と通信する必要がありません。[イベント] ロジックユニットの行番号にはコードは表示されません。[項目更新] 処理コマンドも、ニュートラルとして扱われます。これは、どちらからでも実行することができます。

クライアント側

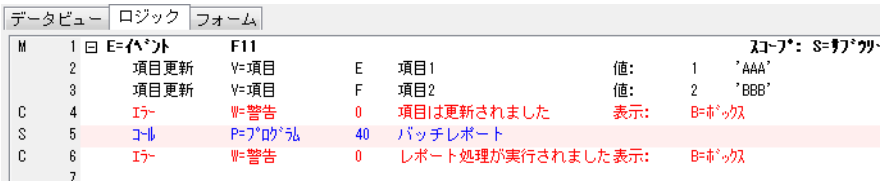


次に、[エラー] 処理コマンドが追加されると、[イベント] ロジックユニットはクライアント側で実行されるようになり、「C」というコードが表示されます。エンドユーザーにメッセージを表示する必要があるため、[エラー] 処理コマンドはクライアントでのみ実行することができるからです。[項目更新] 処理コマンドはニュートラルで、呼び出された場所で実行されます。[エラー] 処理コマンドを実行する必要がある場合、(すなわち、[エラー] 処理コマンドの実行条件が True と評価された場合)、[エラー] 処理コマンドはクライアントで実行されます。



代わりに、バッチタスクへの呼び出しが追加されると、[イベント] ロジックユニットはサーバ側となり、「S」というコードが表示されます。これは、バッチタスクが常にサーバで実行されるからです。

混在



しかし、2 つの処理が混在している場合はどうなるでしょうか？

クライアント側とサーバ側の両方の処理コマンドが使用された場合、イベントは混在しているものとして扱われ、「M」というコードが表示されます。クライアントとサーバ間の通信によってタスクの動作が遅くなる可能性があるため、処理コマンドを混在して使用することはできるだけ避けてください。

不明

処理コマンドによっては、どちらで実行されるか判らないものもあります。例えば、[イベント実行] 処理コマンドは、このタスクでは、親タスクか、メインプログラム、または、コンポーネントで実行されるかもしれません

ロジックユニット内の処理コマンド

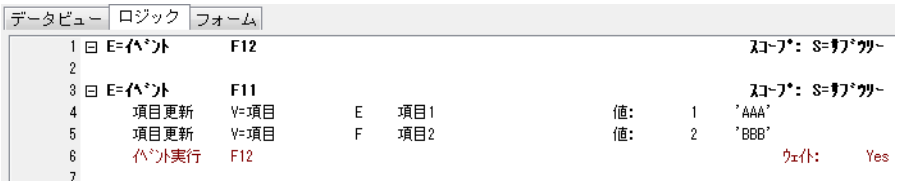
各処理コマンドは、サーバ側かクライアント側、またはニュートラルか不明として扱われます。ニュートラルな処理コマンドは、サーバかクライアントのどちらかで実行されます。

処理コマンド	実行場所		
エラー		クライアント	常にクライアントで実行
コール	バッチ	サーバ	常にサーバで実行
	［出力先］特性を使用		クライアントへの即時アクセスが、現在サブフォーム上で実行しているタスクを閉じるためにある場合、混在と扱われます。
	［ウィンドウタイプ］が「モーダル」	混在	クライアントとサーバで部分的に実行します。これはタスクを読み込まむためにサーバにアクセスして、それを開始するためにクライアントに制御が移ります。そして、呼び出しタスクは停止します。
	［ウィンドウタイプ］が「非モーダル」	サーバ	タスクを読み込むためにサーバにアクセスします。次に、呼び出されたタスクの実行を待つことなく、処理が継続します。
外部コール	UDP	サーバ	処理の一部はサーバ側で実行されます。
	Web サービス	サーバ	処理の一部はサーバ側で実行されます。
	OS コマンド	状況依存	［実行］特性に依存します。
	COM	なし	サポートしていません。
アクション		状況依存	評価される処理に依存します。
ブロック		状況依存	ブロック式に依存します。
項目更新		状況依存	更新されるデータ項目と項目で使用される式に依存します。
イベント実行		状況依存	発行されるイベントに対応するロジックユニットに依存します。
フォーム		なし	サポートしていません。 サーバ側でで帳票出力処理を実行して、結果をクライアントに表示することができます。 「PDF を表示する」を参照してください。

処理コマンドは、それらの特性（例えば、条件）で使用される式にも影響します。

［イベント実行］処理コマンド

ニュートラル



通常、Magic xpa は、［イベント実行］処理コマンドがどこで処理されるかを見分けることができます。たとえば、この例では、イベントはこのタスク内で処理されます。F12 を押下した場合のロジックユニットはニュートラルで、［伝播］特性を「No」設定すると、［イベント実行］処理コマンドもニュートラルになります。

サーバ側

しかし、F12 を押下することで実行されるロジックユニットがサーバ側である場合、[イベント実行] 処理コマンドはサーバ側で実行されます。この場合、F11 に対するロジックユニットもサーバ側になります。

不明

この例では、F12 に対応するイベントロジックユニットは、タスク内に存在していません。スタジオは、[イベント実行] 処理コマンドが実行時にどちらで処理されるかを判断できないため、不明として扱われます。これは、以下のような場合において発生します。

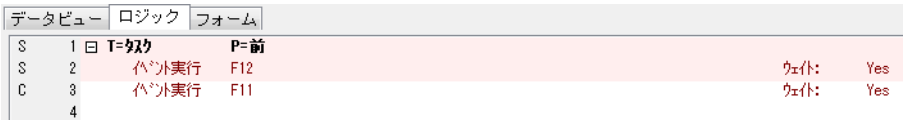
- イベントのロジックユニットが、異なるタスクに定義されている場合。実行時にどのタスクがイベントを処理するかは開発時には判断できません。このような場合、主に実行時のタスクツリーに依存します。
- 現在のタスクのイベントに対するロジックユニットは、[伝播] 特性が「Yes」に設定されている場合。実行中は、タスクがイベントをどこで処理するかは、開発時には、判断できません。この場合も、主に実行時のタスクツリーに依存します。

[イベント実行] 処理コマンドでのメモ：

- **[イベント実行] 処理コマンドの [ウェイト] 特性が「No」に設定されている場合、サーバ側の処理コマンドの定義行内の一部でないならば、全てのイベントの処理の後で実行されるため、(どちらの側にも即時にアクセスが発生しないため) それはニュートラルとして扱われます。**
- **サーバ側の処理の行内に [イベント実行] 処理コマンドが含まれている場合、実行時にエラーが発生する場合があります。**

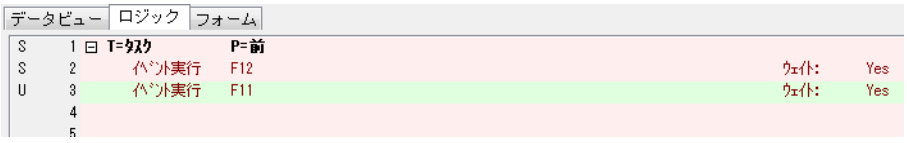
[タスク前] の [イベント実行] 処理コマンド

[タスク前] は、サーバ側で常に実行されます。



[タスク前] でクライアント側の処理コマンドを入力すると、エラーが発生します。

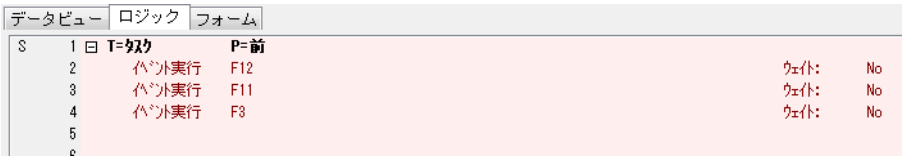
サーバ側のロジックユニットには、クライアント側の処理コマンドを含めることができません。[タスク前] の後でこの処理を実行するために、[ウェイト] 特性を「No」に設定することができます。



[タスク前] に不明の処理コマンドが定義されている場合、警告が発生します。

不明の処理コマンドがサーバ側のロジックユニットに定義されていて、実行時にクライアント側で評価されたる場合、エラーが発生する点に注意してください。

その際、不明の処理コマンドが本当はサーバ側であることを確認するかどうかは、開発者に依存します。

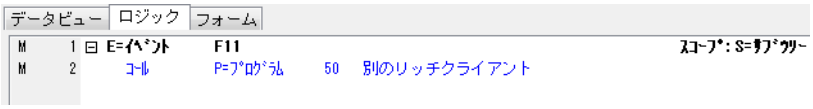


しかし、[イベント実行] 処理コマンドの [ウェイト] 特性を「No」に設定することで問題は解決します。サーバ側のイベントがサーバで発生した後、クライアント側のイベントはクライアントで発生します。

[コール] 処理コマンド

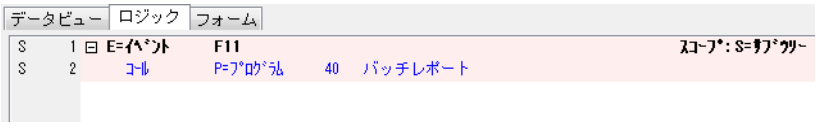
リッチクライアント・タスクは、オンライン・プログラムと同じような方法で、別のリッチクライアント・タスクを呼び出すことができます。その際、[コール] 処理コマンドを使用して、パラメータを渡すことができます。呼び出されたプログラムは、サブフォームまたはフレームセット内に表示したり、個別のウインドウを表示して実行したり、モーダルウィンドウを表示して実行したり、並行に実行されることもできます。

別のリッチクライアント・タスクを呼び出す



リッチクライアント・タスクが別のリッチクライアントタスクを呼び出すとき、混在処理として扱われます。サーバはタスクをパッケージ化する必要があります。そして、クライアントで実行する必要があります。

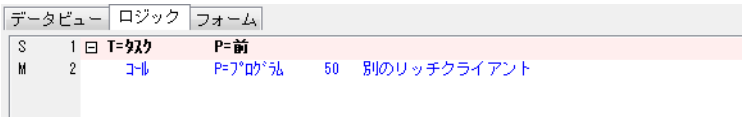
バッチタスク



バッチタスクは、サーバ側で処理されます。このため例えば、複雑な帳票を作成したい場合、帳票出力用のバッチタスクはサーバ側で実行されます。最終結果としての帳票は、バッチタスクが終了した後、クライアントで表示させることができます。

