eDeveloper™ V10

インタラクティブな
リッチクライアントの開発と実行

MAGIC
www.magicsoftware.com

Enabling Business with Superior Technology

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 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 eDeveloper® と Magic Client® は、は Magic Software Japan K.K. の登録商標です。
Magic Studio、Magic Enterprise Server、および Magic RichClient Server は Magic Software Japan K.K. の商標です。
Pervasive.SQL® は Pervasive Software, Inc. の商標です。
Microsoft® および FrontPage® は、Microsoft Corporation の登録商標です。また、Windows™, WindowsNT™ および ActiveX ™ は Microsoft Corporation の商標です。
Oracle® は Oracle Corporation の登録商標です。
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。

1 インタラクティブなリッチクライアントアプリケーション

なぜリッチクライアントを使用するのでしょうか？ 6
リッチクライアントベースの開発 6
リッチクライアントベースの実行 7

2 サポートするアーキテクチャ

アプリケーションサーバのインフラストラクチャー 8
分散されたモジュール 8
WWW サービス機能 8
インターネットリクエスタ 8
MRB 8
Magic アプリケーションサーバ 8
モジュールの分散 9

3 リッチクライアントタスクのライフサイクル

クライアントタスクの初期設定 10
実行時のリッチクライアントタスク 10
ロジックユニット 11
タスクモード 11
再計算 11
サーバ側とクライアント側のロジック 11
透過的な処理 11
処理コマンド 12
動的なインタフェース定義 12
リンクコマンド 12
コンテキスト管理 12
コンテキスト ID 12
コンテキストの内容 12
コンテキスト非稼働タイムアウト 12
データ操作 13
トランザクション 13

4 リッチクライアントタスクの構成

リッチクライアントの開発概念 14
SDI 14
イベントドリブンエンジン 14
基本的な定義 15
リッチクライアントタスクのタイプ 15
データビュー定義 15
並行性 15

ユーザインタフェース 16
リッチクライアントフォーム 16
リッチクライアントコントロール 16
サブフォーム 16
サブフォームが必要な場合 16
サブフォームコントロール 16
ネストされたサブフォーム 17
サブフォームのライフサイクル 17
サブフォームまたはフレームの内容を置き換える 17
フレームレイアウト 17
色とフォント 18
リッチクライアントタスクの設定 18
チャンクサイズ（式） 18
ビュー事前読込 18
メインフォーム 18
アイコンファイル名 18
トランザクションモード 18
リッチクライアントプログラムの設定 19

5 実行時の動作

コール処理コマンド 20
別のリッチクライアントタスクを呼び出す 20
モーダルウィンドウ 20
タスクの初期設定 21
タスクの終了 21
他のタスクを呼び出す 21
同期呼び出し 21
処理の開始と終了 21
エラー処理コマンド 21
クライアント側のメッセージ 21
表示 21
タスクの初期設定 21
タスクの終了 22
システムイベントハンドラ 22
リッチクライアントの内部処理 22
エラー処理 22
リッチクライアントタスクでの例外 22
コール処理コマンドとエラー処理 22
Rollback 関数 22
クライアントのリソースにアクセスする 22

実行ファイルの起動 22
クライアント側のファイルやフォルダへのアクセス 22
ローカルマシンの環境変数の取得 23

6 実行環境

実行の準備 24
リッチクライアントモジュール 24
リッチクライアントフォルダ 24
Web エイリアス 24
動作環境設定 25
Java Web Start 25
JNLP 25
Magic の JNLP ビルダ 25
Magic の実行モード 26
初期プログラム 26
メニューとツールバー 26
ステータスバー 26
ユーザ認証 26
XML データの暗号化 27
デプロイメントアシスタ 27
実運用環境への移行 27
Magic エンジンが別のサーバ PC 上で実行される場合 28
実行ユーザ数の制限 29
Java 環境の設定 29
プロキシサーバの設定 29

7 MDI 動作環境のシミュレート

MDI 環境とはどのようなものでしょうか？ 30
なぜ、MDI フレームワークを使用するのでしょうか？ 30
MDI フレームワークの作成 30
サブフォームを使用する 31
異なるプログラムを表示する 31
実行時のサブフォーム 31
位置に関する考慮事項 31
終了ボタンをシミュレートする 31
トランザクション 32
メニュー 32

8 ブラウザコントロール

なぜ、他のアプリケーションにアクセスする必要があるのでしょうか？ 34
ブラウザコントロール 34
ブラウザコントロール特性 35
ブラウザコントロールイベント 35
ブラウザコントロール関数 35
帳票の作成と表示 - ブラウザコントロールの実装 36

9 パフォーマンス

なぜ制限があるのでしょうか？ 37
クライアント側とサーバ側 37
クライアント側の特性 37
実装 38
サーバ側の特性 38
実装 38
サーバ側のイベント内でのクライアント側の関数 38
実装 38
項目の代入特性の使用 38
範囲と位置付の式 39
実装 39
混在した処理 39
クライアント側とサーバ側の順番の混在 40
推薦方法 40
ブロック IF 40
推薦方法 40
ブロック Loop 41
コントロールとレコードの処理コマンド 41
項目変更 41
タスク前での未確認イベント 41
起動時間と操作時間 42

10 アプリケーションのモニタ

目の前にある問題 43
ロギング 43
Magic のロギング 43
動作のロギング 43
アクティビティモニタ 43
フィルタ 43
サーバ同期 44
クライアントのクラッシュ 44

11 プログラムの書き直し

サポートしていない機能 46
オンラインからリッチクライアントプログラミングへ 46
プルダウンメニュー 47
終了プログラム 47
他のプログラムを呼び出す 47
帳票出力 47
フォーム特性 47
フォームの分割 47
子ウィンドウ 48
コントロール 48
ステータスバー 48
画像表示 48
ブラウザクライアントからリッチクライアントへのプログラミング 49
プルダウンメニュー 49
認証 49
終了プログラム 49
帳票出力と外部ファイル 49
マージ出力 49
JavaScript 49
フォームとコントロール 49

12 要約

Chapter 1

インタラクティブなリッチクライアントアプリケーション

インタラクティブなリッチクライアントアプリケーションの簡単な開発と実行

Magic はアプリケーションのフロントエンドとして、Java ベースのクライアントを使用した高度なビジネスアプリケーションの開発と実行を容易に行うとができます。Magic はインタラクティブな Java クライアント機能を提供します。これは、Magic のアプリケーションサーバによってサポートされ、全てテーブルをもとに作成されたものです。このドキュメントは、インタラクティブな Web アプリケーションの開発と実行のための新しい技術に関する概要を説明したものです。

なぜリッチクライアントを使用するのでしょうか？

アプリケーションをリッチクライアントとして実行させるべきいくつかの理由があります。

アプリケーションをリッチクライアントにすることで、共通のインターネットネットワークを使用して世界中でアクセスすることができる、Web アプリケーションになります。

リッチクライアントベースの開発

Magic はインターネット対応のインタラクティブなアプリケーションを迅速に開発することができます。テーブルをもとに簡単に開発する機能を使用することで、アプリケーションの実行環境やコンポーネント、およびロジックを簡単に定義することができます。Java や JavaScript などのようなリッチクライアントのプログラミングに関する知識もあまり必要としません。Magic のリッチクライアント開発機能では、以下のことが可能になります。

- **容易なフォーム設計** …… Magic Studio 上でリッチクライアントタスクのフォームを設計することが可能な、統合された［フォーム］エディタを提供しています。外部ツールは必要ありません。
- **クライアント / サーバ間のロジックの切り分けの自動化** …… Magic は定義内容に基づいて、ロジックがサーバまたはクライアントのどちらで実行されるかを自動的に決定します。
- **ローカルなクライアント OS 環境との対話機能** …… リッチクライアントを使用することで、ローカルな環境変数の読み込みやローカルコマンドの実行などローカル PC との対話処理を行うことができます。

リッチクライアントベースの実行

マルチスレッドで動作する Magic のアプリケーションサーバは、大量のトランザクションとリクエストを迅速に処理することができます。アプリケーションサーバは、個々のクライアントの処理を扱い、各ユーザのコンテキストを透過的に管理します。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。
- 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、Java SWT によって作成されており、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。
- コンテキスト管理 …… リッチクライアントタスクのために、アプリケーションはサーバ側でコンテキストを作成します。開始から終了するまで、コンテキストはタスクの状態を記録します。
- 自動化されたデータ管理とクライアント / サーバの同期化 …… サーバは自動的にデータ管理とトランザクションを処理します。
- プラットフォームに依存しない …… クロスプラットフォームで実行させることができます。
- ブラウザを使用しない …… 実行には Web ブラウザを必要としません。

Chapter 2

サポートするアーキテクチャ

エンドユーザはどのようにアプリケーションにアクセスするのでしょうか？そして、アプリケーションサーバはどのようにリクエストを処理するのでしょうか？

Magic のアプリケーションサーバの実行環境は、Magic によって提供される分散アプリケーションアーキテクチャを使用することで、簡単に構築することができます。アプリケーションサーバは、多数のユーザからの同時アクセスを処理するために設計されたコンテキスト管理機能によって自動的に最適化されます。

アプリケーションサーバのインフラストラクチャー

Magic のアプリケーションサーバのエンジンは、リッチクライアントからのリクエストを処理するために開発されています。

分散されたモジュール

Magic のアプリケーションサーバを構成する基本的なモジュールは、以下の通りです。

WWW サービス機能

Web サーバは、リモート上のリッチクライアントからリクエストを受け取るために必要です。Magic のインターネットリクエスタを使用することで、Webサーバはリクエストをアプリケーションサーバに送ることができます。

インターネットリクエスタ

Magic はインターネットリクエスタのモジュールを提供します。これは実行モジュールとして Web サーバと組み合わせて利用されます。クライアントからのリクエストがインターネットリクエスタに送られると、モジュールは添付データを含めたリクエストをアイドル状態のアプリケーションサーバに送ります。Magic のインターネットリクエスタは、MRB（Magic Request Broker）によって管理されているサーバエンジンのリストを使用してアイドル状態のアプリケーションサーバを見つけることができます。

MRB

Magic は MRB と呼ばれるミドルウェアエージェントを提供します。MRB は利用可能なアプリケーションサーバエンジンを処理し、インターネットリクエスタから利用可能なアプリケーションサーバエンジンにリクエストを送信します。MRB には、ロードバランシング機能と障害発生時に対する回復機能があります。

Magic アプリケーションサーバ

Magic のアプリケーションサーバは、インタラクティブなリッチクライアントアプリケーションを実行する上で中心的な役割を果たします。これが実際の実行ユニットです。これは各リクエストを処理し、受信された各タ

イプのリクエストに対するアプリケーションロジック全体を実行します。アプリケーションサーバは、MRB が実行されている場所を指定することで、MRB と接続しサーバエンジンをインターネットリクエスタで利用可能になるようにさせる必要があります。
アプリケーションサーバエンジンは、1 つのエンジンのプロセスを使用して、複数のリクエストを処理するように設計されています。これは、Magic サーバエンジンのマルチスレッド機能を使用して実現しています。

モジュールの分散

上記のモジュールは同じマシン上にインストールしたり、異なる OS を使用した異なるマシン上に分散させることもできます。

Magic のインストール処理とアプリケーションサーバによって、必要なモジュールのインストールと実行のための環境設定は自動的に行われます。Web サーバは、インストールされた Magic 製品と同じマシン上になければなりません。

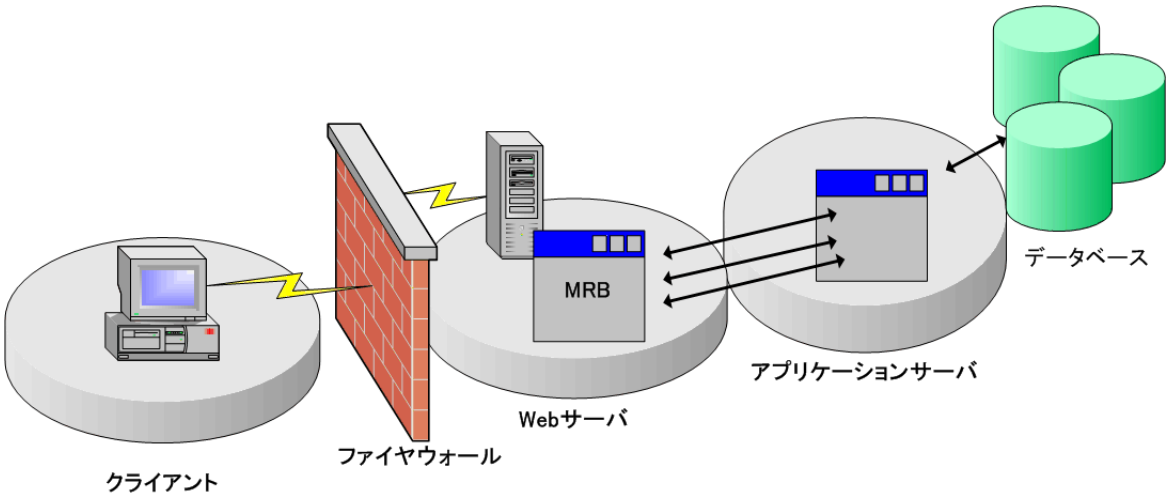

図 2-1 この図は、リッチクライアントがどのようにアプリケーションサーバと通信し、どのように返り、分散されている Magic モジュール間でどのように相互作用しているかを表しています。

