Magic eDeveloper V10

マスタリング eDeveloper

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。
IBM®, iSeries™, xSeries®, DB2® および WebSphere® は、IBM Corporation の商標または登録商標です。
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。
Linux® は Linus Torvalds の登録商標です。
GLOBEtrotter® と FLEXlm® は、Macrovision Corporation の登録商標です。
Interstage® は、富士通株式会社の登録商標です。
JBoss™ は、JBoss Inc. の商標です。
Systinet™ は、Hewlett-Packard Company 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。

マスタリング eDeveloper

第 1 版　2007 年 3 月 31 日

第 2 版　2007 年 6 月 30 日

第 3 版　2007 年 9 月 30 日

第 4 版　2008 年 4 月 30 日

第 5 版　2009 年 1 月 23 日

目次

第 1 章 :*操作方法とワークスペース*

プロジェクトのオブジェクトを整理するには 1
フォルダを作成する 1
フォルダを削除する 1
フォルダを移動する 1
フォルダにオブジェクトを移動する 2
フォルダカラムを使用してフォルダにオブジェクトを移動する 2
移動コマンドを使用して移動する 2
パレットを分離するには 3
結合されたパレットを分離する 3
開発用テーブルの行位置付けを行うには 4
行に位置付ける 4
行番号を指定して移動するには 5
行に移動する 5
オブジェクトがどこで使用されているかを確認するには 6
クロスリファレンスを使用する 6
よく使用するオブジェクトにブックマークを設定するには 7
現在の場所にブックマークを定義する 7
最近使用したプロジェクトを迅速に開くには 8
最近使用したプロジェクトを開く 8
最近開いたプロジェクトの表示数を変更するには 9
最近開いたプロジェクトの数を設定する 9
入力行を複写するには 10
複写機能を使用する 10
入力行を移動するには 11
入力行を移動する 11
入力行を他の行の内容に置き換えるには 12
行を置き換える 12
ペイン間の移動を行うには 13
1 つのセクションを表示している特性シートを保持するには 14
特性シートの表示を変更する 14
プロジェクトを切り替えるには 15
プロジェクトを切り替える 15
モジュールを追加する 15
モジュールを削除する 16
特性シートのセクション表示を切り替えるには 17
特性シートのセクション間を移動する 17
特性シートのノードを拡張表示させる 17
特性シートのノードを縮小表示させる 17

第 2 章 :*プロジェクトとアプリケーション*

プロジェクトを新規作成するには 19
プロジェクトを新規作成する 19
アプリケーションのアイコンを設定するには 21
アプリケーションのアイコンを設定する 21

アプリケーションのタイトルを指定するには 22
アプリケーションのタイトルを設定する 22
ステータスバーにタイトルを表示する 22
アプリケーションにコンテキストメニューを設定するには 23
アプリケーションにコンテキストメニューを設定する 23
アプリケーションにプルダウンメニューを設定するには 24
アプリケーションにプルダウンメニューを設定する 24
アプリケーション用に基本色、フォント、キーボード割付を設定するには 25
アプリケーション用の定義ファイルを設定する 25
プロジェクトのディレクトリのファイルを読み書きするには 26
Magic ディレクトリのファイルを読み書きするには 27
システム一時ディレクトリ内のファイルを読み書きするには 28
既存のプロジェクトをオープンするには 29
プロジェクトを開くダイアログを使用する 29
最近使ったプロジェクトを開く 29
プロジェクトファイル（.edp）を直接アクセスする 30
別のプロジェクトにオブジェクトを転送するには 31
オブジェクトを出力する 31
オブジェクトを入力する 31

第 3 章 :*モデル*

再利用可能なインタフェースオブジェクトを定義するには 33
コントロールモデルを作成する 34
再利用可能なデータオブジェクトを定義するには 35
項目モデルを作成する 36
モデルを使用してデータソースのカラムを定義するには 37
モデルを使用してデータソースのカラムを定義する 37
プロジェクトのデータ項目とコントロールを標準化するには 38
項目モデルに対するコントロールを定義する 38
モデル特性の変更内容をオブジェクトに反映させないようにするには 39
手動で継承を解除する 39
自動的に継承を解除する 39
継承が解除された特性を継承させるには 40
特性値を継承させる 40
モデルのクラスを変更するには 41
コントロールモデルを使用して、フォームにコントロールを自動配置するには 42
パレットからモデルを指定してコントロールを選択する 42
モデルを含めてプログラムやデータソースを出力するには 43
モデルを含めて出力する 43
複数のプロジェクトでモデルを共有するには 44
コンポーネントとしてモデルを共有する 44

第 4 章 :*Magic エンジン*

アプリケーションレベルのイベントを定義するには 45
グローバルイベントを作成する 45
Magic エンジンをイベント駆動型で動作させるには 47
イベントのコンセプト 47
ロジックユニットを作成する 48
イベントの階層を利用するには 49
システムイベントをブロックする 49
システムイベントに機能を追加する 49
行削除のような内部イベントの発生を防止するには 50
内部イベントをブロックする 50
入出力ファイル名に動的な名前を設定するには 53
入出力ファイル名を設定する 53
データソースに動的な名前を設定するには 54
タスクレベルでデータソース名を指定する 54
同じファイルやデータソースを扱っているタスク内でファイルやデータソースを削除するには 55
出力する内容がない場合にファイルを作成したり空白ページを出力させないようにするには 56
入出力ファイルをオープンするタイミングを指定する 56
タスク作成時にタスクレベルのロジックユニットを自動的に作成するようにするには 57
タスクとレコードレベルのロジックユニットを自動的に作成する 57
タスク作成時にロジックユニットが作成されないようにするには 58
ロジックユニットの自動作成を止める 58
イベント処理中にエディットコントロールの値を取得するには 59
エンドユーザがタスク内でレコードを修正することを防ぐには 60
データソースのアクセスモードを設定する 60
タスク特性の初期モードを使用する 60
タスク特性のオプションを使用する 61
項目レベルで修正する 61
レコードが更新されなくてもレコード後を実行させるには 62
強制的にレコード後を実行させる 62
エンドユーザがタスクモードを変更することを防止するには 63
ユーザがタスクモードを変更することを防止する 63
２つのタスクを並行に実行させるには 64
変更実行するようにプログラムを設定する 65

第 5 章 :*タスク*

プログラムのデータビューを定義するには 67
メインソースを定義する 68
リンクテーブルを定義する 68
変数とパラメータを定義する 69
簡単なプログラムを作成するには 70
Magic で「Hello World」を作成する 70
データビューを作成する 70
ロジックを作成する 71

表示画面を作成する 71
プログラムを実行する 72

起動元のプログラムに戻り値を返すように設定するには 74
戻り値を定義する 74
戻り値を使用する 74

データリポジトリに定義されている名前と異なるデータソース名を使用するには .. 76
タスクレベルでデータソース名を指定する 76

レコードの表示順を動的に変更するには 77
インデックス用の式を使用する 77

簡単なバッチプログラムを作成するには 78
簡単なバッチタスクを作成する 78

永久ループするバッチタスクを止めるには 79

データソースからレコードをまとめて削除するには 80
削除用のバッチタスクを作成する 80

データソースの内容をテキストファイルに出力したり、テキストファイルをデータソースに入力するには 81
データソースの内容をテキストファイルに出力する 81
テキストファイルの内容を読み込む 81

プロジェクト内のどこでもアクセス可能なグローバル変数を定義するには 83
グローバル変数を定義する 83

表示するメインフォームを動的に指定するには 84
実行時に表示するフォームを指定する 84

起動元と起動先のパラメータを同期させるには 85

エンドユーザがレコードを追加することを防止するには 86
ユーザが登録モードに切り替えることを防止する 86

エンドユーザがレコードを削除することを防止するには 87
ユーザが削除モードに切り替えることを防止する 87

ユーザがレコードを修正することを防止するには 88
ユーザが修正モードに切り替えることを防止する 88

エンドユーザが特定の項目の内容を変更することを防止するには 89
項目を修正不可に設定する 89

選択プログラムを作成するには 90
選択用プログラムを作成する 90
選択プログラムを使用する 92
指定したプログラムに移動する 92

エンドユーザが入力したデータを検証するには 93
コントロール検証ロジックユニットを定義する 93

Tab によるコントロールの移動順序を設定するには 94
TAB 順序を設定する 94
複数のコントロールの TAB 順序を一度に設定する 94

サブタスクレベルからプログラムを終了するには 95
終了イベントを伝播させる 95

タスクの編集中に編集内容を保存するには 96
プログラムの編集中に保存する 96

タスクにロジックを作成するには 97
ヘッダ行を入力する 97
処理コマンドを入力する 97

ロジックヘッダを簡単に作成するには 98
事前定義ロジックを追加する 98
データ項目に対応したロジックヘッダを作成する 98
任意ロジックヘッダを作成する 98
コントロールに対応したロジックヘッダを作成する 99
タスクに処理コマンドを設定するには 100
条件特性を使用する 100
項目更新 100
コール 100
外部コール 101
イベント実行 101
アクション 101
ブロック 101
エラー 102
フォーム 102
サブタスクとしてタスクをコピーするには 103
プログラムをサブタスクとしてコピーする 103
サブタスクをルートタスクにするには 104
サブタスクを先頭レベルに移動する 104
テーブルコントロールのカラムを選択するには 105
強調表示カラムの左側にコントロールや項目をドロップしてカラムを作成するには 106
複数のコントロールを同じカラムに配置するには 106
フォームにコントロールを追加するには 107
項目パレットを使用する 108
コントロールを連続して複数回ドロップするには 109
子コントロールを選択しないでコンテナコントロールを選択するには 110
再利用のためにフォームの書式を保存するには 110
フォームをテンプレートとして保存する 110
テンプレートフォームを使用する 110
複数のコントロールを同時に選択するには 111
ラバーバンドを使用してコントロールを選択する 111
Ctrl+ クリックを使用してコントロールを選択する 111
テーブルコントロールに交互色を表示させるには 112
テーブルで交互色を使用する 112
テーブルコントロールの区切り線の表示／非表示を行うには 113
フォーム上のコントロールのサイズを表示テキストに合わせるには 114
コントロールの TAB 順序を変更するには 116
TAB 順序を設定する 116
コントロールを他のコントロールより前面に表示させるには 117
Z オーダを表示する 117
Z オーダを手動にする 117
コントロールをリンクする 118
タスクのメインフォームに移動するには 119
すべてのコントロールの TAB 順序表示するには 120
フォームエディタの修正内容を取り消すには 120

複数のコントロールの幅や高さを同じにするには 121
複数のコントロールの特性を同時に設定するには 122
複数のコントロールの特性を変更する 122
テーブルコントロールの幅を変更するには 123
テーブルコントロールのカラムを移動するには 124
テーブルコントロールのカラムを移動する 124
タブコントロールの編集中にタブの切り替えを行うには 125
コントロールを透過表示するには 126
背景色を設定する 126
フォームにデフォルトのプッシュボタンを設定するには 127
デフォルトのプッシュボタンを設定する 127
デフォルトのフォームレイアウトを自動的に作成するには 128
フォームジェネレータを使用する 128
ウィンドウタイトルを動的に表示させるには 129
フォームタイトルに式を使用する 129
フォームのアイコンを設定するには 130
フォームのアイコンを選択する 130
フォームのコントロールにデフォルトのコンテキストメニューを設定するには ... 131
フォームにデフォルトのコンテキストメニューを設定する 131
個々のコントロールにコンテキストメニューを設定するには 132
コントロールレベルのコンテキストメニューを作成する 132
タスクがバッチタスクから繰り返し呼ばれる場合のパフォーマンスを改善するには 133
バッチのサブタスクのパフォーマンスを改善する 133

第 6 章 :*拡張されたロジック*

電子メールを送信するには 135
サーバと接続する 135
電子メールを送信する 135
電子メールにファイルを添付するには 137
電子メールを受信するには 138
サーバのタイプ 138
添付ファイルを受信するには 139
Web ページやその他の URL コンテンツを受信するには 140
HTTPGet() 関数を使用する 140
キー操作をシミュレートするには 141
KbPut() 関数を使用する 141
複数のレコードをマークしたり、マークしたレコードを処理させるには 142
テーブルをマルチマーキング対応に設定する 142
マークされたレコードを処理する 142
マルチマーク関数を使用して処理する 143
メニューを非表示 / 無効 / チェック表示に設定するには 144
論理メニュー名を指定する 144
メニューパス関数を使用する 144
メニュー番号を指定する 144
MnuCheck() 関数を使用する 144

MnuEnabl() 関数を使用する 145
MnuShow() 関数を使用する 145
MnuAdd() 関数を使用する 146
MnuRemove 関数を使用する 146
MnuReset() 関数を使用する 147
MnuName() 関数を使用する 147

DLL 関数を呼び出すには 148
CallDLL() 関数を使用する 148
コール UDP 処理コマンドを使用する 149

構造体のパラメータを DLL に渡すには 150
構造体を作成する 150
Magic からバッファを送る 150
変数項目を準備する 150
API を呼び出すロジックユニットを定義する 151

公開名でプログラムを呼び出すには 153
公開名でプログラムを起動する 153

プログラム番号を指定して動的にプログラムを呼び出すには 154
コール式処理コマンドを使用する 154
CallProg() 関数を使用する 154
プログラムを設定する 154

データの暗号化と復号化をするには 155
Cipher() 関数を使用する 155
Decipher() 関数を使用する 156
サポートする暗号化方法 156

ユーザがパークしたコントロール名を取得するには 157
LastPark() 関数を使用する 157

ユーザがクリックしたコントロール名を取得するには 158
LastClicked() 関数を使用する 158

システムの環境変数から値を取得するには 159
OSEnvGet 関数を使用する 159

ディスク上のファイルを削除するには 160
FileDelete() 関数を使用する 160

ディスク上のファイルをコピーするには 161
FileCopy() 関数を使用する 161

ディスク上のファイルの存在をチェックするには 162
FileExist() 関数を使用する 162

ファイルディレクトリの内容を取得するには 163
FileListGet() 関数を使用する 163

ディスク上のファイルをリネームするには 164
FileRename() 関数を使用する 164

ディスク上のファイルのサイズを取得するには 165
FileInfo() 関数を使用する 165
FileSize() 関数を使用する 165

他の Windows アプリケーションにフォーカスを移すには 166
SetWondowFocus() 関数を使用する 166

第 7 章 :*実行*

アプリケーションを実行するには 167

キャビネットファイルを作成するには 168
キャビネットファイルを作成する 168
アプリケーション用のショートカットを作成するには 169
アプリケーションファイルのショートカットを作成する 169
Magic エンジンのショートカットを作成する 169
独自のアイコンを使用する 170
実行エンジンがキャビネットファイルを自動的に読み込むように設定するには ... 171
動作環境でデフォルトプロジェクトを設定する 171
ショートカットにキャビネットファイルを設定する 172
起動アプリケーション用のスプラッシュイメージを表示するには 173
ロゴファイルを設定する 173

第 8 章 :*サブフォーム*

サブフォームコントロールを起動されるプログラムフォームの寸法に合わせるには 175
自動調整を設定する 175
サブフォームの再表示を手動にするには 176
サブフォームの再表示を手動化する 176
複数のパラメータをサブフォームに渡した場合、最後のパラメータを修正した時のみ再表示するには 178
サブフォームの自動再表示特性を無効にする 178
タブコントロールに配置したサブフォームの表示を制御するには 180
サブフォームをタブに配置する 180
親フォームのコントロールからサブフォームに Tab 入力できるようにするには ... 181
サブフォームコントロールの TAB 順序を設定する 181
サブフォーム上のコントロールから親のコントロールに自動的に戻るには 182
サブフォームを終了するイベントを設定する 182
最初にサブフォームタスクが起動されたときのみタスク前 / 後を実行させるには .. 183
サブフォームが最初に起動された場合のみ実行するブロックを定義する 183
ユーザがサブフォームに入ると常にタスク前 / 後が実行されるようにするには 184
ユーザがサブフォームに入るときにのみ実行するブロックを定義する 184
サブフォームが再表示されたときにサブタスクのタスク前 / 後を実行させるには .. 185
サブフォームが再表示されたときにのみ実行するブロックを定義する 185
自動調整オプションをどのように選択するか 186
自動調整オプションの結果 186

第 9 章 :*分割*

分割機能を利用して２つのフォームを表示するには 187
分割フォームを設定する 188
分割画面の初期設定を行うには 190
親タスク表示を分割画面の別の側に表示させるように設定するには 191
親タスクの表示位置を設定する 191
分割画面の割合を動的に指定するには 192
割合を動的に設定する 192
現在の分割割合の値を取得するには 192

エンドユーザによって実行時に設定された分割位置を維持するには 193
フォーム状態 ID を設定する 193
分割領域を最小サイズに設定するには 194
最小サイズを設定する 194

第 10 章 :*ツリーコントロール*

データをツリー形式で表示するには 195
ツリーコントロールを定義する 195
ツリー表示させるための階層的なデータソースを定義するには 197
ツリーのノードにアイコンを設定するには 198
アイコンを使用する 198
展開 / 縮小ボタンを表示 / 非表示するには 199
展開 / 縮小ボタンの表示 / 非表示を切り替える 199
実行時にノードを追加するには 200
子ノードを作成する 200
実行時に兄弟関係のノードを追加するには 201
兄弟関係のノードを作成する 201
実行時にツリーノードを明示的に展開 / 縮小させるには 202
ノード展開 / ノード縮小イベントを実行する 202
ツリーを表示する際にノードを自動的に展開させるには 203
特定のノードに対して自動展開を設定する 203
自動展開をツリーレベルで設定する 204
子ノードを持つノードのみ展開ボタンを表示させるには 205
事前読込を有効にする 205
ツリーコントロールの接続線を表示 / 非表示させるには 206
接続線の表示を切り替える 206
ルートノードの表示 / 非表示を切り替えるには 207
ルートノードの表示 / 非表示を切り替える 207
特定のノードに移動するには 208
TreeNodeGoto() 関数を使用する 208
エンドユーザが行ったノード展開 / 縮小の操作に対応する処理を実行させるには ...209
ノード展開 / 縮小イベントを取り込む 209
ユーザのノードから別のノードに移動する操作に対応するには 210
ノードに入る動作を取り込む 210
ノードから抜け出る動作を取り込む 210
現在のツリーノードの行を強調表示させるには 211
行ハイライトの指定を切り替える 211

第 11 章 :*いろいろなロジック*

ユーザ定義関数を作成するには 213
関数の有効範囲を定義する 213
ユーザ定義関数を作成する 214
ユーザ定義関数のパラメータを設定するには 215
関数のパラメータを定義する 215

ユーザ定義関数の戻り値を設定するには 216
ユーザ定義関数に戻り値を設定する 216
現在のタスクでのみ有効な関数を作成するには 217
ローカル関数を作成する 217
プロジェクト全体で有効な関数を作成するには 218
グローバル関数を作成する 218
複数のプロジェクト間で関数を共有するには 219
グローバル関数を作成する 219
ユーザ定義関数のヘルプを定義するには 220
ユーザ定義関数にコメントを定義する 220
一連の処理を繰り返し実行させるには 221
LoopCounter() 関数を使用する 221
ブロック While 処理コマンドを定義する 221
データ項目を更新するには 222
項目更新処理コマンドを使用する 222
項目の値を処理コマンドの実行条件に設定するには 223
処理コマンドの条件式を入力する 223
バッチプログラムで処理されたレコードの連番を取得するには 224
Counter() 関数を使用する 224
オンラインタスクで最初のレコードの場合のみ実行する条件を設定するには 225
IsFirstRecordCycle() 関数を使用する 225
特定のコントロールにカーソルを移動させるには 226
CtrlGoto() 関数を使用する 226
コントロールに入ったり抜けたりした場合に実行するロジックを作成するには 227
コントロール前を使用する 227
コントロール後を使用する 227
カーソルを一定の方向に移動させた場合のみ処理を実行させるには 228
順番にカーソルを移動したり、特定のコントロールをスキップした場合のみ処理を実行させるには 229
コントロール前を使用する 229
コントロール上（エディット / リッチエディット / 複数選択リストボックス）にフォーカスが残っている状態で新規入力データを検索するには 230
EditGet() 関数を使用する 230
イベントが発行されたコントロールを識別するには 231
HandledCtrl() 関数を使用する 231
特定のコントロールにパークしている時にのみ実行されるイベントロジックユニットを定義するには 232
イベントのコントロール名特性を使用する 232

第 12 章 :*日付と時刻*

現在の日付を取得するには 233
日付の格納形式 233
Date() 関数を使用する 233
現在の時刻を取得するには 234
Time() 関数を使用する 234

現在の時間をミリ秒単位で取得するには 235
mTime() 関数を使用してタイムスタンプを作成する 235
日付データの計算を行うには 236
時刻データの計算を行うには 237
日付と時刻を組み合わせたデータを加算するには 238
AddDateTime() 関数を使用する 238
項目の指定方法について 238
２つの日付 / 時刻データの差分を計算するには 239
DifDateTime() 関数を使用する 239
項目の指定方法について 239
指定された日付に対応する月の最初の日付を取得するには 240
BOM() 関数を使用する 240
指定された日付に対応する年の最初の日付を取得するには 241
BOY() 関数を使用する 241
指定された日付に対応する月の最後の日付を取得するには 242
EOM() 関数を使用する 242
指定された日付に対応する年の最後の日付を取得するには 243
EOY() 関数を使用する 243
指定された日付に対応する曜日名を取得するには 244
CDOW() 関数を使用する 244
指定された日付に対応する月名を取得するには 245
CMonth() 関数を使用する 245
指定された日付の年 / 月 / 日の各部分を取得するには 246
指定された時刻の時 / 分 / 秒の各部分を取得するには 247
指定された日付の週単位の日数を取得するには 248
DOW() 関数を使用する 248
日付データを文字列に変換するには 249
日付書式 249
DStr() 関数を使用する 250
文字列で格納された日付を日付データに変換するには 251

第 13 章 :*GUI の処理*

マウスを使用した場合のみコントロールへのパークを可能にするには 253
TAB で移動特性を設定する 253
マウスカーソルのアイコンを変更するには 254
SetCrsr() 関数を使用する 254
ドラッグ & ドロップでデータをコピーするには 255
ドラッグ & ドロップを使用して外部アプリケーションにデータを移動するには 256
ドラッグされたファイルからファイル名を取得するには 257
外部アプリケーションから Magic にデータをドラッグするには 258
Magic から Excel にテーブルのデータをドラッグするには 259

外部アプリケーションがデータを取得できないように Magic 内だけでドラッグ＆ドロップを許可するには 260

複数のコントロールを一緒にドラッグするには 261

複数のコントロールをドラッグできるようにする 261

データソース間での複数のレコードをドラッグするには 262

ユーザインタフェースの作成 262

２つのテーブルを使用したプログラムを作成する 262

２つのロジックユニットを定義する 262

ドラッグ開始イベントのロジックユニットを定義する 262

ドロップイベントのロジックユニットを定義する 262

２つのリンクコマンドを定義する 263

チョイスコントロールで定義されている値と異なるテキストを表示するには 264

チョイスコントロールに特殊文字を表示するには 265

フォームエディタ上でチョイスコントロールオプションを切り替えるには 266

アクセラレータをチョイスコントロールオプションに設定するには 267

コンボボックスで表示されるドロップダウンリストの長さを制限するには 268

タブコントロールの異なるタブにコントロールを関連付けるには 269

１つのタブでコントロールを表示させる 269

すべてのタブでコントロールを表示させる 270

タブコントロールの指定されたタブにタスクを関連付けるには 271

垂直タブコントロールを定義するには 272

タブコントロールの各タブにイメージを割り当てるには 273

ラジオボタンを定義するには 274

1 つのコントロール内にボタンを含める 274

複数のコントロール内にボタンを含める 275

チョイスコントロールにデフォルトオプションを設定するには 276

チョイスコントロールの選択肢を動的に設定するには 277

動的な選択肢を設定する 277

データソースの内容を選択肢にするには 278

データベーステーブルにリンクしたチョイスコントロールを作成する 278

データソースにリンクされたチョイスコントロールに追加オプションを結合するには 279

プッシュボタンとロジックを組み合わせるには 280

ユーザイベントを作成する 280

フォームにプッシュボタンを配置する 281

イベントが発行されると実行されるロジックユニットを作成する 281

プッシュボタンにキーボード操作を許可させるには 283

パーク可能なプッシュボタンを作成する 283

プッシュボタンがクリックされたときに実行される検証ロジックをスキップするには 284

イメージボタンを作成するには 285

イメージボタンを作成する 285

イメージとテキストを組み合わせてボタンを表示するには 286

テキスト上のイメージボタンを定義する 286

パーク可能なプッシュボタン上にテキストを指定するには 287

プッシュボタンのテキストを指定するために代入を使用する 287
プッシュボタンのテキストを指定するためにデフォルト値を使用する 288
プッシュボタンのテキストを指定するために書式特性を使用する 288
サブフォームやその親タスクに影響するプッシュボタンを定義するには 289
実行元特性をフォーカス上のタスクに設定する 290
アクセラレータをプッシュボタンに設定するには 291
動的なリッチテキストを作成するには 292
複数選択のリストボックスからデータを取得するには 293
複数選択のリストボックスを使用する 294

第 14 章 :*XML*

最初から XML ドキュメントを作成するには 295
XML ドキュメントを初期設定する 295
XML スキーマを見つけるには 296
XML スキーマ 296
XML ビューを作成するには 298
XML ビューを作成する 298
ノード ID と親 ID を更新するには 300
XML ファイル内の合成要素にアクセスするには 301
繰り返し要素 301
合成要素を選択する 301
既存の XML ドキュメントを修正するには 302
親レコードにアクセスする 302
子レコードにアクセスする 302
XML データ型に対応する Magic のデータ型を決定するには 304
スキーマの設定を表示する 305
任意の順番で XML ドキュメントからデータを取得するには 306
XML ビューのために代替インデックスを作成する 306
XML ドキュメント内のマルチ発生要素を処理するには 307
繰り返し要素を表示する 307
同じ XML ドキュメントを異なるユーザで共有させるには 309
XML ドキュメントにアクセスモードを設定する 309
同じスキーマを使用した異なる XML ドキュメントを作成するには 310
Magic のデータ項目に格納された XML ドキュメントにアクセスするには 311
データソースとして BLOB を使用する 311
入出力ファイルとして BLOB 項目を使用する 311
XML ドキュメントを検証するには 313
XMLValidate() 関数を使用する 313
XML ドキュメントの検証エラーを取得するには 314
XML にアクセスしている間に発生したエラーを処理するには 315
グローバルなエラーハンドラを作成する 315
ドキュメント対するエンコードを指定するには 316
XMLGetEncoding() 関数を使用する 316
Base64 でエンコードされた XML データを処理するには 317

XML ドキュメントをパラメータとして渡すには 318
スキーマを使用しないで XML ドキュメントを処理するには 319
XML ドキュメント内のデータの取得、更新、挿入を行うには 320
XML データを取得する 320
XML データを更新する 320
XML データを挿入する 320
XML データを削除する 321
XML ドキュメント内のデータ要素と階層構造を識別するには 322
XML ドキュメント内の混在内容を扱うには 323
XML ビューの存在を確認するには 324

第 15 章 :*COM*

使用する COM オブジェクトを定義するには 325
COM オブジェクトを定義する 325
同じタイプの複数の COM オブジェクトに対して 1 つのイベントハンドラを設定するには 327
クラスによるハンドラを設定する 327
下位タスクが実行中に、COM オブジェクトに対するイベントを処理できるようにするには 328
COM オブジェクトのメソッドを呼び出すには 329
COM メソッドを呼び出す 329
COM オブジェクトのプロパティを設定 / 取得するには 331
COM オブジェクトでプロパティの設定 / 取得オプションを使用する 331
COM オブジェクトの参照先を変更するには 333
COM ライブラリの参照先を変更する 333
列挙型パラメータ値を設定するには 334
Variant 型に対してデータの設定 / 取得を行うには 335
Variant を作成する 335
Variant からデータを取得する 336
COM オブジェクトの Variant 値のタイプと対応する Magic データ型を決定するには 338
VariantAttr() 関数を使用してデータ型を取得する 338
VariantType() 関数を使用してデータタイプを取得する 339
COM オブジェクトから配列値を取り出したり設定したりするには 340
COM オブジェクトに配列を渡す 340
COM オブジェクト内のコレクションを処理するには 342
コレクションを取り出す 343
COM オブジェクト間でパラメータとして COM オブジェクトの受け渡しを行うには 345
COM オブジェクトを受け取る 345
COM オブジェクト定義を再利用するには 346
プログラム全体に渡って COM オブジェクトのインスタンスを保持するには 347
メインプログラム内で COM オブジェクトを定義する 347
COM オブジェクトによって起動されたイベントを受け取るには 348
COM イベントに対するイベントハンドラを作成する 348

COM オブジェクトによって発生したエラーを処理するには 349
フォーム上に表示しないで ActiveX コントロールを処理するには 350
不可視の ActiveX オブジェクトを使用する 350
COM メソッドと Magic のロジックを公開するには 351
COM インタフェースを作成する 351
公開するロジックのクラス ID を設定するには 353
Magic にローカルにアクセスする COM クライアントを設定するには 354
Magic ロジックを公開する際に COM のデータタイプを決定するには 355
メソッドパラメータの詳細を表示 / 修正する 355

第 16 章 :*コンポーネント*

プロジェクト間で Magic のオブジェクトを再利用するには 357
Magic コンポーネントを作成する 357
オブジェクトを公開するには 361
プロジェクトにコンポーネントを読み込むには 362
Magic コンポーネントを使用する 362
ホストアプリケーションで使用している既存のコンポーネントの変更を有効にするには 363
実行環境でコンポーネントを変更する 363
開発環境でコンポーネントを変更する 363
コンポーネント内のオブジェクトの名前を変更する 363
コンポーネント用のヘルプファイルを提供するには 364
ヘルプファイルを組み込む 364
コンポーネントのオブジェクトに関する詳細情報を参照するには 366
コンポーネントが存在するディレクトリにアクセスするには 367
現在実行中のアプリケーションがコンポーネントかどうかを確認するには 368
コンポーネントとして定義されない別のアプリケーション内のプログラムを呼び出すには 369
コールリモートを使用する 369
コール公開名を使用する 370
アプリケーション間で再帰呼び出しを処理するには 371
コンポーネントの動作環境を組み込むには 372
コンポーネントへのアクセスを最適化するには 374
コンポーネント用プロジェクトを一括管理するには 375
モジュールとしてプロジェクトを登録する 375
登録したモジュールを開く 375
登録したモジュールを削除する 375

第 17 章 :*動作環境*

Windows のログイン ID を使用して Magic にログオンするには 377
Magic にログインするために Windows のユーザ ID を使用する 377
Magic Studio 起動時に自動的にプロジェクトをオープンするようにするには 379

デフォルトプロジェクトを設定する ... 379
サーバアプリケーションをテストする ... 379

SDI アプリケーションとして実行するように指定するには ... 381
SDI アプリケーションを作成する ... 381

SDI プログラムとして実行させるには ... 382
SDI コンテキストを定義する ... 382
SDI コンテキストを修正する ... 383

バッチタスクでの画面の再表示間隔を指定するには ... 384
バッチイベント間隔を指定する ... 384

エンジンが非同期イベントのチェックを行う間隔を指定するには ... 385

ISAM ファイルのトランザクションを組み込むには ... 386
ISAM トランザクションを設定する ... 386
トランザクション内の ISAM の強制ロック ... 386

Magic のロックファイルの位置を指定するには ... 387

入出力ファイルが実際に使用されるまで作成されないようにするには ... 388

実行時のイメージ処理を最適化するには ... 389
イメージキャッシュサイズ ... 389
イメージキャッシュ ... 389

表示されたチェックメッセージを制御するには ... 390
最小チェックレベル ... 390
メッセージのグループ化 ... 390
チェック項目の自動位置付 ... 391
チェックメッセージの表示 ... 391

プロジェクトファイルをコンポーネントとしてアプリケーションを開発するには . 392
プロジェクトファイルをコンポーネントとして使用する ... 393

ANSI を Unicode に変換するためのコードページを指定するには ... 394

第 18 章 : *データソースの定義*

Magic でデータベーステーブルを作成するには ... 395
1. ゲートウェイと DBMS がロードされることを確認する ... 396
2. データベース定義を設定する ... 396
3. テーブルを作成する ... 398
4. カラムを作成する ... 399
5. インデックスを作成する ... 400
6. テーブルの構文チェックを実行する ... 400
7. レコードを作成してテーブルを確認する ... 400

既存のデータベーステーブルにアクセスするには ... 402
既存のデータベーステーブルにアクセスする ... 402

データベースのビューからデータを取得するには ... 404

データベーステーブル定義を変更するには ... 405
Magic 内でのみ定義されているデータベーステーブルを変更する ... 405
外部テーブルと共有している Magic のデータソースを変更する ... 405