バッチタスクへの呼び出しは、常に同期型です。これは、クライアントはバッチタスクが完了するまでの待つことを意味しています。リッチクライアントからバッチタスクを呼び出す場合は、開始と停止を行うためのダイアログはサポートされません。

[タスク前] の [コール] 処理コマンド



[タスク前] からリッチクライアント・プログラムを呼び出そうとすると、エラーが発生します。[タスク前] では、クライアント側のプログラムを呼び出すことができません。

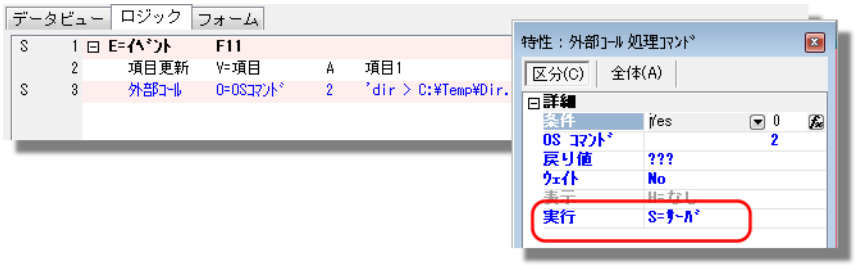
しかし、「クライアント側の処理コマンドを使用する」（ページ 11）で説明しているように、クライアント上でタスクが開始された時点で実行されるプログラムを呼び出すようにするには、[イベント実行] 処理コマンドの [ウェイト] 特性を「No」に設定して使用することで可能になります。

[コール OS] 処理コマンド

[コール OS] 処理コマンドは、興味深い処理コマンドです。これは、OS のネイティブなコマンドを呼び出します。1 つの PC 上で実行している場合は、明確に理解できるはずです。しかし、リッチクライアント環境では、どの様な意味を持つのでしょうか。実際は、クライアント、またはサーバ上で OS コマンドを実行したいかもしれません。

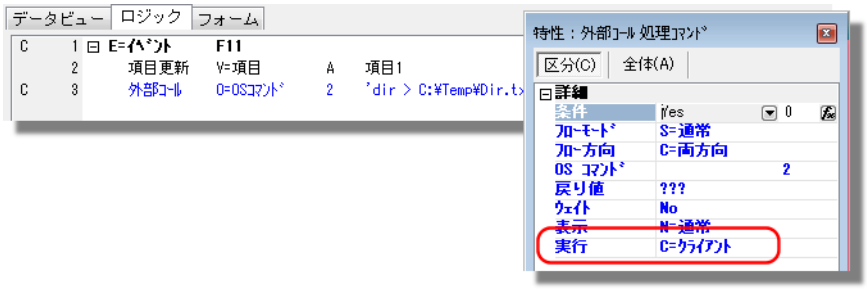
どのような環境でコマンドを処理するかを指定することは、開発者に依存します。

サーバ側で呼び出す



ここでは、簡単な OS コマンドとして、現在のディレクトリを表示し、それを一時的なファイルにリダイレクトする例を示しています。ここでの問題は、サーバ PC、または、クライアント PC のどちらのディレクトリを表示するのでしょうか？ これは、［実行］特性を設定することで開発者によって制御できます。ここでは、サーバ側に設定されています。これによって、この処理コマンドはサーバ側の処理コマンドとなります。

クライアント側で呼び出す



ここでは、同じ処理コマンドが定義されていますが、［実行］特性は「クライアント」と設定されています。これによって、処理コマンドはクライアント側で実行されます。このコマンドが実行された場合、クライアント側のディレクトリが表示されます。

［コール OS］処理コマンドの用途

クライアントまたはサーバ上で実行している OS に対応した［コール OS］処理コマンドを使用することができます。

上記の例では、同じコマンドは実行されます。

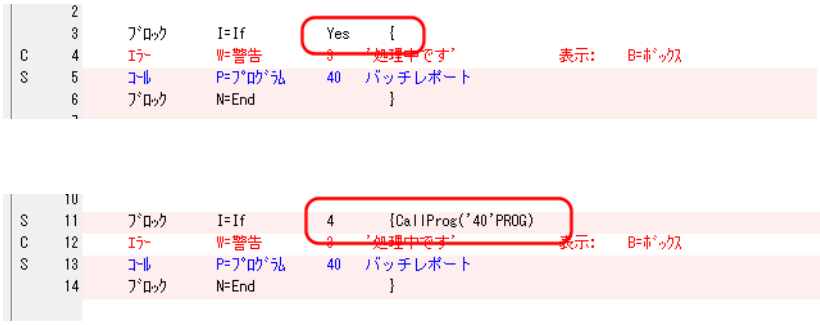
```
dir > C: ¥Temp¥Dir.txt
```

しかし、1 つのインスタンスでは、サーバ側のディレクトリが表示され、別のインスタンスでは、クライアント側のディレクトリが表示されます。

OS にアクセスする Magic xpa の関数もあります。このため、［コール OS］処理コマンドで OS に直接アクセスすることは一般的に必要ではありません。たとえば、OS の環境変数の値を取得したい場合は、OSEnvGet() 関数を使用することができます。

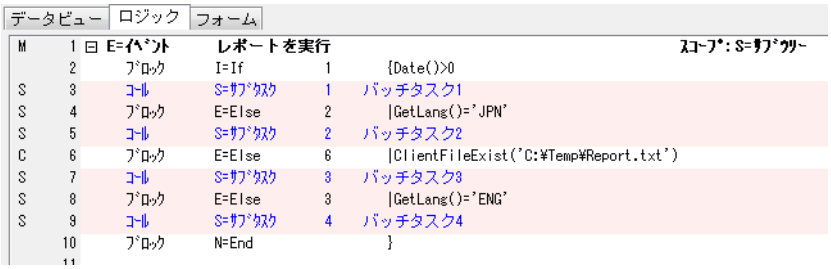
OSEnvGet() 関数は、サーバ環境変数を取得します。この関数のクライアント版（ClientOSEnvGet）は、クライアント側の環境変数を取得できます。別の関数ペアとして、ファイル情報（例えば、FileInfo() と ClientFileInfo() にアクセスするものもあります。

［ブロック］処理コマンド



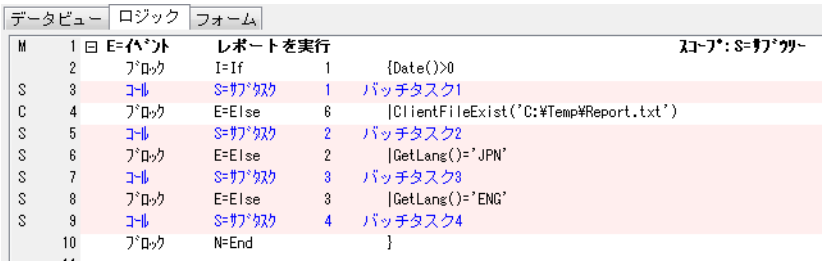
［ブロック］処理コマンドは、条件式の内容に基づいてクライアント側またはサーバ側で実行されます。

[ブロック] 処理コマンドの最適化



[ブロック IF] 処理コマンドが定義されている場合、クライアント側とサーバ側をグループ化するようにしてください。True が設定された行が見つかるまで、モジュールは各条件を評価します。そして、True の条件が見つからない場合、この例のようになります。

- #1. クライアント側で実行されます。
- #4. サーバ側で実行されます。
- #6. クライアント側で実行されます。
- #8. サーバ側で実行されます。
- #10. クライアントに戻ります。

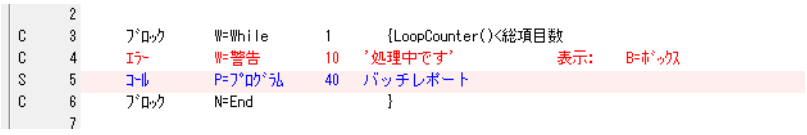


次に、行を入れ替えることで、以下のように変更することができます。

- #1 と #4 は、クライアント側で実行されます。
- #6 と #8 は、サーバ側で実行されます。
- #10 で、クライアントに戻ります。

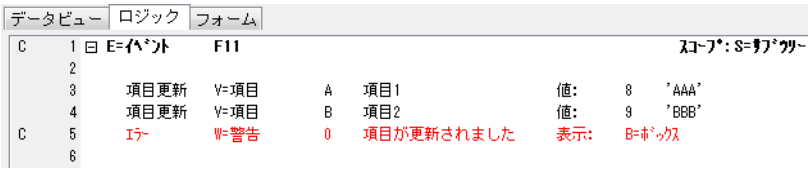
GetLang() 関数の値を保持するために変数を使用して、これをさらに最適化することで、二度も実行する必要がなくなります。GetLang() が［タスク前］で実行させるようにすることで、ブロック内の比較処理はクライアント側になります、バッチタスクが呼び出されるまで、サーバへのアクセスは必要なくなります。

ブロック Loop



上図のように同じループ処理内にサーバとクライアントの両方の処理が混在して定義すると、クライアントとサーバ間でのやり取りが頻繁に行われるため、あまり良いことではありません。

[エラー] 処理コマンド



[エラー] 処理コマンドは、クライアント側の処理コマンドです。ダイアログボックスや、ステータスバーにメッセージを表示するために使用することができます。

[表示] 特性を「ライン」に設定してメッセージを表示させた場合、SDI ウィンドウのステータスバーに表示されます。SDI ウインドウを使用していない場合、メッセージは表示されません。[表示] 特性を「ボックス」に設定した場合、普通のオンライン・プログラムと同じようにメッセージボックスが表示されます。