Chapter 3

リッチクライアントタスクのライフサイクル

タスクのライフサイクル（リクエストが送られてから、タスク処理が完了するまでにタスク内で行われる処理内容）についてより詳細に知っておいてください。

リッチクライアントタスクには、オンラインタスクのほぼ全ての機能が備わっています。タスクはレコードまたはタスクのレベルでトランザクションをオープンし、どのようなロジックユニットも処理することができます。リッチクライアントタスクは、暗黙のうちにタスクやレコード、コントロールの各ロジックユニットを実行し、基本的なデータの参照や操作（スクロールや修正、削除、作成）を行うことができます。また、ほとんどの Magic の処理コマンドや関数を実行することができます。

クライアントタスクの初期設定

リッチクライアントプログラムを実行させるための最初のリクエストによって、Magic エンジンは要求されたリッチクライアントタスクを開きます。このリクエストのためにコンテキストがオープンされ、ユニークなコンテキスト ID が作成されます。コンテキストが作成されたことを知らせるために、短い応答がすぐにリッチクライアントエンジンに戻ります。

応答を受信すると、2 番目のリクエストが Magic サーバに送られます。これはサーバ上で以下の処理を実行することをリッチクライアントタスクに示すものです。

1. 暗黙の初期設定。これは、以下の処理を実行します。
   - タスクに定義されたデータベーステーブルをオープンします
   - 変数項目を初期設定します
   - 範囲と位置付を行います
   - 最初のデータビューを作成します
2. ［タスク前］ロジックユニットに定義された処理コマンドを実行します。

タスク定義自体は、Magic のエンジンによって定義されていますが、アプリケーションサーバはタスクのインタフェース情報やロジック、データ内容を 1 つの XML データに変換します。XML データはリッチクライアントエンジンに送られ、エンジンはその構造や関数、定義されたタスクロジックに従ってインタフェースを表示します。

XML データの内容は、リッチクライアントエンジンに送られる情報量を減らすために圧縮されます。

これらのサーバ側の処理コマンドに続いて、［タスク前］と［レコード前］の更新処理によって影響するすべての関連データを使用して XML 形式の結果ページが作成されます。このファイルは、リッチクライアントエンジンに送られ、組み立てられて、エンドユーザで参照できる状態になります。

実行時のリッチクライアントタスク

リッチクライアントタスクがクライアント側で構築されると、データおよびタスクとそのデータのために定義されたロジックが実行され、エンドユーザで利用可能になります。Magic の標準的な実行機能に加え、受け取った XML データに基づいてアプリケーションに定義されたタスクロジックが実行されます。

ロジックユニット
リッチクライアントモジュールは、タスクに定義された関連するロジックユニットを自動的に実行します。

タスクレベル
タスクの起動時、［タスク前］で初期設定処理が実行されます。これはサーバ側で実行されるロジックユニットで、クライアント側の処理コマンドはここでは実行されません。タスクの終了時には、［タスク後］が実行されます。ここでは定義されている処理コマンドに依存し、サーバ側で実行されたり、クライアント側で実行されたりします。

レコードレベル
エンドユーザが現在のレコードにカーソルを移すと、そのレコードに対して［レコード前］が実行されます。［レコード後］は、修正されたレコードを抜けた場合や、［タスク特性］の［強制レコード後］特性が「Yes」に設定された場合、または、レコードの削除を行った後に実行されます。

コントロールレベル
コントロールにパークされると、そのコントロールの［コントロール前］が実行されます。カーソルがそのコントロールから抜けた場合、そのコントロールに対する［コントロール後］が実行されます。レコードが修正された状態でそのコントロールを抜けると、［コントロール後］の前にそのコントロールに対する［コントロール検証］が実行されます。

エラーレベル
対応するエラーが発生した場合、エラーに対する［イベント］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、サーバ上で実行されます。エラー処理の詳細について、Magic の『リファレンスヘルプ』内のエラー処理の説明を参照してください。

その他のイベントロジックユニット
タスクがアイドル状態にある場合にイベントが発行されすると、このイベントに対応した、システム、内部、ユーザ、タイマ、および式の各［イベント］ロジックユニットが実行されます。

タスクモード
リッチクライアントタスクは、照会、修正、作成、および削除の各基本的なタスクモードが指定可能で、その実行規則に基づいて実行されます。例えば、照会モードではデータの修正や削除ができず、修正モードでは可能になります。

再計算
リッチクライアントエンジンは、データを修正することで再計算を発生させることができます。修正されたデータに基づいて、項目やリンクレコード、および［可視］特性の値が自動的に再計算されます。

サーバ側とクライアント側のロジック

リッチクライアントタスクのロジックには、Magic で利用可能なほとんどの処理コマンドや関数を利用することができます。しかしその特性上、サーバ上で実行させる必要があるため、クライアント側で実行させることができないものがあります。1 つの例として、［フォーム出力］処理コマンドと DbDel 関数があります。［フォーム出力］処理コマンドは、リッチクライアントタスクで利用できないことに注意してください。

透過的な処理
リッチクライアントエンジンは、処理コマンドと関数をサーバ側で実行させるかクライアント側で実行させるかを自動的に区別しています。XML データが作成される際に、処理コマンドや処理コマンドが使用する関数のタイプに基づいて、サーバ側なのかクライアント側で実行されるかの情報を含めています。

ノート

リッチクライアントは、処理コマンドが定義されるテーブル内で、どちらで実行されるかが定義内容に基づいて表示されます。これは、混合モードとして処理コマンドを定義した場合、リッチクライアントエンジンは、サーバ側とクライアント側の異なる処理モードを切り替えながら実行することを意味しています。この場合、パフォーマンス上の問題が発生します。**Magic** はモードが混在するような設定を行った場合、構文チェックユーティリティによってチェックされます。このような場合は、サーバ側で実行される関数と処理コマンドの使用を最小化するようにしてください。

処理コマンド
［エラー］処理コマンドは、クライアント側でのみ実行可能です。

［リンク］コマンドは、サーバ上でのみ実行されます。

他の処理コマンドは、それらのパラメータと特性に基づいてサーバ側またはクライアント側で実行されます。

動的なインタフェース定義
［コントロール特性］は常にクライアント側で評価されます。その結果、サーバ側の関数を含む式が定義された場合はパフォーマンスが低下します。このため、サーバ側の関数で定義された［コントロール特性］は無視されます。

リンクコマンド
［リンク］コマンドは、手続上の処理コマンドではありませんがサーバ側で実行されます。これは、リンクが再計算されるたびに新しいリンクを実行するために、リッチクライアントがサーバを参照することを意味しています。

コンテキスト管理

並行実行のリッチクライアントタスクを新規に実行した場合、呼び出された初期プログラムはサーバ側でコンテキストを作成します。それが実行する瞬間から終了する瞬間までコンテキストはタスクの状態を記録します。各コンテキストはコンテキスト ID によって識別されます。これは、サーバ上にある同じコンテキストの連続的なリクエストを識別させるために使用されます。

コンテキスト ID
新しいコンテキストが作成された場合、ユニークなコンテキスト ID が新しいコンテキストのために作成され、クライアントに送り返されます。リッチクライアントタスクのライフサイクル中に作成されたすべてのリクエストによって、クライアントはコンテキスト ID をアプリケーションに戻します。このように、アプリケーションサーバは、リクエストがどのコンテキストに属するリッチクライアントなのかを認識することができ、そのタスクのコンテキスト内でこの（リッチ）クライアントを提供し続けることができます。

コンテキストの内容
コンテキストにはタスクの構造に関するすべての情報が含まれています。例えば、全てのタスクやメインのタスクからオープンされるサブタスク、実行ツリー構造内のクライアントの位置などが含まれています。またコンテキストは、データベースカーソルをタスクとそのサブタスクのためにオープンできるようにしておきます。

コンテキストは、サーバに返されるすべてのデータ操作ステートメントを、定義されたトランザクション内に保持しています。

コンテキストは、リッチクライアントタスクの実行情報を保持しています。これは指定されたコンテキストに対してローカルに処理されます。実行情報には、メモリテーブルや常駐テーブル、メインプログラムの項目、SetParam 関数によって設定されたグローバルパラメータ、および環境設定が含まれます。これらの実行情報ユニットのインスタンスは、新しいコンテキスト毎に別々に発生し、修正内容はそのコンテキストでのみ参照できます。

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 6000（10 分）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。

データ操作

リッチクライアントエンジンは、データの修正処理（レコードの修正、登録、削除）を行うことができます。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エンジンによって保持され、以下のインスタンスでサーバに送られます。

1. レコードレベルのトランザクションでレコードを抜けた場合
2. タスクレベルのトランザクションでタスクを終了した場合

トランザクション

Magic エンジンは、リッチクライアントタスクによって生成されたデータ修正によるトランザクションを処理します。リッチクライアントタスクでは、「遅延トランザクション」と呼ばれるトランザクションを使用します。

遅延トランザクション

クライアントから送られたすべてのデータ操作はコンテキストによって保持され、トランザクションが完了するまではデータベースエンジンに送られません。トランザクションがロールバックされた場合、保持されているトランザクション情報は廃棄されます。Magic エンジンがデータベースエンジンの物理トランザクションを遅延させるようなデータ操作方法を、「遅延トランザクション」と呼びます。

トランザクションのこの処理は、すべての特定のデータ操作ステートメントに対して、繰り返される送信処理をデータベースエンジンに保存し、アプリケーションサーバの能力より大きなスケーラビリティを提供します。

遅延トランザクションの詳細は、Magic の『リファレンスヘルプ』の「データ管理」のセクションか『データ管理』のホワイトペーパを参照してください。

Chapter 4

リッチクライアントタスクの構成

Magic には、リッチクライアントタスクのロジックとそのインタフェースを定義するための容易で簡潔なパラダイムがあります。

データビュー定義とリッチクライアントタスクのロジックは、イベントドリブンで SDI 形式のオンラインタスクと同じように構成されています。リッチクライアント タスクは、インタフェース定義がオンラインタスクと異なっています。またタスクの一般的な動作も、リッチクライアントの性質上、異なるものがあります。

リッチクライアントの開発概念

SDI

リッチクライアントの概念は、Magic の SDI プログラムのパラダイムの機能や可能性の一部分として成り立っています。SDI プログラムは、それ自身でメニューやツールバー、およびステータスバーを持ち、他のタスクと並行して実行させることができます。SDI プログラムは、フローティングウィンドウやモーダルタスクのような並行して実行させることのできない別のタスクを呼び出すこともできます。

リッチクライアントの動作が SDI のため、MDI 環境が全くない状態になります。利用可能なメニューは SDI クライアントに表示されます。これを前提としてアプリケーションを設計する必要があります。このドキュメントでは、「MDI ライク」なインタフェースを設計する方法についても説明します。

SDI の環境と特性の詳細については、Magic の『リファレンスヘルプ』を参照してください。

> **ノート**
>
> リッチクライアントの開発と実行は、**SDI** のパラダイムに基づいていますが、**Java** 環境の制限のため、すべての機能が利用できるわけではありません。動作環境による制限については後で説明します。

イベントドリブンエンジン

リッチクライアントアプリケーションを設計するために、Magic のイベントドリブンアーキテクチャを使用すると、以下のような利点があります。

- **アプリケーションのロジックが理解しやすくなります。**
- **アプリケーションがより柔軟になります。**
- **分析が容易になります。**
- **コードが再利用できます。**

イベントドリブン方式を使用することで、エンドユーザの操作とは独立したコードを作成することができ、より

効率的なアプリケーションを作成することが可能になります。アプリケーションのビジネスプロセスに名前を付けたり、イベントと対応するロジックユニットの関係を明確にすることで、アプリケーションがより理解しやすく、より扱いやすいものにすることができます。