テーブル内のデータを暗号化するには ... 406
データベーステーブルを暗号化する ... 406

シークレット名を使用するには ... 407
シークレット名を設定する ... 407
シークレット名を使用する ... 407

Magic の項目とデータベースカラム間の割付を定義するには 408
デフォルト /NULL セクション 408
格納セクション 409
SQL セクション 409
複数の DBMS で動作するように設定するには 410
移行可能な項目を定義する 410
移行可能な WHERE 句を定義する 410
永続性のないデータ用のテーブルを定義するには 411
メモリテーブルを作成する 411
決められた順番でデータベーステーブルからレコードを取得するには 412
インデックスを指定する 412
メインソースに対する並び順を指定する 412
リンクテーブルの並び順を指定する 413
動的なインデックスを作成する 413
データベーステーブルを参照するには 414
参照プログラムを作成する 414
データベーステーブルからテキストにデータを出力するには 415
テキスト出力プログラムを作成する 415
テキストファイルのデータをデータベーステーブルに入力するには 416
テキスト入力プログラムを作成する 416
データベーステーブルから一度にデータを取り込むには 417
常駐特性の設定オプション 417
データベーステーブルからレコードを取得する際のデータサイズを指定するには ..418
DbSize() 関数を使用する 418
DbRecs() 関数を使用する 418
DbViewSize() 関数を使用する 419
動的にデータソース名を設定するには 420
定義された名前と異なるデータソース名を使用する 420
関数でデータソース名を指定する 420
データソース名に論理名を使用する 421

第 19 章 :*メインプログラム*

アプリケーションを初期設定するには 423
メインプログラムのデータビューエディタ 423
メインプログラムのロジックエディタ 424
メインプログラムのフォームエディタ 424
初期設定のプログラムを省略するには 425
アプリケーションの背景 / 壁紙を設定するには 426
メインプログラムに背景を定義する 426
グローバル変数の設定と使用を行うには（コンテキスト単位） 427
グローバル変数を使用する 427
アプリケーション起動時の手続き処理を実行するには 428
アプリケーション終了時の手続き処理を実行するには 429

第 20 章 :*データビュー*

タスクのデータビュー内のレコード数を取得するには 431
DbViewSize() 関数を使用する 431
タスクのメインソースを定義するには 433
メインソースを入力する 433
タスクで処理されるレコードの順番を動的に設定するには 434
インデックス番号を指定する 434
インデックス式を使用する 435
タスクのソートテーブルを使用する 436
プログラムに受け渡しするパラメータを定義するには 437
パラメータ項目を定義する 437
戻り値を定義する 438
タスクのデータ項目を定義するには 439
カラムを定義する 440
変数項目を定義する 440
タスクのデータビューに範囲を定義するには 442
範囲の最大 / 最小を使用する 442
範囲指定を異なる場所から確認する 443
範囲式を使用する 445
SQL Where 句を使用する 446
タスク起動時に特定のレコードに位置付けるには 447
位置付特性を使用する 447
位置付の最小 / 最大の両方を使用する 448
位置付式を使用する 448
テーブルの中央に位置付ける 448
読み込み専用のテーブルにアクセスするには 449
データソースを読み込み専用でオープンする 449
データソースを複数のユーザ間でアクセスさせるには 450
タスクのデータ項目の値を設定するには 451
自動的に値を設定する 451
デフォルト値特性を使用する 451
代入特性を使用する 451
タスクで項目を更新する 452
項目処理コマンドを使用する 452
VARSet() 関数を使用する 452
ユーザによって項目を更新する 452
1 つのタスク内で複数のテーブルからデータを検索するには 453
リンクコマンドを使用する 453
リンクの位置付カラムを設定する 454
リンクの種類を設定する 455
リンクの戻り値特性を設定する 456
リンク条件を設定する 456
リンクカラムの範囲を使用する 456
テーブルの最初または最後のレコードを取得するには 458
メインソースの最初または最後のレコードの取得 458
1. インデックスを定義する 458
2. 検索順を設定する 458
3. サイクル数をチェックする 459
リンクテーブルの最初または最後のレコードの取得 460
1. インデックスを定義する 460
2. 検索方向を設定する 460

第 21 章 : *式*

式エディタ内で式の体裁を整えるには 461
式を改行入力する 462
関数名の入力を簡単に行うには 463
オートコンプリート機能を使用する 463
式の拡張表示で色付けされる要素の色を設定するには 464
式で使用する色を設定する 464
関数の関連ヘルプを表示させるには 466
任意の場所から関数のヘルプトピックを探す 466
式エディタでヘルプを使用する 466
関数一覧からヘルプを表示する 466
長い式の編集を容易にするために式の入力行を拡張するには 467
広域モードの式をチェックするには 467
文字型項目内で改行データを入力するには 467
タスクエディタで直接に式を編集するには 468
クイック式エディタを使用する 468
式エディタから項目一覧に移動するには 469
項目一覧からデータ項目を選択する 469
式エディタから関数一覧を開くには 470
関数一覧から関数を選択する 470
式がどこで使用されているかを調べるには 472
式エディタ上でクロスリファレンスを使用する 472
式の中で別の式を再利用するには 473
ユーザ定義関数を使用する 473
ExpCalc() 関数を使用する 474
実行時に式の構文の確認と評価を行うには 475
EvalStr 関数を使用する 475
EvalStrInfo 関数を使用する 476
既存の式を複写するには 477
@Line を使用する 477
使用していない式を検索するには 478
未使用の式を検索する 478
すべての未使用の式を削除するには 479
式の中で文字列を定義するには 480
簡単な式が重複して定義されることを防ぐには 481

第 22 章 : *レポート*

タスクのデータビューを外部ファイルに出力するには 483
動的にデータを出力させるには 484
データ表示プログラムを作成する 484
データ出力ウィザードを使用する 484
簡易参照用プログラムを作成するには 486
簡単な参照プログラムを作成する 486

現在のビューをテキスト出力するには 487

データをテキストファイルとして出力するには 488

DataViewToText() 関数を使用する 488

データを CSV ファイルとして出力するには 490

データを HTML ファイルとして出力するには 491

DataViewToHTML() 関数を使用する 491
HTML テンプレートファイルを使用する 492

データを XML ファイルとして出力するには 494

DataViewToXML() 関数を使用する 494
スキーマファイルを作成する 495
XML テンプレートファイルを使用する 495

帳票を作成するには 496

Magic で帳票を作成する 496

1. “ 起動画面 ” を作成する 496
2. 簡単なテキスト出力プログラムを作成する 497
3. ソート順序と範囲を確認する 497
4. バッチタスクを定義する 498
5. 入出力デバイスを設定する 498
6. フォームを作成する 499
7. フォームを編集する 500
8. 明細フォームの出力処理を定義する 501
9. ヘッダを出力する 502
フッタを出力する 502

ページヘッダ / フッタの情報を定義するには 503

グローバルなページヘッダ / フッタを定義するには 504

グローバルフォームを作成し使用する 504
グローバルなページヘッダ / フッタ上でデータ項目を扱う 504

帳票ページを列挙するには 506

1. ２つの入出力ファイルを定義する 506
2. タスクの呼び出し処理を定義する 507
3. タスク前で出力項目を初期化する 507
4. ページヘッダイベントのロジックユニットを作成する 507
5. 最初の処理で取得されたページ総数を格納する 508
6. ２つの入出力ファイルに出力する 508

複数のプログラムで同じ入出力ファイルを使用して出力するには 509

プログラムの呼び出し処理を設定する 509
起動されるプログラムを設定する 510

帳票のブレイクレベルを作成するには 511

1. ブレイクレベルに対応したレコードの並び順を設定する 511
2. ブレイクの発生対象となるデータ項目を決定する 511
3. グループレベルを設定する 512
4. 合計値の計算を行う 512

ブレイクレベル毎の総計を定義するには 513

1. 総計用項目を定義する 513
2. 総計用項目を更新する 513
3. 総計用項目を初期化する 513

帳票に複数行のコントロールを定義するには 514

1. コントロール項目特性を変更する 514
2. フォーム特性を変更する 514

帳票のテーブルに対して見出しを繰り返し出力させるには 516

PDF ドキュメントを作成するには 517

ユーザが出力先を選択する際に PDF の設定を行う 517

デフォルトの印刷プレビューアーとして PDF を設定する ... 518
バッチジョブ用に PDF を設定する ... 518

ページ毎に計算処理を実行させるには ... 520

帳票にマージンを設定するには ... 521

プリンタテーブルに設定する ... 521
スタイル設定ユーティリティを使用する ... 521
全てのキューで同じマージンを設定する ... 521

第 23 章 : マージ

テキストファイルにデータをマージするには ... 523

1. テキストのテンプレートを作成する ... 523
2. 出力ファイルを指定する ... 524
3. マージフォームを作成する ... 524
4. マージフォームの特性を設定する ... 525
5. タグを選択する ... 525
6. 各タグをデータと関連付ける ... 526
7. マージフォームを出力する ... 526

動的に Word ドキュメントを作成するには ... 528

動的に Excel ドキュメントを作成するには ... 530

テンプレートに繰り返し可能なデータを挿入するには ... 531

HTML テンプレートのテーブルに繰り返し可能なデータを挿入するには ... 532

テンプレートへのデータ挿入を調整するには ... 533

条件を指定する ... 533

テンプレートに置き換え可能なトークンを設定するには ... 534

HTML タグとマージのトークンを区別するには ... 535

好みの HTML エディタを利用するように設定するには ... 536

複数のタスクで 1 つのドキュメントにデータをマージするには ... 537

同じプログラム内の２つのタスクからのデータをマージする ... 537
２つの異なるプログラムのデータをマージする ... 537

テンプレートを使用して、データをグループ化して出力するには ... 539

1. テンプレートを設定する ... 540
2. 詳細行を書き込む ... 540
3. ヘッダを書き込む ... 541

テンプレートにファイルを埋め込むには ... 542

マージ Web アプリケーションを作成するには ... 543

基本的なマージ Web アプリケーション ... 543

表示するデータソースの内容 ... 543
HTML ファイルを作成する ... 543
マージプログラムを作成する ... 544
タスクを定義する ... 544
マージプログラムを実行する ... 544

Cookie を使用するには ... 546

Cookie の書き込み ... 546

HTML ファイルの先頭に 書き込み処理を記述する ... 546
META タグを使用する ... 546
JavaScript を使用する ... 546
RqHTTPHeader 関数を使用する ... 547

Cookie の読み込み ... 547

GetParam 関数を使用する 547
HTTP ヘッダ情報から取り出す 547
セッション管理を行うには 548
セッション管理の必要性 548
セッション管理の方法 548
セッション ID を生成する 548
セッション ID を実装する 548
セッション ID を保存する 548
コンテキスト管理を行うには 549
コンテキスト管理の有効化 549
タスク特性 549
コンテキスト ID の取得 549
プログラム起動時の URL 549
コンテキスト管理プログラム 549
クライアントの認証 549
HTML 環境変数 550
認証項目を指定する 550

第 24 章 :*メッセージング*

メッセージを MSMQ に送るには 551
Open/Send/Close を使用する 551
1. メッセージキューをオープンする 551
2. メッセージを送信する 552
3. キューをクローズする 553
Quick Send を使用する 553
MSMQ メッセージを受信するには 554
1. メッセージキューをオープンする 554
2. メッセージを読み込む 554
3. キューをクローズする 555
送信メッセージにラベルを設定するには 556
MSMQ のラベル特性を設定する 556
テーブルをコンポーネントに送る 556
送信メッセージにプライオリティを設定するには 557
MSMQ のプライオリティ特性を設定する 557
テーブルをコンポーネントに送る 557
MSMQ のパブリックキューにアクセスするには 558
メッセージが読み込まれたことを確認するには 559
MSMQ の Ack 特性を設定する 559
テーブルをコンポーネントに送る 559
外部メッセージテーブルをセットアッププログラム に送るには 560
メッセージングのエラーを捕捉するには 561
MSMQ アプリケーションをデバッグするには 562
Windows 上でメッセージを確認する 562
PC に MSMQ を設定するには 563
MSMQ をインストールする 563
キューを追加する 564
メッセージングコンポーネントをインストールする 564

第 25 章 :*マルチタスク*

複数の対話型タスクを同時に実行させるには 565
タスクを並行に実行させる 566
並行処理のバッチプログラムを実行させるには 567
バッチタスクを並行に実行させる 567
並行実行しているタスク間の通信を管理するには 568
実行中のコンテキストのリストを取得するには 570
現在のコンテキスト名を取得するには 571
現在のコンテキストに異なる名前を設定するには 572
任意のコンテキストにフォーカスを移すには 573
呼び出された並列プログラムのコンテキスト ID を取得するには 574
コンテキスト間で項目を共有するには 575
SharedValSet() 関数を使用する 575
SharedValGet() 関数を使用する 575
コンテキスト間でメモリテーブルを共有するには 576
並行タスクの初期設定をコントロールするには 577
メインプログラムの初期化 577
グローバル変数の複写 577
実行プログラムの単一インスタンスを強制するには 578
単一インスタンスプログラムの再起動を識別するには 579

第 26 章 :*ウィンドウインタフェース*

エンドユーザがカスタマイズしたフォーム状態を保持するには 581
フォーム状態 ID 特性を使用する 581
ウィンドウにアイコンを設定するには 582
動的にウィンドウタイトルを設定するには 583
フォーム名特性を式で設定する 583
ユーザがウィンドウのサイズ変更をできないようにするには 584
サイズ変更を防止する 584
ウィンドウの最小サイズを設定するには 585
最小サイズを設定する 585
ウィンドウタイトルを表示させないようにするには 586

第 27 章 :*メニュー*

メニューにショートカットキーを定義するには 587
内部イベント以外に対してショートカットキーを割り当てる 587
内部イベントに対してショートカットキーを割り当てる 587
実行プログラムのメニューを変更するには 589
メニュー表示の変更 589

サブメニューの追加 **589**
MnuAdd() 関数を使用する 589
MnuRemove() 関数を使用する 591
MnuReset() 関数を使用する 591
SDI プログラムのメニュー変更 **592**
キーボードでウィンドウの切り替えを有効にするには **593**
ウィンドウメニューを使用する 593
メニューの表示／非表示を行うには **595**
MnuShow() 関数を使用する 595
メニューバーを削除するには **596**
メニューにアイコンを追加するには **597**
メニューアイコンを指定する 597
独自のアイコンを指定する 598
ツールバーにアイコンを追加するには **599**

第 28 章 :*Unicode*

多言語データをサポートできるようにするには **601**
Unicode フォント **601**
Unicode 型 **602**
Unicode のテキストファイルの作成 / 読み込みを行うには **603**
ANSI と Unicode 間でデータを変換するには **604**
Unicode コードを ANSI に変換する 604
Unicode 文字の文字コードを取得するには **605**
Unicode の文字コードを文字に変換するには **606**

第 29 章 :*アプリケーションのデバッグ*

デバッガを使用してデバッグするには **607**
1. デバッグモードを有効にする 607
2. モニタ用ウィンドウを開く 607
3. ブレイクポイントとウォッチ設定する 607
4. プログラムやプロジェクトを実行する 608
5. プロジェクトをステップ実行する 608
並行プログラムの実行中にデバッガを使用するには **609**
アプリケーションのブレイクポイントを設定するには **610**
処理コマンドでの設定 **610**
ブレイクポイントを設定する 610
ブレイクポイントを切り替える 610
ブレイクポイント特性を設定する 611
ソースに移動する 611
データビューエディタでの設定 **612**
ブレイクポイントを設定する 612
項目にウォッチに設定する 612
ブレイクの使用 **613**
デバッガによって記録される情報を制御するには **614**
Logging() 関数を使用する 614
パフォーマンスの問題 614

デバッグ中にデータを操作するには 615
コンポーネントをデバッグするには 616
データベースの動作を記録するには 617
ログレベル 617
DBMS のロギング 617
パフォーマンスの問題 617
リモートアプリケーションをデバッグするには 618
リモートデバッガを有効にする 618
リモートアプリケーションに接続する 618
リモートアプリケーションを切断する 619

第 30 章 :*イベントとハンドラ*

イベントによって起動された選択プログラムの戻り値でデータ項目を再表示するには 621
問題点 : 項目が再表示されない 621
解決方法 : 621
解決方法 1: イベントの強制終了特性を使用する 622
解決方法 2: 選択プログラム特性を使用する 622
解決方法 3: コンボボックスを使用する 623
確定されていない更新値で強制的に更新するには 625
同一イベントで複数のハンドラを実行させるには 626
イベントの伝播を許可する 626
ユーザハンドラによって実行された内部イベントに Magic の処理を定義するには ..627
イベントが定義されたタスクでのみ有効にするには 628
内部イベントを処理しないようにするには 629
条件付きの伝播特性を使用する 630
イベントの処理を特定のコントロールに限定するには 631
コンポーネント内に定義されたハンドラを使用して、ホストアプリケーションでイベントを処理するには 632
コンポーネント内でイベントを定義する 632
ホストアプリケーションでイベントを発行する 633
コンポーネント内でイベントを処理する 633
コンポーネントからホストアプリケーションで定義されたイベント実行するには ..634
ホストイベントを発行する 634
ホストアプリケーションでイベントを処理する 634
イベントを即時に処理させるには 635
イベントの処理を延期させるには 636
イベント名を動的に指定してイベントを発行するには 637
メインプログラム内でイベントを設定する 637
イベントを公開名で発行する 637
一定時間の経過後に処理を実行させるは 638
タイマーイベントを使用する 638
条件が満たされた場合に処理が実行されるようにするには 639
式イベントを作成する 639
パラメータ付きのイベントを作成する 639
イベントハンドラでパラメータを受け取るようにする 640

イベントを発行する 640
ユーザイベント選択時にイベントを登録するには 641
ユーザイベントを追加する 641

第 31 章 :*セキュリティ*

データの暗号化 / 復号化を行うには 643
Cipher() 関数を使用する 643
Decipher() 関数を使用する 644
サポートされる暗号化モード 645
テーブルのデータを暗号化するには 646
データベーステーブルを暗号化する 646
データベースのログイン情報を非表示にするには 647
シークレット名を設定する 647
シークレット名を使用する 647
アプリケーションの管理者権利を定義するには 648
セキュリティファイルを削除する 648
特定プログラムの実行を制限するには 649
ログオンユーザにもとづいてメニューをカスタマイズするには 650
メニュー項目に権利を設定する 650
ログオンユーザにもとづいて機能を制限するには 651
Rights() 関数を使用する 651
権利を持たないユーザに対してコントロールを非表示にする 652
権利を持たないユーザに対してロジックの実行を防止する 653
アプリケーション全体のアクセスを制限するには 654
セキュリティキーを使用する 655
権利グループを定義するには 656
1. 権利を設定する 656
2. グループを設定する 657
3. ユーザを定義する 658
ActiveDirectory サーバを使用して認証させるには 659
ActiveDirectory 認証を使用する 659
環境設定を行う 659
クライアント PC をドメインにログオンする 659
LDAP 認証を使用する 660
環境設定を行う 660
Magic でログオンする 661
自動ログオンを設定する 661
ユーザログオン名を使用する 661
OpenLDAP サーバを使用して認証させるには 663
環境設定を行う 663
Magic でログオンする 664
ディレクトリのデータ検索を行う 664

第 32 章 :*ユーティリティ*

データソースを参照するには 665
データソースを参照する簡単なプログラムを作成するには 667
参照プログラムを作成する 667

プログラムの構文チェックを行うには 669
1 つのプログラムの構文をチェックする 669
複数のプログラムを一度にチェックする 670
データソースの構造を検証するには 671
１つのデータソースを構文チェックする 671
複数のデータソースを一度にチェックする 671
構文チェックの表示メッセージを絞るには 672
構文チェックの最小レベルを設定する 672
メッセージのレベルを設定する 673
チェック結果を使用するには 674
チェック結果のリストを使用する 674
キーボード操作で移動する 674
プロジェクトをバックアップするには 675
OS 上のファイルとしてバックアップする 675
リポジトリ出力ファイルを作成する 676
プログラムをコピーする 676
オブジェクト内のテキストを検索 / 置換するには 677
テキストを検索する 677
テキストを置換する 678
検索結果を保存／印刷するには 679
検索結果を保存する 679
検索結果を印刷する 679
Magic とデータベース間でのテーブル構造の不整合に対応するには 680
Magic で自動的にテーブルを変換できるようにする 680
互換性をチェックする 681
外部ツールを Studio に追加するには 682
外部ツールを追加する 683
自動的に外部プロセスを実行するには 684
コマンドファイルを作成する 684
自動的に実行するコマンドファイルを設定する 684
バッチ内に自動処理を設定する 684
Magic ライセンスの使用状況を確認するには 685
Magic Studio/Magic Client の場合 685
Magic EnterpriseServer の場合 685

第 33 章 :*開発環境*

特性シートやナビゲータペインが自動的に表示されないようにするには 687
複数のペインを一つに統合するには 688
結合されたペインを分離する 688
使用するフォントや色を変更するには 689
開発用基本色を変更する 690
開発用フォントを変更する 691
Magic Studio 起動時に、自動的にプロジェクトを読み込ませるには 692
プロジェクトのソースファイルの格納場所を変更するには 693
既存の .edp のソースディレクトリ定義を変更する 693
デフォルトのソースディレクトリを変更する 694
Magic.ini を指定して起動するには 695
異なる Magic.ini ファイルを使用する 695

Magc.ini の環境情報を追加指定するには 696
Magic.ini の設定をオーバライドする 696
オーバライドを使用する 696

第 34 章 :*Web サービス（コンシューマ）*

Web サービスにアクセスするには 697
Web サービスクライアントを作成する 697
Web サービスにアクセスする 698
Web サービス定義の再読込を行うには 699
既存のサービスを再読込する 699
複雑なパラメータの送受信を行うには 700
WSDL が複雑なパラメータを使用しているかどうかを確認する 700
BLOB 項目を定義する 700
XML BLOB の作成 / 読込を行う 701
パラメータと BLOB 項目を使用する 702
Web サービスに安全にアクセスするには 703
サービスにセキュリティを設定する 703
タスクでセキュリティを設定する 704
Web サービスアタッチメントを使用するには 705
添付ファイルを Web サービスプロバイダに送信する 705
Web サービスプロバイダからの受信データから添付ファイルを取り出す 705
Web サービスへの呼び出しをトレースするには 706
SOAP Spy を起動する 706
SOAP Spy のトレースを開始する 706
Magic 側で SOAP Spy を有効にする 706

第 35 章 :*Web サービス（プロバイダ）*

Magic で Web サービスを提供するには 709
Magic のセットアップ 709
Java の確認 709
Systinet の確認 710
Magic アプリケーションのデプロイ 710
1. バッチプログラムを作成する 711
2.Systinet server を起動する 711
3.Web サービスインタフェースビルダを起動する 712
4. サーバの詳細ダイアログ 713
5.Web サービスの詳細ダイアログ 713
6. プログラムの選択ダイアログ 714
7. 処理一覧ダイアログ 715
8. 確認ダイアログ 716
9. 終了 716
10. デプロイ結果を確認する 717
Web サービスのモジュールをデプロイするには 718
Web サービスをテストするには 719
アタッチメント付きの Web サービスを提供するには 721
パラメータとして添付ファイルを受け取る 721
関数で添付ファイルを受け取る 721
プロバイダとしての Web サービスから複雑なパラメータの送受信を行うには 723

SOAP リクエストを追跡するには 724
デプロイされたサービスに対するセキュリティを設定するには 725
ユーザとパスワードを定義する 725
サービスへのアクセスを許可する 727
サービスの認証方法を定義する 728

第 36 章 : データベース

データベースとの接続を定義するには 731
1. データベースゲートウェイの設定を行う 731
2. データベースを定義する 732
3. 接続を確認する 734
DBMS 別の設定方法 735
Oracle 735
DB2 UDB 736
ODBC 737
Pervasive ISAM 737
Magic からデータベーステーブルを作成するには 738
既存のデータベーステーブルまたはビューにアクセスするには 739
データベースに送信される SQL ステートメントを参照するには 740
Magic 内での SQL ステートメントのロギング機能を有効にする 740
独自の SQL ステートメントをデータベースに送るには 741
WHERE 句を手動で入力する 741
他の SQL ステートメントを手動で入力する 742
データベースエラーまたは例外を処理するには 743
エラーハンドラを作成する 743
エンドユーザのアクセスとデータ操作を制限するには 744
データベースのレコードの取得順を指定するには 745
テーブルインデックスを設定する 745
タスクソートを使用する 745
特定の SQL タイプにアクセスするには 747
読込専用データのアクセスを最小化するには 748
常駐テーブルの設定を変更する 748
常駐テーブルの内容を更新する 748
データベースのアクセス頻度を減らすには 750
読込専用テーブル 750
レコードのフェッチの制御 750
配列サイズ 750
キャッシュ 751
ビュー事前読込 751
範囲とダイレクト SQL の使用 751
データベースに送られる SELECT ステートメントを制御するには 752
複数のユーザが同じレコードを修正した場合の Magic の動作を決定するには 753
物理トランザクション使用時の修正行の識別 753
遅延トランザクション使用時の修正行の識別 753
データベース制約がある場合に、1 対多の関係を実現するには 756
ネストされたトランザクションを実現するには 757

トランザクションをオープンしないようにするには 758
明示的にトランザクションをロールバックするには 759
データベースオプティマイザの動作を制御するには 760
データベーストランザクションを初期化するには 761
トランザクションのタイプの定義 761
トランザクションツリー 761
トランザクションの開始と終了の場所の定義 762
関係するテーブルの定義 763

第 37 章 : *ブラウザクライアント*

よく利用する HTML エディタを設定するには 765
HTML エディタを設定する 766
アプリケーションに JavaScript 関数を実装するには 767
1 対多の関係を構築するには 768
1. ヘッダタスクを作成する 768
2. サブタスクを作成する 769
3. サブフォームコントロールを使用してヘッダとサブタスクを結合する 770
タスク / プログラムが呼び出された時に、新しいウィンドウが開かないようにするには 771
指定したフレームに表示するプログラムを呼び出す 771
同じウィンドウ内に異なるプログラムのインタフェースを表示するには 772
ブラウザウィンドウを再利用する 773
最上位のプログラムが終了する際にオープンする URL を指定するには 774
終了時に URL を表示する 774
終了時に Magic プログラムに移る 775
Magic の内部イベントにキーボードを割り当てるには 776
ロジックユニット内でキー操作を使用する 776
ユーザイベントでキー操作を使用する 776
サーバ側とクライアント側の処理コマンドと関数を区別するには 777
テーブルの繰り返し行を指定するには 778
ブラウザに渡されるレコードセット内のレコード数を設定するには 779
チャンクサイズを設定する 779
コントロールを既存のフォームに追加するには 780
新しいコントロールを追加する 780
スタイルとクラスを実装するには 781
スタイルシート 781
HTML クラスを使用する 782
Magic でクラスを使用する 782
Magic 内でスタイルをコード化する 783
他のコントロール特性を使用する 783
ページ上に ActiveX コントロールを実装するには 784
HTML ページに ActiveX コントロールを配置する 785
スクリプトを使用して ActiveX コントロールを動作させる 785
Magic からスクリプトを呼び出す 786
Magic 内のスクリプトからイベントを処理する 786
HTML ページ内の外部スクリプトからの Magic のロジックを起動するには 787

プログラムを外部から呼び出せるようにするには 788

第 38 章 :*ブローカ*

Magic がリクエストを受信できるように Web サーバ（IIS）を設定するには 789
1.Microsoft IIS をインストールする 789
2.Magic をインストールする 792
3.MRB が実行していることを確認する 792
4.Web サーバが実行していることを確認する 793
Magic がリクエストを受信できるように Apache Web サーバを設定するには 794
1.Apache サーバをインストールする 794
2.Magic のリクエスタがインストールされていることを確認する 794
3.Apache のディレクトリを設定する 795
4.MRB が実行していることを確認する 796
MRB の動作を監視するには 798
Magic のリクエスト関数 798
コマンドラインの情報 798
MRB のメインログ 799
Magic が同時に処理するリクエスト数を制限するには 800
MRB が自動的にアプリケーションを起動するようにするには 801
MRB のパスワードを定義するには 802
代替えの MRB を定義するには 803
Mgreq.ini で MRB を定義する 803
実行エンジンが MRB のリクエストを受けないようするするには 804
キュー内で待ち状態のリクエストを削除するには 805
ロードバランシングを実装するには 806

第 39 章 :*ソース管理*

バージョン管理ソフトによって管理されるプロジェクトを作成するには 807
CVS サーバへのログインユーザを指定する 808
既存のプロジェクトをバージョン管理データベースに追加するには 809
Source Safe の場合 809
CVS の場合 809
VC データベース内のプロジェクト 810
VC データベースに定義した場合の効果 810
プロジェクトをバージョン管理から削除するには 811
バージョン管理プロジェクトを新たにオープンするには 812
Source Safe の場合 812
CVS の場合 813
バージョン管理サーバが無効な場合に、バージョン管理プロジェクトを開発するには 814
オフラインの場合に変更される項目 814
VC への再接続 815
更新内容を追跡するには 816

モデルとデータソースをチェックアウトする ... 816
プログラムのチェックイン / チェックアウトを行う ... 816
自動的にプログラムリポジトリをチェックアウトする ... 816
手動でプログラムリポジトリをチェックアウトする ... 816
Source Safe での履歴 ... 817
CVS での履歴 ... 818
バージョン管理プロバイダを決めるには ... 819
オブジェクトを前の状態にロールバックするには ... 820
1 つのオブジェクトの最新バージョンを取得するには ... 820
プロジェクト全体をロールバックする ... 820
CVS を使用して開発するには ... 821
CVS コントロールパネル ... 821
ファイルの位置 ... 821
CVS の実行 ... 822

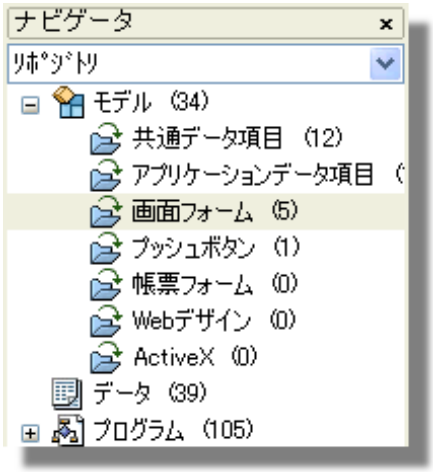
第 1 章： 操作方法とワークスペース

プロジェクトのオブジェクトを整理するには

Magic のプロジェクトは、大規模な基幹業務用のアプリケーションの場合があります。従って、開発内容を整理しておくことが重要になります。Magic は、開発内容をリポジトリ（モデル、データ、プログラム、ヘルプ、権利、メニューやリソース）に分けることでこのような作業を支援します。

右の図は、これらのリポジトリを示しています。

これらのリポジトリ内で、さらにフォルダを使用して入力項目を分類することができます。右の例では、異なるタイプのモデルを含んだモデルリポジトリ用に 7 つのフォルダが作成されています。



フォルダを作成する

1. 作成したいフォルダの上位にカーソルを置きます（この例では、画面フォーム）。
2. *F4* を押下します（または、コンテキストメニューから行作成を選択するか、プルダウンメニューから 編集 → 行作成 を選択）。
3. フォルダ名を入力します。

*ヒント：各入力項目の右側に表示される数値は、各セクション内の項目数を表しています。例えば、［モデル］セクション内には、現在は 34 のモデルが定義されています。また、この内 12 のモデルは「**共通データ項目**」フォルダ内に定義されています。*

*新しいフォルダを追加すると、フォルダ名として「**新しいフォルダ n**」が表示されます。ここでの数値（n）は、ただの連番を表しています。この例の場合、**新しいフォルダ**は、9 番目に追加されるフォルダとなります。*

フォルダを削除する

必要条件： フォルダを削除する前に、フォルダ内のオブジェクトが空になっている必要があります。フォルダ内のオブジェクトを別のフォルダに移動してください。また、バージョン管理を使用している場合、最初にこれらをチェックアウトする必要があります。

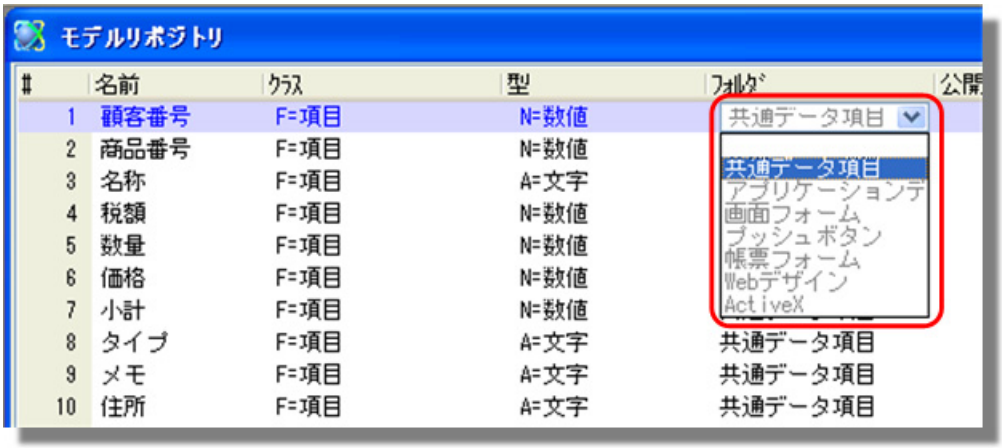
1. 削除したいフォルダ上にカーソルを置きます。
2. *F3* を押下します（または、コンテキストメニューから行削除を選択するか、プルダウンメニューから 編集 → 行削除 を選択）。
3. フォルダの削除の確認ダイアログが表示されたら、はいをクリックします。

フォルダを移動する

1. 移動したいフォルダをクリックします。その際、対応するワークスペースも開きます。
2. フォルダを移動したい位置にドラッグします。
3. フォルダをドロップすると、フォルダのドラッグ＆ドロップの確認ダイアログが表示されます。はいをクリックします。

フォルダにオブジェクトを移動する

既存のオブジェクトをフォルダ内に移動させるには、以下にあげる方法があります。



必要条件： バージョン管理を使用している場合、最初にリポジトリを**チェックアウト**してください。

フォルダカラムを使用してフォルダにオブジェクトを移動する

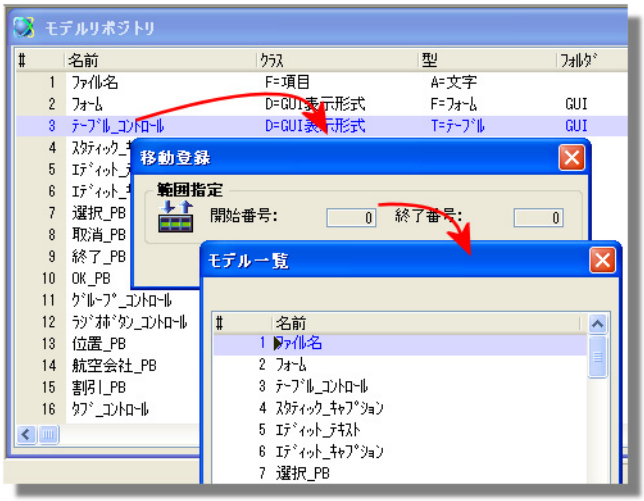
1. 移動したいオブジェクトを選択します。
2. **フォルダ**カラムで、コンボボックスを**クリック**します。
3. 移動するフォルダを選択します。

オブジェクトが現在のフォルダから消え、選択されたフォルダに移動されます。.