[エラー] 処理コマンドはクライアント側で実行されるため、[タスク前] 使用することはできません。

[項目更新] 処理コマンド

[項目更新] 処理コマンドはニュートラルです。実行される場所は、使用されている式と更新されているデータ項目に依存します。

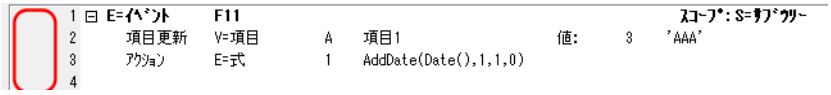
[項目更新] 処理コマンドの実行場所は、以下内容に依存します。

- ロジックユニットが実行される場所
- 更新式で使用される関数とデータ項目
- 更新されるデータ項目の処理場所

[アクション] 処理コマンド

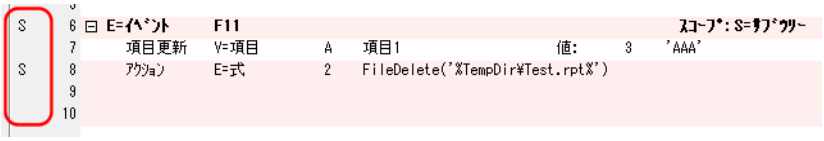
[アクション] 処理コマンドが実行される場所は、どのような式が実行されるかに依存します。式は、複数の関数とデータ項目から構成される場合があります。各々がクライアント側であったりサーバ側、ニュートラルになります。

ニュートラル



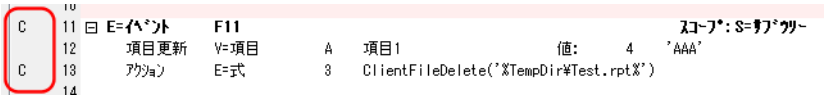
ここでは、AddDate() 関数が式で使用されています。この関数は。ニュートラルです。このため、[アクション] 処理コマンドもニュートラルになります。

サーバ側



しかし、FileDelete() 関数が使用されると、処理コマンドはサーバ側になります。FileDelete() は、サーバ側の関数です。

クライアント側

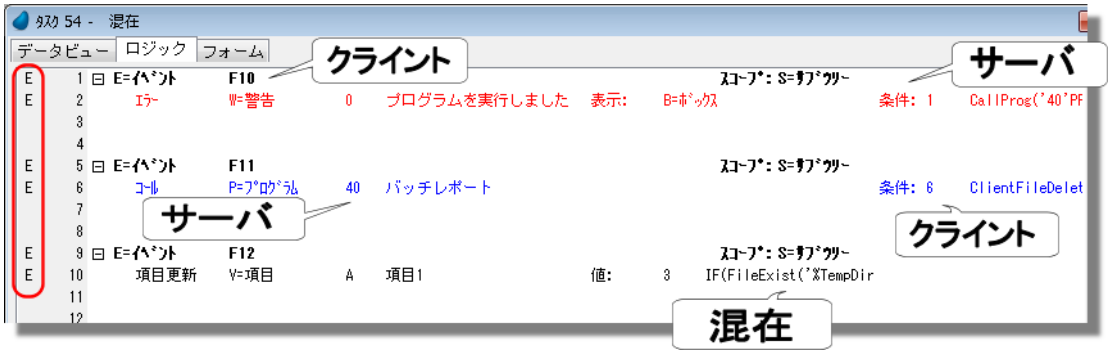


また、ClientFileDelete() 関数はクライアント側です。このため、[アクション] 処理コマンドはクライアント側になります。

エラー

クライアントとサーバ側の関数またはデータ項目を結合する式を作成することもできます。この例の場合、［アクション］処理コマンドによって評価される式はエラーとして扱われ、「E」というコードが表示されています。これは、混在した式として使用しているからです（「M」というコードが表示されます）。混在式には、サーバで実行される関数（FileExist）とクライアントで実行される関数（ClientOSEnvGet）が含まれています。

混在した処理

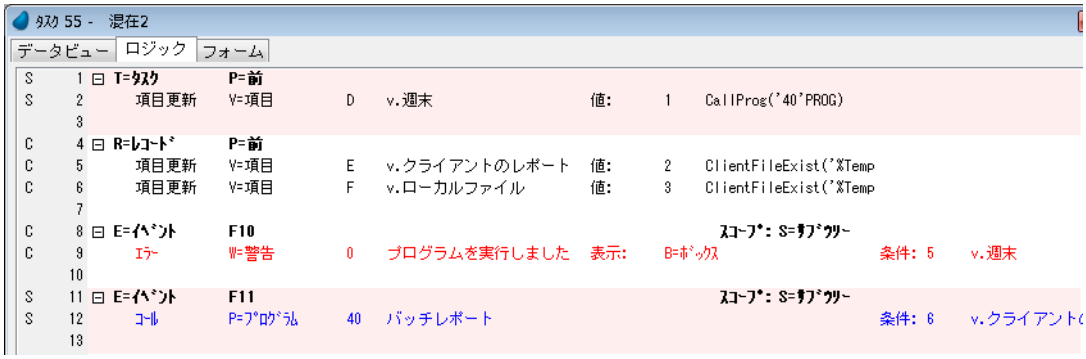


混在した処理は、サーバ側の処理コマンドまたは式が、クライアント側の処理コマンドと一緒に使用されているものです。

このようなことが発生する要因には、以下のものがあります。

- クライアント側の処理コマンド（［エラー］処理コマンドのような）が、処理コマンドの特性（条件）の一部にサーバ側の関数を使用している場合（上記の F10 の［イベント］ロジックユニットを参照）。
- サーバ側の処理コマンドが、特性（例えば条件）の一部にクライアント側の関数を使用している場合（上記の F11 の［イベント］ロジックユニットを参照）。
- 混在した式。一つの式に、クライアント側とサーバ側の関数を使用している場合（上記の F12 の［イベント］ロジックユニットを参照）

これらの場合、構文チェッカーによってエラーが発生します。



この種のロジックを実装するには、複数の処理コマンドを使用して、変数項目に式の結果を保存する必要があります。変数項目は、同じ場所での処理が連続するように最も有効な場所で更新する必要があります。

この例では、変数項目（v. 週末）を使用しています。そして、［タスク前］で CallProg() 関数を使用して更新しています。この処理は、F10 の［イベント］ロジックユニット内で使用されています。この場合、クライアント側になります。

もう一つの変数項目（v. クライアントのレポート）は、［レコード前］で更新されます。これは、F11 の［イベント］ロジックユニット内で使用されています。この場合、サーバ側になります。

類似したロジックは、F12 の［イベント］ロジックユニット内で使用される混在式でも使用されています。元の式は混在しており、「M」というコードが表示されます。

修正した式は、ClientFileExist() 関数の結果を［レコード前］（クライアント側で処理される場所であればどこでも結構です）で変数項目に格納します。これによって、式はサーバ側となるため動作します。

> **ノート**
>
> 使用したい関数がクライアント側かサーバ側、または混在かが分からない場合は、関数一覧を表示してください。関数一覧の左下に表示されます。

グループ処理

混在した［イベント］ロジックユニットがある場合、できる限りサーバ側の処理コマンドとクライアント側の処理コマンドをまとめることが最良です。これによって、ネットワーク上のトラフィックが低減されます。

リッチクライアントでの処理の並び方は以下の通りです。

1. クライアント側を開始（イベントが発行された時、セッションが開始された場所）
2. GetLang() 関数で変数項目更を更新（サーバ）
3. 警告を表示（クライアントに戻ります）
4. プログラムの呼び出（サーバ）
5. プログラムの表示（クライアントに戻ります）

この場合、クライアントとサーバ間の処理が 4 回切り替わります。

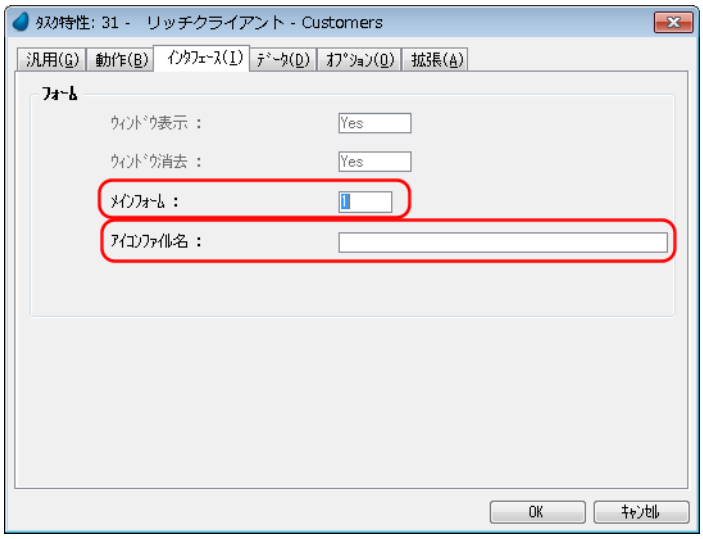
処理行の順序を変更することで、この切り替えを 2 回に変更することができます。

1. 同じようにクライアントを開始します。クライアント上で警告を表示します。
2. サーバで GetLang() 関数を実行します。サーバ上でプログラムを呼び出します。
3. プログラムを表示するためクライアントに戻ります。

当然ですが、可能な最適化の順序は、プログラムのロジックに依存します。

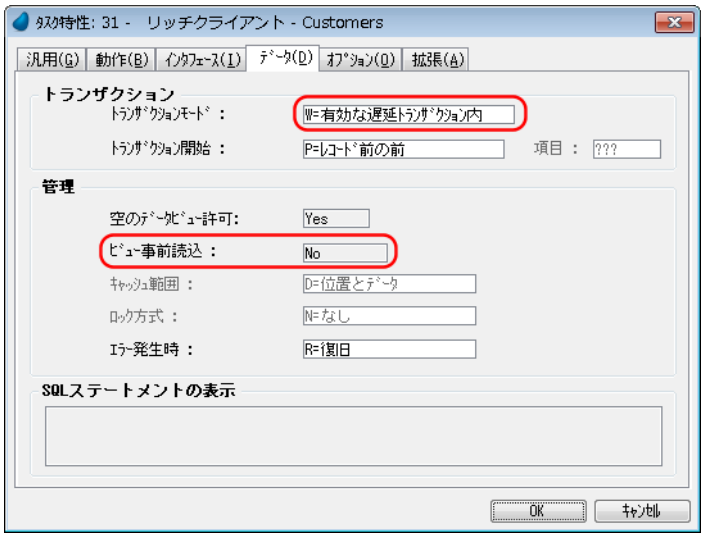
リッチクライアントタスクの設定

ほとんどのリッチクライアントタスクの設定は、オンラインタスクと同じように行うことができます。ただし、リッチクライアントタスクには関係ない設定もあります。また、リッチクライアントタスクにだけ関係する設定もあります。ここでは、これらの設定のいくつかについて説明します。