すべてのアプリケーション要素を処理するために、それがタスクやレコード、コントロールであったとしても、イベントドリブンアーキテクチャを使用するようにしてください。

再使用可能なグローバルなロジックユニットを定義するために、ハンドラの階層構造を利用することができます。

イベントドリブンアーキテクチャの詳細については、『イベントドリブンアーキテクチャ』のホワイトペーパを参照してください。

基本的な定義

リッチクライアントタスクのタイプ

［タスク特性］内で、［タスクタイプ］特性を「C= リッチクライアント」と定義することができます。タスクをリッチクライアントと指定することで、開発及び実行時にリッチクライアントタスクとしての適切な機能が提供されます。

データビュー定義

リッチクライアントタスクのデータビューは、そのメインソースと任意の数のリンクテーブルを使用してオンラインタスクと同じように定義することができます。

ノート

［リンク］コマンドによる再計算が発生すると、リッチクライアントエンジンは、アプリケーションサーバエンジンに接続し、新しいレコードを取得します。このため、あまりにも多くの［リンク］コマンドを使用すると、サーバ側との処理に負荷がかかる可能性があります。従って、［リンク］コマンドはできるだけ使用しないようにすることを推奨します。例えば、データコントロール（データソースの内容をもとに、選択肢を表示させるコンボボックスのような選択コントロール）を使用することで［リンク］コマンドの代わりになる場合があります。
また、［結合リンク］コマンドを使用する場合もあります。この場合、コマンドはメインソースと一緒にデータベースからフェッチされ、データの **1** つの集まりとしてリッチクライアントエンジンに送られます。

アプリケーションのデータ構造を定義したり、タスクのデータビューを定義する内容に関する詳細は、Magic の『リファレンスヘルプ』を参照してください。

並行性

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。しかし、階層内の親タスクまたは他の全てのタスクを終了した場合、下位の全てのスクは自動的に終了します。

タスクの［並行実行］特性を「True」に設定すると、新しいコンテキストでタスクが起動されます。新規コンテキストで実行されるタスクの優位性は、起動元のタスクが終了しても呼び出されたプログラムが終了しないということです。

ユーザインタフェース

リッチクライアントフォーム

リッチクライアントフォームは、通常のオンラインフォームと同じ方法で定義することができます。リッチクライアントの場合のフォーム定義は、SDI のメカニズムに基づいていますが、いくつか利用できない特性があります。例えば、MDI が存在しないフォームの場合、[開始位置] 特性の「MDI の中央」のオプションが無効になります。このようなオプションは指定できません。

リッチクライアントコントロール

リッチクライアントの［フォーム］エディタでは、通常のオンラインフォームと同じように様々なユーザインタフェースを定義することができます。通常のオンラインタスクのコントロールと同じように、ページの色々なコントロールを定義し、データをそれに割り当て、特性を自由に定義することができます。

インタフェース定義によって、コントロールの表示を様々に変えることができます。各コントロールの［特性］シートには、設定可能な色々な特性が表示されています。リッチクライアントでは、オンラインタスクとまったく同じコントロールをサポートしているわけではありません。サポートされないコントロールは無効表示されます。

サブフォーム

Magic は、1 つの単一のインタフェース内に異なるタスクのデータを簡単に表示させることができます。各タスクは定義されたロジックに従って、指定されたインタフェース部分を処理します。リッチクライアントは、カーソルの位置に基づいて、あるタスクから別のタスクに透過的に制御を切り換えることができます。

サブフォームが必要な場合

リッチクライアントが、メインのビューの上に（データが複数のレコードから構成される）特別なデータを表示するように設計する場合、サブフォームが必要になります。これは通常「1 対多数関係」と呼ばれます。

このような場合、通常は複数のタスクが関連しますが、［サブフォーム］コントロールを使用することで、2 つの個別のタスクを同じインタフェース内に表示させることができます。

[サブフォーム]コントロールの境界内に別のタスクのフォームを表示させることで､1 対 1 の関係でもサブフォームを使用することもできます。

サブフォームコントロール

［サブフォーム］コントロールは、メインのリッチクライアントタスクのコントロールの 1 つとして定義されます。ここでタスクが指定されると、子タスクを呼び出します。［サブフォーム］コントロールは、メインのタスクに別のタスクの表示の一部を処理するという論理的な定義になります。

パラメータ

データや項目、または式を指定することで、呼び出すタスクにパラメータを渡すことができます。渡されたパラメータは、呼び出されたタスクに単にデータを渡すためだけに必要なのではありません。パラメータは、リッチクライアントが子タスクのビューを再表示させるための基準となります。

サブフォームのパラメータとして渡された項目が変更されると、そのサブフォームのビューは自動的に再表示されます。パラメータが式で渡される場合、値が変更されてもサブフォームのビューは再表示されません。パラメータとして何も渡されない場合、サブフォームはメインのタスクレコードをスクロールしても再表示されません。サブフォームの再表示は、［サブフォーム］コントロールの［自動再表示］特性に依存します。この特性の詳細については、Magic の『リファレンスヘルプ』を参照してください。

ノート

下位のタスクがメインのタスクのサブタスクの場合、メインのタスクの［データビュー］エディタに定義された項目は、パラメータとして渡さなくても直接アクセスすることができます。
しかし、子タスクのデータビューに関連する項目は、サブフォームの再表示機能を実行させるためにパラメータとして渡す必要があります。

ネストされたサブフォーム
メインのタスクに複数のサブフォームを定義することができます。各サブフォームには、独自にネストされたサブフォームを定義することもできます。

サブフォームのライフサイクル
Magic のアプリケーションサーバは、サブフォームとして定義されているすべてのタスクを自動的にオープンします。その際、サブフォームタスク（サブフォームに定義されたタスク）に対する［コール］処理コマンドを定義する必要はありません。

> **ノート**
>
> サブフォームタスクは、親タスクに対する子となります。このため、サブタスクの［フォーム特性］の［ウィンドウタイプ］特性が「**C=** 子ウィンドウ」として定義されたように動作します。

リッチクライアントの初期設定
メインのタスクの最初のレコードの［レコード前］を実行した後に、各サブフォームで定義されたタスクがオープンされ、［タスク前］と最初のレコードに対する［レコード前］が実行されます。タスクはコントロールリスト内に定義された［サブフォーム］コントロールの順番に従ってオープンされます。ネストされたサブフォームは、親のサブフォームの最初のレコードの［レコード前］の後にオープンされます。

実行中のリッチクライアントタスク
メインのタスクをスクロールし、パラメータとして渡される項目の内容が更新されると、サブフォームのデータビューは［自動再表示］特性に従って再表示されます。サブフォームタスクが再表示されても、サブフォームタスクの［タスク前］や［レコード前］は実行されません。

メインタスクのコントロールからサブフォームタスクのコントロールにカーソルが移動されると、サブフォームタスクに制御が切り替わり、対応するサブフォームタスクのレコードの［レコード前］が実行されます。

リッチクライアントの終了
サブフォームが定義されたタスクを終了すると、各サブフォームタスク の［タスク後］がその親タスクの［タスク後］の直前に実行されます。

> **ノート**
>
> サブフォームタスクは、単独で終了させることはできません。例えば、［イベント実行］処理コマンドで［終了］の内部イベントを発行させてサブフォームタスクを終了させようとした場合、メインタスクも終了します。

サブフォームまたはフレームの内容を置き換える
［出力先］と呼ばれる新しい特性が、［コール］処理コマンドに追加されました。この特性には、プログラムをどのサブフォーム上で実行させるかを指定します。Magic エンジンは実行しているプログラムを終了させ、新しいプログラムの初期設定を行います。これにより、［サブフォーム］コントロールに表示されるタスクを動的に変更させることが可能になります。

この機能は、［フレーム］コントロールに対しても有効です。

フレームレイアウト

タスクをリッチクライアントタスクとして定義した場合、フォームの（［クラス］カラムが「0」の場合の）インタフェースタイプは、自動的に「C= リッチクライアント表示形式」が設定されます。ただし、「A= リッチクライアントフレーム形式」に変更することもできます。

フレームを使用することで、1 つのウィンドウ内に複数のリッチクライアント画面を表示させることができます。各画面では、個別にプログラムを実行させることができます。

色とフォント

オンラインタスクと同じ方法でコントロールの色とフォントを定義することができます。

リッチクライアントモジュールの初期設定の段階で、[アプリケーション特性] に定義されたアプリケーションの基本色とフォントが表示に使用されるリッチクライアントモジュールに渡されます。

> **ノート**
>
> 指定されたフォントがクライアント側のマシンで利用できない場合、**OS** が代わりにどのデフォルトフォントを表示するかを決定します。

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。

チャンクサイズ（式）

タスク \ 特性 \ 高度な設定

[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。

ビュー事前読込

タスク \ 特性 \ データ

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。

メインフォーム

タスク \ 特性 \ インタフェース

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

タスク \ 特性 \ インタフェース

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。タスクにアイコンが定義されていない場合、[アプリケーション特性] で定義されたアイコンが表示されます。

トランザクションモード

タスク \ 特性 \ データ

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。

リッチクライアントプログラムの設定

リッチクライアントプログラムでは、公開名を設定する必要があります。公開名を指定しない状態で、リッチクライアントプログラムを実行させることはできません。

さらに、外部からリッチクライアントプログラムを呼び出す場合は（第 6 章「実行環境」(24 ページ）で詳細を説明します)、プログラムの［外部］カラムをチェック状態に設定する必要があります。

Chapter 5

実行時の動作

実行環境でのリッチクライアントタスクは、特殊な動作を行います。

Java 環境での制約条件のため、リッチクライアントタスクのロジックを作成する前に、実行時の動作に関していくつか考慮する必要があります。

コール処理コマンド

別のリッチクライアントタスクを呼び出す

［コール］処理コマンドを作成し、別のリッチクライアントタスクを指定することで、リッチクライアントタスクから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。また、呼び出すリッチクライアントタスクにパラメータを渡すこともできます。

サブフォームを使用してタスクの内容を表示させたい場合は、［出力先］特性に［サブフォーム］コントロールの名前を指定する必要があります。この内容についての詳細は、第 7 章「異なるプログラムを表示する」(31 ページ) 後で説明します。

フレームセット内にタスクの画面を表示させたい場合は、［出力先］特性に［フレーム］コントロールの［フレーム名］特性に設定された名前を指定する必要があります。

モーダルウィンドウ

［フォーム特性］で、［ウィンドウタイプ］特性を「M= モーダル」に設定することができます。リッチクライアントタスクが、「モーダル」と設定された別のタスクを呼び出した場合、呼び出されたタスクが終了するまで、起動元のタスクの処理は停止されます。呼び出されたタスクの実行中は、起動元のタスクにフォーカスを移動させることができません。これは通常のオンラインプログラム間での動作に似ています。

リッチクライアントフォームが「M= モーダル」と指定されていない場合は、呼び出されたタスクのウィンドウがオープンされた状態でも、起動元のタスクの動作は中断されず、呼び出されたタスクから起動元のタスクにフォーカスを自由に移動させることができます。

ヒント

メインフォームの［ウィンドウタイプ］特性は、タスクウィンドウの様式を定義します。［コール］処理コマンドで呼び出されたタスクの様式を指定したい場合は、様式を設定するパラメータを定義し、［コールプログラム］処理コマンドで設定したい値をパラメータで渡すことにより実現できます。この時、このパラメータの値を使用した式に基づいて呼び出されたプログラムのウィンドウタイプを指定することもできます。
この方法によって、起動元のプログラムは、呼び出されたプログラムの動作を制御することができます。

タスクの初期設定
［タスク前］（タスクの初期設定フェーズ）は、すべてサーバ側で実行されることを知っておいてください。これは、初期設定フェーズでリッチクライアントタスクを呼び出す［コール］処理コマンドを定義した場合、以下の規則に基づいて実行されます。

- すべての［コール］処理コマンドは指定された時間で実行されますが、呼び出されたタスクのウィンドウは［タスク前］の終了後にクライアント側でオープンされます。これは、他の処理コマンドが実行された後に、すべての［コール］処理コマンドが実行されるということを意味しています。

タスクの終了
［タスク後］から呼び出されたリッチクライアントタスクは、呼び出されたタスクがすでに終了しているため表示されません。呼び出されたリッチクライアントタスクは実行されますが、サーバ側で直ちに終了します。この場合、［タスク前］と［タスク後］がサーバ側で自動的に実行されます。