移動コマンドを使用して移動する

1. オブジェクトを移動させたいフォルダに移動します。
2. ***Ctrl+Shift+M*** を押下します。**移動登録**ダイアログが表示されます。
3. **開始番号**に移動したいオブジェクトグループの最初の項目を選択してください（**ズーム**して、一覧から選択することもできます）。
4. 1 項目のみ移動する場合は、ここで ***Enter*** を押します。
5. 複数の項目を移動する場合は、**終了番号**に移動したいオブジェクトグループの最後の項目を選択して ***Enter*** を押します。

オブジェクトは現在のカーソル位置に表示され、以前の位置からは削除されます。

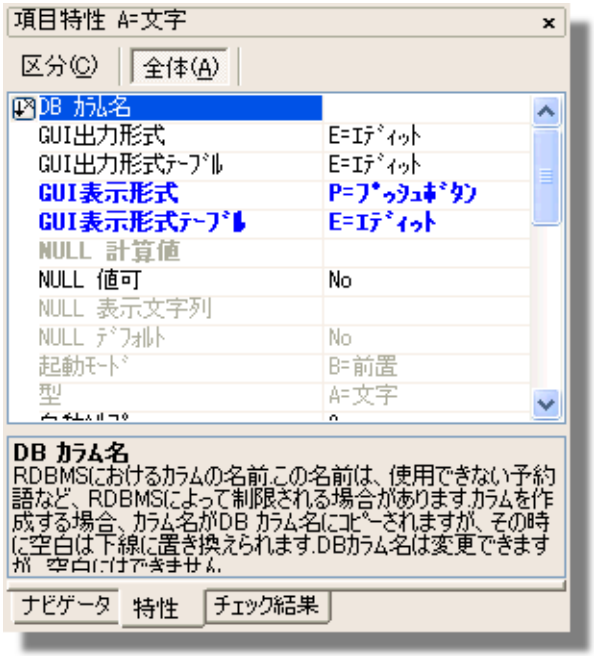


ヒント：*この方法は、連続した複数のオブジェクトを移動する場合に有効です。*

参照： 第 1 章：「入力行を移動するには」（11 ページ）

パレットを分離するには

開発エンジンのパレットは、サイズを変更したり、移動したり、Magic ウィンドウの縁に結合したりすることができ、作業内容に応じて柔軟に対応することができます。また、複数のパレットを 1 つのウィンドウに結合させることでスペースを節約することができます。1 つのパレットを別のパレット側に移動させることでパレットは結合されます。

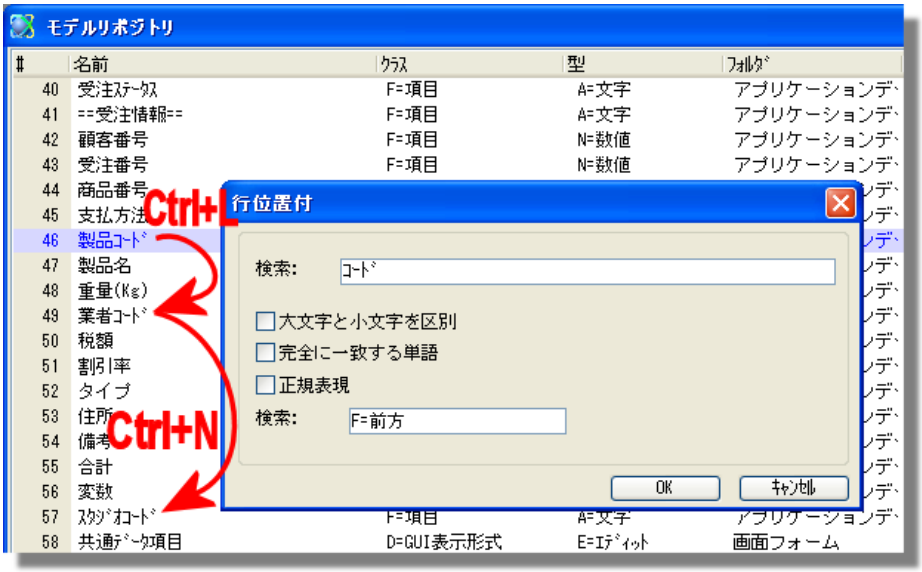


結合されたパレットを分離する

1. パレットにフォーカスを移します。
2. *Ctrl* を押下した状態で、パレットのタイトルバーを任意の方向にドラッグします。
3. *Ctrl* を離します。
4. ３つ以上のパレットが統合されている場合は、同じ操作を繰り返してください。

上記の操作でパレットは分離されます。

開発用テーブルの行位置付けを行うには



リポジトリまたは、プログラム内で特定の行を見つけたい場合があります。このような場合は、位置付け機能を利用することで実現することができます。位置付け機能には、いくつかのオプションがあり、これによって柔軟に対応することができます。デフォルトでは、現在の行から以降の行に対するテキストをもとに位置付けることができます。

例えば、上の例のように、N コードを指定することで、製品コードにもスタジオコードにも位置付けることができます。入力された文字列によって行内に設定されたいる文字列を検索します。**大文字と小文字を区別**というチェックボックスを**チェック**すると、文字列の大文字と小文字を別の文字として検索します。

正規表現のチェックボックスをチェックした場合、複雑なマスク指定を行うこともできます。『リファレンスヘルプ』には、正規表現で使用できる書式の説明があります。

行に位置付ける

1. ***Ctrl+L*** を押下します（または、**編集→クイックアクセス→位置付**）。
2. **行位置付**ダイアログボックスに検索したい文字列を入力します。
3. ***Enter*** を押下します（または、OK を**クリック**）。

入力した文字列をもとに検索処理が実行され、カーソルが最初の行に移動します。***Ctrl+N***（または、**編集→ クイックアクセス→次に位置付**）によって次の検索対象に位置付けられます。***Ctrl+Shift+N***（または、**編集→ クイックアクセス→前に位置付**）によって前の検索対象に位置付けられます。

ヒント :行位置付 *は、現在オープンされているテーブルの行に対して実行されます。このため、特定のフォルダ内で実行した場合は、そのフォルダ内の行のみを検索対象とします。リポジトリ全体を検索したい場合は、リポジトリ全体を開いた状態で実行してください。*

参照 : 「行番号を指定して移動するには」（5 ページ）

行番号を指定して移動するには

Magic 内の全ての項目は、連番を持っています。この番号は、覚えておく必要はありません。使用したい項目を見つける手段として**ズーム**や**位置付け機能**を利用することができます。しかし、行番号を知っていた場合、その番号の行に移動することができます。

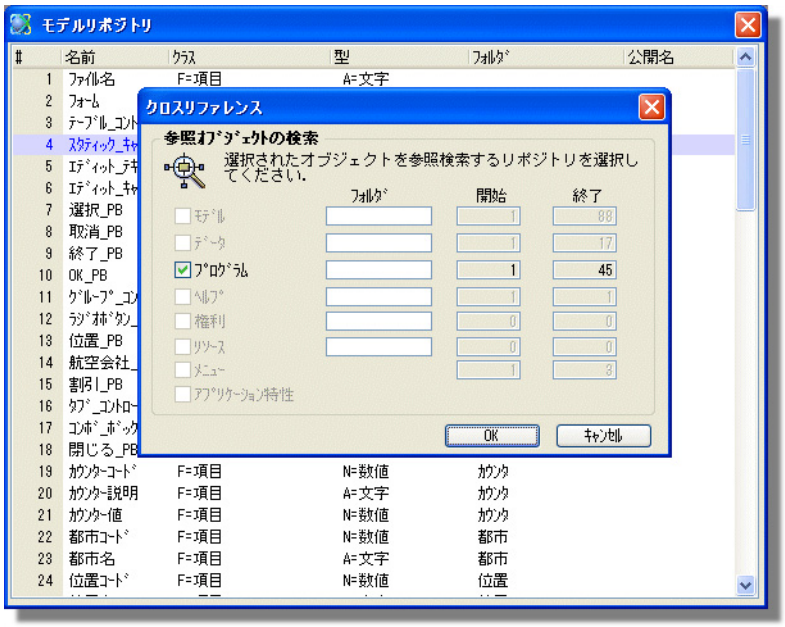
行に移動する

1. アクセスしたいリポジトリにフォーカスを移します。
2. ***Ctrl+J*** を押下します（または、**編集 → クイックアクセス →行ジャンプ**）。
3. 位置付けたい行番号を入力します。
4. ***Enter*** を押下します。

指定した行に位置付けられます。

オブジェクトがどこで使用されているかを確認するには

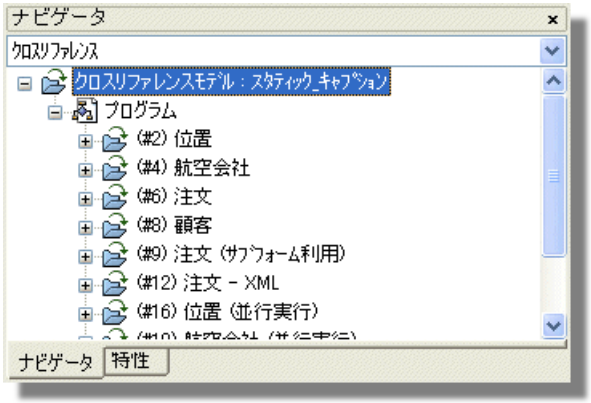
全てのプログラミングにおける主要な問題の 1 つが「もしこの項目を変更したら、他のどの項目に影響するのか?」ということです。1 つの簡単な変更によって、意図しない影響を引き起こす場合がかがあります。Magic には、クロスリファレンスという特定のオブジェクトを参照しているオブジェクトを検索する機能があります。



クロスリファレンスを使用する

1. 検索対象の項目にカーソルを移動します。
2. *Ctrl+F* を押下します（または、**編集 → 検索と置換 → クロスリファレンス**）。
3. クロスリファレンスダイアログが開きます。デフォルトでは、参照可能な全てのリポジトリをチェックするように設定されています。必要に応じてチェック対象を絞ることができます。
4. *Enter* を押下します（または OK をクリック）。

クロスリファレンスを実行すると、指定されたオブジェクトを使用している全てのオブジェクトが表示されます。



この表示は以下のように利用できます。

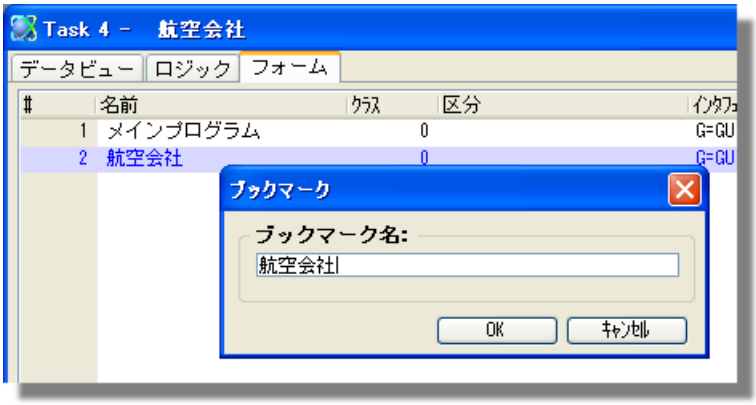
オブジェクト名をクリックすることで、そのオブジェクトに移動することができます。1 つのオブジェクトを複数のオブジェクトが参照しているような場合に有効です。

この表示内容は削除することもできます。参照項目の修正を行った後で、*F3* を押下して（または **編集→行削除**：Magic の通常の行削除と同じです）表示を削除してください。

クロスリファレンスの結果をテキストファイルに保存したり、印刷することもできます。（**編集→検索と置換→検索結果の保存** と **検索結果の印刷** ）

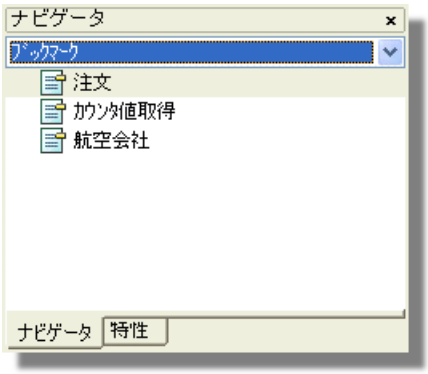
よく使用するオブジェクトにブックマークを設定するには

プログラミングやテストの際に同じオブジェクトに何回も繰り返しアクセスする場合があります。これらのオブジェクトは、タスクツリー内のオブジェクトの場合もあります。これらのオブジェクトに対して、**ブックマーク**という機能を利用することでマークすることができます。ブックマークが定義されたオブジェクトは、**ブックマーク**ペイン（**ナビゲータ→ブックマーク**）に表示されます。



現在の場所にブックマークを定義する

1. ***Ctrl+Shift+B*** を押下します（または、**オプション→ブックマーク**）。
2. **ブックマーク**ダイアログが表示されます。ブックマークとして登録したい名前を入力します。
3. ***Enter*** を押下します（または OK を**クリック**）。.



定義されたブックマークが **ブックマーク**ペイン（**ナビゲータ→ブックマーク**）に表示されます。このブックマーク名を**クリック**すると対応するオブジェクトにカーソルが移動します。

コンテキストメニューの**ツリー編集**を使用することでブックマークの名前を変更することができます。また、***F3***（**編集→行削除**）を使用して削除することができます。

ヒント : *登録できるブックマークの最大値は、**動作環境**ダイアログで指定できます。最大値を変更するには、**ブックマークの最大登録数（オプション→設定→動作環境→動作設定）**の設定値を変更します。*

最近使用したプロジェクトを迅速に開くには

複数のプロジェクトで開発を行うことがあります。通常、プロジェクトを開くには *Ctrl+O* を押下します（ファイル→プロジェクトを開く）。しかし、今まで作業していたプロジェクトを簡単に開くこともできます。

最近使用したプロジェクトを開く

1. プルダウンメニューからファイル→最近使ったプロジェクトを選択します。
2. 開きたいプロジェクトをクリックします。

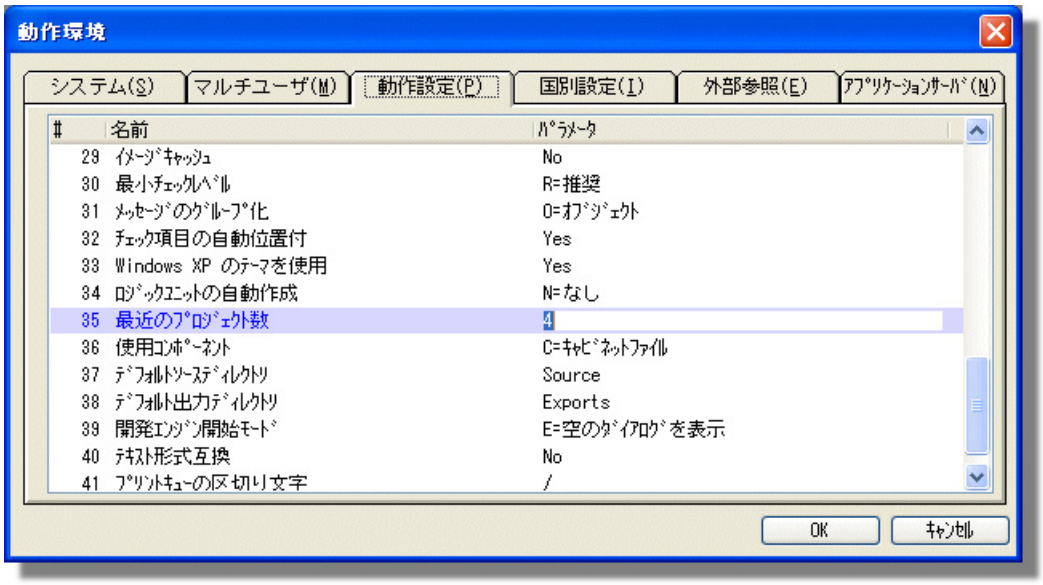
現在作業中のプロジェクトは保存され、選択されたプロジェクトが開きます。

注： プロジェクト名の一覧表示をスクロールするとステータスバーにプロジェクトファイル名がフルパスで表示されます。同じ名前や似たような名前のプロジェクトをいくつか使用している場合は、この表示を参照してください。

参照： 「プロジェクトを切り替えるには」（15 ページ）
「最近開いたプロジェクトの表示数を変更するには」（9 ページ）

最近開いたプロジェクトの表示数を変更するには

最近開いたプロジェクトの表示数を変更することができます。デフォルトは *4* ですが、最大 *99* まで表示させることができます。



最近開いたプロジェクトの数を設定する

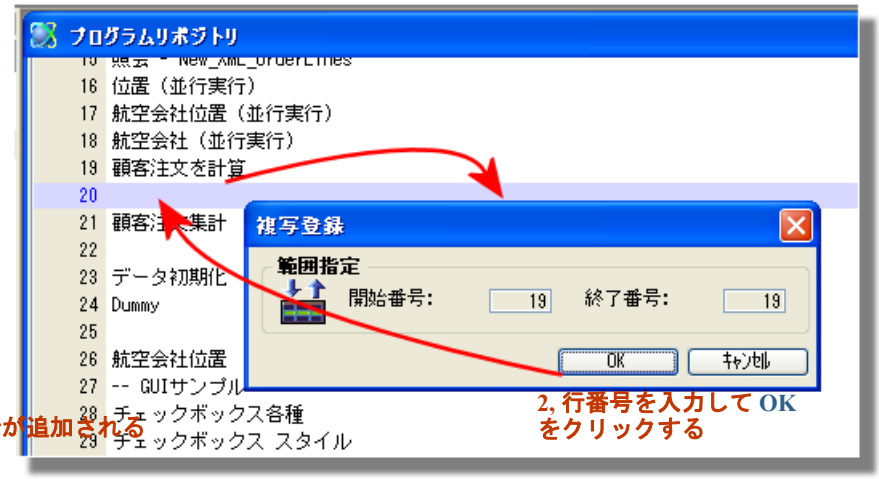
1. プルダウンメニューから**オプション→設定→動作環境→動作設定**を選択します。
2. **最近のプロジェクト数**の値を変更します。

大きな値を設定すると、表示されるプロジェクト数が増えます。

Magic.ini ファイルの **NumberOfRecentProjects** の設定値を変更することで表示数を変えることもできます。

入力行を複写するには

Magic で新しいオブジェクトを作成する必要がある場合、類似したオブジェクトをコピーして利用することで開発時間を短縮することができます。 1 つのオブジェクトをコピーする場合は、この操作を繰り返す必要があります。



必要条件 : バージョン管理を使用している場合、最初にリポジトリをチェックアウトしてください。

複写機能を使用する

1. 新規にオブジェクトを作成したい行にカーソルを移動します。
2. *Ctrl+Shift+R* を押下します（または、編集→登録→複写登録）。
3. ダイアログボックスが表示されたら、複写したい行番号を入力します。行番号があらかじめ分かっている場合は、そのまま入力します。分からない場合は、開始番号からズームして項目一覧を表示して選択します。
4. 終了番号のデフォルトは、開始番号と同じ値になります。1 つだけコピーする場合はこのまま *Enter* を押下します。連続した複数の行をコピーする場合は、複写行グループの最後の番号を入力します。
5. *Enter* を押下します（または OK をクリック）。

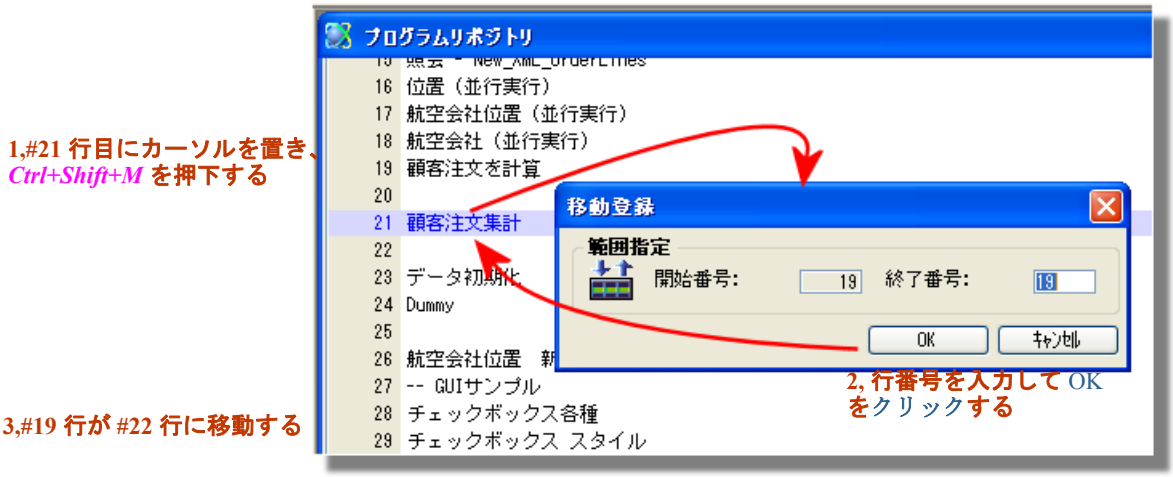
選択された項目は、現在行のすぐ下にコピーされます。

ヒント : コピーした項目は、すぐ名前を変更するようにしてください。そうしないと、コピー元の項目との区別が付かなくなります。また、バージョン管理を使用している場合、最初にリポジトリをチェックアウトしてください。チェックアウトしない場合はこの機能は利用できません。

注 : リポジトリ内でのカット＆ペースト操作は、テキストのみに対して行うことができます。リポジトリ内の項目名のみが対象で、オブジェクト自身のコピーは行われません。

入力行を移動するには

作業内容を整理しておくために、リポジトリ内の項目を入れ替える必要がある場合があります。例えば、アルファベット順であったり、機能にもとづいたグループに分けようにします。（メインプログラムを除いた）どのような項目もリポジトリ中で移動させることができます。



必要条件： バージョン管理を使用している場合、最初にリポジトリがチェックアウトされていることを確認してください。

入力行を移動する

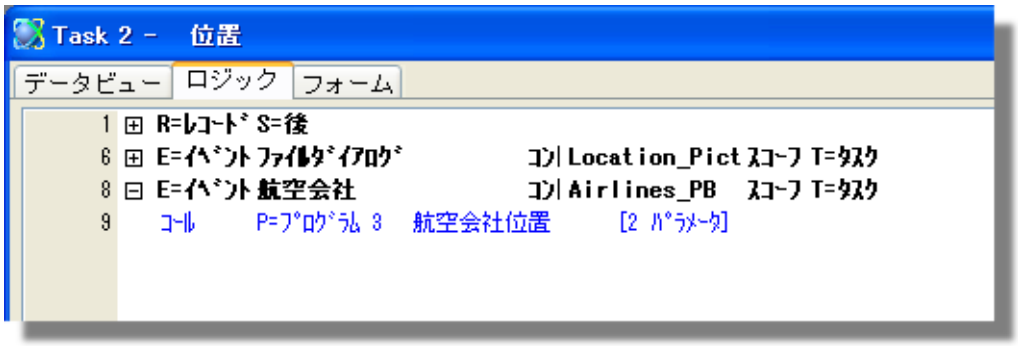
1. 項目を移動したい行にカーソルを移動します。
2. *Ctrl+Shift+M* を押下します（または、編集→登録→移動登録）。
3. ダイアログボックスが表示されたら、**開始番号**に移動する項目の行番号を入力します。番号を知っているのであれば、直接入力します。分からない場合は、ここからズームして一覧から選択します。
4. **終了番号**のデフォルトは**開始番号**と同じになるため、1 行だけ移動する場合は *Enter* を押下するだけです。複数の連続した行を移動する場合は、移動する行の最終番号を入力します。
5. *Enter* を押下します（または、OK をクリック）。
6. 選択された項目は、現在の位置に移動します。

ヒント : *Magic は、オブジェクトを名前ではなく番号で参照しているため、移動オプションを使用して移動することは何の問題もありません。Magic は、新しい位置を示すため参照情報を変更し、背後で様々なオブジェクトとの関連性を保持するために内部番号を使用しています。*

ヒント : DSOURCE または PROG *リテラルを使用しない場合は、問題がでる場合があります。このようなプログラムを使用している可能性がある場合は、（特に* **DbDel()** *関数などを）テキスト検索機能を利用してチェックし、修正するようにしてください。*

入力行を他の行の内容に置き換えるには

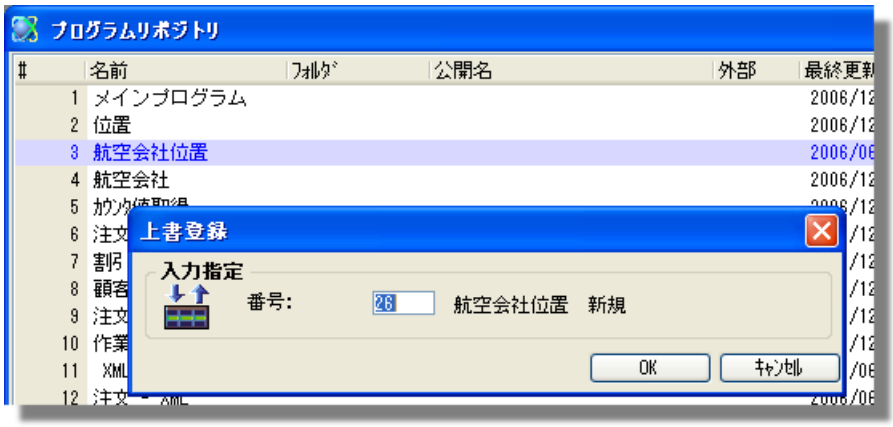
Magic における項目は、これらの連番にもとづいて参照されます。例えば、以下のようなプログラムを呼び出すようにしたとします。



プログラム **#3** の航空会社位置プログラムをプログラム **#26** に移動した場合、何の問題も起きません。Magic は、自動的に起動先を **#26** のプログラムに変更します。

しかし、作成したばかりの新バージョンの航空会社位置プログラムを呼び出すようにしたいと考えた場合、プログラム **#3** を新しいプログラムに置き換える必要があります。

行を置き換える



1. 差し替えたいプログラムの行にカーソルを移動します（この場合は、行 **#3**）。
2. ***Ctrl+Shift+O*** を押下します（または、**編集→登録→上書登録**）。
3. 置き換える項目の行番号（この場合 **#26**）を入力するか、**ズーム**して一覧から選択してください。
4. ***Enter*** を押下します（または OK を**クリック**）。

上記の操作によって、行 **#11** は新しいプログラム（この場合、航空会社位置 新規）に置き換えられます。

ヒント： *上記の操作によって、現在作業中のオブジェクトをそのままにした状態で必要に応じて実際に利用するオブジェクトを変更することができます。複写機能を使用して作業中のオブジェクトをコピーした場合は、後で混乱しないようにコピーされたオブジェクトの名前を変更しておきます。必要であれば、このような方法でいくつかの作業用コピーを取っておき、容易にそれらを比較し、各バージョンに切り替えるために上書き機能を使用することができます。この場合は、開発中のバージョンに戻すため、ロールバック用のオブジェクトも準備しておく必要があります。*

ペイン間の移動を行うには

作業中は、通常 1 つ以上のパレットがオープンされます。これらのパレット間の移動は、複数の方法で行うことができます。

- 使用するパレットを**クリック**します。
- ***Ctrl+Tab*** を使用してパレット間でフォーカスを移動します。
- パレットを表示させるには、以下のキーを使用します。

  Alt+F1（**表示→ナビゲータ**）

  Alt+F2（**表示→特性シート**、***Alt+Enter*** も可）

  Alt+F3（**表示→チェック結果**）

Ctrl+Tab は、***Alt*** とは少し動作が異なります。***Ctrl+Tab*** の場合、結合されたパレットは 1 つのウィンドウとして動作しますが、***Alt*** は他のウィンドウによって隠されていてもパレットが開きます。また、**タスク特性**ダイアログや**タスクエディタ**のようにタブを持つウィンドウを開いた場合、***Ctrl+Tab*** は、タブの切替として動作します。

参照： 「パレットを分離するには」（3 ページ）

1 つのセクションを表示している特性シートを保持するには

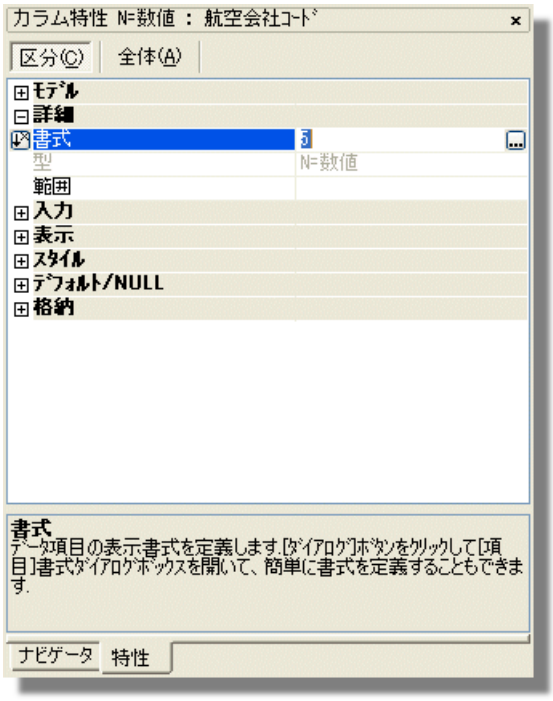
特性シートが開いた場合、デフォルトではすべてのセクションは展開されて表示され、全ての有効な特性値を参照することができます。しかし、よりコンパクトに表示させたい場合があるかもしれません。

特性シートの表示を変更する

1. 現在のプロジェクトを閉じます。
2. 単一拡張パレット特性（オプション→設定→動作環境→動作設定）を選択します。
3. 単一拡張パレット特性を *Yes* に設定します。

この設定によって、1 つのセクションを開くと、以前に開いていたセクションが縮小表示されます。また、新しい項目をクリックすると、すべてのセクションは縮小表示されます。

全体タブで表示されたいる特性は、この設定に影響しません。

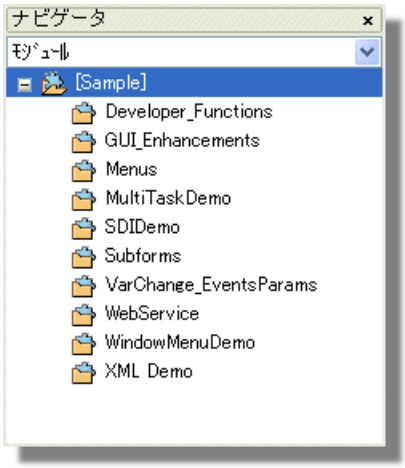


プロジェクトを切り替えるには

複数のプロジェクトが、必ずしも互いにコンポーネントという関係でなくても、グループとして一緒に作業することがあるかもしれません。

ナビゲータペインの*モジュール*を選択することで、これらのグループに簡単にアクセスすることができます。

以下の例では、Sample は 10 のモジュールを持っています。モジュールがオープンされている場合、アイコン名に括弧が設定されます。各モジュールは、Magic のプロジェクトを表しており、これらはどこにでも位置付けることができます。