メインフォーム：
通常のオンラインタスクと同様に、複数のフォームを定義し、［メインフォーム］特性を使用して実行時に表示されるフォーム番号を式で指定することができます。

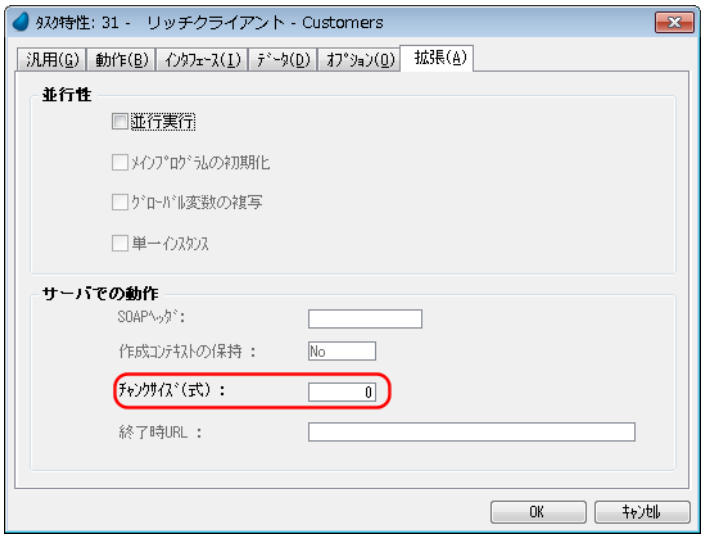
アイコンファイル名：
通常のオンラインタスクと同様に、リッチクライアント・タスクで表示されるアイコンを定義することができます。アイコンファイルが定義されない場合は、メインプログラムの［タスク特性］で定義されるアイコンが表示されます。



トランザクションモード：
リッチクライアントタスクは、「W= 有効な遅延トランザクション」（上位タスクでオープンされているトランザクションを利用する場合）または「N= 新規のトランザクション」（トランザクションが既にオープンされていても、別のトランザクションを新規にオープンしたい場合）、および「N= なし」のみ設定できます。

「トランザクション」のセクションで詳細に説明します。

ビュー事前読込：
この特性は、事前にデータビュー全体を取得するかどうかを定義します。値が「True」に設定されると、サーバはタスクの初期設定中にデータベースからすべての関連するレコードを取得します。これは小さなテーブルに対しては、非常に便利です。この特性を「True」に設定した場合、［チャンクサイズ］特性の値をデータベースから取得されたすべてのレコードを保持する上で十分なサイズに設定することを推奨します。



チャンクサイズ：

［チャンクサイズ（式）］特性は、追加レコードとしてクライアントに渡されるレコード数を定義します。

例えば、この特性が「100」に設定された場合、アプリケーションサーバでタスクが起動されると、最初の 100 レコードがクライアントに渡されます。これによって、エンドユーザはローカル上で最初の 100 のレコードを参照することができるようになります。エンドユーザが、レコードの与えられた範囲を越えてスクロールしようとすると、クライアントはサーバに通知し、追加の 100 レコードを受け取ります。

レコードの集まりは、レコードのローカルキャッシュとしてクライアントに蓄積されます。エンドユーザがテーブルの最後や先頭に移動すると、ローカルキャッシュは消去され、キャッシュはレコードの 1 回分のチャンク数のレコード群を格納します。

ノート

チャンクサイズが大きすぎると、パフォーマンスが低下する可能性があります。データが転送される度に、多くのデータがサーバからクライアントに渡ります。

他方、レコード長が大きい場合（たとえば、大きな BLOB 型項目が含まれている場合など）は、チャンクサイズを小さくした方が良い場合もあります。

クライアント関数

クライアント側のファイルやフォルダへのアクセス

リッチクライアントタスクでは、クライアント側にアクセスするための専用の関数を使用することができます。

関数には、以下のものがあります。

- ClientBlb2File
- ClientDirDlg
- ClientFile2Blb
- ClientFileCopy
- ClientFileDelete
- ClientFileExist
- ClientFileInfo
- ClientFileListGet
- ClientFileRename
- ClientFileSize
- ClientFileOpenDlg
- ClientFileSaveDlg
- ClientFileToServer

ローカルマシンの環境変数の取得

クライアントマシンの一時ディレクトリのような環境情報を取得する必要があるかもしれません。以下の関数を使用することで、このようなことが可能になります。

- ClientGetUniqueMachineID
- ClientOSEnvGet

ローカル環境の設定

環境設定を行うには、次の関数を使用します。

- ClientSessionSet

セッション情報を取得

現在のセッションに関する情報を取得する場合、以下の関数を使用することができます。

- ClientSessionStatisticsGet

第 4 章 複数のタスクで動作

［コール］処理コマンドを使用してリッチクライアント・タスクから、別のプログラムを呼び出すことができます。呼び出すプログラムは、リッチクライアントかバッチ・タスクである必要があります。

オンライン・プログラミングで使用するような形式でプログラムを呼び出すことができます。

- **選択プログラム** …… オンラインアプリケーションと同じように使用することができます。
- **モーダルウィンドウ** …… そのウィンドウが閉じるまで別のウインドウにアクセスできなくなります。
- **並行タスク** …… 呼び出したタスクと独立して実行することができます。

サブフォームやフレームを使用してプログラムを暗黙的に呼び出すこともできます。サブフォームまたはフレームは、［出力先］特性を使用することで実行時に、異なるプログラムを表示させることもできます。

ウィンドウタイプ

実行時に異なる方法で実行するタスクを設定することができます。たとえば、サブフォームで実行するタスクを定義したり、スタンドアロンで実行するタスクを定義することができます。このうちのほとんどが、サブフォームでも動作するため、自動的に行われます。

しかし、異なる［ウィンドウタイプ］特性を使用して実行するタスクが必要な場合、そのタスクのために 2 つの異なるフォームを準備する必要があります。そして、どのフォームを表示するかを［メインフォーム］特性で指定します。

他のプログラムを呼び出す

呼び出されたタスクの初期化

［タスク前］は、サーバ側で常に実行されます。したがって、リッチクライアント・タスクまたはプログラムを呼び出す場合、サーバへのアクセスが行われる間、クライアントの現在の処理は休止しなければなりません。呼び出されたタスクの［タスク前］はサーバ側で実行されます。そして、ウインドウはクライアントで表示されます。

バッチタスクの呼び出し

リッチクライアントからバッチタスクを呼び出すことができます。バッチタスクが終了するまでリッチクライアントの処理は休止します。

バッチタスクは常にサーバで実行されます。たとえば、OS ファイルを使用する場合、気をつけておいてください。入出力ファイルは、サーバ PC 上のファイルにアクセスします。場合は、この処理のための関数があります。

リッチクライアントからバッチタスクを呼び出した場合は、開始／終了のダイアログボックスはサポートされません。

コンテキスト管理

リッチクライアントタスクを新規に実行した場合、自のコンテキストで動作します。コンテキストは、タスクが実行する瞬間から終了する瞬間までタスクの状態を記録します。各コンテキストには、個別にユニークなコンテキスト ID が割り当てられます。

並行タスクを開始すると、新しいコンテキスト ID を持ったコンテキストは作成されます。並行タスクの子タスクは、（子タスクも並行実行の設定がされていない限り、設定されている場合は、さらにもう一つの新しいコンテキストが割り当てられます）同じコンテキストで開きます。

各コンテキストは、他のコンテキストとは独立しています。タスクツリーやデータベース・カーソル、SetParam() 関数によって設定されるデータ、データ操作ステートメントのすべては、コンテキストごとに個別に追跡されます。

Magic xpa は、各コンテキストの動作を追跡します。サーバは、各クライアントのリクエストを属しているコンテキストをもとに識別します。どのタスクまたはサブタスクが開いているか、データベースカーソルとデータ操作ステートメントに関する情報は、コンテキストごとに全て維持されます。

ノート

一旦、コンテキストが Magic xpa エンジンで作成されると、クライアントからのすべてのリクエストは常に同じエンジンに送られます。

外部 HTTP ロードバランサーを使用した場合、クライアントとサーバ間の粘着性（つまり、各クライアントから最初のリクエストが作成された Web サーバへのダイレクトリクエストへのロードバランサー機能）を確実にする必要があります。。これをサポートするために、各 RIA リクエストには、MgxpaRIAglobalUniqueSessionID という名前のヘッダが含まれます。このヘッダは、特定のエンドユーザセッションの各 HTTP リクエストに同じユニークな値を保持します。ヘッダには、各クライアント・インスタンスに対する異なるユニークな値が含まれています。外部 HTTP ロードバランサーは、粘着性をサポートするために、このヘッダを使用することができます。

コンテキスト非稼働タイムアウト

サーバ側で保持されるコンテキスト情報は、メモリリソースを必要とします。たくさんのシステムリソースを使用する可能性があるコンテキストを保持する場合、サーバの負荷を軽減するためコンテキストにタイムアウトを設定することができます。

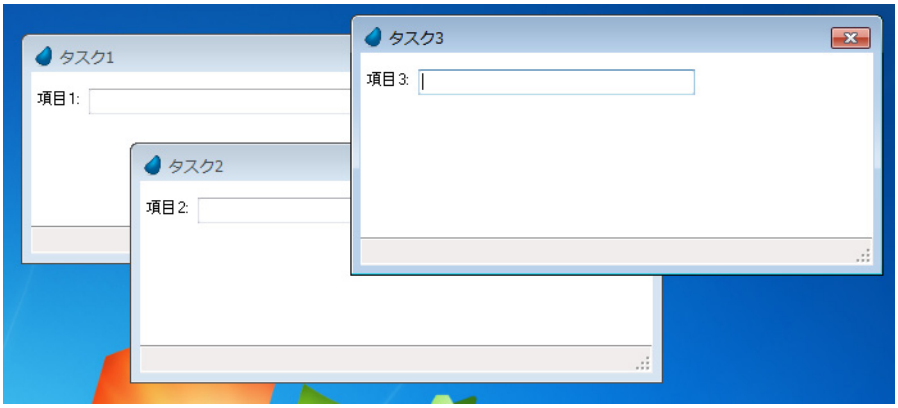
［コンテキスト非稼働タイムアウト］の設定は、［オプション \ 設定 \ 動作環境 \ アプリケーションサーバ］にあります。この設定は、クライアントが非稼働状態をチェックする時間間隔を指定します。「コンテキスト非稼働」とは、リッチクライアントタスクの実行中にクライアント / サーバ間のアクセスがない状態を意味します。