しかし、呼び出されたタスクが並行実行するように設定されている場合、このような制限はありません。

他のタスクを呼び出す

リッチクライアントタスクからは、どのようなバッチタスクでも呼び出すことができます。

バッチタスクはサーバ側で実行されるタスクで、制限のない通常のバッチタスクとしてすべての機能を持っています。

リッチクライアントタスクは、他のリッチクライアントタスクとバッチタスクを呼び出すことができます。

同期呼び出し
バッチタスクを呼び出す［コール］処理コマンドは常に同期モードで実行されます。これは、バッチタスクが終了するまで、クライアントの処理が待たされることを意味しています。

処理の開始と終了
バッチタスクの開始と終了を指定するオプションメニューは、リッチクライアントタスクから起動された場合はサポートされません。

エラー処理コマンド

クライアント側のメッセージ

アプリケーションの実行中にエンドユーザに対しメッセージを表示させるため、［エラー］処理コマンドを使用することができます。

表示
確認メッセージは、以下の 2 つの方法で表示させることができます。

- B= ボックス …… ダイアログボックスがクライアントに表示されます。このボックスを閉じるまで、タスクの処理は中断されます。
- S= ライン …… クライアントのステータスバーにメッセージが表示されます。メッセージが表示される間もタスクの処理は継続され、ユーザ側の操作は必要ありません。フォームにステータスバーが定義されていない場合、メッセージは表示されません。しかし、リッチクライアントは SDI メカニズムに似ているため、クライアントのステータスバーは実質的には SDI プログラムのステータスバーになります。従って、確認メッセージをフローティングウィンドウのステータスバーに表示しようとした場合、対応する SDI フレームのステータスバーに表示されます。

タスクの初期設定
［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。［タスク前］はサーバ側で実行されるロジックユニットになるため、［エラー］処理コマンドは［タスク前］では定義できません。

タスクの終了
［エラー］処理コマンドが［タスク後］に定義された場合、現在のタスクが終了されるため、メッセージが表示されなくなる可能性があります。しかし、［表示タイプ］が「S= ライン」に設定されている場合、メッセージは実行タスクツリー内の SDI プログラムのステータスバーに表示されます。

システムイベントハンドラ

リッチクライアントの内部処理

実行中のリッチクライアントのウィンドウは、オンラインタスクと同じようにキーボードコマンドなどのシステムイベントに対する処理機能を持っています。

エラー処理

リッチクライアントタスクでの例外

実行中のリッチクライアントタスクで発生したエラーは、Magic の他のタスクと同じような方法で処理することができます。

基本的なエラー処理の詳細については、Magic の『リファレンスヘルプ』の「エラー処理」のトピックを参照してください。

コール処理コマンドとエラー処理
エラーによる［イベント］ロジックユニットは、サーバ側で実行されます。これは、別のリッチクライアントタスクを呼び出す［コール］処理コマンドなどの処理が、エラーによる［イベント］ロジックユニットの終了後に実行されることを意味しています。

Rollback 関数
［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上では実行されません。このため、Rollback 関数の最初のパラメータは無視されます。

クライアントのリソースにアクセスする

アプリケーションを実行する場合、アプリケーション内のプログラムが、クライアント側に存在する特定のファイルにアクセスしたり、実行形式のファイルを起動するなどの処理が必要になることがあります。

ここでは、リッチクライアントプログラムがローカル側のリソースにアクセスする方法について説明します。各方法の詳細については、Magic の『リファレンスヘルプ』を参照してください。

実行ファイルの起動
ローカル側にある実行形式ファイルを起動させることができます。

［コール OS コマンド］処理コマンドに、［実行］特性が追加されています。この特性は、OS コマンドをサーバ側で実行させるのか、クライアント側で実行させるのかを指定することができます。ここでは、クライアント側の実行形式ファイルを起動させるため、「C= クライアント」を選択してください。

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
- ClientFile2Blb

- ClientFileCopy
- ClientFileDelete
- ClientFileExist
- ClientFileListGet
- ClientFileRename
- ClientFileSize

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。

- ClientOSEnvGet
- ClientOSEnvSet

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、JNLP 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。

Chapter 6

実行環境

リッチクライアントアプリケーションは、どのように起動され、実行時にどのように表示されるのでしょうか？

前の章では、開発上の概念と動作環境について説明しました。そして、実行時のルック＆フィールに関する一般的な項目、特に、実行環境を設定するために必要な様々な特性についていくつか説明しました。

この章では、実行環境とリッチクライアントの環境をどのように設定するかについて説明します。

実行の準備

リッチクライアントの実行モジュールが Java の実行環境で実行できるように、リッチクライアントアプリケーションをデプロイする必要があります。

リッチクライアントの実行環境は、Web 上で実行されることを意図しているため、使用されているテクノロジを理解しておくことを推奨します。

リッチクライアントモジュール

XML フォーマットで提供されるすべてのタスク情報とデータは、Java 環境で処理され実行されます。このモジュールは、実際のクライアントエンジンとして実行されます。ウィンドウ表示は、Java の SWT テクノロジで処理されます。

リッチクライアントフォルダ

Magicのインストール処理によって､Magicのインストールフォルダ内に以下の4つのサブフォルダを作成します。

- Scripts …… インターネットリクエスタファイルが格納されます。
- RichClientCache …… タスクやイメージ、メニューのためにキャッシュデータを格納します。
- RichClientModules …… リッチクライアントタスクを実行するために必要な JAR（Java Archive）ファイルやメッセージ表示用データを格納しています。
- Projects …… 開発したプロジェクトのデフォルトの格納先

これらのフォルダは、インストールディレクトリ内に作成されます。これらは、次のセクションで説明する Web エイリアスで管理されます。

Web エイリアス

Web エイリアスは、リッチクライアントアプリケーションを実行するために必要です。

- Magic101Scripts …… Scripts ディレクトリを参照しています。
- Magic101RCCache …… RichClientCache ディレクトリを参照しています。

- Magic101RCModules …… RichClientModules ディレクトリ を参照しています。
- Magic101RCProjects …… Projects ディレクトリ を参照しています。

これらの設定内容は、アプリケーションの内容に応じて変更することもできます。

動作環境設定

リッチクライアントキャッシュの位置は、必要に応じて変更することができます。この設定は、Magic の［動作環境］ダイアログの［アプリケーションサーバ］タブで行うことができます。

- リッチクライアントキャッシュパス …… アプリケーションサーバがリッチクライアントのキャッシュファイルを書き込む物理ディレクトリを指定します。
- リッチクライアントキャッシュエイリアス …… クライアントマシンがキャッシュパスにアクセスできるように、上記のディレクトリの Web 上のエイリアス名を指定します。

Java Web Start

Java Web Start（JWS）は、Web から直接 Java アプリケーションを実行させることを可能にする Sun Microsystems によって開発されたフレームワークです。これは、インターネットブラウザの制約条件に縛られることなく動作します。このテクノロジによって、クライアントアプリケーションをインターネットからだけでなくデスクトップ環境からも実行させることができるようになります。

JNLP

アプリケーションを実行するためのフレームワークを取得したら、次にアプリケーションをどのようにデプロイするかを定義する必要があります。すなわち、Magic アプリケーションの取得方法やデプロイの方法を Java Web Start に知らせる必要があります。JNLP（Java Network Launching Protocol）は、アプリケーションをどのように実行するかを Web Start に通知するためのプロトコルです。JNLP は、Web Start の起動メカニズムをどのように実装すべきかを定義する規則のセットから構成される XML ファイルです。このファイルには、一般に JAR（Java アーカイブ）ファイルの位置とアプリケーションのメインクラス（プログラムの公開名）の名前などの情報が含まれています。JNLP ファイルは、リッチクライアントタスクの実行で使用される JRE（Java Runtime Environment）に渡されます。

Magic の JNLP ビルダ

Magic では、「リッチクライアントインタフェースビルダ」と呼ばれる、簡単に JNLP を定義することができるユーティリティを提供しています。これは、プロジェクトを開いている状態でプルダウンメニューから［オプション / インタフェース ビルダー / リッチクライアント］を選択することで起動することができます。

このユーティリティは、必要な手順をウィザード形式で示しながら JNLP ファイルを作成していくものです。このウィザードでは、Java Web Start で起動されるリッチクライアントモジュールの名前を指定します。1 つの Magic アプリケーションに対して複数のリッチクライアントモジュールと JNLP ファイルを定義することもできます。

ユーティリティ内で、リッチクライアントモジュールが読み込まれた場合、最初に起動するプログラムを指定します。起動されるプログラムは、［公開名］カラムと［外部］カラムが設定されていなければなりません。

リッチクライアントインタフェースビルダは、２つの外部ファイルを作成します。

- JNLP ファイル …… Java Web Start によって呼び出されるファイルです。
- HTML ファイル …… Java Web Start を含む JRE がクライアントにインストールされているかどうかを確認するために、この HTML ページが最初にチェックし、それから、JNLP ファイルで定義されたアプリケーションを起動します。この HTML ページは、必要に応じて修正することもできます。

> **ノート**
>
> クライアントマシンに **JRE** がインストールされていることを既に知っている場合は、**JNLP** ファイルから直接実行することもできます。知らない場合は、**HTML** ファイルを開いてチェックするようにしてください。

Magic の実行モード

Magic のリッチクライアントアプリケーションを正しく実行させるには、[実行モード]（動作環境 \ システムタブ）を「バックグラウンド」に設定する必要があります。Magic Studio からアプリケーションをデバッグする場合は、オンラインモードで実行させます。

ノート

オンラインモードで起動した場合、リッチクライアントアプリケーションは **1** ユーザでしか実行させることができません。

初期プログラム

外部からプログラムを呼び出す場合は、このプログラムの［外部］カラムをチェックさせる必要があります。

このプログラムのフォームは SDI として定義することを推奨します。

このプログラムが起動されると、新しいコンテキストが自動的にオープンされ、その後のすべてのプログラムはそのタスクツリー内で実行されます。

メニューとツールバー

MDI 環境がない場合、唯一のメニューは SDI フレームに表示されるものになります。どのメニューを表示するか、またはツールバーを表示するかどうかなどは、リッチクライアントフォームの［フォーム特性］の［SDI］セクションで定義することができます。

［フォームタイプ］が「SDI」の場合のみ、これらの特性が有効になります。

リッチクライアント環境において、最初に起動されたタスクは定義されたメニューを表示します。従って、SDI プログラムではプルダウンメニューを定義し、それをプログラムに割り当てる必要があります。

ステータスバー

MDI 環境がない場合、基本的にはステータスバーは表示されません。これは、ステータスバーに送られたメッセージが表示されないことを意味しています。これはアプリケーションの要求仕様をもとに必要に応じて修正してください。ステータスバーを表示するかしないかは、リッチクライアントフォームの［フォーム特性］の［SDI］セクションにある［ステータスバー表示］特性で定義することができます。

注意

SDI ウィンドウのステータスバーは、**MDI** フレームのステータスバーと同じ機能を持っていません。**SDI** ウィンドウ内で自動ヘルプやエラーメッセージを表示する場合だけに使用されます。

ユーザ認証

ユーザ認証は、オンライン環境での認証方法と同じように行うことができます。

JNLP を起動する際に、（バックグラウンドモードで起動された）Magic 実行エンジンは、Magic.ini の設定をもとにユーザ認証する必要があるかどうかを確認します。［ログオン］ダイアログが必要であれば（［MAGIC_ENV］セクションで「InputPassword=Y」が設定された場合）、実行エンジンは Magic の［ログオン］ダイアログを表示します。ここで入力されたユーザ ID やパスワードは、Magic のセキュリティファイルをもとにチェックされます。

［ログオン］ダイアログで［キャンセル］ボタンをクリックしてもアプリケーションは実行されます。［ログオン］ダイアログから入力されたユーザ ID は User() 関数で取得できるため、［キャンセル］ボタンをクリックされた場合は、User 関数を使用して（空白が返ります）チェックすることができます。

例えば、[キャンセル] ボタンをクリックされた場合は、Guest ユーザとして扱ったり、アプリケーションを即終了させたりするようにアプリケーション側でチェック処理を組み込むことで柔軟に対応できます。

また、あらかじめプログラムに実行権を設定しておくことで、実行させるプログラムを制限することができます。

LDAP を使用した認証方式を利用することもできます。LDAP 認証を使用するための設定方法は、『リファレンスヘルプ』を参照してください。

> **注意**
>
> **Active Directry** 認証は使用できません。

XML データの暗号化

リッチクライアントアプリケーションでは、クライアント PC とサーバ間は XML データでやり取りします。このデータはデフォルトでは平文で処理されるため、機密上の問題が発生する場合があります。XML データを暗号化させるには、MAGIC.INI の［MAGIC_SPECIALS］セクションに以下のパラメータを設定してください。

```
[MAGIC_SPECIALS]
SecureBrowserClient=Y
```