プロジェクトを切り替える

1. ナビゲータペインに移動します。
2. *モジュール*を選択します。
3. 開きたいモジュールにカーソルを移動します。
4. ダブルクリックするか、*F5*（ズーム）を押下します。

そのモジュールに対応するプロジェクトが自動的にオープンされ、現在のプロジェクトがクローズされます。その際、保存されていない変更内容が保存されます。

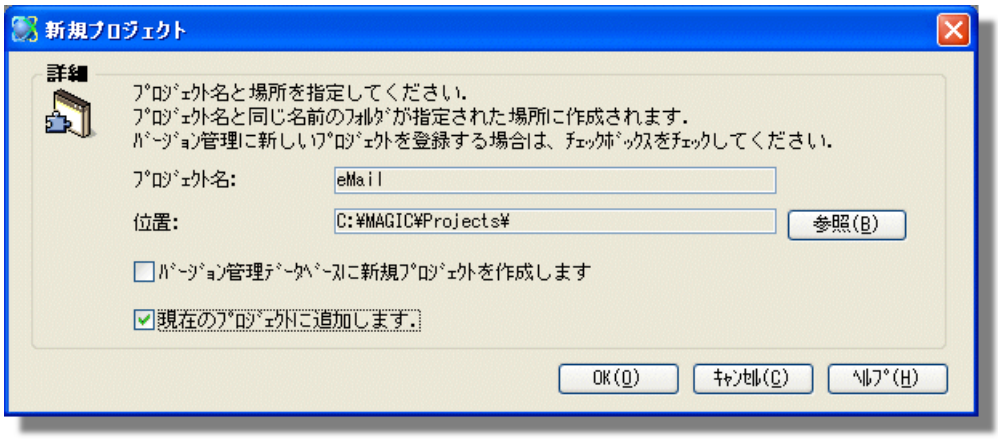
モジュールを追加する

必要条件： バージョン管理を使用している場合、最初にリポジトリをチェックアウトする必要があります。

1. プルダウンメニューからプロジェクト→モジュール追加を選択します。
2. プロジェクトを開くダイアログが表示されます。追加したいプロジェクトファイルを選択します。

追加されたプロジェクトは、モジュールペインに表示されます。

また、プロジェクトを新規作成する場合、現在のプロジェクトに追加します。のチェックボックスをチェックすることで自動的に現在のプロジェクトのモジュールペインに追加されます。



必要条件： バージョン管理を使用している場合、最初にリポジトリがチェックアウトされているかどうかを確認してください。

モジュールを削除する

必要条件： モジュールを削除する前に、モジュールの上位にカーソルを置く必要があります。現在開いているモジュールや、モジュールの上位のオブジェクトを削除することはできません。また、バージョン管理を使用している場合、最初にリポジトリを**チェックアウト**とする必要があります。

1. 削除したいモジュールのノード上にカーソルを置きます。
2. *F3* を押下します（または、**編集→行削除**）。
3. **削除確認**ダイアログではいを**クリック**します。

ノードは、現在の**モジュールツリー**から削除されるます。ただし、プロジェクトは削除されません。再度、モジュールとして追加することができます。

参照： 「最近使用したプロジェクトを迅速に開くには」（8 ページ）

特性シートのセクション表示を切り替えるには

特性シートのセクション間を移動する

特性シートのセクション間を移動するには、以下の３つの方法があります。

- マウスで該当セクションを**クリック**します。
- キーボードで ***Ctrl+*** ↑（上方向のセクションに移動）、または ***Ctrl+*** ↓（下方向のセクションに移動）を押下します。
- **編集→ツリー→前のセクション**（上方向のセクションに移動）、または**編集→ツリー→次のセクション**（下方向のセクションに移動）

特性シートのノードを拡張表示させる

特性シートのノード表示を拡張表示させるには、以下の３つの方法があります。

- マウスで該当するセクション名の前に表示されている ⊟ アイコンを**クリック**します。
- キーボードで ***Ctrl+Plus***（+）を押下します。
- **編集→ツリー→サブツリー展開**

特性シートのノードを縮小表示させる

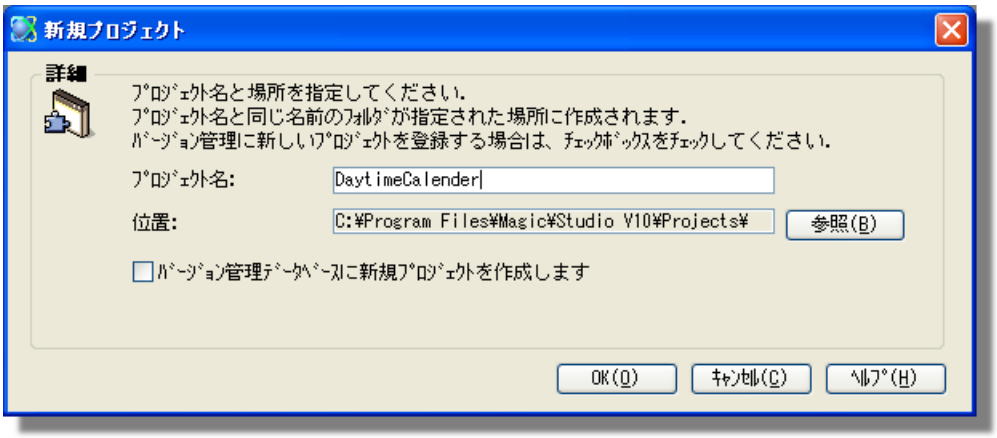
特性シートのノード表示を縮小表示させるには、以下の３つの方法があります。

- マウスで該当するセクション名の前に表示されている ⊞ アイコンを**クリック**します。
- キーボードで ***Ctrl+Minus***（-）を押下します。
- **編集→ツリー→サブツリー縮小**

［このページは意図的に空白にしています。］

第 2 章： プロジェクトとアプリケーション

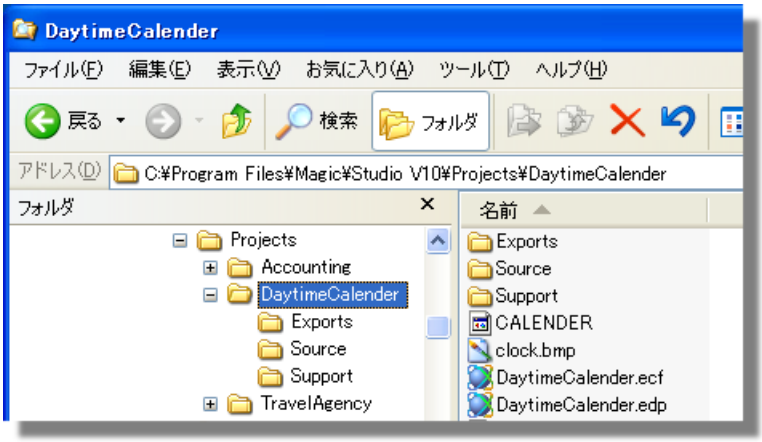
プロジェクトを新規作成するには



プロジェクトを新規作成する

1. プルダウンメニューから、ファイル→新規作成 を選択します。
2. プロジェクト名を入力します。
3. 位置にプロジェクトファイルを作成するディレクトリを入力します。
4. OK をクリックします。

現在プロジェクトを開いている場合は、そのプロジェクトがクローズされ、新規プロジェクトが作成されます。



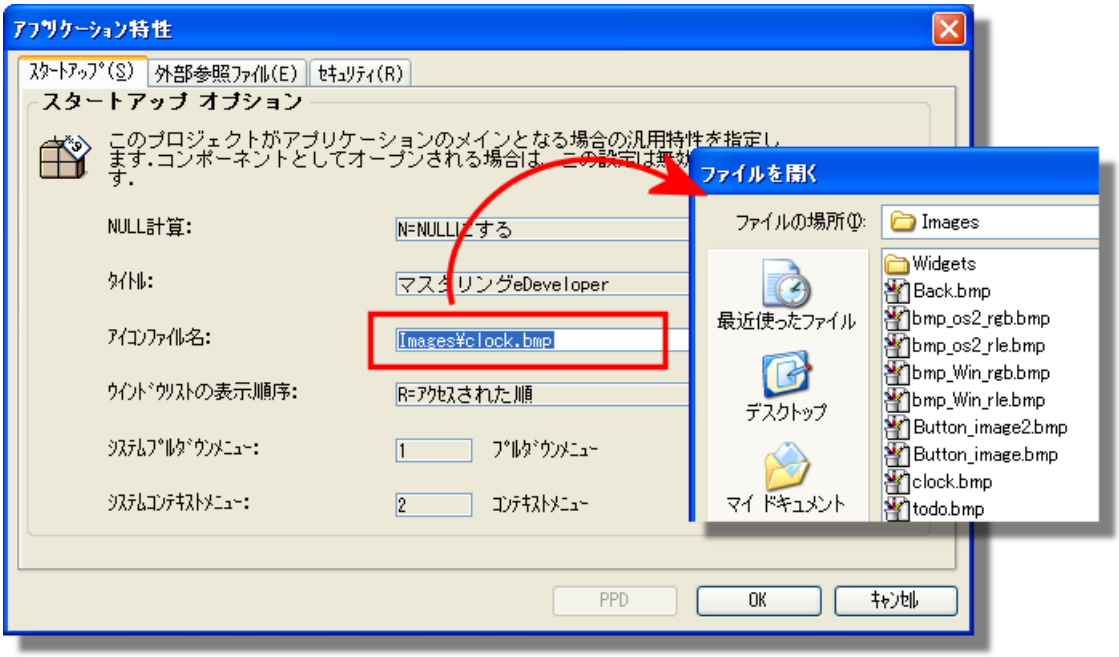
プロジェクトが作成されると、指定されたディレクトリ内にサブディレクトリが作成され、その中に関連するファイルが格納されます。プロジェクト名はディレクトリ名になり、プロジェクトファイルの名前になります。プロジェクトファイルの拡張子は、「.edp」になります。

プロジェクトが既にオープンされている場合、現在のプロジェクトに追加します。というチェックボックスが表示されます。チェックボックスを**チェック**すると作成されたプロジェクトは、既存のプロジェクトのモジュールとして追加されます。

参照 : 「既存のプロジェクトをオープンするには」（29 ページ）

アプリケーションのアイコンを設定するには

アプリケーションを開発する場合、アプリケーションに独自のアイコンを割り当てることがあります。このような場合は、以下で示す方法で設定することができます。



アプリケーションのアイコンを設定する

1. アプリケーション特性（ファイル→アプリケーション特性または、*Ctrl+Shift+P*）を開きます。
2. アイコンファイル名特性にファイル名を入力するか、ズーム を使用してファイルを選択します。
3. OK をクリックします。

アプリケーションを実行すると、ウィンドウの左上にアイコンが表示されます。また、タスクバーや *Alt+Tab* によってウィンドウを切り替える際にも表示されます。

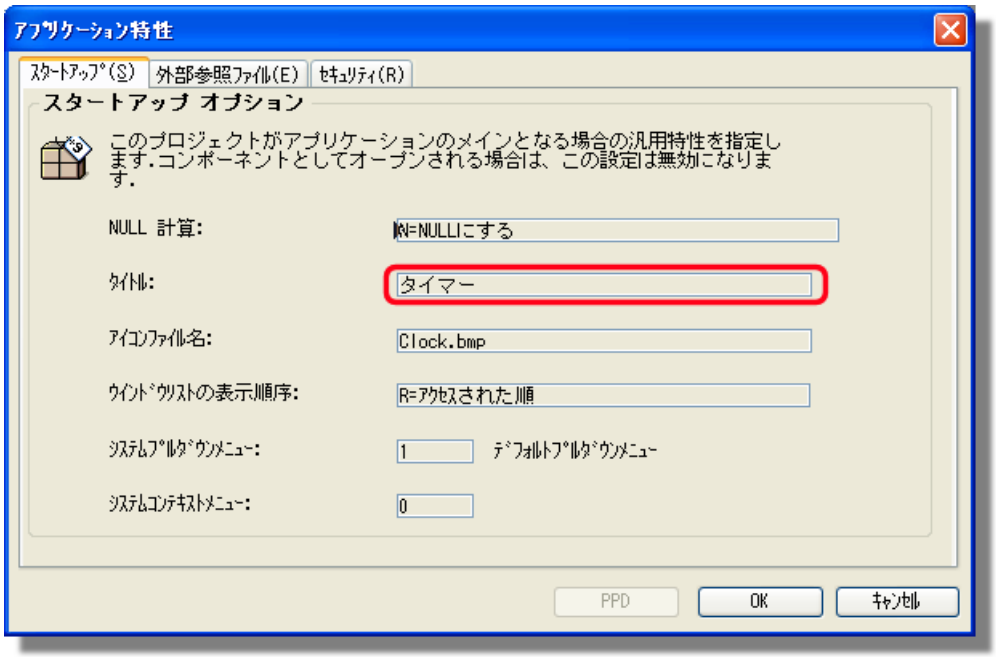
ウィンドウタイトルのアイコン　システムトレイのアイコン　　アプリケーション切り替え時の表示

ヒント：*アプリケーションのユーザは、開発時とは異なるパスでセットアップする場合があるため、このような内部ファイルを指定する場合は、固定のパス名を使用しないでください。デフォルトのパスは、作業ディレクトリ（プロジェクトファイルが存在している場所）になるため、アイコンファイルはそこに置くことか、相対的なサブディレクトリを指定してください。*

参照： 「プロジェクトのディレクトリのファイルを読み書きするには」（26 ページ）

アプリケーションのタイトルを指定するには

ウィンドウに表示されるアイコンの隣に、アプリケーションの名前を示すテキストを表示させることができます。この設定は、アイコン指定と同じ場所で行います。



アプリケーションのタイトルを設定する

1. アプリケーション特性（ファイル→アプリケーション特性または、*Ctrl+Shift+P*）を開きます。
2. タイトル特性にタイトル名を入力します。タイトル名に論理名を指定することで、動作環境に応じてタイトルを変更させることができます。
3. OK をクリックします。

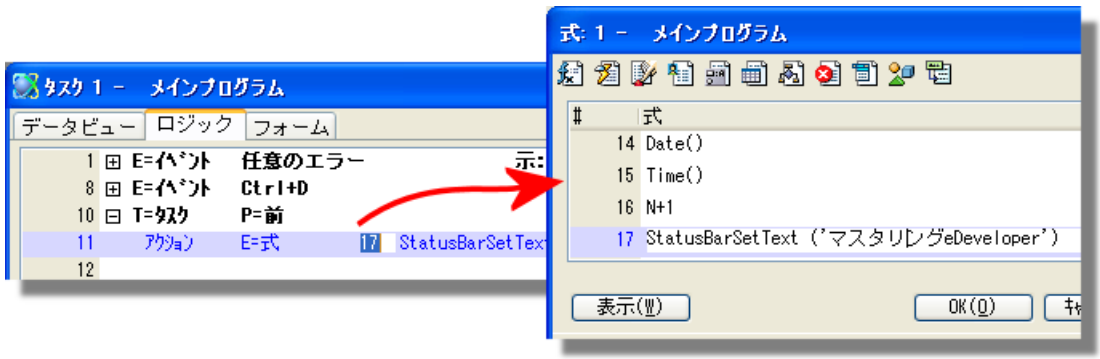
アプリケーションを実行すると、ウィンドウの左上にタイトルが表示されます。また、タスクバーや *Alt+Tab* によってウィンドウを切り替える際にも表示されます。

ステータスバーにタイトルを表示する

アプリケーション起動時に、ステータスバーにアプリケーションのタイトルを表示させることもできます。

1. メインプログラムのロジックエディタを開きます。
2. タスク前のロジックユニットを作成し、１行追加します。
3. アクション処理コマンドを定義します。式でズームし、以下の式を定義します。

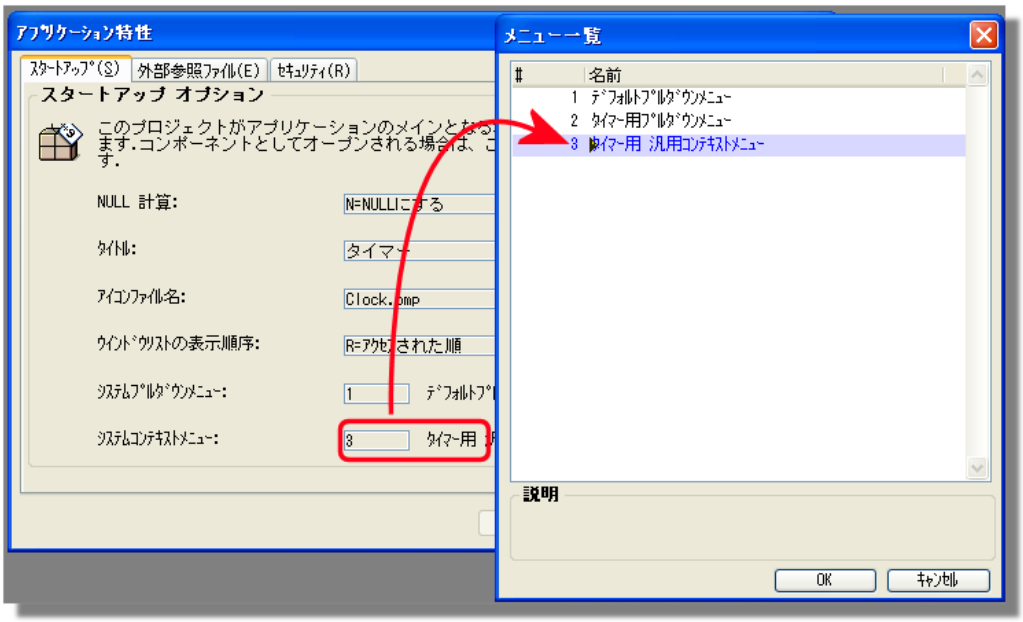
   StatusBarSetText (*アプリケーション名*)



アプリケーションを実行すると、ステータスバーにタイトルが表示されます。

アプリケーションにコンテキストメニューを設定するには

オンラインタスク用にコンテキストメニューを定義する場合、特定のプログラムに設定する方法とは別に、アプリケーション全体で有効なデフォルトのコンテキストメニューを設定することもできます。

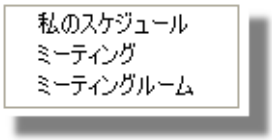


設定するメニューは、あらかじめ作成しておく必要があります。

アプリケーションにコンテキストメニューを設定する

1. アプリケーション特性（ファイル→アプリケーション特性または、*Ctrl+Shift+P*）を開きます。
2. システムコンテキストメニュー特性でズーム（*F5* または、ダブルクリック）を実行します。
3. 設定するコンテキストメニューにカーソルを置きます。
4. OK をクリックして、アプリケーション特性に戻ります。
5. OK をクリックしして、ダイアログを閉じます。

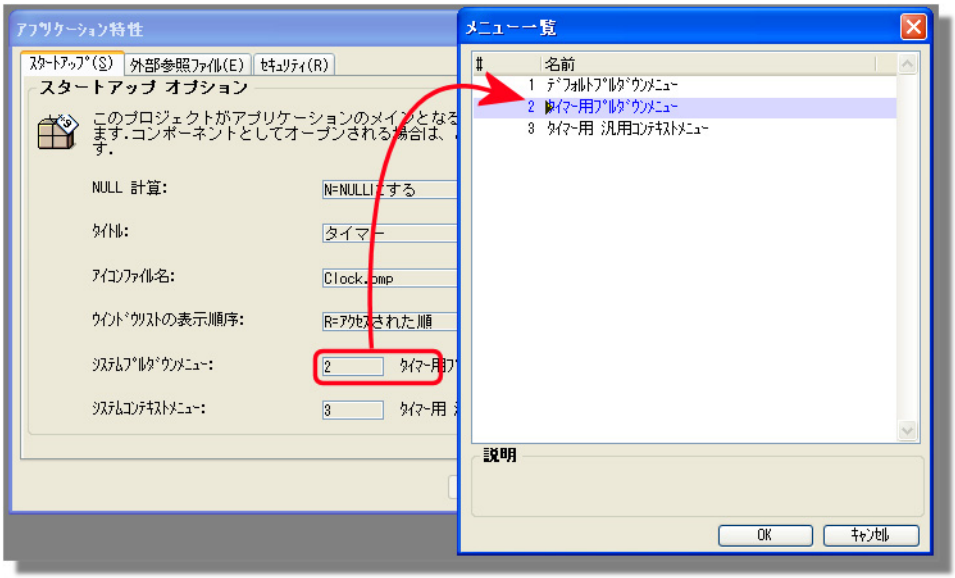
アプリケーションを実行し、ユーザがマウスの右クリックを行うと、設定されたメニューが表示されます。



参照： 第 5 章：「フォームのコントロールにデフォルトのコンテキストメニューを設定するには」（131 ページ）
第 5 章：「個々のコントロールにコンテキストメニューを設定するには」（132 ページ）

アプリケーションにプルダウンメニューを設定するには

Magic は通常、デフォルトのプルダウンメニューが表示されますが、ここには基本的な編集用のコマンドのみ表示されています。アプリケーションを使用するユーザ用に、専用のプルダウンメニューを表示させることができます。

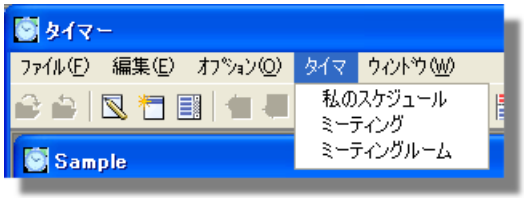


必要条件： 設定するメニューは、あらかじめ作成しておく必要があります。

アプリケーションにプルダウンメニューを設定する

1. アプリケーション特性（ファイル→アプリケーション特性または、*Ctrl+Shift+P*）を開きます。
2. システムプルダウンメニュー特性でズーム（*F5* または、ダブルクリック）を実行します。
3. 設定するコンテキストメニューにカーソルを置きます。
4. OK をクリックして、アプリケーション特性に戻ります。
5. OK をクリックしして、ダイアログを閉じます。

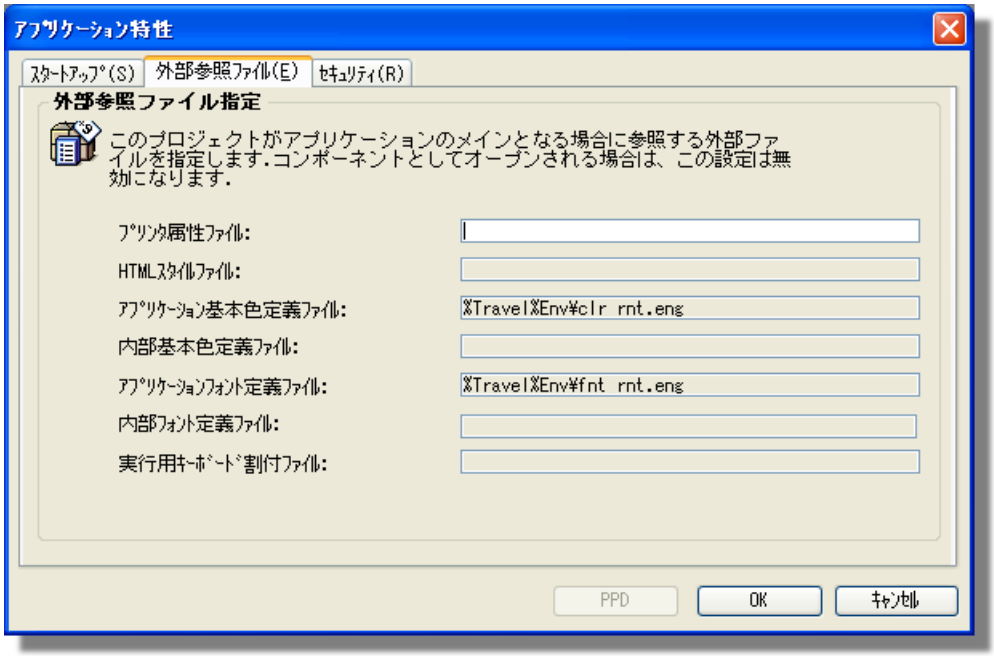
アプリケーションを実行すると、指定されたプルダウンメニューが表示されます。



注： メニューリポジトリ上では、*コンテキスト*と *プルダウン*の違いはありません。必要であれば、両方に同じメニューを使用することができます。

アプリケーション用に基本色、フォント、キーボード割付を設定するには

Magic は、デフォルトでは、Magic のインストール時に転送された（基本色、フォント、キーボード割付）定義ファイルを使用します。しかし、アプリケーション用にこれらのファイルをカスタマイズする必要があります。各アプリケーションは、独自の定義ファイルを使用することができます。これらのファイルは、個別のテキストファイルのため、ユーザによって任意にカスタマイズすることができます。



必要条件： 最初に特定のディレクトリ内に定義ファイルをコピーしておく必要があります。

アプリケーション用の定義ファイルを設定する

1. アプリケーション特性（ファイル→アプリケーション特性または、*Ctrl+Shift+P*）を開きます。
2. 変更したい特性値（基本色、フォント、キーボード割付の各定義ファイル）にカーソルを移動します。
3. ファイル名を入力します。

各ファイル名の入力欄でズームすると、アプリケーション用に定義内容を変更するためのダイアログが表示されます。

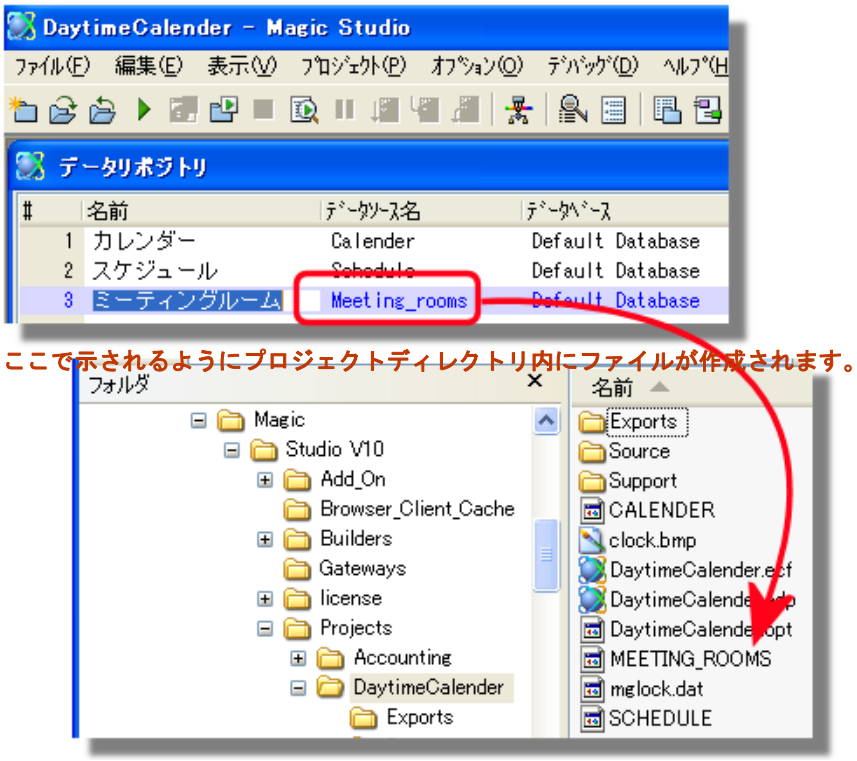
注： SP4 以降、フルパスまたは論理名が指定されない場合、MAgic エンジンが存在するディレクトリ（%EngineDir%）をアクセスします。アプリケーションフォルダ内のファイルをアクセスさせる場合は、%WorkingDir% を指定するようにする必要があります。

参照： 「プロジェクトのディレクトリのファイルを読み書きするには」（26 ページ）

プロジェクトのディレクトリのファイルを読み書きするには

プロジェクトの実行中、デフォルトディレクトリはプロジェクトディレクトリ（.edp または .ecf ファイルのが配置されているディレクトリ）です。従って、アプリケーションによってファイルを作成する際、明示的なパスを指定しない場合、プロジェクトディレクトリ内に作成されることになります。

以下のような DayTimerCalender アプリケーションに 3 つの DB テーブルがある場合を例に説明します。



入出力ファイルなどを使用してプログラムから他のファイルを参照する場合も同じように動作します。プロジェクトディレクトリ以下にファイルを置くことで、相対パスを使用してこれらのファイルにアクセスすることができます。

アプリケーションが複数のプロジェクトによって構成されている場合、プロジェクトディレクトリは一番上位のプロジェクトのプロジェクトディレクトリによって決まります。従って、DayTimerCalender プロジェクトがプロジェクトディレクトリに書き込まれたコンポーネントを呼び出す場合、これらのファイルは DayTimeCalendar ディレクトリ内にあります。

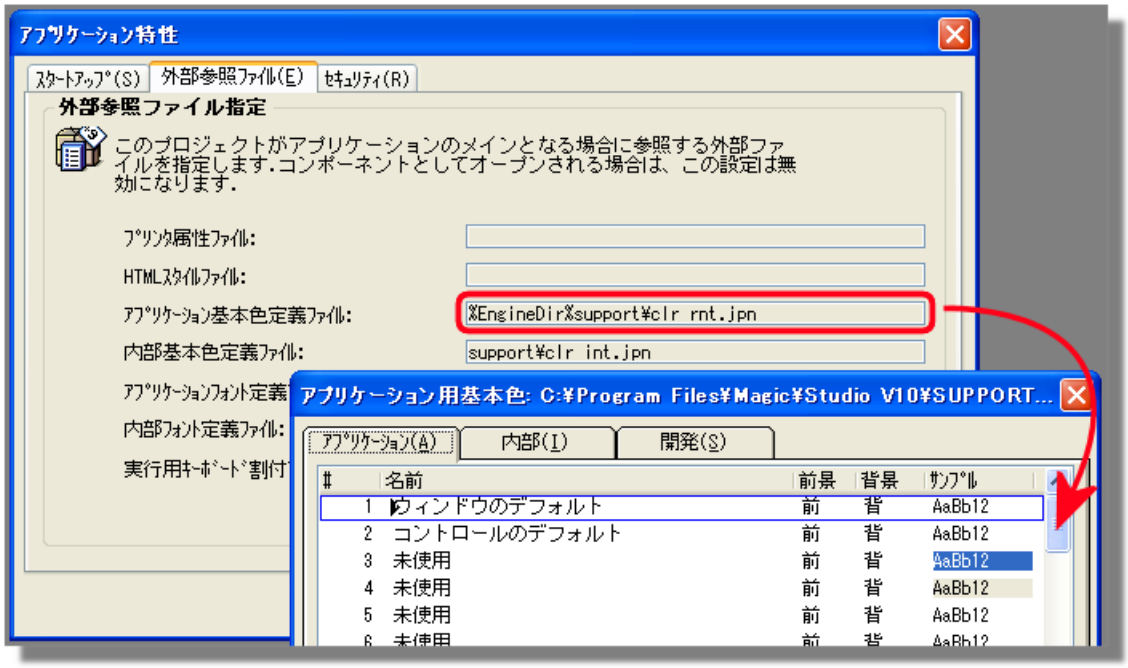
明示的にプロジェクトディレクトリを参照したい場合は、論理名（%WorkingDir%）を使用してください。

Magic ディレクトリのファイルを読み書きするには

*Magic ディレクトリ*は、Magic エンジンが存在するディレクトリを表しています。インストール時のデフォルトディレクトリは、Windows の上の Programe Files ディレクトリの下に作成されます。

Magic は、データソースや入出力ファイルなどのプロジェクト固有のファイルと一般的なエンジンファイルを区別します。例えば、**動作環境**ダイアログの**外部参照**タブ（**オプション→設定→動作環境**）に定義されている基本色、フォルト、キーボード割付の各定義ファイルは、すべて Magic ディレクトリに配置されます。

アプリケーション特性内で、作業ディレクトリの代わりに Magic ディレクトリを参照するようにするには、論理名 %EngineDir% を使用することができます。例えば、DayTimerCalender プロジェクト内で、Magic と一緒にインストールされた**基本色定義ファイル**にアクセスする必要がある場合は以下のようにします。