［コンテキスト非稼働タイムアウト］は、1/10 秒単位で設定します。デフォルト値は 36000（1 時間）です。

コンテキストがタイムアウトに到達すると、コンテキストはサーバから削除されます。このタイムアウトを設定することで、放棄されたコンテキストがサーバ上に蓄積されることを防止することができます。

リッチクライアントタスクが開発モードで実行された場合、［コンテキスト非稼働タイムアウト］は無制限になります。

並行処理



リッチクライアントタスクの並行動作は、オンラインタスクとは異なります。リッチクライアントタスクは、その親タスクとその実行時のタスクツリー内の他のすべてのタスクと並行して実行されます。この例では、タスク 1 はタスク 2 を呼び出しています。さらにタスク 3 を呼び出しています。しかし、一旦すべてのタスクが開いた場合、ユーザは自由に 3 つのウインドウの間をクリックすることができます。

しかし、非リッチクライアントのオンラインプログラムの場合、サブフォームを使用するか、［ウィンドウ消去］特性を「No」に指定しない限り、タスク 2 をクリックするとタスク 3 は自動的に停止します。この点については、ウインドウの動作をよりオンラインプログラムのようにしたいならば、モーダルに設定することで実現できます。これによって、上位のタスクで変更処理を行う場合、まず下位のタスクを終了させることが必要になります。

リッチクライアントと非リッチクライアントの両方のタスクツリーでは、最上位のタスクを閉じると、全てのツリーを閉じます。この例では、タスク 1 を閉じると、タスク 2 とタスク 3 も閉じます。3 つの全てのタスクを完全に独立させたい場合は、［並行実行］特性を「Yes」にしてください。しかし、この場合は、管理する新規コンテキストが作成されるため、この機能は、必要な時だけ使用してください。

非インタラクティブタスク

バッチタスクはサーバ上で実行されるため、リッチクライアントタスクを呼び出すことはできません。しかし、(インタラクティブなタスクを呼び出すという目的で) 時々クライアント側でバッチ処理を実行させたいという必要性が発生します。これは、[タスク特性] 内の [インタラクティブ] 特性のチェックを外すことで実現できます。

非インタラクティブなリッチクライアントタスクは、ユーザの入力処理を待たずに自動的にレコードを処理し、メインソースのすべてのレコード処理を終了するか、[タスク終了] 特性の条件を満たした場合にタスクが終了されるという点を除いて、インタラクティブなリッチクライアントタスクと同じように動作します。

サブフォーム

サブフォームは 2 つのタスクをマージすることができます。このため、複数のタスクを同フォーム上で実行させることができます。これは、受注入力処理のように、テーブルを表示するために使用されます。

サブフォームを追加する方法は、オンラインタスクの場合とほぼ同じです。サブフォームで使用されるタスクは、リッチクライアントタスクでなければなりません。

サブフォームの初期設定

親タスクが開始されると、親タスクの [タスク前] と [レコード前] が実行されます。

次に、サブフォームの [タスク前] が実行されます。そして、[レコード前] は最初のレコードの処理が実行されます。

サブフォームの再表示

リッチクライアントのサブフォームは、オンラインタスクと同じ規則で再表示されます。[自動再表示] 特性が「Yes」に設定され、パラメータが渡されると、サブフォームはパラメータの値が変更される度に自動的に再表示されます。[サブフォーム再表示] イベントは、サブフォームを再表示するために使用することができます。

しかし、サブフォームを再表示しても [タスク前] は実行されません。[レコード前] は、レコードが再度読み込まれることで実行されます。

[非表示の再表示] 特性

サブフォームには、[非表示の再表示] 特性があります。これは、サブフォームの初期化と再表示のタイミングを制御します。

この特性が「Yes」に設定されると、サブフォームが表示されているか否かに関わらず、動作に違いはありません。これは、以下のこと意味しています：

- サブフォームタスクの初期設定は、データビューの最初のレコードの [レコード前] の後で実行されます。
- サブフォームの再表示は、[自動再表示] 特性や [サブフォーム再表示] イベントによって行われます。

この特性が「No」に設定されている場合、サブフォームが表示されていない時は、サブフォームタスクの初期設定と再表示のみ実行されます。これは、以下のこと意味しています：

- 各サブフォームタスクを実行する前に、その可視性がチェックされ、表示されていない場合、サブフォームタスクは実行されず、遅延されます。一旦クライアントがサブフォームの表示を要求すると、サーバ側を呼び出し、サブフォームタスクは初めて表示されます。
- サブフォームの［タスク前］の実行は、サブフォームが表示されるまで遅延されます。
- サブフォームの［可視］特性の式がクライアント側で実行される場合、タスクが読み込まれた時点でサブフォームが表示されます。
- サブフォームの再表示を実行する前に、サブフォームの可視状態がチェックされ、表示されていない場合、サブフォームのデータは再表示されません。そして、表示されるようになると、Magic xpa はサーバにアクセスし、サブフォームのデータを再表示します。

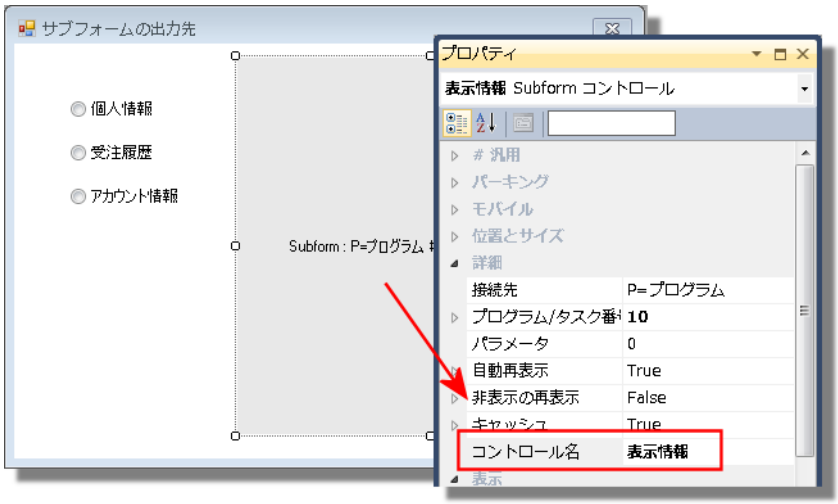
［タブサイクル］特性

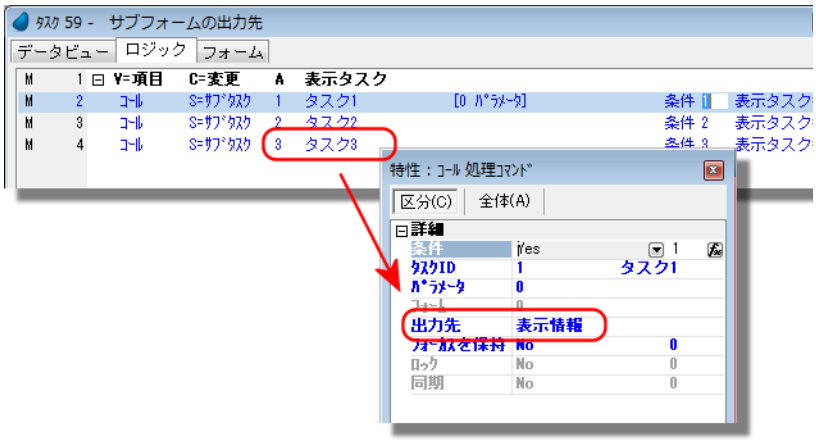
サブフォームタスクには、［タブサイクル］があります。これは、サブフォームの最後のパーク可能なコントロールから TAB キーを押下した場合の動作を定義します。

サブフォーム / フレームでの連続するタスクの動作は、そのタスク内の特性で決定され、親タスクでは定義しません。

［出力先］特性

サブフォームは、実行時に異なるプログラムを表示させることができます。





これは、［コール］処理コマンドの［出力先］特性を使用することで実現できます。

プログラムがサブフォームで実行するように呼び出される場合、プログラムは［出力先］特性で設定された名前で指定されたサブフォーム上で実行されます。別のプログラムを同じサブフォーム上で実行するように呼び出された場合、最初のプログラムが閉じ、呼び出されたプログラムが実行されます。

［出力先］特性は、サブフォームまたはフレームで使用することができます。この特性は、［コントロール名］または［フレーム名］を参照します。

混在処理のタスクを呼び出す場合は、注意が必要です。サブタスクが呼び出される場合は常に、サーバへのアクセスが発生します。この例のように［項目変更］ロジックユニットを使用した場合、呼び出し処理の効率が悪くなる旨の警告が発生します。

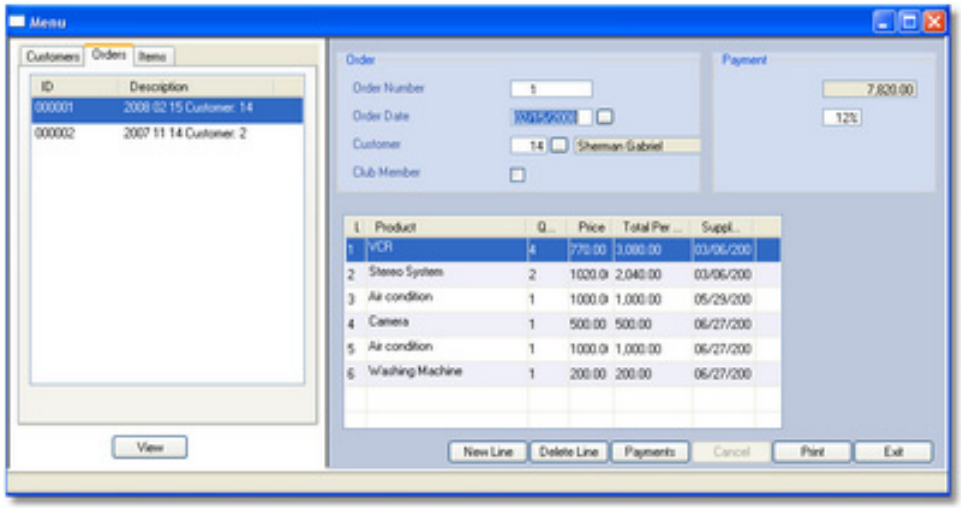
フレームを使用した場合もこの技術を使用するができます。

サブフォームの終了

リッチクライアントでサブフォーム関する違いは、サブフォームが終了すると親タスクも終了するということです。［終了］イベントが発行されたり、ユーザがサブフォーム内で ESC キーを押下すると、親タスクも終了します。これは、フレームの場合も同じです。ユーザが下位タスクで ESC キーを押下してもタスクが終了しないようにするには、ESC キーに対する［イベント］ロジックユニット作成し、［伝播］特性を「No」に設定することで実現できます。

```
6
7 ⊟ E=ｲﾍﾞﾝﾄ  Esc                                    ｽｺｰﾌﾟ S=ｻﾌﾞｳｨﾝ
8                   ESCキーのブロック
```