デプロイメントアシスタ

基本的には、JRE がインストールされているクライアント側で実行環境に関する問題が発生することはありません。しかし、クライアントマシンで Java アプリケーションが起動できるように正しくセットアップされていない場合もあります。

リッチクライアントアプリケーションのデプロイをより簡単にするために、Magic では、クライアントマシン上の Java の実行環境のインストール状況を分析する ActiveX モジュールを提供しています。何らかの問題がある場合、このモジュールは、問題に対応するために必要な処理を提示したり対応処理を実行したりします。

また、どのようなクライアントからでも実行できるように、このオブジェクトが定義された HTML ファイルも提供しています。HTML ファイルは、以下の URL でアクセスすることができます。

```
http://hostserver/Magic101RCModules/RCDeploymentAssistor.htm
```

実運用環境への移行

リッチクライアントビルダで JNLP ファイルを作成後、（エンドユーザなどの）運用環境にシステム関連のファイルを配置することになります。その際、開発環境とはシステム環境が異なる場合があります。ここでは、これらについての配置方法について説明します。

運用環境の構築は以下の手順で行います。

1. Magic RichClient Server V10 をインストールします。その際、リッチクライアント実行モジュールもインストールします。

> **注意**
>
> **Magic Enterprise Server** では、リッチクライアント実行モジュールをインストールすることはできません。またこのライセンス（**MGENTX1**）では、リッチクライアントは動作しません。専用のライセンスを購入する必要があります。

2. アプリケーションファイルをコピーします。（コピー先が Magic のインストール先の Projects フォルダでな

い場合）アプリケーションのコピー先のフォルダに対するエイリアスを IIS に作成します。

3. 必要に応じてアプリケーションが正しく動作するための環境設定を行います。
4. リッチクライアントビルダで作成された JNLP ファイルや HTML ファイルをアプリケーションファイルのコピー先にコピーします。
5. JNLP ファイルや HTML ファイルを運用環境に合わせてサーバ名やエイリアス名を修正します。

- **JNLP ファイルの修正箇所**

```
<?xml version="1.0" encoding="utf-8" ?>
<jnlp spec="1.5+" codebase="http://ServerName/Magic101RCModules/"
href="http://ServerName/Magic101RCProjects/TravelAgency/TravelAgency.jnlp">
<information>
<title>TravelAgency</title>
<icon href="http://ServerName/Magic101RCProjects/TravelAgency/icon.bmp" width="64" height="64" />
<description></description>

.......

<application-desc main-class="com.magicsoftware.richclient.ClientManager">
<argument>protocol=http</argument>
<argument>server=ServerName</argument>
<argument>requester=/Magic101Scripts/mgrqispi101.dll</argument>
<argument>appname=TravelAgency</argument>
<argument>prgname=ProgramName</argument>
```

- **HTML ファイルの修正箇所**

```
function launchApp() {
 if (windowsIE) {
 document.write("<OBJECT CODEBASE=http://java.sun.com/update/1.5.0/jinstall-1_5_0_02-won.....
 document.write("<PARAM NAME=app VALUE=http://ServerName/Magic101RCProjects/TravelAgency/
 TravelAgency.jnlp>");
 document.write("<PARAM NAME=back VALUE=false>");
 document.write("</OBJECT>");
  } else {
    if (navigator.mimeTypes && navigator.mimeTypes.length) {
      if (!webstartVersionCheck("1.5")) {
        // Java Web Start not installed; open browser window to install site
         window.open("http://jdl.sun.com/webapps/getjava/BrowserRedirect?locale=en&host=ja.....
      }
      // Spin quietly, waiting to launch the app from the original window
 launchTID = setInterval('launchJNLP("http://ServerName/Magic101RCProjects/TravelAgency/
 TravelAgency.jnlp")',
    }
 }
```

6. MRB の起動時にアプリケーションが Magic エンジンが自動的に起動され、アプリケーションもロードされるように Mgrb.ini を修正します。

これで、クライアントから JNLP ファイルにアクセスすることでアプリケーションが実行されます。

Magic エンジンが別のサーバ PC 上で実行される場合

デフォルトのインストール環境は、Magic エンジンと MRB、インターネットリクエスタ、リッチクライアントモジュール、IIS が同じ PC 上で稼働していることを想定したものです。しかし、Magic エンジンだけを別の PC 上で実行させて負荷を分散させたい場合もあります。このような場合は、以下のように設定します。

PC1（IIS がインストールされているものとします）

1. 追加コンポーネントのインストールプログラムを実行します。インストールするコンポーネントは以下の通りです。
   - **MRB**
   - **インターネットリクエスタ（リッチクライアントモジュールもインストールされます）**
2. リッチクライアント用モジュールフォルダ（Magic101RCModules エイリアスが参照する物理フォルダ）と、キャッシュフォルダ（Magic101RCCache エイリアスが参照する物理フォルダ）を共有フォルダとして

設定します。キャッシュフォルダは、書き込みができるようにします。

3. プロジェクトフォルダ（Magic101RCProjects エイリアスが参照する物理フォルダ）に JNLP ファイルをコピーします。

PC2

1. Magic RichClient Server のインストールプログラムを実行します。その際、インターネットリクエスタやリッチクライアント実行モジュールはインストールオプションから外します。
2. アプリケーションファイルを任意のフォルダにコピーし、アプリケーションを実行させるための環境設定を行います。
3. Magic.ini ファイルを以下のように修正します。

```
[MAGIC_ENV]
MessagingServer = PC1_Server
RequesterTimeout = 0
WebDocumentPath = ¥¥ServerName¥（リッチクライアント用モジュールフォルダの共有フォルダ名）
InternetDispatcherPath = /Magic101Scripts/mgrqispi101.dll
WebDocumentAlias = /Magic101RCModules
RCCacheFilesPath = ¥¥ServerName¥（リッチクライアント用キャッシュフォルダの共有フォルダ名）
RCCacheFilesAlias = /Magic101RCCache


[MAGIC_SERVERS]
PC1_Server = 0,ServerName/4000,,password,0,,1
```

4. Magic エンジンを起動して、アプリケーションをロードさせます。

これで、PC1 の JNLP ファイルをアクセスすることで PC2 のアプリケーションが実行されます。

実行ユーザ数の制限

リッチクライアントアプリケーションが実行できるユーザ数は、提供されるユーザ数によって決まります。また、このユーザ数を上限として実際に実行できるユーザ数を制限することもできます。

[最大並行ユーザ数]（オプション / 設定 / 動作環境 / アプリケーションサーバ）にユーザ数を設定した場合、Magic エンジンがバックグラウンドで実行している状態では、指定されたユーザ（クライアント）数まで同時にアプリケーションを実行させることができます。これによって、アプリケーション毎に実行できるユーザ数を制限させることができます。

この設定が「0」の場合や、ライセンス数を超えた値を設定した場合は、ライセンス数まで起動できます。また、Magic エンジンがオンラインで起動している場合は、1 ユーザでしか実行させることはできません。

Java 環境の設定

クライアント側の実行環境によっては、設定の変更が必要になる場合があります。

プロキシサーバの設定

クライアント側でインターネットにアクセスする時にプロキシサーバを経由する必要がある場合、JRE の環境設定を以下のように変更します。

[Java コントロールパネル]（Windows のコントロールパネル /Java）を開き、[基本] タブの [ネットワーク設定] ボタンをクリックします。[ネットワーク設定] ダイアログが表示されます。ここで、[プロキシサーバを使用] を有効にし、プロキシサーバのアドレスとポート番号を設定します。

Chapter 7

MDI 動作環境のシミュレート

リッチクライアントはSDI ですが、どのようにしたらMDI として動作しますか？

前の章では、SDI のパラダイムに基づくリッチクライアントの概念について説明しました。このパラダイムでは、プログラムは 2 番目のプログラムを呼び出し、この新しいプログラムはデスクトップ上のフレームワーク内であればどこでも起動させることができます。基本的には、これらの 2 つのプログラムの間は接続されていないように見えますが、プログラムは他の実行ツリーの一部として実行されます。

MDI 環境とはどのようなものでしょうか？

オンラインアプリケーションを開発する場合、プルダウンメニューやステータスバー、背景などを定義します。プログラムを呼び出すと、作成されたプログラムは一定のフレーム内で動作します。プログラムがメニューから起動されると、今まで実行していたプログラムが終了し、起動されたプログラムが表示されます。

リッチクライアントのテクノロジを使用して、オンラインアプリケーションと同じようなルック＆フィールを持つ「MDI ライクな」環境を作ることができます。

なぜ、MDI フレームワークを使用するのでしょうか？

MDI のルック＆フィールを実現する理由として以下のようなことが考えられます。

- ウィンドウ表示数を削減することができます。実行するタスクの表示を 1 つのウィンドウに納めることで、デスクトップ上の何処で実行されるかの制御を行う必要がないため、扱いが簡単になります。
- SDI の場合、1 つのプログラムにフォーカスがある状態でも、実行中のすべてのタスクが同時に表示されます。このため、例えば、リッチクライアントプログラムの実行中に電子メールを受信した場合、受信データを表示させるために各タスクの制御をフォーカスのあるタスクに戻す必要性が発生します。
- メニューやツールバー、およびステータスバーが全て表示され、アクセス可能になります。
- ユーザが既に MDI 環境を使用している場合、新しい（SDI）環境についての説明を行う必要がなくなります。

MDI フレームワークの作成

動作環境を作成する場合、メニューやツールバー、ステータスバー、および壁紙などの通常のルック＆フィールを設定する必要があります。これらのオプションについては、このドキュメントの各所で説明しています。

これらはすべて、リッチクライアントフォームの［フォーム特性］の［SDI］セクション内にある特性で定義します。

フレームが定義されているのであれば、プログラムのロジックを対応させる必要があります。プログラムで「MDI ライクな」ロジックを定義します。

MDI のロジックは、第 4 章「サブフォーム」（16 ページ）で説明したサブフォームを使用することで実装することができます。

サブフォームを使用する

サブフォームを使用することで、別のプログラムを表示するためのフレームを作成することができます。これについては、第 4 章「サブフォーム」（16 ページ）で説明しています。

［サブフォーム］コントロールで表示されるプログラムを変更するだけで MDI 用として利用することができます。

異なるプログラムを表示する

これを実施するための最初のステップは、［サブフォーム］コントロールに［コントロール名］を設定することです。他のすべてのコントロールと同じように、サブフォームにも名前を設定することが可能で、今回の場合には必須事項となります。1 つのフォーム上に複数のサブフォームを定義することが可能なため、コントロール一覧から［サブフォーム］コントロールを参照する場合、選択しやすいように各サブフォームにわかりやすいコントロール名を定義することを推奨します。

次のステップは、［コールプログラム］処理コマンドを修正することです。以前に説明したように、プログラムが Magic Studio によってリッチクライアントプログラムとして認識されると、［コールプログラム］処理コマンドに［出力先］特性という新しい特性が有効になります。ここには、新しいプログラムが表示される場所として［サブフォーム］コントロールのコントロール名を指定します。

実行時のサブフォーム

サブフォーム内でプログラムが実行されている時に、［コール］処理コマンドが実行され別のプログラムが起動されると、サブフォーム上の現在実行中のプログラムが置き換わります。この場合、現在［サブフォーム］コントロールに表示されているプログラムは終了します。新しいプログラムは、サブフォームの規制の範囲内で初期設定され、表示されます。

位置に関する考慮事項

一般に、フォームはさまざまなサイズで表示されます。1 つの［サブフォーム］コントロールで実行している場合、全てのフォームは、指定されたサイズの指定されたコントロールでの制約条件に対応させる必要があります。

- **プログラムが表示されると、プログラム自身をそのコントロールの制約条件に対応させる必要があります。従って、定義されたフォームが小さい場合、フォームが［サブフォーム］コントロールに対応するようにサイズを大きくすることが期待され、フォームが大きい場合、対応してサイズを小さくするように求められます。**
- **親フォームのサイズが大きくなると、［サブフォーム］コントロールのサイズもそれに応じて大きくなることが求められます。［サブフォーム］コントロールのサイズが大きくなると、コントロール内で表示されるデータもがそれに応じて大きくなることが求められます。**

結果として、フォームとテーブルを設計する場合、［位置］特性を考慮する必要があります。

終了ボタンをシミュレートする

オンラインプログラムでは、現在のプログラムを終了させる［閉じる］ボタンを表示させる場合があります。このような［終了］ボタン（［閉じる］や［完了］と表示する場合もあります）は、一般的に［終了］イベントを発行させるようにします。