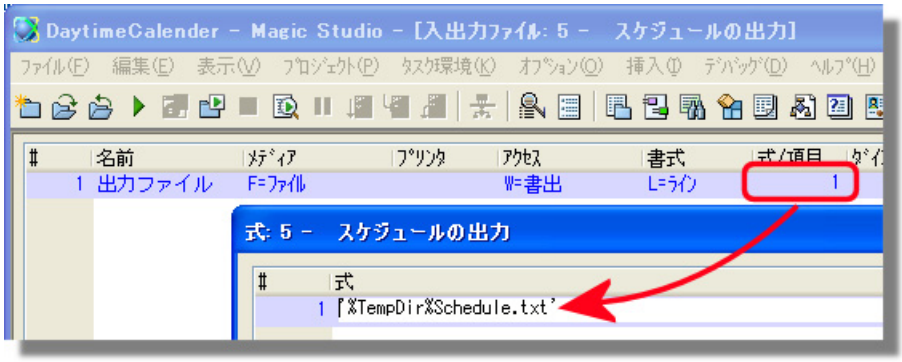


相対パスの前に %EngineDir% を付加することで Magic のインストールディレクトリ内の clr_rnt.jpn ファイルを指定することができます。

システム一時ディレクトリ内のファイルを読み書きするには

Windows™ には、セットアップ時に作成される一時ディレクトリがあります（デフォルトでは、C:¥Documents and Settings の下に作成されます）。これは、多くのアプリケーションが***スクラッチファイル***を書込むために使用されます。

例えば、他の製品にデータをインポートする目的で作成する中間ファイルの作成先などに使用できます。このディレクトリは、論理名 %TempDir% を使用することでプログラムからアクセスすることができます。



この例では、Excel ファイルから読み込み込まれたデータをもとにテキストファイルを作成しています。パス名を指定しない場合、一時ファイルは作業ディレクトリに作成されます。ここで論理名 %TempDir% を追加することで、システム一時ディレクトリに作成するように指定することができます。

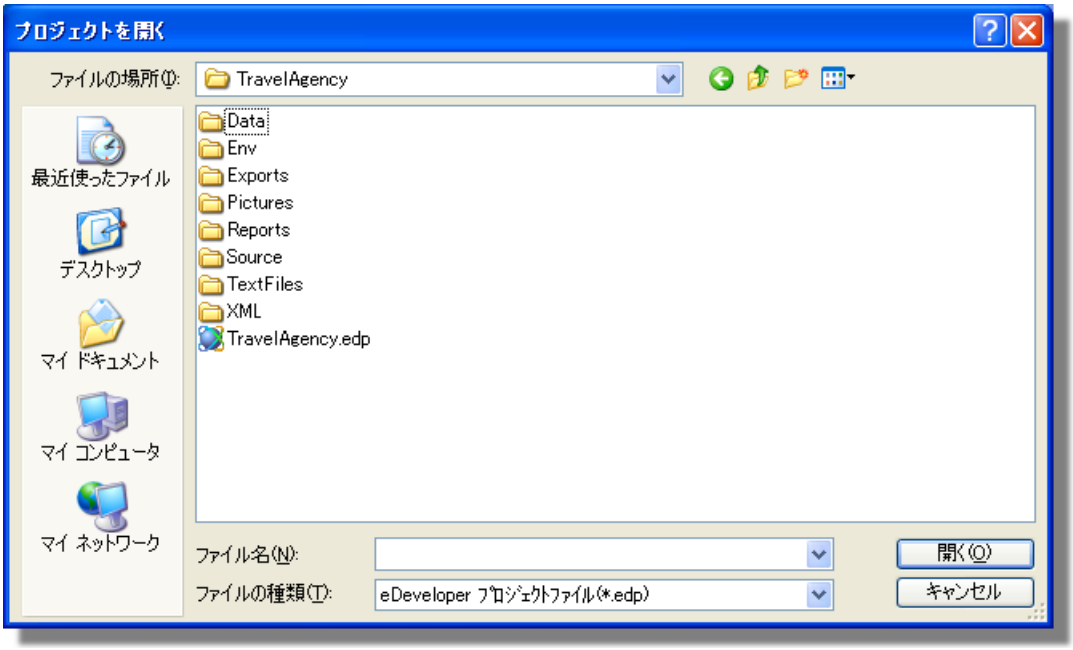
既存のプロジェクトをオープンするには

アプリケーションを 1 つ以上のプロジェクトで構成する場合があります。この場合、いくつかのロジックがコンポーネントを使用してカプセル化されたり、他のサーバ上で動作するように作成されていることもあります。Magic Studio 内でプロジェクト間での移動を行ったり、Windows からプロジェクトをオープンすることができます。既存のプロジェクトをオープンする方法として、以下の 3 つの基本的な方法があります。

プロジェクトを開くダイアログを使用する

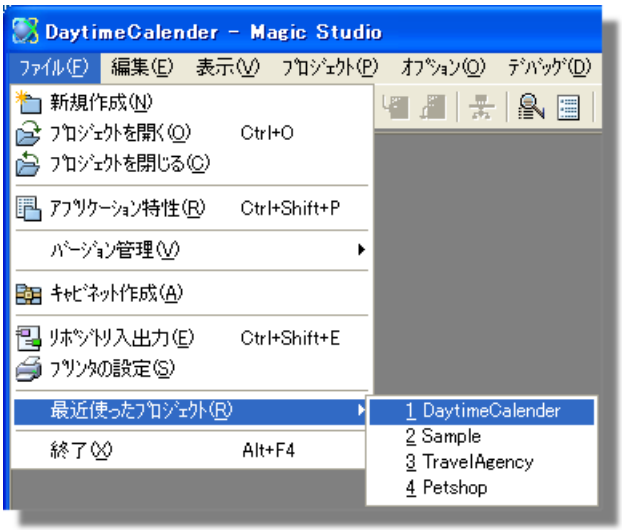
1. プルダウンメニューからファイル→プロジェクトを開く（または、*Ctrl+O* を押下）を選択します。
2. プロジェクトを開くダイアログが表示されます。プロジェクト（***.edp***）ファイルを選択します。
3. 開くをクリックします。

選択されたプロジェクトがオープンされます。



最近使ったプロジェクトを開く

現在のプロジェクトをクローズする必要はありません。プロジェクトのオープン時に現在のプロジェクトは保存、クローズされます。*Ctrl+O*（または、ファイル→プロジェクトを開く）を押下してネットワーク上のプロジェクトをオープンすることができます。しかし、以前作業していたプロジェクトを簡単にオープンすることができます。



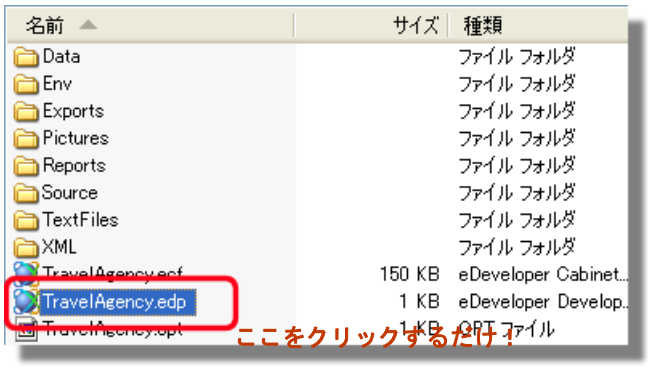
1. プルダウンメニューからファイル→最近使ったプロジェクトを開くを選択します。

2. オープンしたいプロジェクトを探します。表示されているプロジェクト名にカーソルを置くとステータスバーにプロジェクトファイル名がフルパスで表示されるので、この内容も参考にしてください。
3. オープンしたいプロジェクト名をクリックします。

現在オープンしているプロジェクトは保存され、選択されたプロジェクトがオープンされます。

参照： 第 1 章：「最近開いたプロジェクトの表示数を変更するには」（9 ページ）

プロジェクトファイル（*.edp*）を直接アクセスする



1. Windows エクスプローラでオープンしたいプロジェクトファイル（.edp）にカーソルを置きます。
2. プロジェクトファイルをクリックします。

プロジェクトがオープンされます。この方法は他と異なり、オープン中のプロジェクトはそのままの状態で、新しく **Magic Studio** のインスタンスが起動されます。プロジェクトファイルを Windows のショートカットとして登録することでこの方法を利用することもできます。

この方法は、Magic のインストール処理によって、拡張子 ***.edp*** が Magic のプロジェクトファイルに関連付けられることで実現しています。

参照： 第 7 章：「アプリケーション用のショートカットを作成するには」（169 ページ）

別のプロジェクトにオブジェクトを転送するには

必要条件： 他のオブジェクトを参照していないオブジェクトであれば、簡単にコピーすることができます。しかし、他のオブジェクトを参照している場合、コピー先のプロジェクトファイルにも同じ参照オブジェクトが定義されている必要があります。

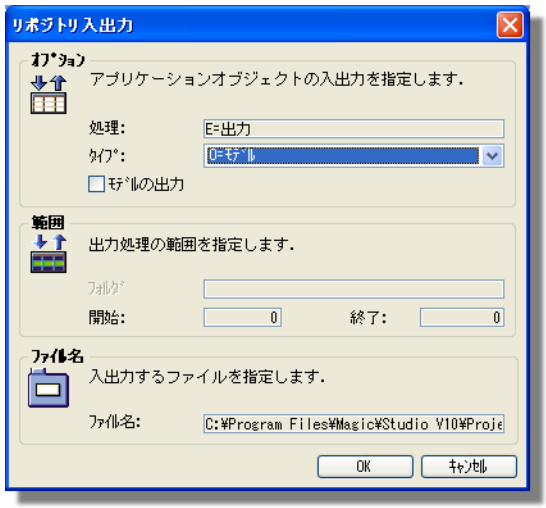
例えば、**プログラム #21** をプロジェクト A からプロジェクト B にコピーする場合を考えます。**プログラム #21** は、**モデル #3、#18、**および **#24** を使用し、また、**プログラム #46** 呼び出しているものとします。

プログラム #21 をプロジェクト B にインポートする場合、**モデル #3、#18、**および **#24** は 2 つのプロジェクトにおいて同じ場所に定義されている必要があります。**プログラム #46** も同じです。

この例の場合、一方のプロジェクトからモデルをエクスポートし、別のプロジェクトにインポートすることから始める必要があります。

オブジェクトを出力する

1. リポジトリ入出力ダイアログ（ファイル→リポジトリ入出力または、*Ctrl+Shift+E*）を開きます。
2. 処理で ***E= 出力***を選択します。
3. タイプで出力したいリポジトリ（モデル、データソース、プログラム、ヘルプ、権利、メニュー、コンポーネント、アプリケーション特性、プロジェクト全体）を選択します。
4. 必要であれば、出力対象のフォルダや出力範囲を指定して、出力範囲を絞り込むことができます。出力範囲指定（開始、終了）では、ズームすることで一覧から選択できます。デフォルトでは、指定されたタイプのオブジェクトがすべて出力されます。
5. ファイル名に出力したいファイル名を入力します。デフォルトは、作業ディレクトリ内の ***Exports*** サブディレクトリ内に出力されます。
6. OK をクリックします。



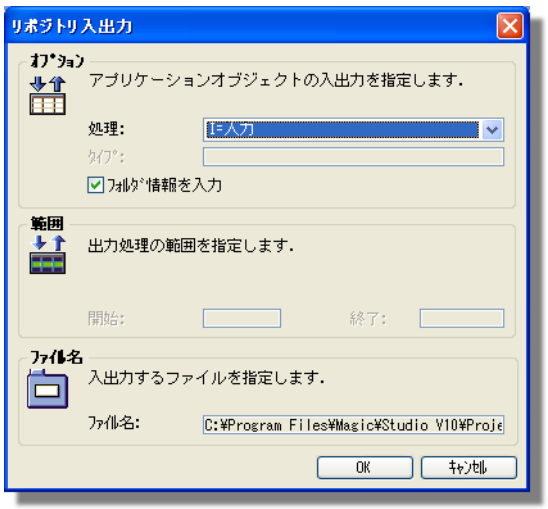
指定したフォルダ内にファイルが作成されます。例えば、モデルを出力した場合 ***Models.xml*** が作成されます。

次に出力したモデルを新規に作成したプロジェクトに入力します。

オブジェクトを入力する

1. リポジトリ入出力ダイアログ（ファイル→リポジトリ入出力または、*Ctrl+Shift+E*）を開きます。
2. 処理で ***I= 入力***を選択します。
3. 入力する XML ファイル名を指定します。
4. OK をクリックします。

出力したオブジェクトと同じ内容が、現在のプロジェクトに追加されます。これらは、常に現在のリポジトリ内に定義されているオブジェクトの下に追加されます。



ヒント：*リポジトリ入出力機能は、通常、同じプロジェクトの異なるバージョン間でプログラムを移動したり、（モデルなどを）空のプロジェクトにコピーしたり、簡単な汎用プログラムをコピーする場合に使用されます。通常、複数のプロジェクト間でオブジェクトを共有する必要がある場合は、コンポーネントを使用してください。コンポーネントは、容易に共有可能で再利用可能なオブジェクトです。*

参照： 第 16 章：「プロジェクト間で Magic のオブジェクトを再利用するには」（357 ページ）

［このページは意図的に空白にしています。］

第 3 章： モデル

再利用可能なインタフェースオブジェクトを定義するには

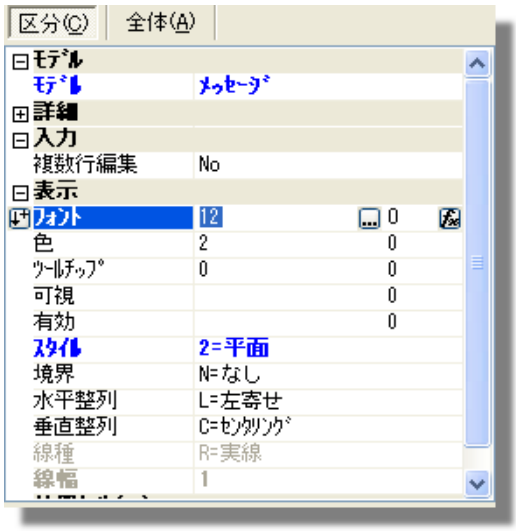
プログラミングにおける難問の 1 つは、複数のウィンドウやブラウザ、あるいは帳票の書式を一貫性のあるものとして維持することです。テキストベースの画面表示は使用されなくなってきているためです。GUI 形式の画面や帳票を設計するには、数百のフォント、サイズ、および色から選択する必要があります。これらの選択作業を各オブジェクト毎に行うと、非常に工数がかかってしまい、更にこれらの一貫性を維持したり、変更したりすることは至難の業です。

Magic はこのような作業を簡単に行うことができます。Magic には、***モデル***という強力な機能が備わっています。モデルは、テキストやラジオボタン、テーブル、テーブルカラム、印刷フォーマット、Web ブラウザ画面、および Windows 互換ウィンドウなどを含んでおり、すべてのインタフェースオブジェクトのルック＆フィールを定義するために使用できます。

各オブジェクトの特性をモデルとして設定することで、容易に使用することができます。フォームを設計する場合は、オブジェクトに対応した使用したいモデルを選択するだけです。これらの書式を変更する場合は、モデルを変更することで自動的にそのモデルを使用するすべてのオブジェクトの書式が変更されます。

ここでは、メッセージ表示用の簡単な**項目モデル**を設定する例を挙げて説明しています。

コントロールモデルを作成する



1. モデルリポジトリに移動し（*Shift+F1*）、アクセスしたい行に移動します。
2. *F4*（編集→行作成）を押下します。
3. モデルの**名前**を入力します（ここでは、メッセージ と入力されています）。
4. **クラス**を選択します。
   - ***GUI 表示形式***：表示画面フォーム用
   - ***GUI 出力形式***：帳票フォーム用
   - ***テキスト形式***：他のアプリケーションに渡すためのテキストファイルやテキストプリンタのフォーム用
   - ***ブラウザ形式***：ブラウザクライアント用
   - ***リッチクライアント表示形式***：リッチクライアント用
5. **型**を選択します。クラスの選択内容によってここに表示される内容は異なります。ここでは GUI 表示フォームの項目が一覧されています。例えば、**テキスト**コントロールを定義する場合は、***S= スタティック***を選択します。
6. 最後に特性を変更します。特性値の表示内容は、選択したクラスと型に依存します。
7. ここでは、他の表示より太字で表示させたいため、**フォント**特性の指定を変更します。また、**スタイル**特性をデフォルトの *3= 凸立体*から *2= 平面*に変更しています。

モデルの定義を行った後に、作成したコントロールにこのモデルを割り当てることで書式が自動的に設定されます。

参照： （*「プロジェクトのデータ項目とコントロールを標準化するには」（38 ページ）*）

再利用可能なデータオブジェクトを定義するには

大規模なアプリケーションを開発する上で重要な点の 1 つにデータ項目の一貫性を維持することがあります。例えば、住所用のデータ項目を考えてみます。このデータ長が、あるプログラムでは 30 桁に定義されており、別のプログラムでは 40 桁に定義されていることは問題が発生する元になります。また、起動元のレコード ID が 10 桁であるのに対し、起動先のプログラムのパラメータとして定義されたレコード ID が 8 桁になっていては不都合です。

また、項目の有効な値は一貫性があり、開発者が確認しやすいようにすべきです。例えば、ステータスコードが定義されている場合、開発者はそのコードの値が何を意味しているかを知っている必要があります。

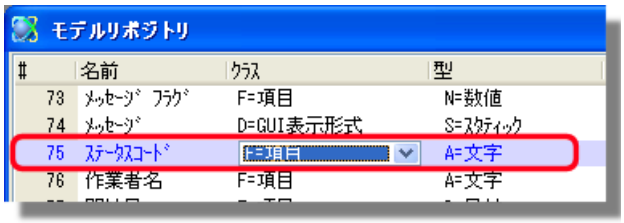
そして最も重要なことは、項目長を変更した場合、その項目を使用するすべてのデータソースやプログラムに反映される必要があることです。

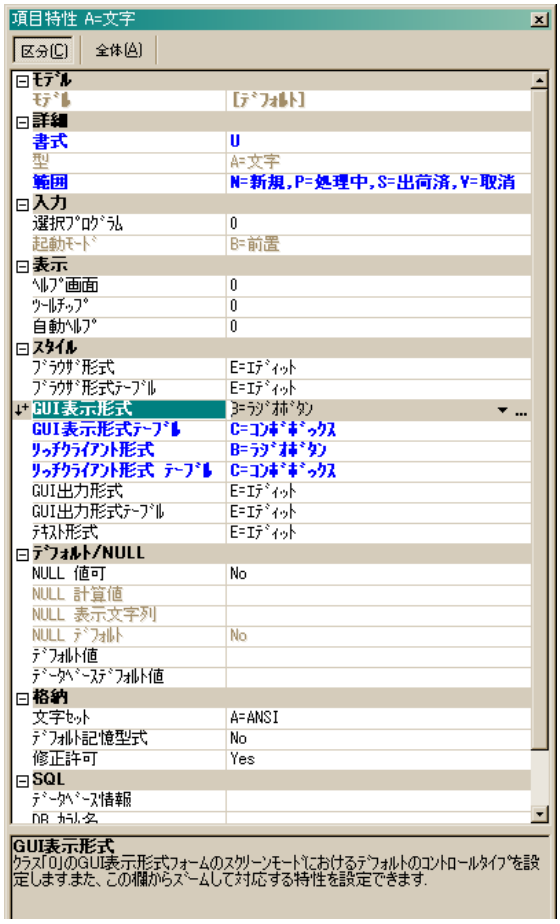
モデルを使用することで、Magic はこれらのことを容易に行うことができます。データにモデルが割り当てられている場合、そのモデルはデータの項目長と有効な値を定義することができます。さらに、モデルは実際の DBMS での記憶形式や、カプセル化された選択プログラム、ヘルプ画面、自動ヘルプ、ツールチップなどを定義することができます。設定内容は、開発者が簡単にアクセスすることができるため、データのタイプを開発時に確認することができます。

モデルの設定は容易に変更することができます。ステータスコードを追加する必要がある場合、ステータスコード用のモデルを変更することで、ステータスコードを使用しているデータソースやプログラムは自動的に変更されます。さらに、変更することでロジックに影響することがないかどうかを確認するために、**クロスリファレンス**（***Ctrl+F***）ユーティリティを利用することでステータスコードを使用しているすべてのオブジェクトを検索することができます。

データを定義するモデルは、**項目**クラスを指定することで作成することができます。

項目モデルを作成する





1. モデルリポジトリに移動し（*Shift+F1*）、目的の行にカーソルを位置付けます。
2. *F4*（編集→行作成）を押下します。
3. モデルの名前を入力します（ここでは、ステータスコードです）。
4. クラスカラムで *F= 項目* を選択します。
5. 型を選択します。項目モデルでは、以下のデータ型が選択できます。文字、Unicode、数値、日付、時刻、BLOB、ActiveX など。
6. 次に、モデル特性（*Alt+Enter*）を開き、必要な特性を設定します。ここにはたくさんの選択項目があります。各特性値の内容は『リファレンスヘルプ』を参照してください。

   詳細： このセクションでは項目の書式や長さを定義することができます。範囲特性に連続値や不連続値を指定することで、有効な値を指定することもできます。ここでは、1 桁の大文字で、4 つの範囲値が指定されています。

   入力： 値を選択するために *F5* を押下するか、ダブルクリック（ズーム）することで起動されるプログラムを指定できます。

   表示： ここでは、ヘルプ画面や、ツールチップ、ステータス行に表示される自動ヘルプを定義することができます。

   スタイル： このセクションでは、データが表示されるフォームごとにどのように表示されるかを指定することができます。左の例では、ステータスコードが GUI 表示フォーム上ではラジオボタンとして表示され、テーブル形式の GUI 表示フォームではコンボボックスとして表示されるように設定されています。

   デフォルト /NULL: NULL がどのように使用されるか、またデフォルト値を持つかどうかを指定します。

   記憶形式 /SQL: データが DBMS 内でどのように保存されるかを指定します。

項目モデルを使用することで、データの書式がどのように設定され、アプリケーションでどのように使用されているかを制御することができます。項目モデルが作成された場合、該当するデータタイプであればどこでも使用することができます。上記の場合、レコード内のカラムとして指定したり、パラメータとして渡されたり、プログラム内の変数として使用することができます。

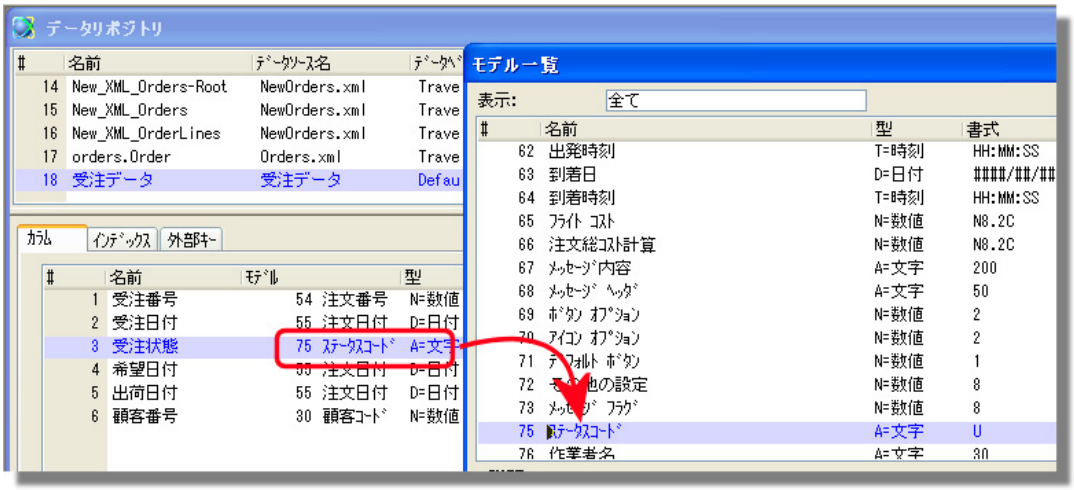
参照： 「プロジェクトのデータ項目とコントロールを標準化するには」（38 ページ）
「モデルを使用してデータソースのカラムを定義するには」（37 ページ）

モデルを使用してデータソースのカラムを定義するには

データを一貫性を持って定義することは非常によいことです。モデルを使用することで一貫性を実現することが容易になります。一度、データモデルを定義すると、すべてのデータソース内で同じデータ型を持たせることができます。

モデルを使用してデータソースのカラムを定義する

必要条件： 項目モデルは、すでにモデルリポジトリに定義されているものとします。



1. データリポジトリのカラムタブに移動します。
2. F4（または、編集→行作成）を押下して、カラムを追加します。
3. 名前カラムは空白の状態にしておいてもかまいません。モデルを指定すると、モデル名が継承されて名前カラムに設定されます。
4. モデルカラムでズームすると（*F5* またはダブルクリック）選択可能なモデルが一覧表示されます。ここでは、ステータスコードモデルを選択します。

ここで追加されたカラムの特性シートを表示します。ステータスコードモデルから継承された特性値が表示されているはずです。これは、モデルを使用することでステータスコードの関する設定を行う必要がないことを意味しています。もし、ステータスコードモデルの特性値を変更すると、カラムの特性値も変更されます。

必要であれば、この特性値をオーバライドすることもできます。特性値がオーバライドされた場合、モデルとの継承関係が切れ、モデルの特性値が変更されてもこれらの特性値は変わらなくなります。

参照： 「再利用可能なデータオブジェクトを定義するには」（35 ページ）

プロジェクトのデータ項目とコントロールを標準化するには

どのようなプロジェクトでもデータに関する２種類の問題が考えられます。

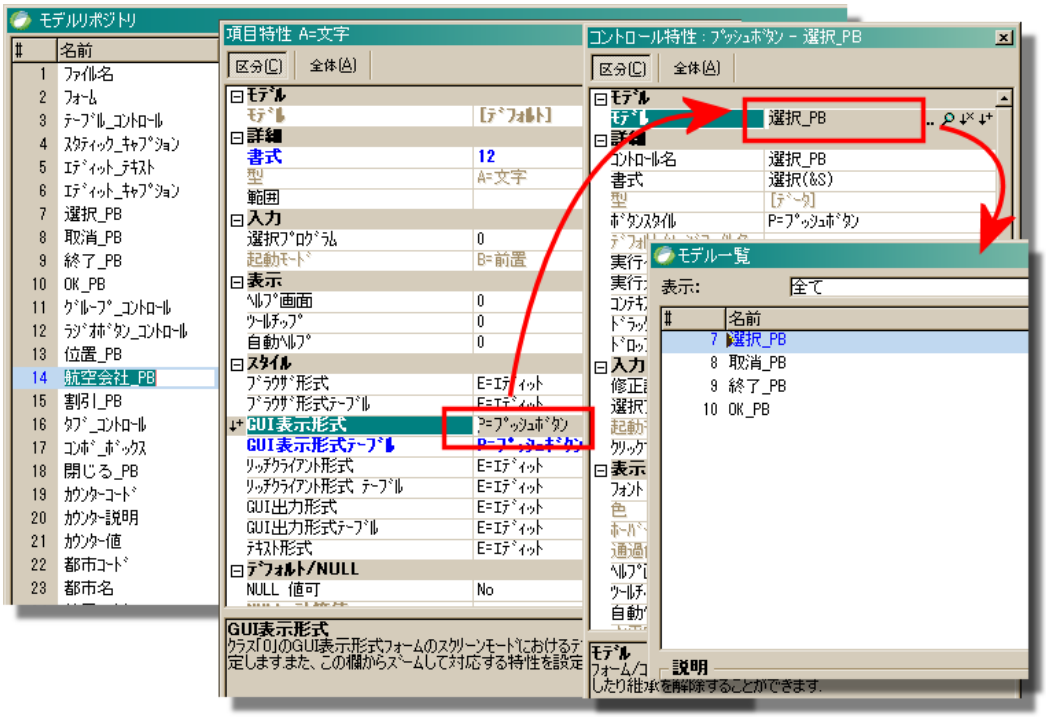
1. データ自身について
   - 各項目の長さがどれくらいか？、
   - どのような形式で保存するか？
   - データタイプは何にするか？
   - どの値を正しいものとするか？

   などです。開発時は、**項目モデル**としてこの情報をカプセル化します。
2. データをどのように表示させるか、またユーザがどのようにアクセスするようにするか？
   コントロールモデルとなる **GUI 表示形式**, **GUI 出力形式**, **リッチクライアント形式**, **ブラウザ形式** そして **テキスト形式** の各モデルは、ルック & フィールをカプセル化します。

フォームにデータ項目を配置したとき、コントロールのすべての特性値を設定するかコントロールのモデルを選択することができます。また、**項目モデル**の一部としてコントロールモデルを指定することもできます。この２つはもっとも基本的なレベルでリンクされています。これによって、プログラミングにかかる時間を短縮し、フォーム上のコントロールとして表示されるデータ形式を標準化することができます。

項目モデルに対するコントロールを定義する

必要条件： **コントロール**モデルはすでに定義されているものとします。



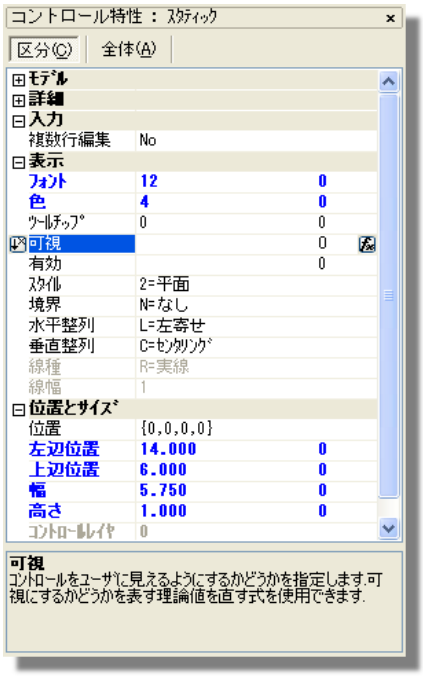
1. **項目モデル**にカーソルを位置付けます。
2. モデルの**特性シート**を開きます（*Alt+Enter*）。
3. **スタイル**セクションの変更したいフォームに移動します。例えば、スクリーンモードでのウィンドウに表示される内容を変更したい場合は、**GUI 表示形式**特性に移動します。
4. ここで**ズーム**すると**コントロール特性**と表示されるペインが新たに表示されます。必要であれば、ここに特性値を設定することができますが、ここではモデルを設定します。
5. **コントロール特性**内の**モデル**特性 から**ズーム**（*F5* または**ダブルクリック**）して一覧からモデルを選択します。

これによって**項目モデル**が**コントロールモデル**とリンクされました。**項目モデル**が、データを定義するために使用されと、**コントロールモデル**で指定された**スタイル**特性を持つことになります。

参照： 「再利用可能なデータオブジェクトを定義するには」（35 ページ）

モデル特性の変更内容をオブジェクトに反映させないようにするには

すでにモデルが定義されている場合、モデルの内容を変更するとそのモデルを使用しているすべてのオブジェクトに反映されます。これはとても効果的です。例えば、すべて***イタリック体***に設定されている項目を標準に戻したい場合、モデルを変更することですべての表示が簡単に変更されます。



ここで、特定のウィンドウの表示だけモデルの変更を反映させたくない場合があるとします。この場合、継承関係を***解除する***必要があります。Magic では、特性値を変更することで表示が**青いボールド表示**となり、継承が***解除された***ことを示すようになっています。

注： 継承状態を表す色とフォントは変更することができます。

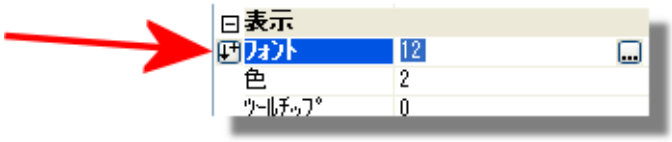
- 色を変更するには、**アプリケーション用基本色定義**テーブル（**設定→オプション→基本色→開発**）を開いて、#44 と #45 の色を変更します。
- フォントを変更するには、**アプリケーション用フォント定義**テーブル（**設定→オプション→フォント→開発**）を開いて、#34 と #35 のフォントを変更します。

右の例において、**フォント**特性と**色**特性は、**ボールド体の青色**になっています。これらは、継承関係が解除されていることを表しています。

継承関係を解除するには、手動と自動の２つの方法があります。

手動で継承を解除する

特性値の左側に表示されるアイコンを**クリック**することによって継承を解除する（継承させる）ことができます。



このアイコンは、特性値にカーソルを置いた場合のみ表示されます。アイコンが  と表示されている場合は、**クリック**すると継承が解除され、特性値が**青色**に変わります。アイコンが  と表示されている場合は、**クリック**すると継承された状態になり、特性値は黒色に変わります。

自動的に継承を解除する

オブジェクトの特性値を変更するとその特性に対する継承関係は自動的に解除され、特性値が**青色**に変わります。もしオブジェクトの継承された**色**特性が **2** の場合、この値を **4** に変更すると **色**特性の継承関係は解除されます。

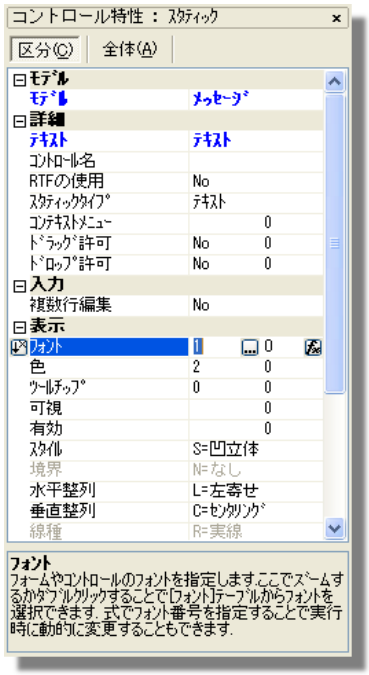
参照： 「継承が解除された特性を継承させるには」（40 ページ）

継承が解除された特性を継承させるには

特性シートでモデルが定義されているオブジェクトの特性値を変更すると、変更された特性はモデルと継承関係が解除されてしまいます。一端継承関係が解除されると、モデルの特性値を変更してもその特性値には反映されません。

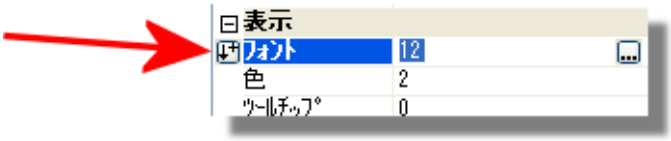
一端解除された継承関係を再び有効にするには、特性値の左側に表示されるアイコンを使用します。

.