フレーム

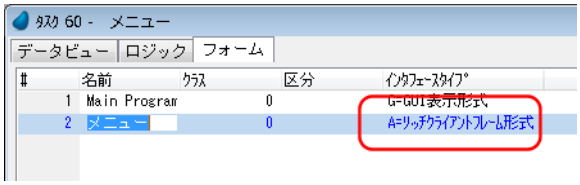


より洗練されたフォームを作成するために、リッチクライアントのフレームセットを使用することができます。これを使用することでタスクを複数のセクションに分けることができます。そして、それぞれが異なるプログラムを表示します。各フレームに表示されるプログラムは、リッチクライアントのプログラムまたはサブタスクでなければなりません。

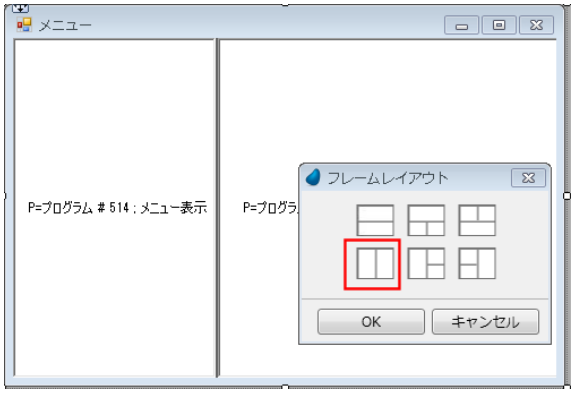
フレームは、サブフォームと同じように動作します。しかし、フレームでは、ユーザが仕切りバーをドラッグすることによって、簡単に各セクションの大きさを変更することができます。

フレームはネストすることができ、さらにサブフォームやタブと併用することで、より複雑なユーザインタフェースを作成することができます。この例では、左側のフレームでユーザーや受注情報、製品情報が表示されるようにようにしています。選択品内容が右側のフレームに表示されます。

フォームタイプ



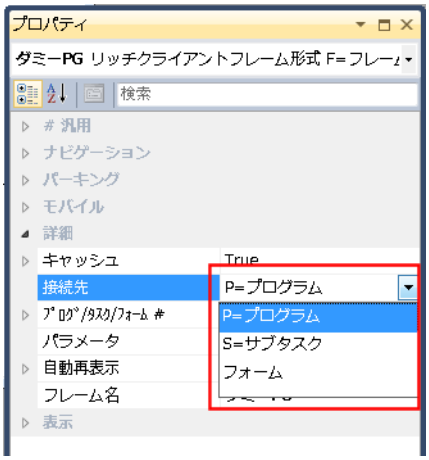
リッチクライアント用のフレームを作成するには、最初にフォームの［インタフェースタイプ］で「リッチクライアントフレーム」を選択する必要があります。



次に、フォームを開くと［フレーム］パレットが表示されます。パレットをクリックすることで使用したいフォーマットを選択することができます。ここでは、2 つの垂直フレームを持つフレームセットを使用します。

これらのフレームは、ネストさせることもできます。このため、実行時には 6 種類以上の組み合わせで表示させることができます。

フレームの接続先

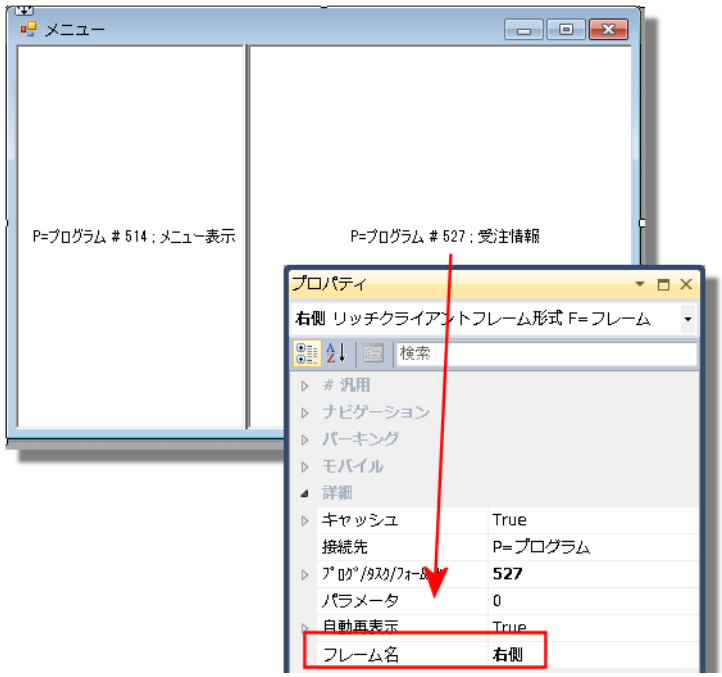


各フレームは、プログラムまたは、サブタスクやフォームに接続することができます。これはサブフォームの場合と同じように動作させることができます。［パラメータ］や［自動再表示］の特性を使用することもできます。

現在のタスクからデータを表示させたい場合、「フォーム」のタイプを使用します。この場合は、サブタスクを呼び出しません。

各フレームで何を表示させるかを選択する必要があります。実行時にフレームに表示させる内容を変更させたい場合、何も表示させないダミーのタスクを作成して、フォームから呼び出すようにしてください。

フレーム名



各フレームは、フレーム名を定義することができます。これは、特定のフレームでプログラムを表示するために使用されます。

この例では、2 つのフレームがあります。左上はメニューリストを表示します。これは、フレーム内に常に表示される唯一のプログラムです。右側には、何も表示されないダミープログラムを呼び出します。実行時には、メニューリストから呼び出されプログラムをもとに、このフレームを表示します。

プログラムの呼び出し

次に、プログラムを呼び出すとき、どのフレームに表示させるかを指定します。ここでは、「ユーザー表示」というプログラムを「右側」という名前のフレームで表示させるようにします。

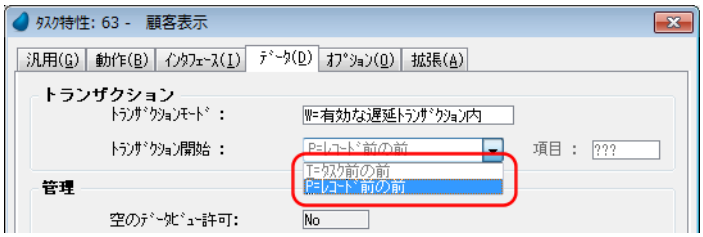
トランザクション

リッチクライアント・プログラムは、遅延トランザクションを使用します。DBMS はサーバ上にあるため、物理トランザクションを使用することができません。

Magic xpa は、遅延トランザクションにデータの変更内容を保存し、トランザクションが終了した時点でコミットします。トランザクションがロールバックされると、保持されたトランザクションは廃棄されます。リッチクライアントのこの動作は、オンラインプログラムの場合と同じです。

しかし、リッチクライアントでトランザクションが処理される方法にはいくつかの違いがあります。このセクションでは、類似点と違いについて説明しています。

トランザクション開始



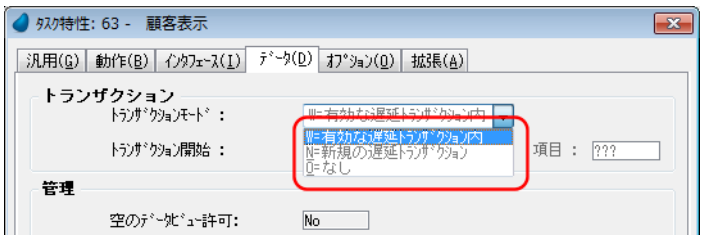
レコードを更新しているか、新規レコードを作成しているか、レコードを削除しているかどうかにかかわらず、リッチクライアントモジュールはデータのどのような修正も処理します。すべてのデータ操作ステートメントは、リッチクライアントモジュールによって維持され、［トランザクション開始］の設定に基づいてサーバーに送られます。

- **レコード前の前** …… 変更内容は、レコードレベルのトランザクション内のレコードを終了した後にコミットされます。
- **タスク前の前** …… 変更内容は、タスクレベルのトランザクション内でタスクを閉じた後にコミットされます。

トランザクションモード

リッチクライアントで使用することができるトランザクションモードは以下の通りです。

- なし
- 新規の遅延トランザクション
- 有効な遅延トランザクション内



なし

［トランザクションモード］が「なし」の場合、トランザクションがオープンされません。

子タスク内のトランザクションをまとめて、ラッピングされたトランザクションがオープンされることを防止するために、データの保存処理が行われないタスク上でのみ使用してください。メインのバックグラウンド画面やバックグラウンドフレームを作成する場合、この設定が必要です。

新規の遅延トランザクション

［トランザクションモード］が「新規の遅延トランザクション」に設定された場合、トランザクションが親タスクで開始されるか否かに関わらず、新しいトランザクションが開始されます。親のトランザクション内で発生した内容に関係なく、子のトランザクションは子タスクレベルでコミットされます。

もし、トランザクションが開始されると：

- 新しい遅延トランザクションが開始されます。
- トランザクションが閉じた後（呼び出し側のトランザクションに関係なく）、このトランザクションの変更内容がコミットされます。
- このトランザクションがまだ開いていても、呼び出し側のトランザクションを閉じることができます。呼び出し側のトランザクションを閉じても、このトランザクションは影響しません。
- 新しいトランザクションは、独自のトランザクションキャッシュを持っています。トランザクションキャッシュは共有できません。そして、このトランザクションで行われる変更内容は、他のトランザクションで参照することはできません。