しかし、サブフォームでこのイベントを発行させると、第 4 章「サブフォーム」（16 ページ）で説明したように親のプログラムも一緒に終了することになります。従って、ロジックを工夫する必要があります。

以下ような方法を推奨します。

1. サブフォームに表示させるプログラムを作成します。このプログラムはどのような種類のオンラインプログラムでもかまいません。例えば簡単なテキスト表示させたり、通常のプログラムとして動作するものであれば任意に作成したものが利用できます。
2. メインプログラム内に「サブフォームの消去」という名前のユーザ定義のイベントを作成します。
3. 親プログラム内で、この新しいイベント（「サブフォームの消去」）に対するロジックユニットを作成します。このロジックユニット内で、［コールプログラム］処理コマンドを使用してデフォルトプログラムを

呼び出します。[出力先]特性には、定義したサブフォームのコントロール名を指定します。

4. 子タスクの中で、[終了]イベントに対する[伝播]特性を「No」に設定します。このロジックユニット内で、作成したユーザイベント(「サブフォームの消去」)を発行するようにします。

上記のプログラムを実行すると、現在のプログラムが終了し、デフォルトプログラムが表示されます。

ヒント

作成された「**MDI** ライク」な環境を実行すると、メインタスクのウィンドウ内には何も表示されず、メニューとステータスバーのみが表示された状態になります。
このような環境をシミュレートするには、親タスクをダミータスクとして作成します。このダミータスクには、**1** つの文字型の変数項目を定義します。このタスクのフォームには、[サブフォーム]コントロール以外は、タスクに定義された **1** 項目のみを表示させます。また、コントロールの寸法を全て「**0**」に設定します。

トランザクション

第 3 章「トランザクション」(13 ページ)では、遅延トランザクションがリッチクライアントプログラムで使用されるトランザクションであることについて説明しました。

「MDI ライク」なプログラムでは、親タスクはマネージャとして動作します。このマネージャは継続的に実行され、必要に応じて[サブフォーム]コントロール内に別のタスクを呼び出すようになっています。起動されたタスクは、サブフォーム内で実行され、別のタスクが起動されると自動的に現在のタスクは終了されます。

親タスクがトランザクションをオープンした場合、サブフォーム内で起動されるすべてのタスクはこのトランザクション内で実行されます。サブプログラムが終了した場合、親のトランザクションがまだオープンされている状態のため、子プログラムのトランザクションはコミットされません。

このような場合、コミットさせる必要のあるラージトランザクションを作成します。この動作を考慮する必要があります。

マネージャ機能を持つ親タスクでは、トランザクションを使用しないように定義することを推奨します。このような場合、サブフォームで起動された各プログラムは、タスクの起動時にオープンされ、終了時にコミットされられる個別のトランザクションを持つことになります。

しかし、親タスクがトランザクションをオープンする必要があり、さらにサブフォーム内で実行するタスクが個別のトランザクションを持つ必要がある場合は、そのタスクは新しい遅延トランザクションを定義する必要があります。このような場合は、[タスク特性]の[トランザクションモード]特性を「N= 新規の遅延トランザクション」に設定してください。

ノート

このような動作は、フレーム内でプログラムを実行させる場合も同じです。
タスクにフレーム形式フォームのみ定義されている場合、異なるフレーム内で実行されるプログラムが同じトランザクション内にある必要はないため、トランザクションを使用しないように設定してください。
親タスクが、トランザクションをオープンする必要があり、フレーム内で実行されるタスクが、個別のトランザクションを持つ必要がある場合は、新しい遅延トランザクションをオープンするように定義してください。

メニュー

今まで説明してきたように、MDI 環境をシミュレートするには、[コール]処理コマンドの一部の特性を使用します。[メニュー]リポジトリで[メニュータイプ]が「P= プログラム」に定義された場合は、このオプションがありません。

メニューから MDI 環境でプログラムを実行させるには、[メニュータイプ]を「V= イベント」に設定し、ユー

ザ定義のイベントを設定するようにします。これらのイベントは、メニューを表示する親プログラムで処理させるようにします。親プログラム内のロジックユニット内で、表示させる［サブフォーム］コントロールを指定した［コール］処理コマンドを定義します。

Chapter

8

ブラウザコントロール

新しいパラダイムは独自の世界に存在するわけではありません。それは、他のアプリケーションと接続する必要があります。どのようにして接続しますか？

理想的な世界では、外の世界にアクセスすることなく自己完結型のアプリケーションとして開発することができます。独自の知識と経験を利用することで、さまざまな限界を克服することになります。

しかし、現実の世界はこれとは異なり、今日のアプリケーションは外部リソースへのアクセスが必要になります。

なぜ、他のアプリケーションにアクセスする必要があるのでしょうか？

最近のアプリケーションは相互接続が要求されます。これは、文書の PDF を作成し Acrobat Reader で表示させるようにしたり、Excel ワークシートのデータを表示させるなどといった、より複雑なものになるかもしれません。

しかし、分散環境においては、クライアント側が表示を主目的とし、サーバ側でディスクへのアクセスを行うように処理を分けることができます。これにより、何処でどのようなものを作成する必要があるのかを決める必要性が発生します。

ブラウザコントロール

［ブラウザ］コントロールは、他のアプリケーションとの相互接続を可能にするリッチクライアントタスク専用の新しいコントロールです。

［ブラウザ］コントロールは Internet Explorer の処理をシミュレートし、ブラウザの ActiveX コントロールが使用されたかのように動作します。このコントロールをフォームに配置すると、ブラウザが埋め込まれ、必要に応じたコントロールを使用することができます。

ブラウザが埋め込まれると、http://www.magicsoftware.com などのインターネットページを表示するためにこのブラウザを使用することができます。

インターネットブラウザは、次のような様々なタイプのファイルを表示させることができます。

- Word 文書
- Excel シート
- PDF ファイル
- 映画や音楽などのマルチメディアファイル
- その他色々

この機能を任意の目的で使用することができます。例えば、Excel シートを表示させるために［ブラウザ］コントロールを使用することができます。

注意

ファイルを表示させるためには、クライアント側にそのファイルを表示させるためのアプリケーションがインストールされている必要があります。

ブラウザコントロール特性

コントロールは通常のコントロールとして動作しますが、いくつかの特性を定義しておく必要があります。

- データ …… ［ブラウザ］コントロールに表示させる URL を指定します。埋め込まれたブラウザは、その URL にアクセスします。Internet Explorer は、ローカルなファイルシステム上のファイルにアクセスすることができますが、ローカルファイルに対してディレクトリパスではなく URL 指定でアクセスすることを推奨します。この場合、C:¥temp¥Excel.xls などのようにして Excel ファイルを表示する代わりに、http://server/tempDir/Excel.xls という指定で同じファイルにアクセスすることができます。
- コントロール名 …… ブラウザにアクセスする必要がある場合は、コントロール名を指定することを推奨します。
- 位置 …… 実行時にフォームのサイズが変更された場合に、コントロールのサイズも一緒に変更するように指定することができます。

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 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。

ステータスバーがインターネットブラウザのステータスバーであることに注意してください。表示されるかされないかはブラウザ側に依存します。

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。

帳票の作成と表示 - ブラウザコントロールの実装

今まで、[ブラウザ] コントロールの背景となる概念に関して説明してきました。これを応用してみましょう。

アプリケーションを実行させる際に、帳票の提供が要求される場合があります。帳票によっては、単にデータを表示させるだけの簡単なものがあります。また、(例えば、地域毎に分類された前四半期の利益と損失などの) データ操作が必要な複雑な帳票もあります。

リッチクライアントが表示用のメカニズムであり、サーバ側はデータ抽出用のメカニズムを実行するものであることを理解しておく必要があります。プリンタは常にサーバ側に接続されています。

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。

1. 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)
2. そ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。
3. 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。
4. 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。
5. 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://server/pdf/report.pdf
6. 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。

Chapter 9

パフォーマンス

この新しいタイプのパラダイムには、いくつかのパフォーマンス上の問題があります。どのようにして解決したらいいのでしょうか？

前の章では、リッチクライアントモジュールについて説明しました。テクノロジを扱う場合、回避すべきさまざまな落とし穴があります。アプリケーションを開発する場合、アプリケーションがスムーズに動作できるようにこれらの問題に対処する必要があります。この章では、アプリケーションの性能に影響する可能性のある問題の対処方法について説明しています。

なぜ制限があるのでしょうか？

サーバと通信するクライアントプログラムがある場合、それはコミュニケーションバスを経由して実行されます。コミュニケーションバスは、パフォーマンスを低下させるボトルネックになるかもしれません。画面表示に必要な情報を取得するために、プログラムがネットワークを使用してデータのやりとりを行った場合、不要なネットワークトラフィックを発生させると同時に、そのプログラムのパフォーマンスにも影響を与えます。

アプリケーションをスムーズに動作させるためには、タスクの性能と効率を改善する必要があります。

クライアント側とサーバ側

今までの章で説明したように、処理コマンドによって、クライアント側でのみ実行するものや、サーバ側でのみ実行するものがあります。

関数やそれを使用する式についても同じです。関数によっては、クライアント側で評価されるものや、サーバ側で評価されるものがあり、またクライアントかサーバのどちらかで評価されることを意味するニュートラルな関数もあります。クライアント側の関数とサーバ側の関数のリストは、Magic の『リファレンスヘルプ』を参照してください。

同じ式の中でクライアント側の関数とサーバ側の関数が混在して定義された場合、リッチクライアントモジュールがその式を評価するためにコミュニケーションバスを行き来することでパフォーマンスに影響を与えることになります。

この種のパフォーマンス上の問題を防止するために、Magic ではこのような混在状態が見つかった場合、構文チェックユーティリティでエラーメッセージを表示します。

クライアント側の特性

［フォーム特性］と［タスク特性］などのような特性は、サーバへのアクセスを最小限にするためにクライアント側の特性として定義されます。関連する特性については、Magic の『リファレンスヘルプ』を参照してください。

クライアント側で評価される特性では、INIGet などのサーバ側の関数は使用できません。例えば、画面の再表示

を行う度にサーバ側の関数を使用した場合、リッチクライアントモジュールは、式を評価するためにサーバにアクセスします。

実装
このような処理を実装するには、［タスク前］がサーバ側で実行されることを理解する必要があります。クライアント側の処理が混在しないようにするためには、以下のような定義方法が考えられます。

1. ［データビュー］エディタに変数項目を定義します。
2. ［タスク前］で、サーバ側の関数を使用して変数項目を更新します。
3. クライアント側の特性で、サーバ関数の代わりにこの変数項目を使用します。

サーバ側の特性

タスクが起動される際に、いくつかの［タスク特性］が評価されます。［範囲式］などの特性はサーバ側の特性として処理されます。関連する特性については、Magic の『リファレンスヘルプ』を参照してください。

サーバ側で評価される特性では、ClipRead などのクライアント側の関数を使用することはできません。タスクが初期設定を行っている状態では、まだ、リッチクライアントインタフェースは表示されません。このため、モジュールは、クライアント側の関数が含まれた式を評価するために制御をクライアントに戻すことができません。

実装
このような処理を実装するには、以下のように実行タスクツリー内に存在しているプログラムを使用する必要があります。

1. 親プログラムかメインプログラムに変数項目を定義します。
2. プログラムを呼び出す前に、クライアント側の関数によって変数項目を更新します。
3. サーバ側の特性で、この変数項目を使用します。

サーバ側のイベント内でのクライアント側の関数

［タスク前］は、タスクの初期設定段階で実行されるサーバ側のロジックユニットです。

［タスク前］で MnuShow や CtrlGoto などのクライアント側の関数を使用する必要が発生するかもしれません。クライアント側で評価される関数は、クライアント側の処理が利用できないロジックユニット内で使用することはできません。

実装
このような処理を実装するには、以下のようにイベントを利用する必要があります。

1. ユーザイベントを作成します。
2. イベントに対応したロジックユニットを作成します。
3. ロジックユニット内に、必要なクライアント側のロジックを定義します。
4. サーバ側のロジックユニット内で、［ウェイト］特性を「No」に設定してユーザイベントを発行します。

Magic エンジンはイベントの終了を待たず、サーバがアイドル状態になるまでイベント処理が延期されるため、この実装によって意図した動作になるはずです。クライアントが初期設定され、起動されると、イベントはクライアント側で処理されます。

項目の代入特性の使用

項目の［代入］特性は、場合によってはサーバ側で評価されます。例えば、登録モードでの実項目の場合がこれに該当します。変数項目の場合は、クライアント側で評価されます。