特性値を継承させる

特性値の左側に表示されるアイコンをクリックすることによって継承させることができます。



このアイコンは、特性値にカーソルを置いた場合のみ表示されます。アイコンが と表示されている場合は、クリックすると継承が解除され、特性値が青色に変わります。アイコンが と表示されている場合は、クリックすると継承された状態になり、特性値は黒色に変わります。

モデルのクラスを変更するには

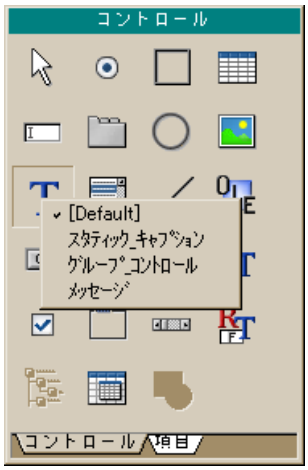
モデルは一度作成されると、クラスを変更することはできません。モデルを作成し値を設定した後、カーソルを別の行に移動するとクラスの設定は確定されます。何らかの理由で間違ったクラスを指定されたモデルを作成してしまった場合、*F3*（**編集→削除**）を押下してそのモデルを削除し、再度作成し直す必要があります。

そのモデルが他のオブジェクトから参照されているかどうかを確認する場合は、削除する前に *Ctrl+F*（**編集→検索と置換→クロスリファレンス**）を押下して使用しているオブジェクトを検索してください。

参照： 第 1 章：「クロスリファレンスを使用する」（6 ページ）

コントロールモデルを使用して、フォームにコントロールを自動配置するには

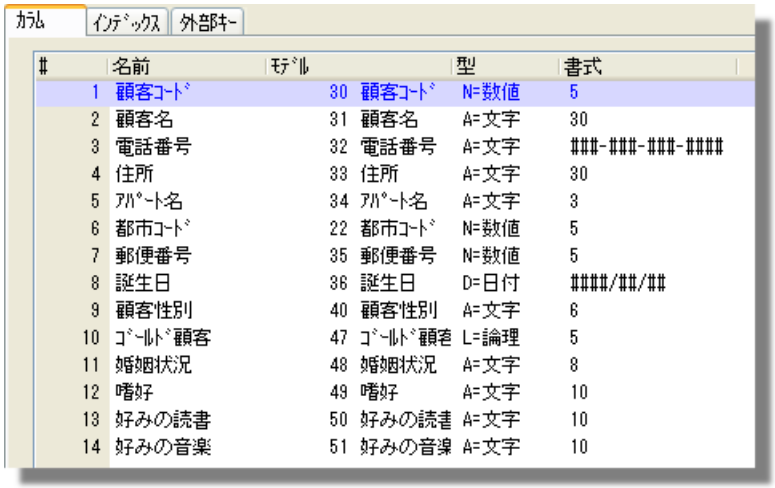
Magic のフォームエディタにはコントロールパレットがあり、ここをクリックすることでコントロールをフォームに配置することができます。この場合、コントロールの特性はデフォルトの設定になっています。しかし、モデルを指定してコントロールを配置することもできます。



パレットからモデルを指定してコントロールを選択する

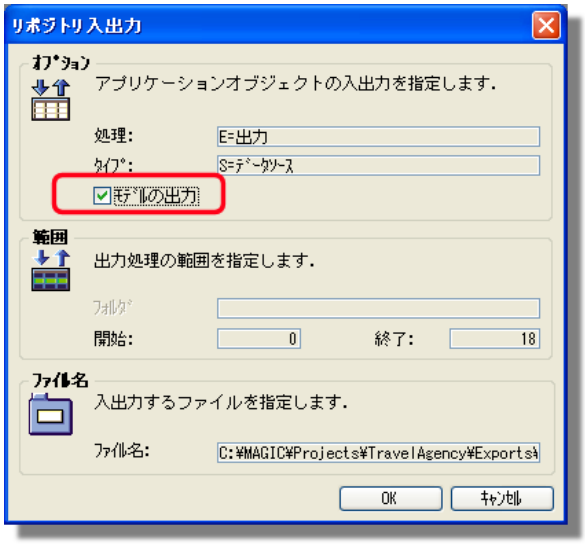
1. コントロールパレットにカーソルを移動します。
2. 配置したいコントロールの上で右クリックします。
3. 設定可能なモデルの一覧が表示されます。設定したいモデル名に移動し、左クリックします。
4. カーソルアイコンの表示が、選択されてことを示すように変わります。フォーム上のコントロールを配置したい場所にカーソルを置き、左クリックするとコントロールが設定されます。

モデルを含めてプログラムやデータソースを出力するには



モデルはとても有効的なので、データソースやプログラムなどによく利用されます。しかし、これらをリポジトリ入出力する場合は、多少複雑になります。なぜなら、入力するプロジェクト内に同じモデルが存在している必要があるからです。

このような場合の簡単な対処方法として、リポジトリ入出力ダイアログに表示されるモデルも出力する オプションを使用することです。このオプションを使用することで、データソースやプログラムが使用しているモデルも一緒に出力されます。



モデルを含めて出力する

1. リポジトリ出力を行う場合、モデルも出力するをチェックします。
2. このリポジトリファイルを入力すると、出力されたモデルが現在のモデルリポジトリの最後に追加されます。これらのモデルへの参照情報が更新され、データソースやプログラムが正しく継承されるようになります。

モデルを持ったプロジェクトにデータソースを入力すると、モデルの項番は異なりますが、入力したデータソースが使用してたモデルが新しいプロジェクトに追加されます。

参照： 第 2 章：「別のプロジェクトにオブジェクトを転送するには」（31 ページ）

複数のプロジェクトでモデルを共有するには

異なるプロジェクト同じ内容のモデルを使用する必要があります。リポジトリ入出力によってモデルを別のプロジェクトにモデルをコピーすることもできますが、プロジェクト間でモデルを共有することの方が保守も楽になります。また、これによってプログラムを変更することなくモデルの機能を変更することができます。例えば、日付項目に対応したポップアッププログラムとしてカレンダーを追加したい場合、またはアプリケーションが実行している国の通貨に対応したコンポーネントが必要な場合などが考えられます。

#	名前	クラス	型	フォルダ	公開名
9	終了_PB	D=GUI表示形式	P=ﾌﾟｯｼｭﾎﾞﾀﾝ	GUI	Exit_PB
10	OK_PB	D=GUI表示形式	P=ﾌﾟｯｼｭﾎﾞﾀﾝ	GUI	OK_PB
11	ｸﾞﾙｰﾌﾟ_ｺﾝﾄﾛｰﾙ	D=GUI表示形式	S=ｽﾀﾃｨｯｸ	GUI	Grouup_CTL
12	ﾗｼﾞｵﾎﾞﾀﾝ_ｺﾝﾄﾛｰﾙ	D=GUI表示形式	B=ﾗｼﾞｵﾎﾞﾀﾝ	GUI	Radio_CTL
13	位置_PB	F=項目	A=文字	GUI	Position_PB
14	航空会社_PB	F=項目	A=文字	GUI	Airline_PB
15	割引_PB	F=項目	A=文字	GUI	Discount_PB
16	ﾀﾌﾞ_ｺﾝﾄﾛｰﾙ	F=項目	A=文字	GUI	Tab_CTL
17	ｺﾝﾎﾞ_ﾎﾞｯｸｽ	D=GUI表示形式	C=ｺﾝﾎﾞﾎﾞｯｸｽ	GUI	Combo_CTL
18	閉じる_PB	F=項目	A=文字	GUI	Close_PB
19	ｶｳﾝﾀｰｺｰﾄﾞ	F=項目	N=数値	ｶｳﾝﾀ	Counter_Code
20	ｶｳﾝﾀｰ説明	F=項目	A=文字	ｶｳﾝﾀ	Counter_Discription
21	ｶｳﾝﾀｰ値	F=項目	N=数値	ｶｳﾝﾀ	Counter_Value
22	都市ｺｰﾄﾞ	F=項目	N=数値	都市	City_Code
23	都市名	F=項目	A=文字	都市	City_Name
24	位置ｺｰﾄﾞ	F=項目	N=数値	位置	Position_Code
25	位置名	F=項目	A=文字	位置	Position_Name
26	位置画像	F=項目	A=文字	位置	Positoin_Picture

コンポーネントとしてモデルを共有する

1. よく使用するモデルを定義します。
2. 共有する際に参照する公開名をモデルに指定します。
3. コンポーネントの作成とインストール処理を実行します。

コンポーネントをプロジェクトに定義した場合、コンポーネントモデルはモデル一覧から選択できるようになります。コンポーネントモデルは、プロジェクトのモデルリポジトリに定義されているものと同じように扱うことができます。

参照： 第 16 章：「プロジェクト間で Magic のオブジェクトを再利用するには」（357 ページ）

第 4 章： Magic エンジン

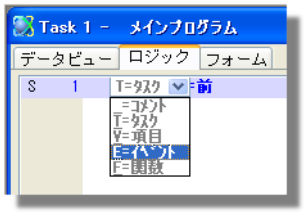
アプリケーションレベルのイベントを定義するには

Magic では、タスクが実行中に発生させることのできるイベントを定義することができます。しかし、どのタスクが実行中でもまたは、タスクが実行していなくても発生させることのできるイベントが必要な場合があります。本章の例は、数分ごとにメッセージをチェックするメッセージ通信システム、または処理されないデータベースエラーに対してメッセージを表示させるグローバルなエラーハンドラ、あるいはホットキーに設定されたポップアッププログラムについて説明しています。これらはどのタスク上でも発生させることができます。

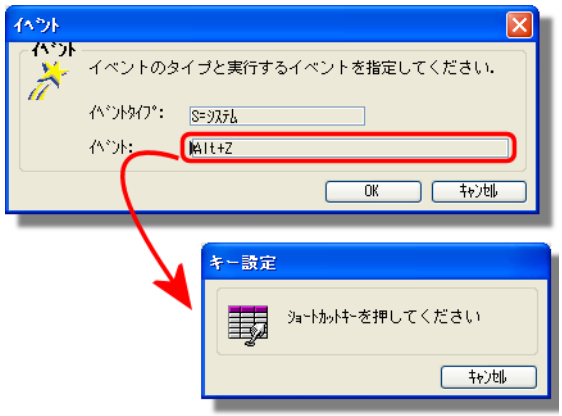
これらのグローバルイベントは、プロジェクトのメインプログラムで定義します。

グローバルイベントを作成する

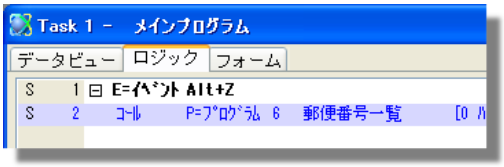
1. メインプログラムを開きます。
2. ロジックタブ（*Ctrl+1*）をクリックします。
3. *F4*（編集→行作成）を押下して行を追加します。2 つの新規行が表示されます。最初の行に移動し、ドロップダウンリストからイベントを選択します。



4. *Tab* で右に移動すると、イベントダイアログが表示されます。
5. 使用するイベントタイプを選択します。この例では、キー操作によるシステムイベントを選択しています。
6. イベントでズーム（*F5* または、編集→ズーム）するとキー定義ダイアログが表示されます。
7. トリガとして定義するキー操作を押下します。この場合は、*Alt+Z* を設定しています。



8. これで、*Alt+Z* を押下することで発生するグローバルイベントが定義できました。次にこのイベントが発生した場合のロジックを定義する必要があります。
   ここでは、郵便番号一覧というプログラムを起動するようにしています。このプログラムは、郵便番号コードの一覧を表示するものです。



参照： 第 11 章：「特定のコントロールにパークしている時にのみ実行されるイベントロジックユニットを定義するには」（232 ページ）

Magic エンジンをイベント駆動型で動作させるには

COBOL の場合、プログラムは主に手続き型です。すなわち、プログラムは上から下にロジックが実行されるように記述します。プログラムの開始処理以外、ユーザからの入力処理はほとんどありません。

今日、ほとんどのプログラムはイベント駆動型です。すなわちプログラムは、実行されるとイベントの発生を待つ状態になります。イベントが発生するとそれに対応した処理を実行します。

簡単な例として、Web サイトや SOAP サービスがあります。Web サイトは常にそこにあり、キーを押したり、特定の場所でマウスカーソルが通過するでそれに対応した処理が実行されます。SOAP サービスは、決まった書式でデータを送信することでデータが返されます。

Magic では、これら両方の種類のプログラムを作成することができます。その際、バッチプログラムであってもイベント駆動方式を利用することがよくあります。これは古い手続き型の方法より簡単で効率的なためです。

イベントのコンセプト

イベントの基本的な考え方は簡単です。何か（トリガ）が発生し、ロジック（ハンドラ）を実行することでプログラムが応答するというものです。

トリガはユーザによって発生されるものがよくあります。例えば、マウスカーソルを通過させたり、キーを押下したりする操作があります。カーソルを項目内に移動させたり、項目外に移動させたりすることもあります。キー操作が何も行われなければ、何も発生しないこともあります。一定の時間が経過することで発生する場合もあります。

しかし、プログラムはまた、他のプログラムと対話するためにイベントを発生することもあります。ActiveX オブジェクトがよい例です。これには、対応することのできるイベントのグループが含まれています。

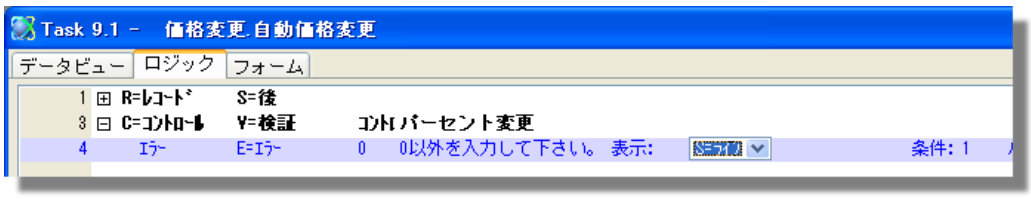
発生させることができる内容を以下に示しました。

発生内容	**Magic でどのように処理されるか**	
イベント	**ロジックハンドラ**	**サブタイプ**
タスク起動	タスク	前
タスク終了	タスク	後
新規レコード読込	レコード	前
ユーザがこのレコードを離れた	レコード	後
ユーザが項目にカーソルを置いた	コントロール	前
ユーザが別の項目にカーソルを移動した	コントロール	後
X 秒間キー操作を行わなかった	イベント	タイマイベント
ユーザがプッシュボタンを押下した	イベント	ユーザイベント
項目の値が変更された	項目変更	
時間が 11:03 AM になった。	イベント	式
ActiveX オブジェクトがイベントを発生させた	イベント	ActiveX
ユーザがキー操作を行った	イベント	システム
DBMS により重複レコードエラーが発生した	イベント	エラー

処理が可能なイベントが数百あっても、現実的には、通常よく利用する処理は Magic によって自動的に実行されます。例えば、明示的に DB テーブルを開いたり、閉じたりする必要はありません。また、初期設定やデフォルト値、データの妥当性をチェックするような処理は**項目モデル**によって処理されます。

また、モデルやフォーム上のコントロールに選択プログラムを定義したり、**チョイス**コントロールをデータソースとリンクさせたりすることで選択プログラムにリンクさせることができます。

処理したいイベント用に、タスクの**ロジックエディタ**に複数の行を追加することになります。ここでは項目への入力を必須化させるための簡単な例を挙げます。



必要条件： フォームにコントロールが配置されており、コントロール名が設定されているものとします。

ロジックユニットを作成する

1. タスクを開き、ロジックタブをクリックします（*Ctrl+1*）。
2. ロジックユニットを入力したい行に移動します。
3. *Ctrl+H*（編集→ヘッダ行作成）を押下してヘッダ行を作成します。
4. ロジックユニットのタイプとして *C= コントロール*を選択します。*Tab* で次のカラムに移動します。
5. コントロールロジックユニットのタイプとして *V= 検証*を選択します。*Tab* で次のカラムに移動します。
6. ズーム（*F5* または、ダブルクリック）して検証対象となるコントロールを選択します。一覧に選択したいコントロール名が表示されない場合は、フォームにコントロールが配置されていないかコントロール名が定義されていないかのどちらかです。

この設定によって、コントロールを通過するたびに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内でエラー処理コマンドを追加します。これにより、項目にデータが入力されるまでユーザは他の操作が何もできなくなります。

参照： イベント処理のホワイトペーパ
「イベントの階層を利用するには」（49 ページ）

イベントの階層を利用するには

イベントについて考える上で問題になる点の 1 つは、誰がイベントを処理するかということです。すなわち、イベントが実行された場合、処理できるタスクにはいくつかのレベルがあるかもしれないということです。

1 つの例とし DBMS エラーがあります。DBMS がエラーを発生した場合、エラーの内容に応じてエラーメッセージを表示するようにしたいと考えられます。また、データベースにすべての DBMS エラーを記録するようなグローバルなエラー追跡プログラムが必要な場合があります。

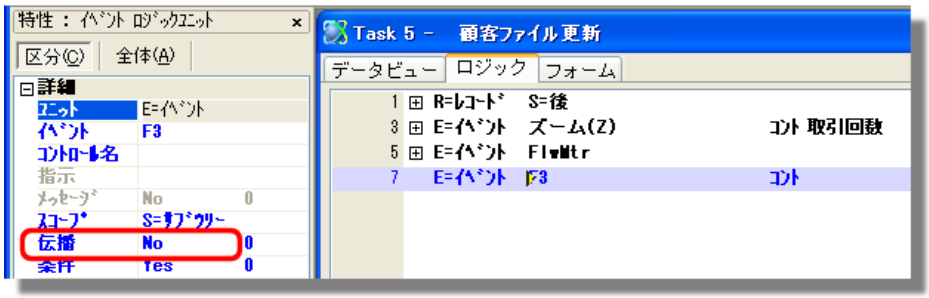
また、例えばキー操作を無効にする必要があるかもしれません。例えば、**F3** はデフォルトでは行を削除するために割り当てられており、**F3** を押下すると自動的に現在のレコードを DB テーブルから削除します。しかし、**F3** に対して削除レコードをマークするように処理を変更する必要があるかもしれません。また、**F3** に対して**ロジックユニット**内で関連するレコードを削除し、現在のレコードを削除させる機能を持たせる必要があるかもしれません。

このようなことは、**伝播**特性を使用することで実現できます。**伝播**特性が ***Yes*** に設定されている場合、**ロジックユニット**が実行され、さらにイベントが上位のタスクレベルに伝播されます。***No*** に設定された場合、現在の**ロジックユニット**が実行するだけで終わります。

Magic は、ロジック階層を決定するための決められた検索順序を使用します。それは、最初に特定のコントロールに対応したイベントが処理され、その後、対応していないコントロールに移ります。この検索処理はタスクツリーを上位方向へと実行され、伝播指定が有効であれば**メインプログラム**まで行われます。

では具体的な例で説明します。

システムイベントをブロックする



ここでは、***F3*** の動作をブロックするイベントがあります。ユーザが ***F3*** を押下するとこのイベントが起動されます。しかし、このイベントは伝播されず、Magic はこれを処理できません。このためレコードの削除は行われません。

システムイベントに機能を追加する

この**ロジックユニット**は、関連するレコードを削除するサブタスクを起動するように定義されています。***F3*** のシステムイベントを Magic に渡すと現在のレコードが削除されます。

ヒント : *この例では、キー操作をブロックする方法について説明していますが、通常は、行削除の内部イベントを使用するようにしてください。これは、ユーザがプルダウンメニューの行削除を使用したり、キー割付が変更されたり、プッシュボタンにイベントを割り当てることでプログラム的に削除される場合があるためです。行削除のイベントの使用方法については、「内部イベントをブロックする」 (50 ページ) を参照してください。*

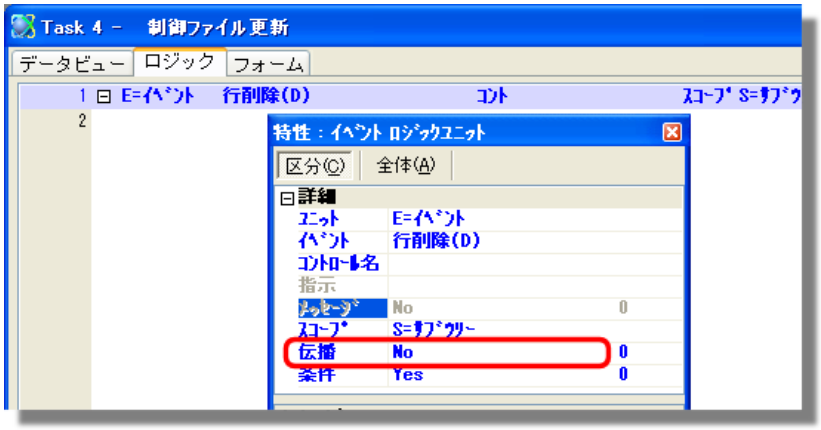
行削除のような内部イベントの発生を防止するには

Magic には、たくさんの組み込み機能を持っています。これによって開発工数を軽減することができます。行作成や削除、終了 のような共通のイベントはエンジンに組み込まれており、プログラムに組み込む必要はありません。

デフォルトのキーボード割付やメニューの設定を変更することでシステムレベルのイベントをブロックすることができます。また、タスクオプション（タスク特性→オプションタブ）内のパラメータを無効にすることで、イベントが発生しても実行させないようにすることもできます。また、内部イベントを受け取るロジックユニットに処理を組み込むことでタスクレベルでこれらを制御したりオプション化することもできます。

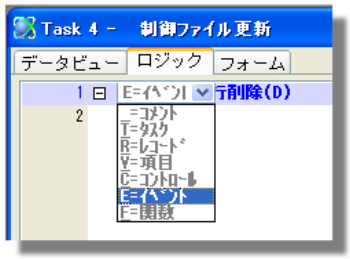
この例では、受注状態が N（新規）でない場合、行削除イベントをブロックします。

内部イベントをブロックする

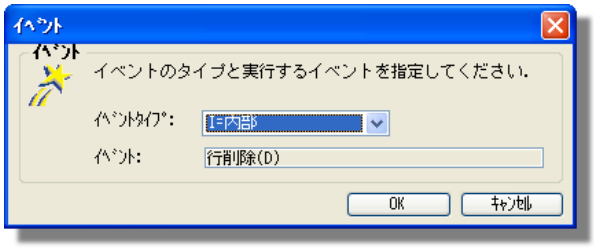


ここでは、受注状態 <>'N' の場合に行削除イベントをブロックするロジックユニットを例に説明します。このロジックユニットではユーザに対してメッセージを表示するようにする必要があるかもしれませんが、ブロック処理には必要ありません。以下にこのイベントをどのように定義するかを説明します。

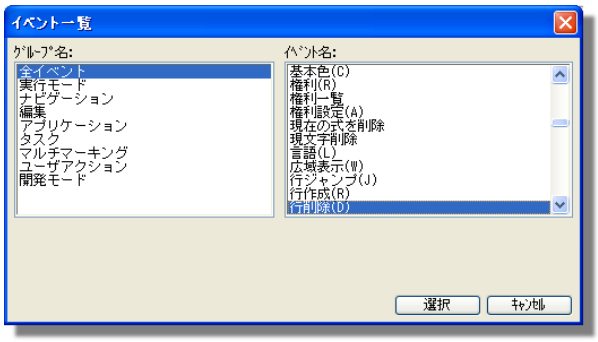
1. タスクを開き、ロジックタブ（*Ctrl+1*）をクリックします。
2. ロジックユニットを入力したい行に移動します。
3. *Ctrl+H* を押下し、ヘッダ行を作成します。
4. ヘッダタイプで *E= イベント*を選択します。



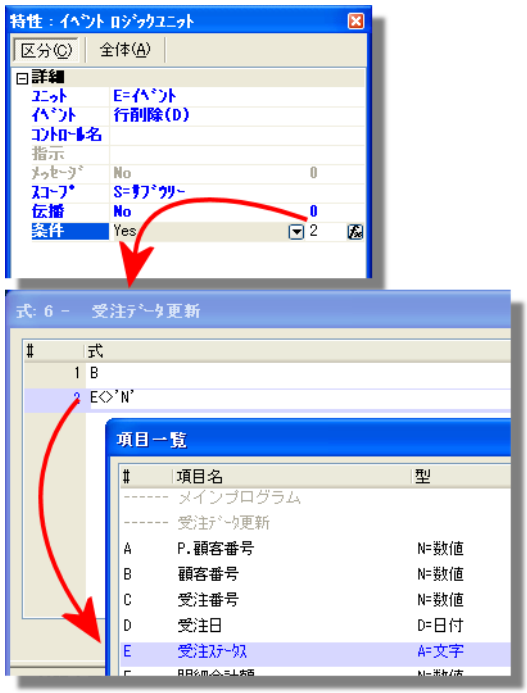
5. ダイアログボックスが表示されます。イベントタイプで *I= 内部*を選択します。*Tab* で次のイベントに移動します。

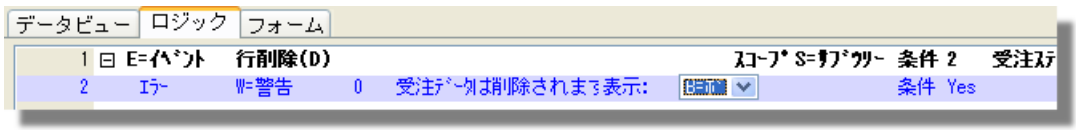


6. *Tab* を押下するとイベント一覧ダイアログが表示されます。
   選択したいイベント名の文字を入力することで該当するイベントに位置づけられ選択することができます。この場合は、行と入力することで*行*という名前を持つイベント名に位置づけることができ、カーソルキーで行削除のイベント名の位置付けることができます。
   使用したいしたいイベント名が見つかったら、選択ボタンを押下するか *Enter* を押下します。



7. もう一回 *Enter* を押下し、イベントダイアログを閉じます。
8. これで行削除イベントが発生した場合に実行されるイベントロジックユニットが定義できました。
   イベントが伝播されることを防ぐため、イベント特性シート（*Alt+Enter*）を開き、伝播特性を *No* に設定します。
9. 行削除イベントトリガが下位のタスクに渡らないようにため、スコープ特性を *T= タスク*に設定します。
10. これで行作成 イベントが発生すると必ずこのロジックユニットが実行されるようになります。
    しかし、ステータスコードが N でない場合のみ実行するようにしなければなりません。
    この処理を実現するには条件特性でズーム（F5, またはダブルクリック）し、式エディタを開きます。ここで条件式を定義します。参照したい項目を見つける場合は、ズームを行い、関数を見つかるには右クリックを行います。（第 21 章：「式エディタ内で式の体裁を整えるには」（461 ページ）を参照してください。）





これで作成できました。このロジックユニットにメッセージを表示するなどの処理を追加したりすることができます。

参照 : 「Magic エンジンをイベント駆動型で動作させるには」（47 ページ）

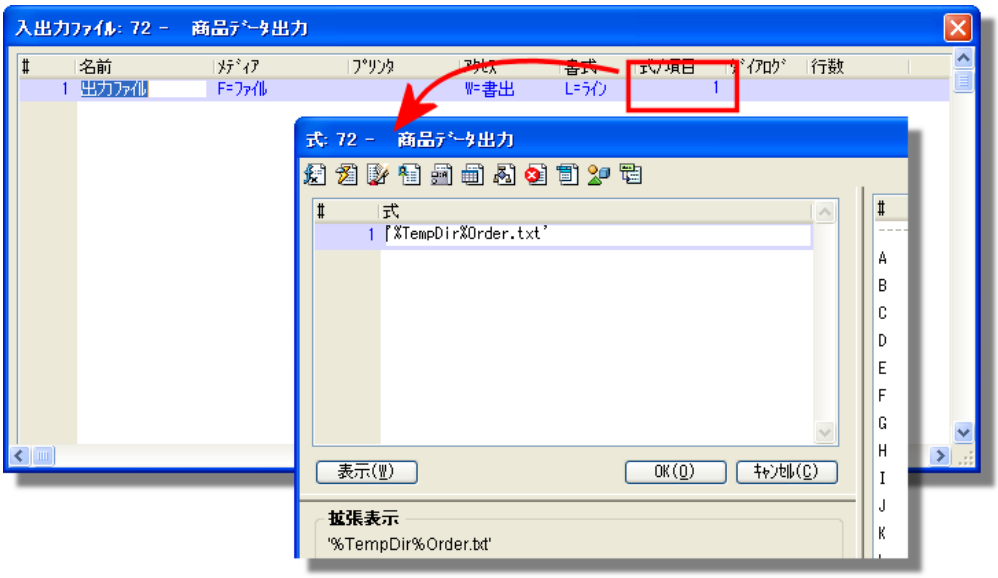
入出力ファイル名に動的な名前を設定するには

Magic で入出力ファイルを定義 / 利用する場合、ファイル名の定義処理に対して、いくつかのオプションがあります。

- 固定のファイル名を指定できます。（例：'C:¥Temp¥Report.txt'）
- データ項目で指定できます（例：**Trim(*AF*)**）。指定されるデータ項目は、データソースのカラムであったり、ユーザによって入力されたデータの場合もあります。ファイル名をランダムに指定したり、項目内で保存することもできます。一時ファイルを使用する場合に有効です。
- ファイル名の一部または全部を***論理名***を使用して設定することができます。パス名を実行時に動的に指定するうえで有効な方法です。よく利用する論理名に、**%TempDir%** があります。これは、Magic エンジンが自動的に設定するものです。

上記のすべての場合において、ファイル名は式エディタで指定されます。ここでは、テキストファイルを出力する方法について説明します。

入出力ファイル名を設定する



1. 最初に、テキストファイルを出力するタスクを開きます。ナビゲータペイン（*Alt+F1*）からタスクをクリックすることで開くことができます。
2. *Ctrl+I*（タスク→入出力ファイル）を押下し、入出力ファイルテーブルを開きます。まだ出力ファイルが定義されていない場合、以下のようにして作成します。
   - *F4*（編集→行作成）を押下して 1 行追加します。
   - 名前カラムに出力ファイルの名前を入力します。
   - メディアカラムでは、*F= ファイル*を選択します。
   - アクセスカラムでは、*W= 書込*を選択します。
   - 書式カラムでは、*L= ライン*を選択します。
3. 次に式カラムに移動し、ズーム（*F5* または、ダブルクリック）して式エディタを開きます。
4. ファイル名が返る式を入力します。ここでは、**'%TempDir%Orders.txt'** が入力されています。指定された論理名が正しく評価される限り、この論理名とファイル名の組み合わせは常に使用することができます。

このプログラムを実行させると、テキストファイルが Windows の一時ディレクトリに作成されます。

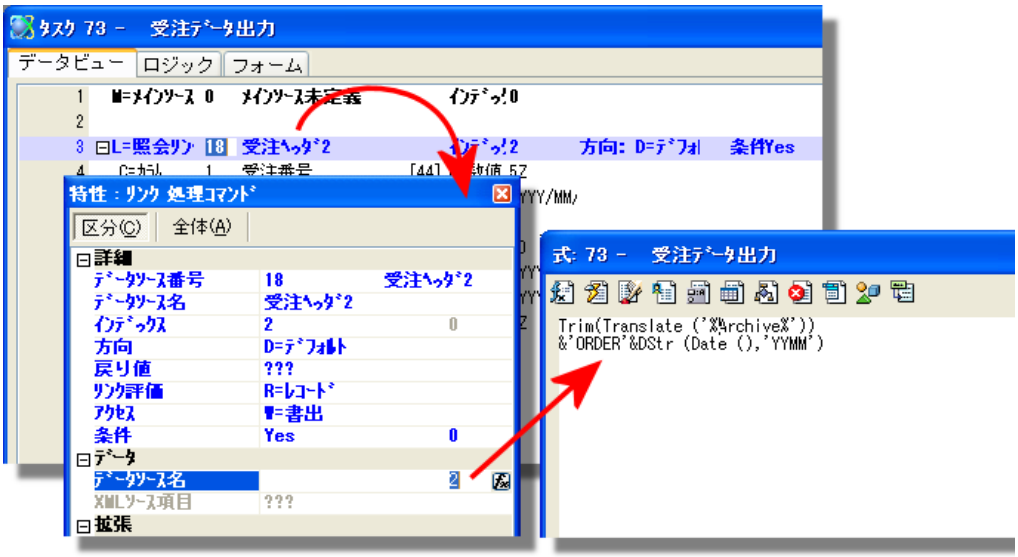
参照： 第 21 章：「式エディタ内で式の体裁を整えるには」（461 ページ）
第 2 章：「プロジェクトのディレクトリのファイルを読み書きするには」（26 ページ）