有効な遅延トランザクション内

［トランザクションモード］が「有効な遅延トランザクション内」の場合、このトランザクションは親のトランザクションの要素となります。

- トランザクションがすでに開始される場合、タスクはそのトランザクションの要素です。
- トランザクションが開始されていない場合、新しい遅延トランザクションが開始されます。

親タスクのトランザクションがロールバックされると、子トランザクションもロールバックされます。

1 つの親タスクが子タスクで複数のトランザクションを開いていることになるため、注意して使用してください。もし、一度にタスクレベルでロールバックする正当な理由がない限り、新規の遅延トランザクションは、より安全です。

オンラインとリッチクライアントでのトランザクションモードの比較

リッチクライアント	オンライン	比較
なし	物理／なし	トランザクションはオープンしません。
新規遅延	遅延	親レベルでは、新規の遅延トランザクションと遅延トランザクションは、同じように動作します。両方とも、新しいトランザクションを開きます。
	ネスト遅延	子タスクレベルでは、オンラインタスクでのネストされた遅延トランザクションは、リッチクライアントでの新規の遅延トランザクションと同じように動作します。
遅延内	遅延内	オンラインとリッチクライアントでは、同じように動作します。現在のタスクのトランザクションは、親のトランザクションの要素になります。
	物理	物理トランザクションモードは、リッチクライアントにはありません。トランザクションが閉じると、遅延トランザクションは物理的なデータベースにコミットされます。

ノート

ローカルデータソースでのトランザクションモードは、下記のように動作が異なります。

ローカルデータソース

ローカルデータソースにおける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上にあります。

トランザクションをオープンしたタスクがローカルデータをコミットしてください。トランザクションが兄弟タスクでオープンされた場合、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

第 5 章 リッチクライアントでの MDI

以下のようにすることで、リッチクライアントアプリケーションで MDI 環境を使用することができます。

1. メインプログラムに、フォームを追加します。
2. ［クラス］を「0」、［インターフェースタイプ］を「リッチクライアント表示形式」に設定します。
3. ［フォーム特性］で、［プルダウンメニュー］特性にリッチクライアント専用のプルダウンメニューを設定します。MDI の背景色や背景イメージ、またはグラデーションスタイルを指定することもできます。
4. ［プロジェクトの実行タイプ］特性を「リッチクライアント」に設定します。
5. プログラムを MDI 内に表示させたい場合は、［ウィンドウタイプ］を「MDI 調整」か「MDI 子ウィンドウ」に設定します。

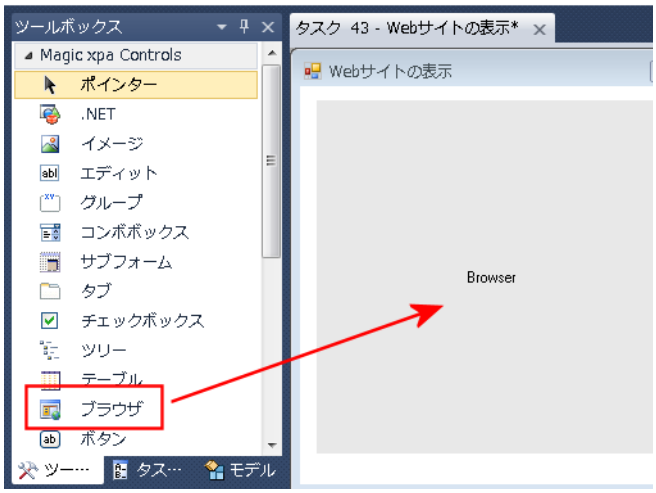
アプリケーションをテストするためにプログラム上で F7 を押下するか、Ctrl+F7 を押下してプロジェクトを実行した場合も、MDI 環境で動作します。

ノート

MDI 環境をメインプログラムに定義すると、リッチクライアント・アプリケーションを公開する際に、開始プログラム名を指定する必要が無くなります。

第 6 章 ブラウザコントロール

他のアプリケーションへのアクセス



リッチクライアント環境で動作する場合、データはサーバにあり、クライアントで表示されます。しかし、異なるアプリケーションを使用しているクライアントにデータを表示する必要があるかもしれません。

たとえば、PDF や Excel のスプレッドシートで帳票を表示させたい場合があるかもしれません。

［ブラウザ］コントロール

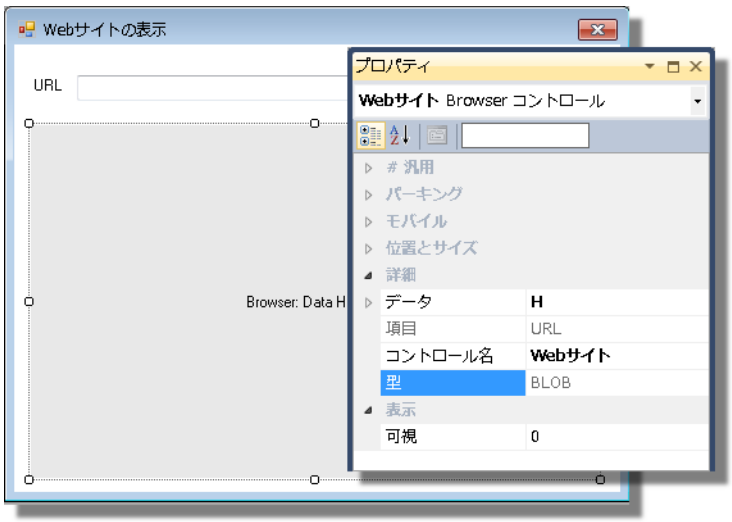
これを処理する 1 つの方法が、［ブラウザ］コントロールの使用です。このコントロールは、Microsoft の Internet Explorer のように動作します。

ヒント

ブラウザ内容がコントロールのサイズより大き過ぎる場合や、スクロールバーを表示させたくない場合、以下のスタイルを使用することができます。

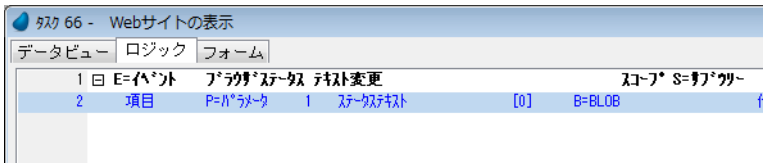
```
body { margin:
0px;
overflow:hidden }
```

ブラウザコントロール特性



[ブラウザ] コントロールには、簡単な特性があります。そのほとんどは、他のコントロールと同じように使用します。[データ] 特性は、Web ブラウザで開く Web サイトやファイルの URL を格納する文字型項目を指定します。

ブラウザコントロールイベント



[ブラウザ] コントロールは、「ブラウザステータス　テキスト変更」という内部イベントを発行させることができます。

[ブラウザ] コントロールによって表示されているアプリケーションのステータスバー内のテキストが変更されると、コントロールはこのイベントを発行します。このイベントは、Magic xpa 内で処理することができます。

このイベントに対するハンドラは 1 つのパラメータ（ステータスバーの現在のテキスト）を受け取ることができます。

ノート

[ブラウザステータス　テキスト変更] イベントによって、リッチクライアントモジュールはブラウザから情報を受け取ることができます。

JavaScript を使用してステータスバーの値を変更することができ、それを行うことで、ブラウザ側で表示するデータをリッチクライアントに返すことでができます。

ブラウザコントロール関数

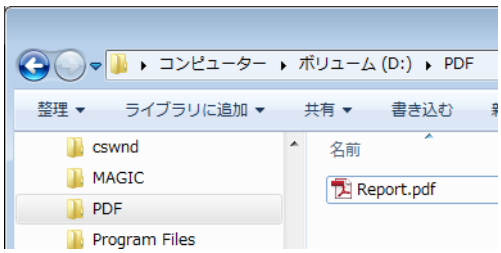
リッチクライアントでは、コントロールに表示させる内容を制御するための関数を使用することができます。

- BrowserSetContent …… コントロールに表示させる内容を設定することができます。この関数は、パラメータで指定された値を使用してコントロールを更新するもので、URL によって参照する内容をもとにコントロールを更新する方法とは異なります。
- BrowserGetContent …… コントロールに表示されている内容を取得します。この関数は、BrowserSetContent 関数とともに使用され、内容を取得し、テキストを処理し、コントロールに戻すことができます。
- BrowserScriptExecute ……［ブラウザ］コントロールに表示されるファイルに定義されている VB Script や JavaScript の関数を実行します。

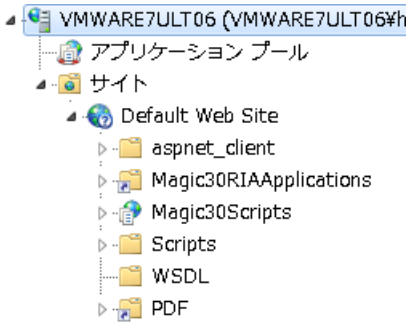
PDF の表示

帳票を作成する場合に、この［ブラウザ］コントロールを使用することができます。エンドユーザは、帳票をクライアント側に表示させたいはずです。しかし、Magic xpa のプリンタの設定は、サーバ側のプリンタにしかアクセスできません。

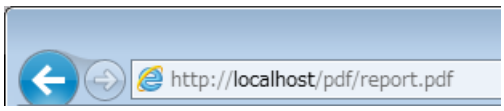
クライアント側で帳票を表示させるためのソシューションは、［ブラウザ］コントロールを使用し、PDF ファイルを表示するためにこれを拡張することです。このためには、以下の処理が必要です。