特性がいつ評価されるかわからない場合、式の中に定義される関数は、サーバ側でもクライアント側でもないことを意味するニュートラルな関数である必要があります。しかし場合によっては、どちらかのグループに属している関数を使用する必要があります。

実装

このような処理を実装するには、以下のような手順を行う必要があります。

1. 変数項目を定義します。ここには関数の値が格納されます。
2. ［レコード前］において、関連する式を使用して変数項目を更新します。
3. データソースのカラムを処理する場合、タスクが登録モードの場合のみ［代入］特性が有効になります。従って、［項目更新］処理コマンドの実行条件を Stat（0,'C'Mode）に設定する必要があります。

範囲と位置付の式

項目の［範囲］特性と［位置付］特性の式は、場合によってはサーバ側で評価されます。例えばタスク初期設定におけるメインソースの実項目の場合がこれに該当します。変数項目の場合は、クライアントで評価されます。

特性がいつ評価されるかわからないため、式の中に定義される関数は、サーバ側でもクライアント側でもないことを意味するニュートラルな関数である必要があります。

実装

このような処理を実装するには、以下のような手順を行う必要があります。

1. 変数項目を定義します。ここには関数の値が格納されます。範囲式で使用される項目は、タスクの初期設定中に評価されるため、親タスクで変数項目を定義し、パラメータとして式をタスクに渡す必要があります。
2. ［レコード前］において、関連する式を使用して変数項目を更新します。

混在した処理

混在した処理とは、サーバ側の処理コマンドや式がクライアント側の処理コマンドと混在して使用されるものです。このような状態は以下のような場合に発生します。

- **［エラー］処理コマンドのようなクライアント側の処理コマンドの特性値（例えば、［条件］特性）にサーバ側の関数が指定された場合。**
- **サーバ側の処理コマンドの特性値（例えば、［条件式］特性）にクライアント側の関数が指定された場合。**
- **1 つの式に、クライアント側とサーバ側の両方の関数を使用する必要がある場合。**

処理を混在させた場合、同じ処理コマンドや同じ式の中であっても、クライアントとサーバの両方で実行させる必要が発生します。このようなことは、パフォーマンス上の問題を最小化するために Magic ではエラーとして扱われます。構文チェックユーティリティを実行すると、エラーメッセージが表示されます。

実装

このような処理を実装するには、以下のように複数の処理コマンドを使用する必要があります。

1. 変数項目を定義します。
2. 関連する関数の値で変数項目を更新します。ネットワークトラフィックを最小化するために、［項目更新］処理コマンドを同じ場所（サーバまたはクライアント）で実行される他の処理コマンドに続けて定義することを推奨します。
3. 関数の代わりに、式の中で変数項目を使用するようにします。

クライアント側とサーバ側の順番の混在

サーバ側の処理とクライアント側の処理、または別のサーバ側の処理が混在して定義されている場合があるかもしれません。例えば、以下のような場合が考えられます。

1. INIGet 関数で項目 A を更新します。
2.「無効です」という確認メッセージを表示させます。
3. プログラム 3 を呼び出します。

このような場合、最初の処理コマンドでリッチクライアントエンジンはサーバにアクセスし、処理コマンドを実行します。2 番目の処理コマンドではクライアントに戻り、3 番目はサーバ側で実行され、再度クライアントに戻ります。このような簡単な処理でも、ネットワークに 4 回のアクセスが発生し、パフォーマンスに影響することになります。

推薦方法

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それぞれグループ化して、同じグループで連続して実行させるようにすることを推奨します。この方法により、ネットワークトラフィックが最小化されます。

上記で説明した処理は、以下のように定義することができます。

1.「無効です」という確認メッセージを表示させます。
2. INIGet 関数で項目 A を更新します。
3. プログラム 3 を呼び出します。

この場合は、サーバ側で［項目更新］処理コマンドと［コールプログラム］処理コマンドが実行されてクライアントに戻るため、ネットワークへのアクセスは 2 回になります。

全ての状況でこのような対応ができるわけではありません。

同じような問題は、［ブロック］処理コマンドを扱う場合にも発生する可能性があります。

ブロック IF

［ブロック IF］処理コマンドを使用した場合、以下の問題に対応する必要がでてきます。

- **IF/ELSE/ELSE …… IF で指定された条件と各 ELSE で指定された条件が定義されている場合、指定された条件のうちどれか 1 つが「True」と評価されるまで各条件が評価されます。条件式にサーバ側とクライアント側の処理が混在している場合、同じようにパフォーマンス上の問題が発生します。**

IF 文で条件設定においても、サーバ側とクライアント側のグループ毎に処理が行われるように検討する必要があります。

推薦方法

サーバ側またはクライアント側の関数を使用する代わりに、変数項目を更新する方が有効な場合があります。IF-ELSE 条件で条件式に項目を使用する代わりに、変数項目を更新することができます。すべてのサーバまたはクライアントの関数をグループ化することで、サーバへのアクセスを 1 回にすることができ、パフォーマンス上の問題を減らすことができます。

簡単な例を紹介します。以下のような式が定義されているものとします。

If（**サーバ側の式**）
　Else （**クライアント側の式**）
　　If（**サーバ側の式**）

この場合、サーバへのアクセスは 4 回になります。以下のような方法でこのコードを修正します。

クライアント側の式で項目 A を更新
If（**サーバ側の式**）
　Else （**項目 A**）
　　If（**サーバ側の式**）

この場合、サーバへのアクセスは 2 回になります。この方法では、項目を更新した後の［ブロック IF］処理コマンドの位置でサーバ側に制御が移ります。［ブロック IF］処理コマンド内に定義する処理内容を変更することが

できます。以下の場合は、全てクライアント側で実行されます。

サーバ側の式で項目 A を更新
サーバ側の式で項目 B を更新
If（**項目 A**）
Else （**クライアント側の式**）
If（**項目 B**）

どのような方法が最善なのかは、ブロック内の一連の処理コマンドの内容に依存します。処理コマンドがクライアント側で実行されるものであれば、最後に示した方法がよりよいものになるはずです。

ブロック Loop

[ブロック Loop]処理コマンド内にクライアント側とサーバ側の両方の処理コマンドが定義されている場合、ループが繰返される度に、リッチクライアントモジュールはクライアントとサーバの両方をアクセスすることになります。できるだけループ内には 1 種類のタイプの処理コマンドだけを定義するようにしてください。

コントロールとレコードの処理コマンド

コントロール間でカーソルを動かすと、オンラインモジュールと同じように、まず現在のコントロールの［コントロール後］が実行され、それから次のコントロールの［コントロール前］が実行されます。レコードレベルでも同様な動作になります。

［レコード前］または［レコード後］のどちらかにサーバ側の処理コマンドが定義されている場合、トランザクション処理が遅くなります。このようなことはできるだけ少なくすることを推奨します。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使用するようにしてください。

項目変更

［項目変更］には 2 つの異なるフェーズ（再計算とコントロール更新）があります。サーバ側の処理コマンドが使用された場合、このフェーズ内では、関連する処理コマンドを実行するためにサーバにアクセスします。

このようなことは、できるだけ少なくすることを推奨します。従って、ここではクライアント側またはニュートラルの処理コマンドのみを使用するようにしてください。

タスク前での未確認イベント

あるイベントがサーバ側で処理されるのか、またはクライアント側なのかを確認できない場合があります。以下のような場合にこのようなことが発生します。

- **イベントに対応した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。**
- **現在のタスクに定義されている［イベント］ロジックユニットの［伝播］特性が「Yes」に設定されている場合。**

  イベントによってどのタスクが実行中に処理されるかを開発時に把握することはできません。これは主に実行ツリーに依存します。

［タスク前］はサーバ側で実行されるロジックユニットで、タスクが初期設定を行っている間はクライアントにアクセスすることはありません。エンジンがロジックユニットの位置を確認することができないため、開発時はこれらのイベントを防止することはできません。

予想外の結果になる可能性があるため、このような状況は避けるようにしてください。これらのイベントがクライアントで処理された場合、実行エラーが発生するかもしれません。

起動時間と操作時間

「リッチクライアントタスクの設定」（18 ページ）で説明しましたが、［タスク特性］の［ビューの事前読込］特性を設定することで、事前にデータビュー全体を読み込むかどうかを定義することができます。

フォーム上に［ツリー］コントロールが定義されたタスクの場合、［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で事前にデータビュー全体を読み込むかどうかを定義することができます。［ツリー］コントロールがフォームに配置された場合、［タスク特性］は無視されます。

これらの特性値が「Ture」に設定された場合、レコード間またはノード間の操作が早くなります。追加レコードを検索するためにサーバに制御が戻る事はありません。しかし、データビューに大量のレコードが含まれている場合、起動に時間がかかる可能性があります。

Chapter 10

アプリケーションのモニタ

実行すると不具合が発生します。どのように原因を見つけ出しますか？

常に正常に動作するアプリケーションを開発し実行させることが開発者の目標です。しかし、不具合は突然、当然のごとく発生し、しかも最も悪い状況で発生するものです。

この章では、リッチクライアントアプリケーションを監視する方法について説明します。

目の前にある問題

リッチクライアントのパラダイムは、主にデータを表示するシンクライアントと実行環境と相互作用するサーバに基づいています。

Magic のデバッガを使用することで、サーバ側で処理されている内容を参照することがができますが、クライアント側の処理内容も確認する必要があります。ブレイクポイントを定義して処理を順番に確認することができないため、リモート側のクライアントを使用して実行させる場合、問題になります。

ロギング

このような問題の解決方法は、ログを出力するように設定し、出力されたログを参照することです。ログには、各レベルや各コンテキストで発生した内容が出力されます。

Magic のロギング

Magic のデバッガには、クライアント側とサーバ側の両方で処理内容を表示させることのできる組込型のロギング機能が備わっています。

動作のロギング

アプリケーションを監視するには、Magic Studio でプロジェクトを読み込む必要があります。これでロギング機能を使用することができます。

[デバッグ] メニューでデバッグモードを有効にすることで、アプリケーションはデバッグモードで実行され、ロギングが開始されます。

アクティビティモニタ

[表示] メニューから [アクティビティモニタ] を選択すると、現在の動作内容が別ウィンドウ内に表示されます。

アクティブティモニタの表示内容についての詳細は、Magic の『リファレンスヘルプ』を参照してください。

フィルタ

Magic のロギング機能には、ログに出力したい内容を絞り込むことができます。フィルタの設定は、プルダウン

メニューから［オプション \ 設定 \ ロギング］を選択することで可能です。

フィルタ機能に関する詳細は、Magic の『リファレンスヘルプ』を参照してください。

> **ノート**
>
> クライアント側の処理内容を参照するには、［ロギング］テーブルの［クライアントの動作］の設定を「**Yes**」にする必要があります。この設定が「**No**」の場合、サーバ側の処理内容のみ記録されます。

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。

クライアントのクラッシュ

クライアントがクラッシュすることで、直近の処理内容がログに残らない場合があります。このような場合、クラッシュする前にどのような処理が実行されていたかを知っておく必要があります。これらの処理内容は、まだサーバ上では消去されていないため、アクティビティモニタには表示されません。

クラッシュ前の処理内容を確認するには、リッチクライアントインタフェースビルダによって作成された JNLP ファイルの設定を変更する必要があります。この設定は、JNLP を再起動したときから有効になります。

<application-desc> と呼ばれるセクションの中に、LogClientSequenceForActivityMonitor と呼ばれるパラメータがあります。この設定を「Y」に変更してください。

```
<application-desc main-class="com.magicsoftware.richclient.ClientManager">
    <argument>protocol=http</argument>
    <argument>server=Localhost</argument>
    <argument>requester=/eDevScripts/mgrqispi101.dll</argument>
    <argument>appname=PetShopDemo</argument>
    <argument>prgname=Maint_Control</argument>
    <argument>envvars=&amp;quot;&amp;quot;</argument>
    <argument>DisplayStatisticInformation=N</argument>
    <argument>LogClientSequenceForActivityMonitor=Y</argument>
    <argument>InternalLogLevel=NONE</argument>
    <!--argument>InternalLogFile=InternalClient.log</argument-->
    </application-desc>
</jnlp>
```

このオプションが設定されると、JNLP はサーバに送られていないクライアント側の処理のログを含む物理ファイルが作成されます。

前に説明したように、これらのログの内容はアクティビティモニタに表示するためにサーバに送られます。それらがサーバに渡されると、ファイルは削除されます。

クラッシュの後にログの内容をアクティビティモニタに送るために、以下の処理を実行してください。

1. 実行中のコンテキストをすべての終了します。これは、他のコンテキストからモニタにログが送られることを防ぐためです。
2. リッチクライアントモジュールを再起動します。