データソースに動的な名前を設定するには

通常、データソース名はデータリポジトリで設定されます。これによって保守が容易になります。一カ所で管理されていることで、データソース名を探したり、変更することが容易になります。

しかし、タスクレベルでデータソース名を変更したい場合があります。例えば、ユーザ毎に個別のテーブルを定義したり、同じデータ定義で日付によって異なるデータソース名にするようなことが考えられます。

タスクレベルでデータソース名を指定する

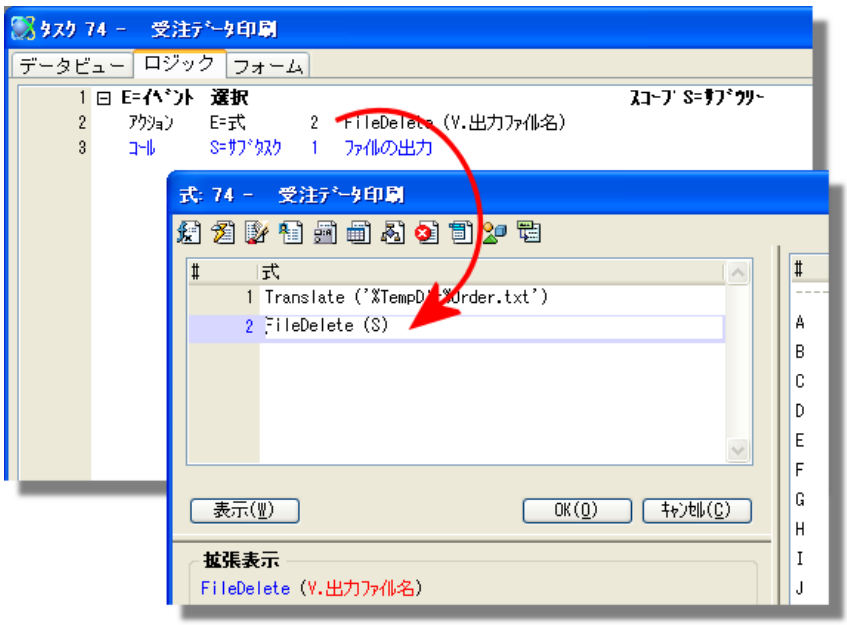


この例では、受注用のヘッダーレコードを複写するサブタスクがすでに定義されているものとします。ここではデータリポジトリ内の同じオブジェクトを開いてはいますが、**データソース名**特性を指定することで、デフォルトのデータソース名を上書きしています。データソース名は、実行時の年と月にもとづいて定義され（例えば、2007 年 12 月の場合は、'0712'）、論理名 **%Archive%** で指定されたディレクトリ内に作成 / 修正されるようになっています。

注： 同じタスク内では、同じデータソースに対して異なる名前を定義することはできません。この例では、親タスクがデフォルト名で同じデータソースを開くようになっているため、サブタスク内では保管用のレコードを作成しています。

参照： 第 18 章：「Magic でデータベーステーブルを作成するには」（395 ページ）

同じファイルやデータソースを扱っているタスク内でファイルやデータソースを削除するには



Magic のタスクが実行されると、プログラムの制御を行うことができる前に、そのタスクが使用するすべてのファイルとデータソースはオープンされます。このため、そのタスク内でファイルやデータソースを削除することはできません。

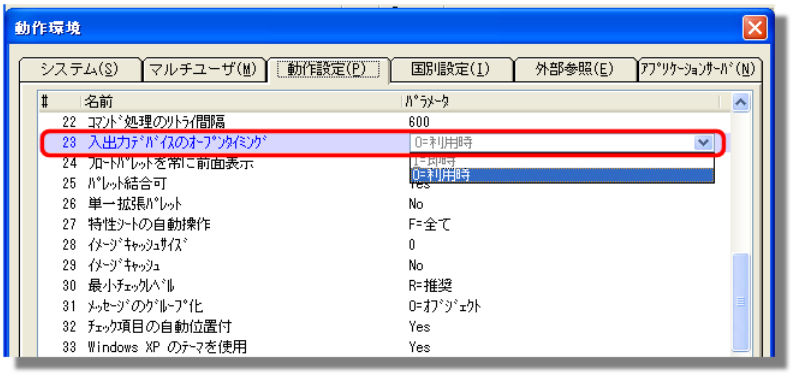
しかし、一般的には、タスクが実行される前にファイルを削除する必要があります。これを実現するには、親タスクでファイルを削除し、その後でタスクを起動する用にします。

この例では、古いファイルを削除し、ファイルを作成するタスクを呼び出すロジックユニットが定義されています。ここでは、**FileDelete** 関数と、ファイル名を格納する変数「C」を使用しています。これらの処理は親タスク内で実行されるため、ファイルを開くサブタスクと競合することはありません。

出力する内容がない場合にファイルを作成したり空白ページを出力させないようにするには

デフォルトでは、タスクが起動される前に出力ファイルやプリンタがオープンされます。これは旧バージョン（Ver8）以前での Magic の仕様でした。しかし、タスクが何も出力しないことがわかっていれば、出力処理を実行しないようにするはずです。

開発者は、処理毎に出力内容が存在するかどうかは判別できません。従って、空データを出力することを防ぐには、実際にデータの存在が確認できるまで出力ファイルのオープンを遅らせる必要があります。



入出力ファイルをオープンするタイミングを指定する

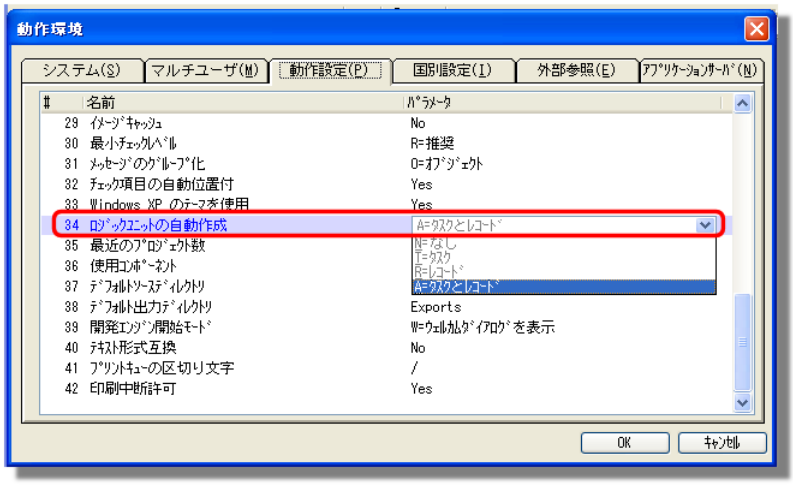
1. **動作環境**ダイアログ（**設定→環境設定**）を開き、**動作設定**タブを**クリック**します。
2. 行 #23 **入出力デバイスのオープンタイミング**に移動します。
3. ***U= 利用時***を選択します。

これにより、Magic エンジンが**ファイルの書込**処理を実行するまで出力デバイスはオープンされなくなります。

タスク作成時にタスクレベルのロジックユニットを自動的に作成するようにするには

旧バージョンの Magic を使用している場合は、プログラム内にタスクとレコードのロジックユニットが常に表示されていることに慣れていると思います。これらは、実際に使用されていなくても常に表示されていました、

Magic V10 では、デフォルトではロジックユニットは作成されません。必要に応じて、*Ctrl+H* を押すことで作成することになります。しかし、これらを自動的に作成したいようであれば、以下のように環境設定を行ってください。



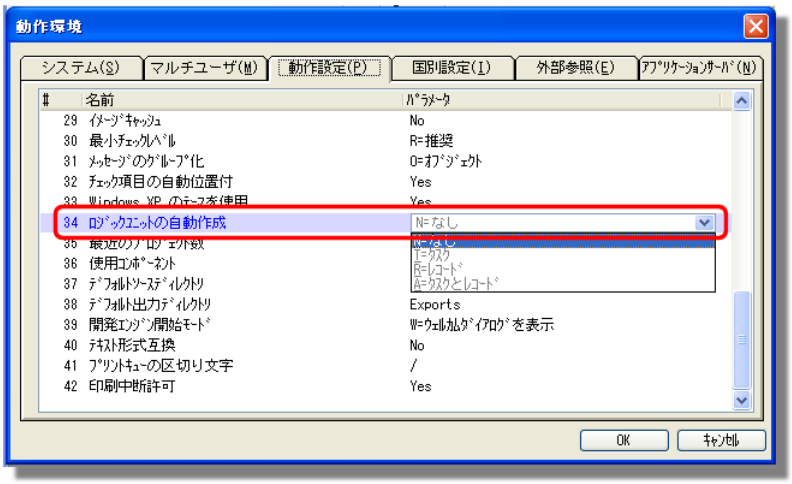
タスクとレコードレベルのロジックユニットを自動的に作成する

1. **動作環境**ダイアログ（**設定→環境設定**）を開き、**動作設定**タブを**クリック**します。
2. 行 #34 **ロジックユニットの自動作成**に移動します。
3. ***A= タスクとレコード***を選択します。

これで新規タスクを開くと、タスクとレコードレベルのロジックユニットが自動的に作成されます。

タスク作成時にロジックユニットが作成されないようにするには

Magic が自動的にロジックユニットを作成するかどうかは、環境設定で決まります。新しいタスクを作成するたびに、ロジックユニットが作成されているようであれば、以下の操作でこの機能を止めることができます。



ロジックユニットの自動作成を止める

1. 動作環境ダイアログ（設定→環境設定）を開き、動作設定タブをクリックします。
2. 行 #34 ロジックユニットの自動作成に移動します。
3. *N= なし*を選択します。

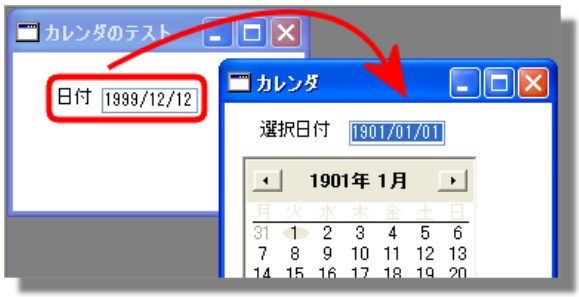
これで新規タスクを開いても、ロジックユニットは作成されなくなります。

イベント処理中にエディットコントロールの値を取得するには

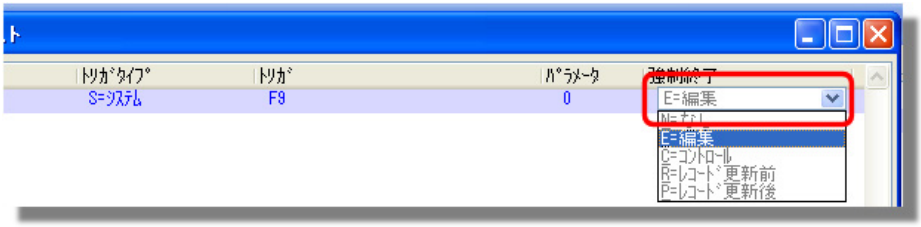
デフォルトでは、カーソルがエディットコントロール上に位置している間に、ハンドラが実行されても、そのコントロールの値は読み込まれません。これは、ユーザが新しい値を入力しても、そのコントロールから移動するまで、入力されたそはデータ項目内に存在していないためです。

これは、コントロール操作で起動されるハンドラによって発生する問題です。例えば、ユーザが *F9* を押下すると、表示されるカレンダが定義されているものとします。

デフォルトの日付は「1901/01/01」です。「1999/12/12」と入力し *F9* を押下します。しかし、起動されたカレンダは修正前の「1901/01/01」が表示されています。これでは、期待する動作にはなりません。



この動作を変更するには、ユーザイベントテーブルの強制終了カラムを変更する必要があります。



強制終了を *E= 編集* に設定することで、ハンドラが実行される前に編集モードの状態のまま、入力された値によって項目が更新され、またこれによって再計算が実行されます。

この設定により日付のデータは期待された値で渡されます。

ヒント :強制終了 *特性は、ユーザイベントでのみ有効です。キー操作にようなシステムイベントで指定したい場合は、上記の例で行ったようにユーザイベントのトリガとしてシステムイベントを割り当てることで可能になります。*

注 : カレンダプログラムは、モデルの選択プログラム特性を使用して日付モデルに割り当てることができます。これによって日付項目は自動的に「ユーザが作成した」フォームに値が渡されるようにすることができます。しかし、ロジックユニットを使用することでより多くのオプションを与えることができます。例えば、*F9* を押下すると通常のカレンダが表示され、*F10* を押下するとスケジュールが表示されるようにできます。

参照 : 「Magic エンジンをイベント駆動型で動作させるには」（47 ページ）
『リファレンスヘルプ』: 強制終了

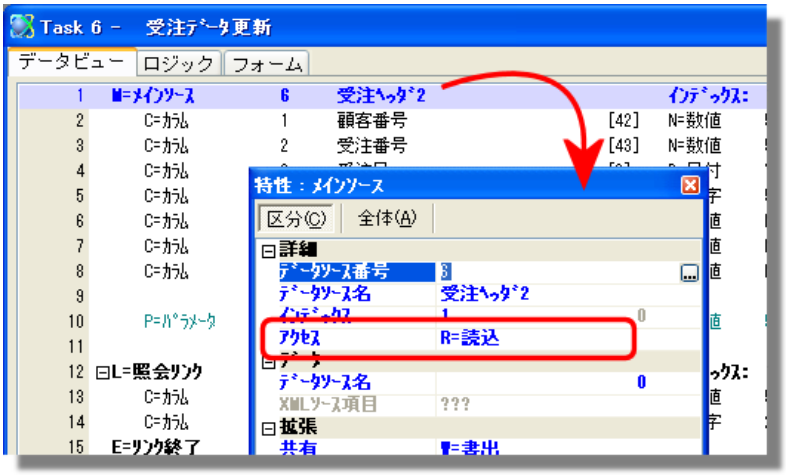
エンドユーザがタスク内でレコードを修正することを防ぐには

デフォルトでは、作成されたオンラインタスクを実行すると、ユーザはレコードに対してあらゆる操作（追加や削除、修正）が可能になります。データソースは書込モードでオープンされ、修正された場合レコードはロックされます。

これは、データソースを保守する際の簡単なプログラムを作成する上では有効なのですが、一般的なアプリケーションではユーザは、特性のレコードに対する修正処理に制限がかかっています。また、多くの画面は表示のみで、レコードの修正やロック処理に関する処理を必要としていません。

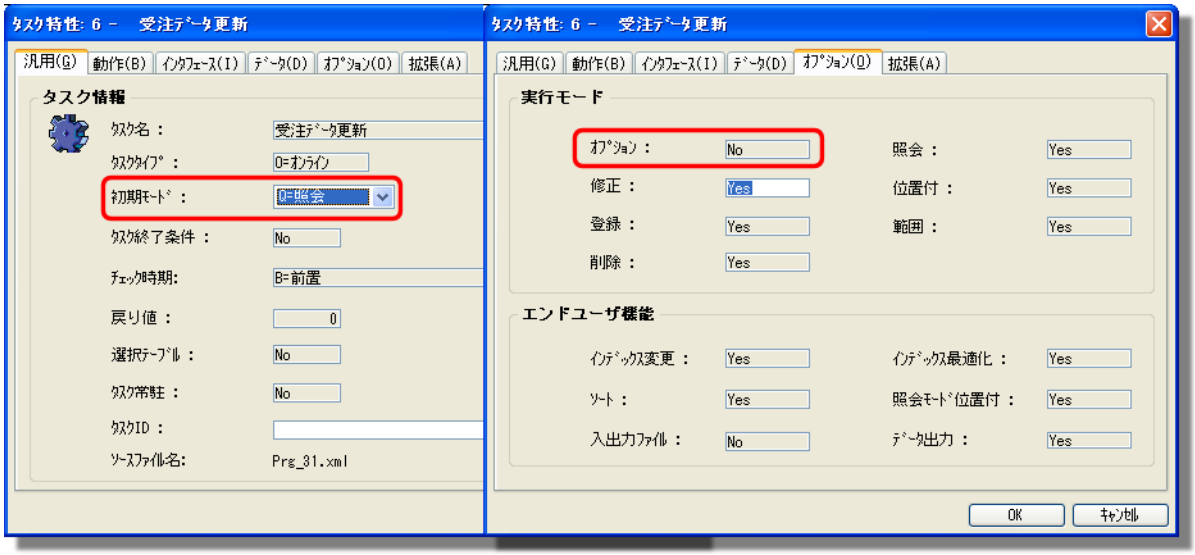
Magic では、データ修正に関にして複数の制御レベルを持っています。以下この件について説明します。

データソースのアクセスモードを設定する



データソースの**アクセス**特性を設定することで、確実にレコードの修正を防ぐことができます。**読込**モードでオープンされたデータソースを修正しようとした場合、DB エラーが発生するだけでレコードは更新されません。またこの場合は、データソースをより早くオープンすることができます。このため、データの更新が必要でないタスクでは**読込**モードでオープンするようにすることを推奨します。

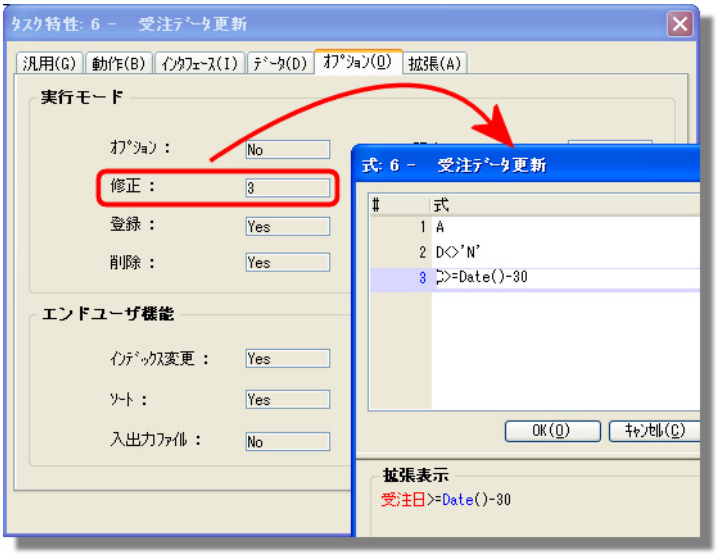
タスク特性の初期モードを使用する



タスク特性の**初期モード** 特性を ***Q= 照会***に設定することで修正できないモードでタスクを実行させることができます。これは、「検索」または「表示」のみのタスク用のモードです。

ただし、右上のフレームに表示されているように**オプション** 特性（**タスク特性→オプション**タブ）を選択し、***No*** を設定することがない限り**オプション→修正**（***Ctrl+M***）を選択することで**照会**から**修正** にモードを切り替えることができます。

タスク特性のオプションを使用する



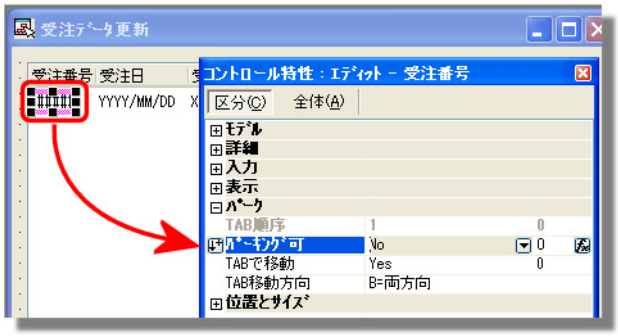
タスク特性（*Ctrl+P*）のオプションは、このタスクで有効な制御内容を指定することができます。各オプションは *Yes* または *No* あるいは、*式*で有効にするかどうかを指定できます。式で指定することでユーザ毎やレコード毎に動的に切り替えることができます。

上記の例では、レコードの削除は許可されていませんが、受注日付が 30 日未満の場合は修正が可能になっています。

項目レベルで修正する

データソースレベルの修正に加え、項目レベルの制御も可能です。

- コントロール特性の**パーク可**特性を *No* に設定することで、ユーザはこの項目にパークすることができなくなります。



- コントロールの**修正許可**特性を使用することで項目にパークはできるが修正できないようにすることができます。ここには、*Yes* または *No* あるいは論理値が返る式で指定できます。

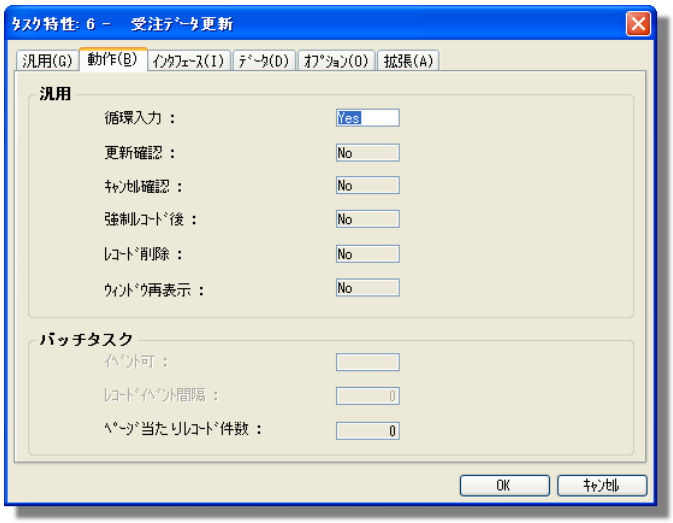
レコードが更新されなくてもレコード後を実行させるには

通常オンラインタスクでは、レコードがユーザによって更新された場合のみ**レコード後**が実行されます。実際、レコードが更新されない限りレコードを書き込む意味はありません。

しかし、変更がなくても**レコード後**を実行させたい場合があります。例えば、何らかのメッセージを表示させたり、計算処理やログを採取するような場合などが考えられます。

強制的にレコード後を実行させる

1. **タスク特性**（*Ctrl+P* または、**タスク→タスク特性**）を開きます。
2. **動作**タブを選択します。
3. **強制レコード後**を *Yes* に設定します。
4. 式で動的に指定したい場合は、**ズーム**（*F5*、**ダブルクリック**、または**編集→ズーム**）して**式エディタ**を開き、論理式を定義することができます。



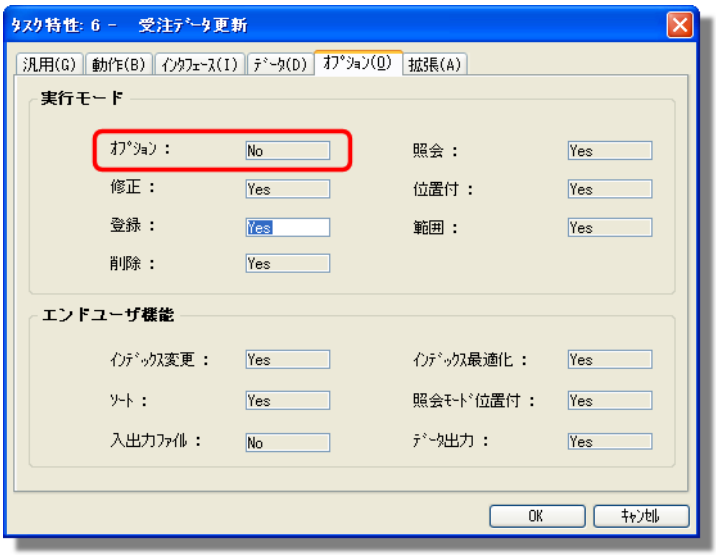
注： 今まで**レコード後**で実行させていたことは、コントロールイベントを使用することで実現できます。コントロールイベントはコントロールに関して最も良いレベルになります。

参照： 第 21 章：「式エディタ内で式の体裁を整えるには」（461 ページ）

エンドユーザがタスクモードを変更することを防止するには

デフォルトでは、Magic はエンドユーザがレコードの作成や削除、修正、照会ができるようにしています。これによって、データビューを簡単に表示させるタスクを作成することができます。しかし、ほとんどのアプリケーションでは、ユーザによってまたは実行するプログラムによって機能を制限するようにしています。例えば、一般のユーザはレコードの参照のみ許可され、管理者だけが修正することができるタスクなどを作成することができます。

ユーザがタスクモードを変更することを防止する

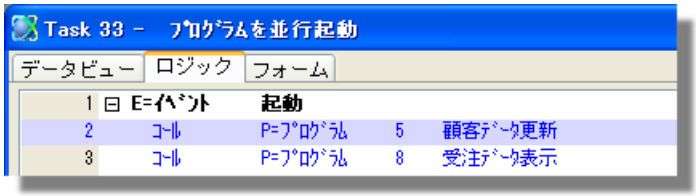


1. タスク特性（*Ctrl+P* または、タスク→タスク特性）を開きます。
2. オプションタブを選択します。
3. オプション特性を *No* に設定します。この設定によってユーザがタスクのモードを変更することができなくなります。
   式で動的に指定したい場合は、ズーム（*F5*、ダブルクリック、または編集→ズーム）して式エディタを開き、論理式を定義することができます。

注： このタブ内にある他のオプションを使用して特定の処理の利用を制限することもできます。例えば、削除特性を *No* に設定することでユーザはレコードの削除ができなくなります。

参照： 第 21 章：「式エディタ内で式の体裁を整えるには」（461 ページ）

２つのタスクを並行に実行させるには



通常、ここに表示されているように２つのプログラムを起動すると、Magic は最初のタスクを実行し、ユーザがそのタスクを終了すると 2 番目のタスクが起動されます。

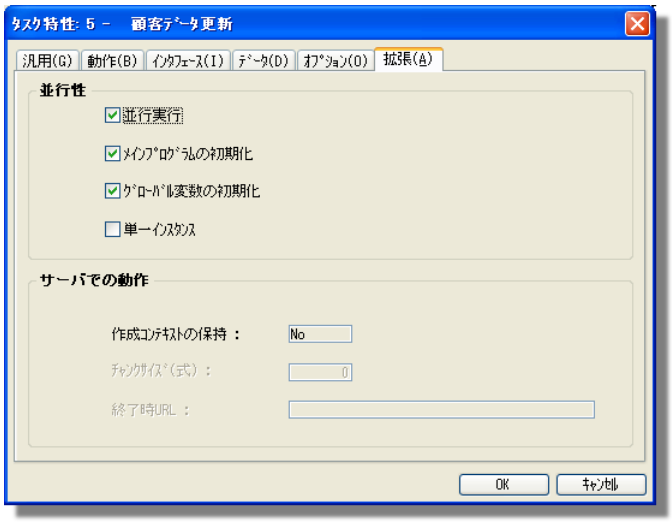
しかし、この２つのプログラムを同時に実行させることもできます。これは２つのウィンドウが同時にオープンされているように少しづつ動作することで実現しています。これらは両方ともオープンされていますが、各々独立して動いています。どのように独立しているかは Magic の設定に依存します。

この例では、並行に実行することのできる２つのタスクを設定しています。そして最初のタスクが実行され、次に２番目のタスクが実行されています。これらは両方とも実行されたままの状態になっています。２つのタスクの間でフォーカスを変更し、個別に更新処理を行うことができます。

ここではロックを防止するような対応はされていません。各タスクに対して、２人のユーザが同じデータをアクセスするような操作を行えば、同じような結果になります。この例では、同じレコードを２つの異なるウィンドウで表示していますが、、表示のみの処理であれば問題は発生しません。

変更実行するようにプログラムを設定する

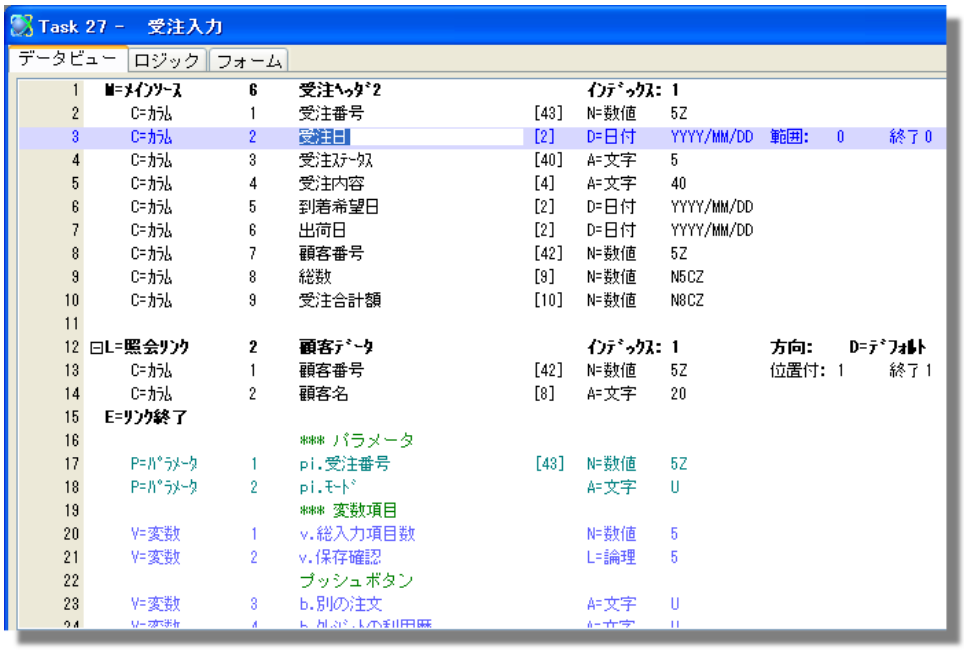
1. **タスク特性**（*Ctrl+P* または、**タスク→タスク特性**）を開きます。
2. **拡張**タブを選択します。
3. **並行実行**特性のチェックボックスを**チェック**します。
4. 必要に応じて他のチェックボックスを**チェック**します。



［このページは意図的に空白にしています。］

第 5 章： タスク

プログラムのデータビューを定義するには



Magic のタスクで使用されるデータは**データビューエディタ**で定義します。データソースを使用している場合、そのデータソースのすべてのカラムを定義する必要はありません。使用したいカラムのみを定義するだけで十分です。また、必要に応じて変数やパラメータを定義することもできます。

データビューエディタでは、５種類の基本的なタイプのデータを定義することができます。最初の３つのタイプはデータソースをもとに定義します。

- **メインソース / ダイレクト SQL：**メインソースが定義されている場合、タスクはデフォルトでデータソース全体を読み込みます。つまりオンラインタスクであれば、ユーザはテーブルのすべてのカラムを参照することができ、バッチタスクであればテーブルのすべてのレコードをアクセスするようになります。しかし、範囲を指定することで、読み込むレコードを制限したり、1 レコードのみ読み込むようにすることもできます。
- **リンクテーブル：**他のデータソースと 1 対 1 にリンクされるテーブル（ソース）です。この例では、顧客の姓名を表示するために、注文データと照合された顧客レコードを取得しています。データビューを取得するには、テーブルのすべてのカラムが必要なわけではありません。必要なカラムだけ定義すれば十分です。
- **宣言：**現在のタスクでは使用しないがデータソースをオープンしておくことができます。これは下位のタスクがそのデータソースを使用する場合、オープン / クロースの処理をしないようにすることで効率化させるための方法です。

最後の２つのタイプは、データソースに含まれない項目です。

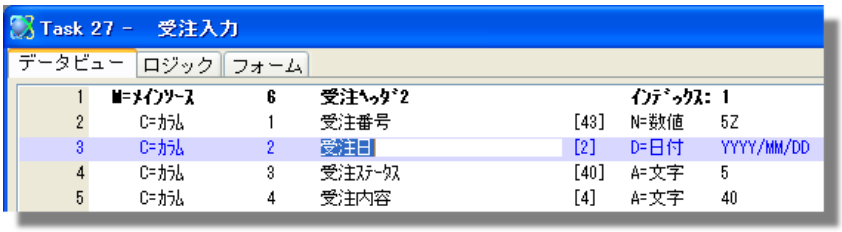
- **パラメータ：**他のプログラムに値を渡すために使用します。
- **変数：**変数項目はタスク実行時にのみ有効な一時的なデータ項目です。

メインソースはデータビューの先頭の定義される必要がありますが、これ以外のデータはどの順番でも定義することができます。しかし、従属関係があることを考慮してください。例えば、リンクテーブルはリンクされるデータソースの

下に定義しなければなりません。これは、データを整理する上で最良の方法です。また、コメントや余白を追加することで、ロジックが理解しやすくなります。

以下は、これらのことをどのように行うかを示した概要です。ここには、これ以外にも学ぶ内容がたくさんがあります。また、Magic のマニュアルを参照することで、これらの理解を助けることになるはずです。

メインソースを定義する



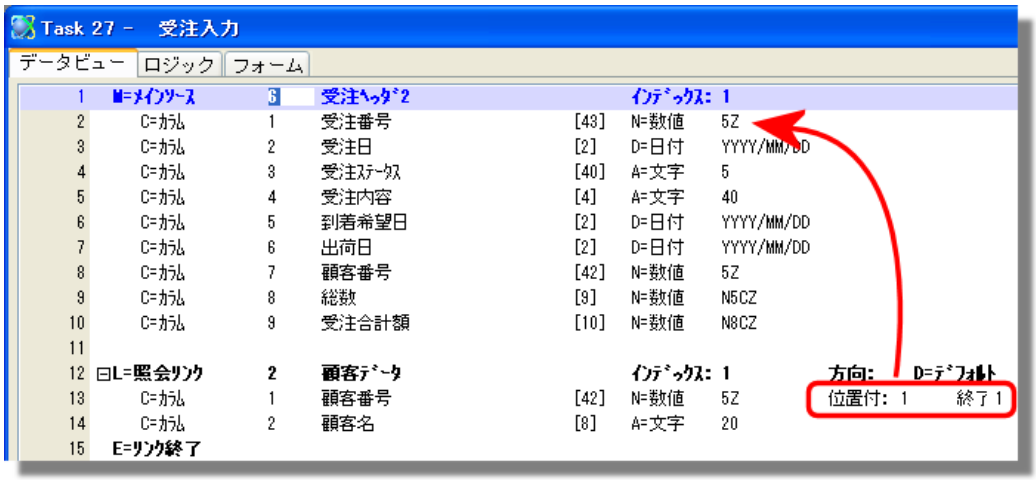
1. メインソースのヘッダ行は常に存在しています。これを作成する必要はありません。*Tab* を押下してメインソースが表示されている次のカラムに移動し、アクセスしたいデータソースの番号を入力するか、ズーム（*F5* または、ダブルクリック）を実行して一覧からデータソースを選択します。再度 *Tab* を押下します。
2. カーソルは、データソース名のカラム上に位置付けられているはずです。この名前は変更することができます。データソースの名前を変更することは、アクセスするデータリポジトリにも、エンジンがデータにアクセスする際にも影響しません。これは同じデータソースを１つのタスクツリー内で何回も定義するような場合に、ロジックを理解しやすくするする利点があります。再度 *Tab* を押下します。
3. 次にカーソルは、インデックスのカラムに移動します。インデックス番号を入力するか、ズームして一覧から選択します。適切なインデックスを選択することは非常に重要です。これによって、レコードを一覧表示させる場合の順番が決定され、範囲指定されている場合の動作の処理速度に影響します。
4. 最後に、マウスの右クリックを行い、コンテキストメニューから特性を選択します。ここでは、データソースをどのようにオープンするかを定義することができます。

メインソースを定義する、*F4* の押下でヘッダ行の下に詳細行が追加され、データソースに定義されているカラムを選択することができます。これらのカラム行では、データビューを限定するための範囲条件をメインソースに定義することができます。例えば、１つの注文情報や１人の顧客を表示させたい場合に指定します。

ヒント : メインソースが定義されたタスク内では、リンク / リンク終了内の行を除いて、カラムと指定されたデータビューはすべてメインソースを参照します。これは、データビュー内に複数のリンクテーブルや変数が定義されていても同じです。しかし、メインソースのヘッダ行内にカラムを集めておいた方が、ロジックは見やすくなります。

リンクテーブルを定義する

リンクテーブルの定義は、メインソース同じように行うことができます。.