サーバ上にファイルが作成されるディレクトリを作成します。（例：D:¥Pdf）

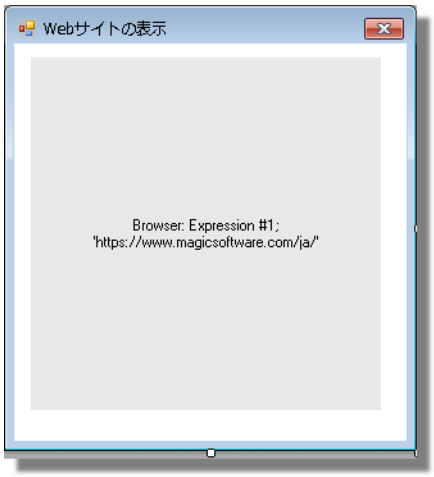


このディレクトリを参照する Web サーバのエイリアス（例：pdf）を作成します。



これで、サーバ上の PDF に Web ブラウザでアクセスすることができます。

帳票作成のバッチタスク内で、PDF を作成するための処理を組み込みます。Adobe Acrobat などのサードパーティ製の PDF 出力ツールを使用して GUI 印刷出力の内容を PDF に変換するようにします。ファイル名は「d:¥Pdf¥report.pdf」と設定します。
PDF 出力ツールは、指定したフォルダ内に自動的に PDF を出力し、出力結果を表示しないように指定できる機能が必要です。ファイル名は、［入出力ファイル］テーブルの［名前］カラムに論理名で指定することで動的に指定できます。



ファイルを表示させるリッチクライアントタスク内で、［ブラウザ］コントロールを定義します。

コントロールには、以下の URL を参照するように指定します。http://localhost/pdf/report.pdf

PDF ファイルの作成タスクと表示タスクを順番に実行します。

ノート
PDF の表示タスク内から作成タスクを実行しても、フォーム内の［ブラウザ］コントロールは再表示されません。ファイルの作成後に表示タスクを起動するようにタスクの構成を工夫してください。

第 7 章 アプリケーションのモニタ

堅牢で複雑なリッチクライアント・プログラムを開発することができます。複雑なロジックや未知の問題を理解する必要がある場合、Magic xpa には、作業を支援するツールが組み込まれています。

デバッガ

Magic xpa には、強力なデバッガとロギングシステムがあります。プログラムが何をしているかを把握することができ、ブレイクポイントを使用して必要に応じてプログラムをで停止させることができるため、リッチクライアントの動作を理解する上で役に立ちます。デバッガについての詳細は、『マスタリング Magic xpa』を参照してください。

リッチクライアントでデバッガーを使用する場合は、［ロギング］設定の［クライアント動作］を「Yes」に設定してください。この設定を行わない場合、サーバの動作のみ参照できます。

アプリケーションが配備された後は、リモート環境でデバッガを実行させることもできます。

サーバ同期

サーバ側のロギングを行うだけの場合は、サーバ側の処理コマンドがそのまま出力されます。しかし、クライアント側の内容をロギングする場合は、アクティビティモニタで参照するために処理内容をサーバに転送する必要があります。

各クライアントのコンテキストは、キャッシュメモリ内の自身の処理内容を記録します。リッチクライアントモジュールは、以下の場合、サーバ側に対しクライアントの処理内容を消去させます。

- サーバに対する次のリッチクライアントモジュールのアクセスが発生した場合
- ロジックユニットが終了した場合。例えば、［レコード前］が終了した場合、リッチクライアントモジュールはサーバ側に対し処理内容を消去させます。

重要

アプリケーションのデバッグを行うとパフォーマンスが低下します。

デバッグ機能を止めたい場合は、［デバッグ］メニューの［デバッグモード］をクリックしてください。

クライアントの異常終了

```
<xml id="rcExecProps">↵
   <properties>↵
      <property key="protocol" val="http"/>↵
      <property key="server" val="VMWare7UIt06"/>↵
      <property key="requester" val="/Magic22Scripts/Mgrqispi.dll"/>↵
      <property key="appname" val="Rich Internet Samples"/>↵
      <property key="prgname" val="Menu"/>↵
      <property key="arguments" val=""/>↵
      <property key="envvars" val=""/>↵
      <property key="UseWindowsXPThemes" val="Y"/>↵
      <property key="HTTPCompressionLevel" val="Normal"/>↵
      <property key="RSACookies" val=""/>↵
      <property key="LogonWindowIconURL" val=""/>↵
      <property key="LogonImageURL" val=""/>↵
      <property key="DisplayStatisticInformation" val="N"/>↵
      <property key="LogClientSequenceForActivityMonitor" val="Y"/>↵
      <property key="InternalLogLevel" val="server"/>↵
      <property key="InternalLogFile" val="InternalClient.log"/>↵
      <property key="InternalLogSync" val="Session">↵
   </properties>↵
</xml>↵
```

クライアントが異常終了する場合が時々あります。このような場合は、最後の処理内容が記録されない可能性があります。障害が発生した直前にどのような処理が実行されたか理解する必要があります。これらの処理は、サーバ上には格納されないため、アクティビティモニターでは参照できません。

クライアント側の異常終了をデバッグするには：

1. HTML ファイルの "LogClientSequenceForActivityMonitor" フラグを「Yes」に設定します。HTML ファイルは、リッチクライアントインタフェースビルダで作成されます。フラグが「Yes」に設定されると、クライアントの動作がクライアント側のローカルログファイルとして書き込まれます。このログファイルは、サーバにまだ送信されなかったすべての情報が含まれています。ログファイルは、各イベントの実行後、またはアプリケーションの終了時に後削除されます。

2. もう一度アプリケーションを起動します。Magic xpa は、既存のログファイルをスキャンします。異常終了時にログファイルが見つかると、これらをサーバに送信します（そして、アクティビティモニターに表示します）。
3. 異常終了が発生すると、他の全ての並行処理も終了します（これらのログファイルは、通常削除されます）。
4. アプリケーションを再開してください。クライアントの最後の処理内容が記録されたログファイルがサーバに送信されます（そして、アクティビティモニターに表示されます）。
5. アクティビティモニターは、クライアントログによって更新されます。これで、問題の発生箇所を確認することができます。

ノート

アクティビティモニターの行は、サーバ処理がピンクで、クライアント処理が白で色分けされます。

ネットワーク情報の表示

［リッチクライアント形式］フォームでは、ステータスバーでネットワーク状態を表示することができます。ネットワーク情報は、ステータスバーの右の表示領域上でマウスポインタをホバーリングさせることで表示されます。プログラムが Magic xpa Enterprise Studio から実行された場合、この機能は常に有効になります。プログラムが実行エンジンで実行する場合は、アプリケーションの起動に使用される HTML ファイルの DisplayStatisticInformation パラメータを以下のように修正することで有効になります。

第 8 章 アプリケーションの実行

リッチクライアントアプリケーションは、リッチクライアント実行モジュールを使用して、.NET フレームワークが利用できる環境で実行できるように配備する必要があります。

リッチクライアントの環境は、インターネット上での使用を前提としているため、使用技術を理解することは役に立ちます。

リッチクライアントアプリケーションの配備に関する情報については、『リッチクライアントアプリケーションの実行』を参照してください。変換する必要があります。

第 9 章 ユーザ認証

ユーザ認証は、オンライン環境と同じように実装されているため、同じように扱うことができます。

アプリケーションを起動するとき、ユーザ認証が必要かどうかは動作環境（MAGIC.INI）で指定します。ログオンダイアログを必要とする場合、実行エンジンは Magic xpa のログオンダイアログを表示します。

アプリケーションの起動時に環境変数を取得する

アプリケーションの起動時に環境変数を取得する場合は、HTML ファイル内の envvars パラメータに環境変数を指定することで可能になります。この設定は、リッチクライアントインタフェースビルダを使用する際に指定することができます。

これらの環境変数はサーバに送られ、リッチクライアントモジュールがクライアント側で読み込まれる前に取得することができます（ClientOSEnvGet 関数とは対照的です）。この方法を使用することで、GetParam 関数で環境変数を取得し、それに応じた初期設定処理を行うことができます。

アプリケーションにパラメータを渡す

起動時の URL を使用してリッチクライアントにパラメータを渡すことができます。このパラメータの値は、GetParam 関数を使用することでアプリケーション側で取得することができます。

例えば、リッチクライアントアプリケーションを以下の URL で起動した場合を想定します。

http://server/Magic3xRIAApplications/projects/project.application?N1=v1&N2=v2

v1 の値は GetParam('N1') で、v2 の値は GetParam('N2') で取得できます。指定できる値は文字データのみで、UTF-8 でエンコードされます。このため日本語を渡す場合は、アプリケーション側で UTF8toAnsi 関数を使用して変換する必要があります。

('x' は、バージョンによって異なります)

第 10 章オフラインアプリケーションの開発

アプリケーションにオフライン機能とローカル・データソースと追加すると、アプリケーションの性能を向上させることができ、ネットワーク上のトラフィックを減らすことができます。

詳細は『オフラインアプリケーションの開発』を参照してください。

第 11 章モバイル用の RIA アプリケーションの開発

モバイル端末からアプリケーションに接続できたら非常に便利になります。モバイル端末上で実行されるアプリケーションを開発するために、RIA テクノロジーを使用することができます。これによりデスクトップ上での RIA アプリケーションと同じルック＆フィールを持つことができます。

詳細は、『モバイル RIA 開発者ガイド』を参照してください。

第 12 章オンラインから RIA への変更

アプリケーションをオンライン用から RIA 用に変更する場合、最初に実行する必要がある手順と、考慮する必要がある動作の違いがあります。

さらに、パフォーマンスの件でも考慮する必要があります。

リッチクライアント準備ユーティリティ

クライアント／サーバアプリケーションを変更する前に、リッチクライアント準備ユーティリティを実行する必要があります([オプション]メニューから「リッチクライアント準備」を選択することで起動されます)。

このユーティリティは、プロジェクトのモデルをスキャンし、既存の GUI 表示モデルに基づいて新たにリッチクライアント表示モデルを作成します。リッチクライアントモデルは、後で GUI 表示フォームをリッチクライアント表示フォームに変換する際に使用されます。

RM コンバータ

注意：RM コンバータは、Magic xpa にはバンドルされていません。Magic Software Japan K.K. にお問い合わせください。

[RM 互換]ロジックユニットを持つタスクは、リッチクライアントタスクに変換することができません。そのようなロジックは、イベントドリブンの動ロジックに変換する必要があります。RM コンバータは、処理されなかったた問題のリストも出力します。

オプティマイザ

注意：オプティマイザは、Magic xpa にはバンドルされていません。Magic Software Japan K.K. にお問い合わせください。

Magic xpa オプティマイザを実行してプロジェクトを読み込んでください。オプティマイザは、手動で変更が必要な場所を示すレポートを出力します。

このツールは、アプリケーションによって使用されているコンポーネントなどのような、すべてのケースに対応しているわけではありません。