クラッシュ前に作成されたログの内容が自動的にサーバに送られ、アクティビティモニタで参照することができるようになります。

Chapter 11

プログラムの書き直し

リッチクライアントのプログラミングは、オンラインプログラミングと異なるものと思われがちですが、実際はどうでしょうか？

このドキュメントでは、リッチクライアントの技術と、アプリケーションを新しい環境の制約内で、よりよく動作させることを保証するための方法について説明してきました。このドキュメントの読者のほとんどは、Magic のプログラミングに関しての初心者ではなく、独自のプログラミング方法を持っている場合があります。今までよく使用していたオプションのいくつかがリッチクライアントプログラミングでは利用できないものがあります。

この章では、リッチクライアント用にプログラムを作成する上で、注意しなければならない変更点について解説しています。新しい方法で開発する場合を含め、今まで使用していたプログラムを修正する場合に発生する可能性のある問題点についても説明しています。

サポートしていない機能

今まで説明してきたように、リッチクライアントプログラムは、通常のオンラインまたはブラウザクライアントのプログラムとは異なった動作をします。今まで使用していたいくつかの機能が、この環境では利用できなくなっています。

簡単な Magic プログラミングを使用して、利用できない機能の代替処理を実装する必要があります。

- **通常の Magic 関数の中にはリッチクライアントタスクでは利用できないものがあります。これらのほとんどは、簡単な回避手段があります。関数の中には、ロジックユニットに定義できないものがあります。これについては、今まで説明してきました。例えば、KBPut () と KBGet () はサポートされません。これらの関数の代わりに、［イベント実行］処理コマンドと［イベント］ロジックユニットを使用することで実装することができます。**
- **（［ユーザイベント n］のような）いくつかの内部イベントは、リッチクライアントタスクでは利用できません。通常のユーザイベントを作成し、これらのイベントの代わりとして使用する必要があります。**
- **いくつかのコントロール特性は、リッチクライアントタスクでは利用できません。例えば、［選択プログラム］特性は、コントロールの［特性シート］には表示されません。この場合、代わりに、該当するコントロールに対応した［ズーム］イベントのロジックユニットを作成することによって実装することができます。**

オンラインからリッチクライアントプログラミングへ

アプリケーションをオンラインプログラムからリッチクライアントプログラムに変更する場合、動作の違いを考慮する必要があります。

いくつかは、今までの章で説明してきました。アプリケーションを変更する前に、まずリッチクラ

イアント準備ユーティリティを実行してください。このユーティリティは、アプリケーションに定義されているモデルをスキャンし、［スタイル］特性の［GUI 表示形式］の内容を［リッチクライアント表示形式］にコピーします。この処理は、継承が解除されている特性に対してのみ行われます。

プルダウンメニュー
これは、第 6 章 「メニューとツールバー」（26 ページ）で説明しています。リッチクライアントアプリケーションは、アプリケーションのメニューを含む MDI フレームまたはコンテナを持っていません。

これは SDI プログラムの中で実装する必要があります。

終了プログラム
オンラインプログラムの場合、ESC キーを押すことでプログラムは終了します。この場合、メニューからこのプログラムを選択した時点の MDI フレームに戻ります。

今まで説明してきたように、リッチクライアントアプリケーションは MDI フレームを持っていません。従って、すべてのプログラムを終了するために、ユーザが ESC キーを押した場合、それらは不用意にアプリケーション自体を終了させることになります。このような状況を避けるために、適切なプログラム内に定義した［イベント］ロジックユニットで ESC キーを処理するようにする必要があります。

他のプログラムを呼び出す
オンラインプログラムの場合、プログラムが別プログラムを呼び出すと、並行起動プログラムでない限り起動元のプログラムの処理は中断されます。しかし、リッチクライアントプログラムが別のプログラムを呼ぶ場合、起動元のプログラムも並行して実行されます。

オンラインプログラムと同じように動作させるには、起動するプログラムをモーダルに指定します。これは、その［ウィンドウタイプ］特性を「M= モーダル」にすることで実現できます。

帳票出力
帳票出力は、PDF ファイルとして出力し、関連プログラムを使用して PDF を表示することで実現できます。この例は第 8 章 「帳票の作成と表示 - ブラウザコントロールの実装」（36 ページ）で説明しています。この例では、帳票を表示するために［ブラウザ］コントロールを使用しています。

フォーム特性
［タスクタイプ］特性を「O= オンライン」から「C= リッチクライアント」に変更すると、タスクに定義されているすべての［GUI 表示形式］フォームが［フォーム］エディタの最終行からコピーされます。元のフォームの［インタフェースタイプ］は、「C= リッチクライアント表示形式」に変更されます。リッチクライアントでサポートされるコントロールと特性のすべては、フォーム上で有効になります。

これらのコントロールと特性は、ほとんどがオンラインタスクの表示フォームに対応するものと同じ特性があります。

式の中で ’3’FORM のように、特定のフォームを指定しているような場合、その指定内容は変更されません。

変更前の（オンラインタスクの）フォームは、欠落したコントロールと特性の確認用として保存されています。必要がなければ、このフォームを削除することができます。

重要

リッチクライアントコントロールは **Java SWT** に基づいて動作します。このため、コントロールのルック＆フィールはオンラインタスクと異なる場合があります。場合によっては、コントロールの幅が、テキスト全体を表示できないかもしれません。これは［コンボボックス］コントロールのようなチョイスコントロールや［カラム］コントロールなどで明確に表れます。
あらかじめフォームの内容を確認しておくことを推奨します。

フォームの分割
フォームが分割されている場合、「リッチクライアントフレーム形式」フォームが追加されます。このフォームは、既存のフォームに対する参照内容をフレームの 1 つに含めた状態で作成されます。

オンラインタスクでは、分割ウィンドウ内でタスクを実行させる場合、タスクフォームの［ウィンドウタイプ］特性を［分割子ウィンドウ］として定義します。起動元のプログラムから、新しいプログラムを呼び出します。分割定義が設定されているフォームをリッチクライアント用に変換すると、分割されたフレームワーク内にどのプログラムが表示されるのかを確認する方法がありません。2 番目のフレーム内に表示させるタスクを手動で指定する必要があります。また、［コール］処理コマンドを削除する必要があります。

複数のタスクを、指定されたフレーム内に表示させる必要がある場合、［コール］処理コマンドの［出力先］特性を使用することができます。

子ウィンドウ

［ウィンドウタイプ］特性が「C= 子ウィンドウ」に設定されたプログラムは、サブフォームに定義することでのみ実行させることができます。これらは新しいウィンドウを開くことはできません。

コントロール

いくつかのコントロールは、リッチクライアント環境では動作が異なります。プログラムを作成する時に、これらを考慮するために、その違いを事前に知っておく必要があります。

ここでは、考慮する必要がある問題について説明しています。

- ［テーブル］コントロール …… テーブル内に表示される変数項目の値を変更すると、行のすべての変数項目が再表示されます。
- ［タブ］コントロール …… タブの値が変更されても、［タブ］コントロールは［ズーム］イベントを発行しません。この処理を［項目変更］ロジックユニットに変更する必要があります。
- ［サブフォーム］コントロール …… オンラインタスクの場合、タスクがサブフォームによって表示されているのであれば、キャレットがタスクを抜けると、タスクは自動的に終了します。リッチクライアントの場合は、コンテナタスクが終了するまで、タスクは実行し続けます。以下は、事前に知っておく必要のある問題点です。
  - 複数のサブフォーム内で同じデータソースを使用する場合、データソースの［名前］特性を使用すると、そのデータソースを使用する全てのサブフォームに影響します。
  - サブフォームタスク内で［終了］イベントを発行すると、サブフォームのコンテナタスク内でイベントを発行することになり、結果として両方のプログラムを終了させることになります。この動作については、この章の初めの箇所で説明していています。
- コントロールの位置 …… リッチクライアントタスク内のコントロールの位置は、フォームではなくコンテナコントロールと関連しています。場合によっては、それに応じて特性値を調整する必要があります。

ステータスバー

ステータスバーについては、第 6 章 「メニューとツールバー」（26 ページ）で説明しています。リッチクライアントアプリケーションは、ステータスバーを含む MDI フレームやコンテナを持っていません。ステータスバーを表示させる必要がある場合は、SDI ウィンドウ内に実装する必要があります。

MDI フレームのステータスバーとは異なることに注意してください。SDI のステータスバーでは、エラーメッセージや自動ヘルプの表示のみを行います。

これは、コントロールに対応する［ズーム］イベントのロジックユニットが定義されていても、ステータスバーに「ズーム」が表示されないことを意味しています。［コントロール特性］や［モデル特性］に自動ヘルプを定義した場合、この機能を実装する必要性がでてきます。

画像表示

［テキスト］コントロールに画像ファイルのファイル名を格納してフォームに表示させている場合は、ファイルの位置を URL 形式で指定するように変更する必要があります。ファイルの位置を論理名で指定している場合は、論理名の実行値を変更するだけなので変更が楽になります。その際、IIS で定義されているエイリアスのスコープ内にファイルが格納されている必要があります。

ブラウザクライアントからリッチクライアントへのプログラミング

アプリケーションをブラウザプログラムからリッチクライアントプログラムに変更する場合、動作の違いを考慮する必要があります。

そのうちいくつかは、今までの章で説明してきました。

プルダウンメニュー
SDI プログラムは、使用している OS のルック＆フィールを持つプルダウンメニューを提供します。独自にプログラムを定義しないで、これらのメニューを使用してください。これはアプリケーションにクライアント / サーバのルック＆フィールを与えることになります。

認証
リッチクライアントは、独自のログオンダイアログを持っています。このような認証ダイアログはプログラムで作成せす、このダイアログを使用するようにしてください。

終了プログラム
ESC キーを使用することでプログラムを終了させることができます。従って、すべてのプログラムを終了するために、ユーザが ESC キーを押した場合、それらは不用意にアプリケーション自体を終了させることになります。このような状況を避けるために、適切なプログラム内に定義した［イベント］ロジックユニットで ESC キーを処理するようにする必要があります。

帳票出力と外部ファイル
ブラウザクライアント環境で IE を使用して表示させる出力ファイル（PDF や HTML ファイルなど）は、［ブラウザ］コントロールを使用することでリッチクライアントで表示させることができます。これによってルック＆フィールの制御を行うことができます。

マージ出力
マージ出力の結果ファイルは、［ブラウザ］コントロールを使用してリッチクライアントで表示させることができます。マージ出力処理プログラムは同じコンテキストで実行していないため、リッチクライアントタスクからは、明示的にパラメータが渡される必要があります。

JavaScript
JavaScript 関数は、［ブラウザ］コントロールを使用して実行させることができます。

フォームとコントロール
［タスクタイプ］特性を「R= ブラウザ」から「C= リッチクライアント」に変更すると、タスクに定義されているすべての［ブラウザ形式］フォームが［フォーム］エディタの最終行からコピーされます。元のフォームの［インタフェースタイプ］は、「C= リッチクライアント表示形式」に変更されます。

HTML からリッチクライアントへの変換方法がないため、［リッチクライアント］フォームは、関連するコントロールによってすべて作り直す必要があります。

- ブラウザクライアントで CSS ファイルを使用していた場合、色とフォントは Magic アプリケーションの基本色定義とフォント定義ファイルを元に設定します。
- ハイパーリンクのルック＆フィールを維持するために、［エディット］コントロールに青色を設定し、このコントロールのために［ダブルクリック］イベントでのロジックユニットを定義します。
- フレームを含むブラウザクライアントプログラムは、［リッチクライアントフレームセット形式］フォームに変換する必要があります。［出力先フレーム］特性を使用しているすべての［コール］処理コマンドは、［出力先］特性を使用する必要があります。
- すべてのイメージは、イメージを参照する URL アドレスを設定する必要があります。
- ［テーブル］コントロールの代替えの背景色を実装するには、［テーブル］コントロールの［交互表示色］特性を使用します。

Chapter 12

要約

Magic プログラミングのこの新しいスタイルに乗り出す前にひとこと

私達は、このドキュメントがリッチクライアントプログラミングを使用する上での手助けになれることを期待しています。

このドキュメントを読んだ後に、リッチクライアントの用語や、制限事項およびその回避方法について理解していただければ幸いです。

私達は、リッチクライアントに基づくプログラミングによって、開発者がより短い期間によりよいインターネットまたはイントラネットのアプリケーションを作成することができるようになるものと確信しています。