1. 最初にリンクテーブルを作成したい行で *Ctrl+H* を押下してヘッダ行を作成します。
2. 左端のコンボボックスからリンクタイプを選択します。各タイプは異なる動作をします。通常は照会リンクを使用します。*Tab* を押下して次のカラムに移動します。リンク終了はヘッダ行の下に自動的に追加されます。

3. リンクしたいデータソースの番号を入力するか、ズーム（*F5* または、ダブルクリック）を実行して一覧からデータソースを選択します。再度 *Tab* を押下します。
4. インデックスカラムでは、リンクレコードの検索に使用したいインデックスを選択します。選択されたインデックスのセグメントとして定義されているカラムが自動的に追加されます。これらを使用してリンクレコードの位置付け定義を行うことができます。
5. マウスの右クリックを行い、コンテキストメニューから特性を選択することでデータソースをどのように処理するかを定義することができます。

リンクと終了リンクの行間で *F4* を押下するとリンクで使用するカラムを選択することができます。位置付特性は、どのレコードをリンクするかを制御するために使用されます。この例では、顧客レコード内の顧客 ID と受注ヘッダ内の顧客 ID がリンクされています。

変数とパラメータを定義する

データソースのカラムではないデータ項目を定義することができます。これらの項目には２つのタイプがあります。***変数***と***パラメータ***です。これらの唯一の違いは、パラメータは別のプログラムからデータを受け取るためのものであり、パラメータの数や書式は、起動元のプログラムの定義内容とチェックされるところです。

1. 変数またはパラメータを作成したい行に移動し、*F4*（編集→行作成）を押下します。
2. ドロップダウンリストから *V= 変数* か *P= パラメータ* を選択します。*Tab* を押下して次のカラムに移動します。
3. 現在カーソルは、名前のカラムにあります。ここには、任意の名前を入力することができます。何らかの命名規約（例えば、変数ならば接頭辞として「v.」を使用する等）を使用することもあるかもしれません。名前を入力後、*Tab* を押下して次のカラムに移動します。
4. 次に、モデルのカラムに移動します。ここからズームして一覧からモデルを選択できます。モデルを使用することで開発工数が減ったり、エラーの防止やタスクの規格化に役立ちます。モデルを使用する場合は、ここでモデルを選択するだけで十分です。モデルを選択しない場合は、*Tab* を押下して次のカラムに移動します。
5. 次は、型のカラムに移動します。モデルを使用しない場合、型の先頭文字を入力してデータ型（文字、論理、日付、時間など）を選択する、プルダウンリストから選択します。再び *Tab* を押下します。
6. ここには、項目の書式を入力するか、ズームして書式ダイアログを表示させます。書式ダイアログでは、*F1* キーを押すことで書式に関するヘルプを表示させることができます。
7. *Alt+Enter* を押下するか、特性シートをクリックして項目特性に移動します。項目特性は、データ項目のサイズや有効値を指定するだけでなく、フォーム上でどのように表示させるかを定義することもできます。モデルを使用した場合、これらの特性値はそのモデルの値が設定されますが必要に応じて、継承された値を上書きすることもできます。

ヒント : *Magic で最も一般的なエラー原因の 1 つに書式を指定しないで項目を作成することがあります。型を選択したら、すぐに特性シートを開き項目の書式を設定してください。データ長が 0 の場合、このプログラムは実行できないかもしれません。*

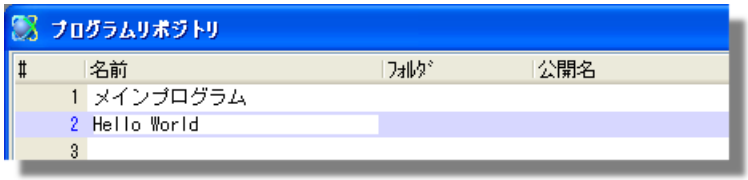
参照 : 第 3 章 :「再利用可能なデータオブジェクトを定義するには」（35 ページ）

簡単なプログラムを作成するには

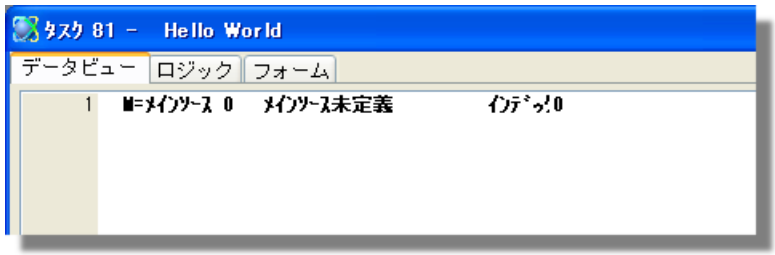
Magic 一般的なプログラミングの書籍では、Hello World という簡単なプログラム例もとに説明しています。ここでは、Magic による Hellow World の説明を行います。

ヒント : *まだ Magic を始めたばかりの場合は、F1 を押下することでヘルプを表示させることができます。ここには、現在表示されているパラメータに関連したトピックが表示されます。*

Magic で「Hello World」を作成する

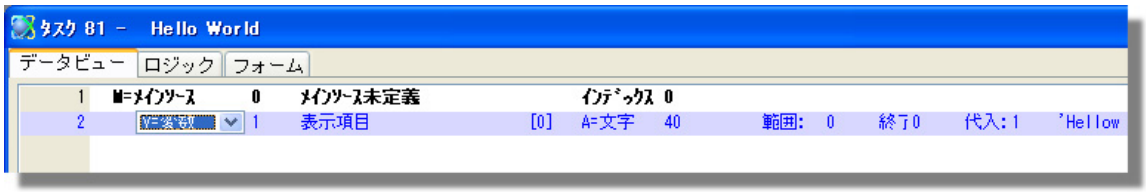


1. プログラムリポジトリに移動し（*Shift+F3*）、プログラムを作成したい行にカーソルを置きます。
2. *F4*（編集→行作成）を押下し一行追加します。
3. プログラムの名前を入力します。この場合は、Hello World です。Magic エンジンはこの名前を実行時は使用しないため、どのような名前でも入力できます。
4. ズーム（*F5* または、ダブルクリック）を押してプログラムを開きます。
5. 新規プログラムのため、タスク特性ダイアログが表示されます。*Esc* か OK ボタンを押下してダイアログを閉じます。簡単なプログラム用にデフォルト値が設定されています。

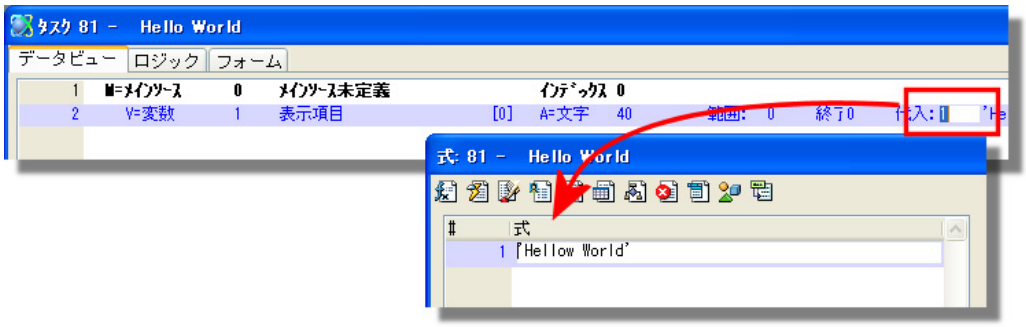


6. 次に作成されたプログラムを見てください。３つのタブが表示されています。データビュー、ロジックそしてフォームです。これらは、Magic でプログラムを作成する場所になります。

データビューを作成する



1. 最初に、データビューを定義します。データビュータブをクリックします。メインソースのヘッダ行がすでに定義されていることを確認してください。この例では、メインソースを使用しないためこの設定を無視してください。
2. *F4* を押下し一行作成します。
3. プルダウンリストから ***V= 変数***を選択します。これは変数項目を作成するという意味です。*Tab* で次のカラムに移動します。
4. 変数の名前を指定します。ここでは表示項目と入力します。
5. *Tab* を２回押下して **A= 文字**と表示されているカラムに移動します。ここはデータの型を指定します。デフォルトは文字型です。プルダウンをクリックすると他の選択肢をみることができます。もう一度 *Tab* を押下します。**書式**を指定するカラムに移動します。ここに *40* と入力します。これは 40 桁の文字項目を意味しています。**代入**と表示されたカラムまでカーソルを移動します。

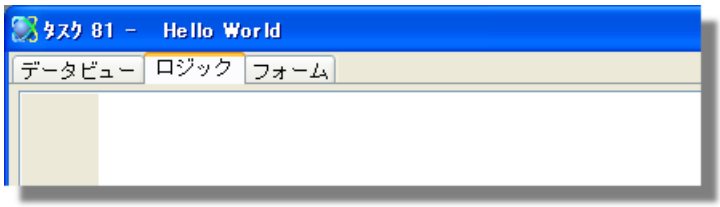


6. 代入特性は、タスク実行時に初期設定する値や実行時に再計算される値を指定します。この場合は、*Hello World* という文字列で初期設定します。*F5* を押下するかズームして式エディタを開きます。
7. 式エディタでは、*F4* を押下して一行追加し、*'Hello world'* という文字列を入力します（シングルクォーテーションも含めます。）。OK をクリックします。
8. 式エディタを閉じると代入特性に *1* という数字が表示されます。これは式番号 **#1** が定義されていることを意味しています。また、右側に式の内容の先頭部分が表示されます。

ヒント：このような簡単な式の場合は、代入カラムに = を入力することでも設定できます。この場合、簡単な入力ボックスが表示され、ここに式の内容を入力します。

これでこのプログラムに対してデータビューエディタで行う作業は終了しました。

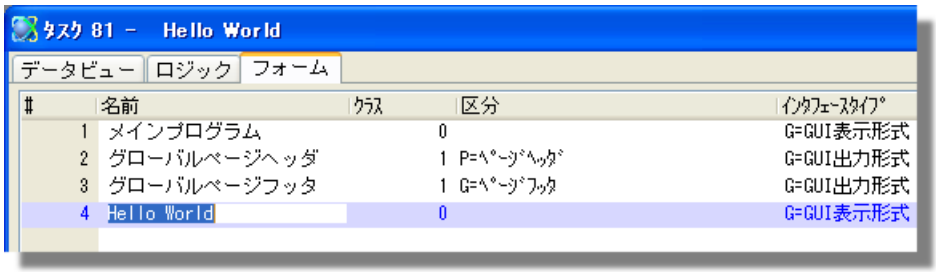
ロジックを作成する



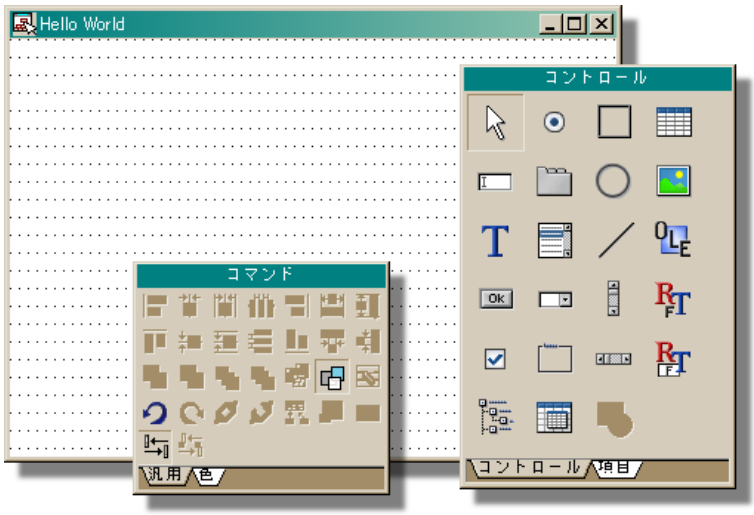
このプログラムでは、ロジックエディタで何かを行う必要はありません。Magic は、通常のプログラムで行うようなほとんどの処理をエンジン内で行うため、そのための明示的なロジックを作成する必要はありません。

ここでは、データ項目の検証や他のプログラムを呼び出すなどの処理を定義することができます。Magic でロジックとイベントを使用する方法については、第 4 章：「Magic エンジンをイベント駆動型で動作させるには」（47 ページ）を参照してください。

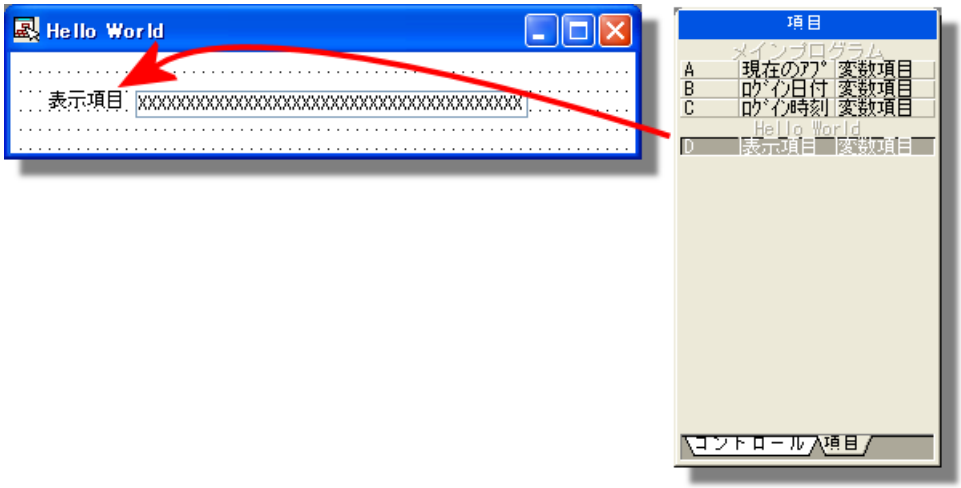
表示画面を作成する



1. フォームタブをクリックします。すでに **Hello World!** という名前のフォームが作成されているはずです。デフォルトではタスクの名前と同じ名前がフォームに設定されますが、必要に応じて変更することができます。またデフォルトでは、この名前がウィンドウのタイトルバーに表示されます。
2. 名前カラムでズーム（*F5* または、ダブルクリック）すると、フォームの内容が表示されます。この時点では基本的な空のウィンドウが表示されます。このウィンドウは位置を変更したりドラッグすることでサイズを変更することができます。特性シート（*Alt+Enter*）には、このウィンドウの表示内容を変更するための色々なパラメータがあります。しかしここではデフォルトのままにしておきます。



3. フォームのそばには**コントロール**と**コマンド**というラベルが表示された２つの特殊なボックスが表示されています。これらは、フォームの編集中に使用するパレットです。同じオプションはプルダウンメニューやコンテキストメニュー、ショートカットキーからも使用できます。これらのオプションやショートカットに慣れると操作が楽になります。
   何らかの理由で**コントロール**と**コマンド**の各パレットがワークスペース上に表示されていない場合は、**表示→フォームエディタ**パレット（またはツールバーの  アイコン）を選択してください。これはパレットの表示と非表示を切り替えます。
4. **コントロール**パレットの**項目**タブを**クリック**します。ここには、データビューとしてタスクがアクセスできるすべての項目が表示されています。ここでは１つだけ表示されています。



5. 項目を**クリック**します。カーソルがボックス表示に変わり、選択されていることを示すようになります。フォームを**クリック**して項目をドロップします。項目が貼り付けられます。項目名もテキストコントロールとして貼り付けられます。このことを考慮して項目の名前を定義していれば、フォーム作成の工数を減らすことができます。

フォームエディタでの作業はこれで終わりです。**オプション→オブジェクトを保存 / 閉じる**を選択して、**プログラム**リポジトリに戻ります。

プログラムを実行する

1. 作成したプログラムを実行する前に、***F8***（**オプション→構文チェック**）を押下します。プログラムに問題がなければ、ステータス行に「プログラムは正常です。」というメッセージが表示されます。何らかの問題点があれば、チェック結果ペインにエラーメッセージが表示されます。どのようなプログラミング言語であれ、プログラムに問題があれば、実行させても予期しない結果になります。プログラムを実行する前にはエラーをチェックして問題がないことを確認する必要があります。

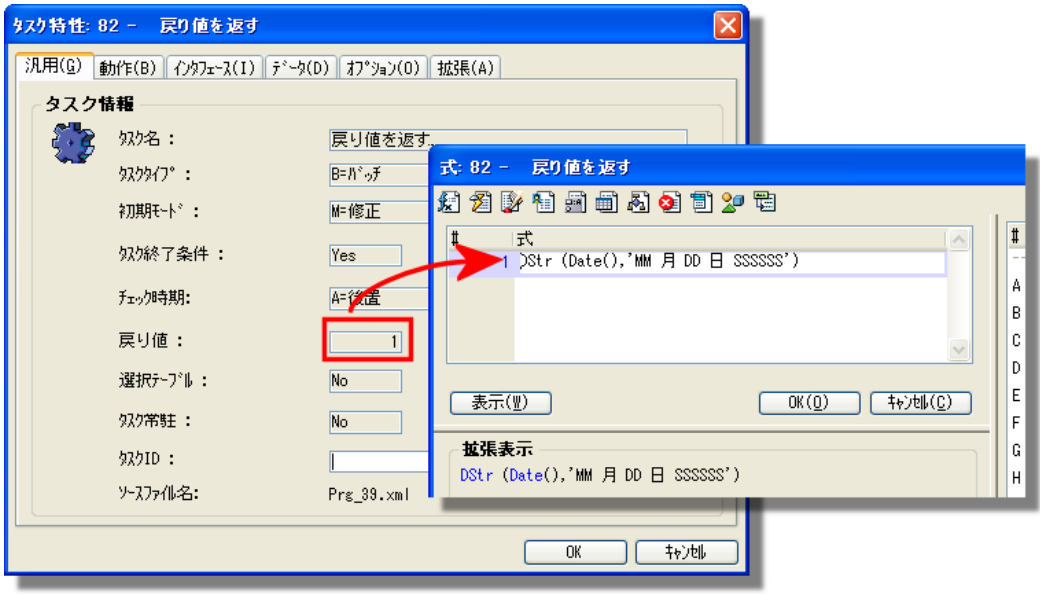
2. プログラムを実行するには ***F7***（**デバッグ→実行**）を押下します。別のウィンドウが表示され、実行エンジンによってプログラムが起動されます。プログラムが実行中は、開発環境はロックされ読み込み専用の状態になります。

3. これで Hello world! プログラムの作成が終わりました。プログラムを終了すると開発エンジンの環境に戻り、プログラム作成を継続したり、実行し直したりすることができます。このようにして Magic を使用することでプログラムのインクリメンタル開発を行うことができます。
   またデバッグ機能も利用できるため、プログラムが実行中にエンジン内でどのような処理が行われているかを確認することができます。

起動元のプログラムに戻り値を返すように設定するには

起動したプログラムに戻り値を返すには２つの方法があります。１つはパラメータを使用する方法です。この方法については、「起動元と起動先のパラメータを同期させるには」（85 ページ）を参照してください。もう１つは戻り値を使用する方法です。

式を使用して Magic プログラムを起動したり、Magic 外のプログラムを呼び出す場合に戻り値が必要になります。ここに、どのように設定するかを示しています。

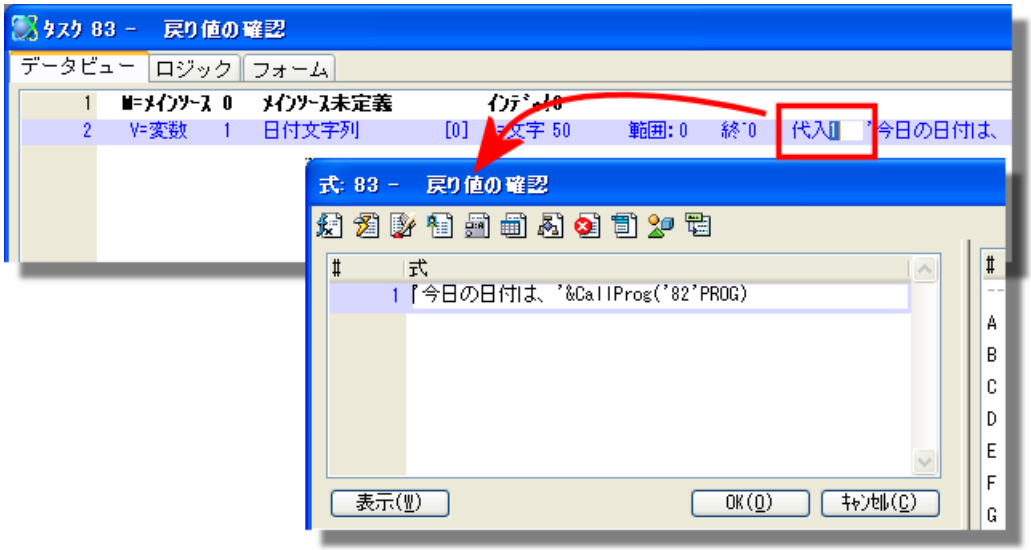


戻り値を定義する

1. 起動されるプログラムを選択し**ズーム**で開きます。**タスク特性**の**戻り値**特性（*Ctrl+P*）を選択します。戻り値としたい値を返す式を定義します。

この場合は、１回だけループを繰り返すバッチタスクで、今日の日付を指定された文字型書式で返すようにしています。もし書式を変更する必要があれば、このプログラムを変更するだけで済むようになります。

戻り値を使用する



1. 式を定義することのできるプログラムであれば、どれでも使用することができます。この例では、**代入**カラムに使用しています。**CallProg** 関数は、プログラムを呼び出し、定義された文字項目に値が自動的に返ります。

   CallProg 関数はプログラム番号を使用して呼び出します。もし、**#82** のプログラムを別の場所に移動したらどうなるでしょうか？ **PROG** リテラルを使用することでこのような心配は不要になります。プログラム番号が変更された場合、式内の番号が自動的に変更されるようになります。

プログラムの**公開名**を使用して **CallProg** 関数で呼び出すこともできます。この場合の構文は以下のようになります。

```
'Today''s date is: ' & CallProg(ProgIdx('DateString','TRUE'LOG))
```

CallProg や **ProgIdx** 関数についての詳細は、**F1** を押下して表示されるヘルプを参照してください。

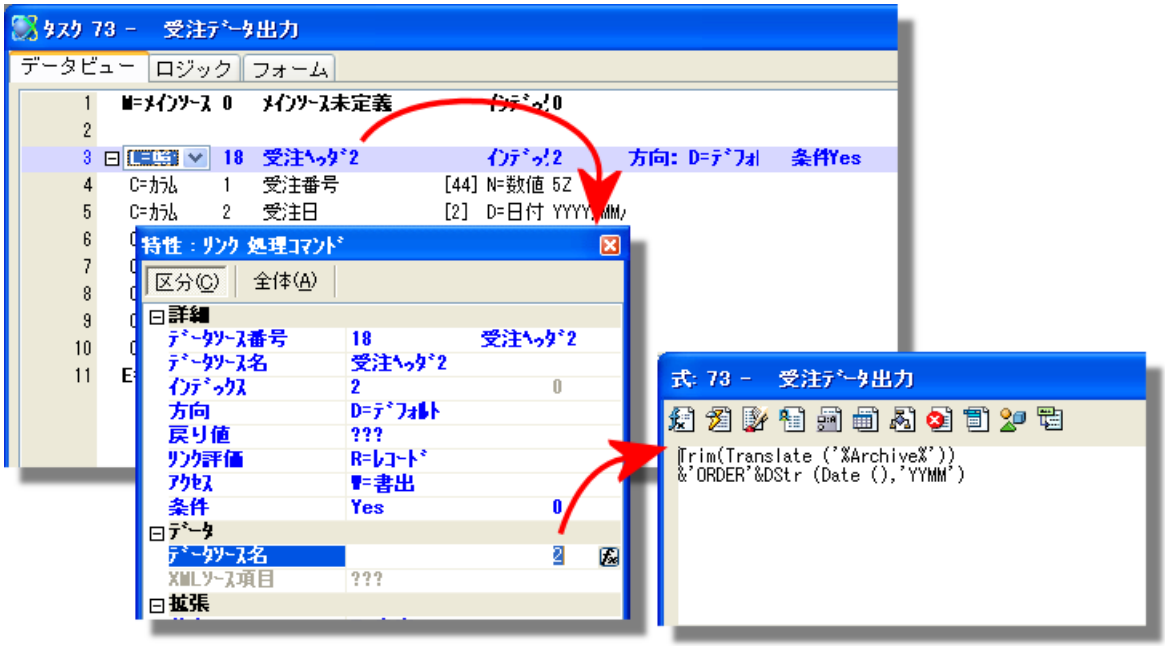
注： Magic にはプログラムを標準化するための様々なオプションがあります。この例の場合は、日付書式の**項目モデル**や**メインプログラム**に処理を定義するようなことが考えられます。

データリポジトリに定義されている名前と異なるデータソース名を使用するには

通常、データソース名はデータリポジトリで設定されます。これによって保守が容易になります。一カ所で管理されていることで、データソース名を探したり、変更することが容易になります。

しかし、タスクレベルでデータソース名を変更したい場合があります。例えば、ユーザ毎に個別のテーブルを定義したり、同じデータ定義で日付によって異なるデータソース名にするようなことが考えられます。

タスクレベルでデータソース名を指定する

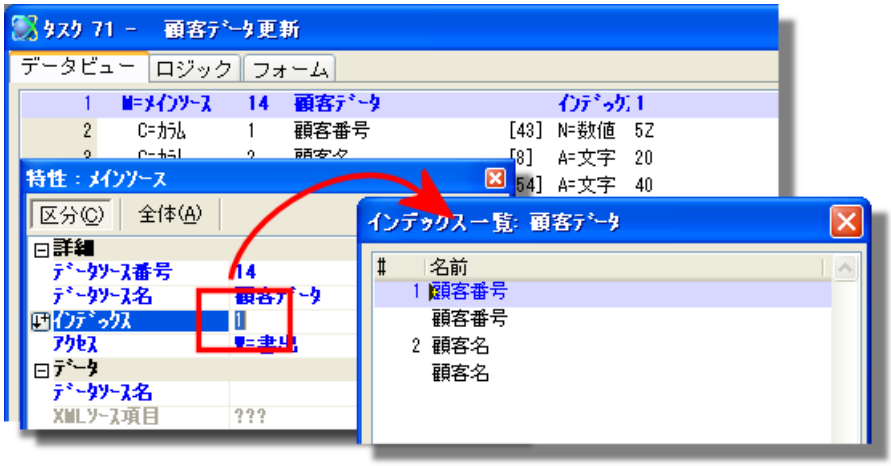


この例では、受注用のヘッダレコードを複写するサブタスクがすでに定義されているものとします。ここではデータリポジトリ内の同じオブジェクトを開いてはいますが、データソース名特性を指定することで、デフォルトのデータソース名を上書きしています。データソース名は、実行時の年と月にもとづいて定義され（例えば、2007 年 12 月の場合は、'0712'）、論理名 %Archive% で指定されたディレクトリ内に作成 / 修正されるようになっています。

注： 同じタスク内では、同じデータソースに対して異なる名前を定義することはできません。この例では、親タスクがデフォルト名で同じデータソースを開くようになっているため、サブタスク内では保管用のレコードを作成しています。

参照： 第 18 章：「既存のデータベーステーブルにアクセスするには」（402 ページ）

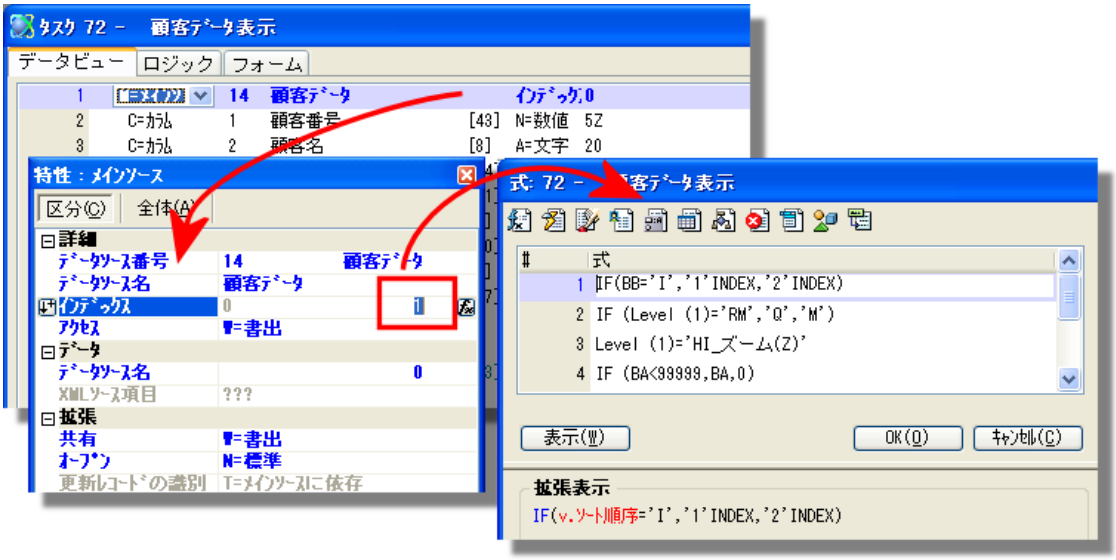
レコードの表示順を動的に変更するには



レコードを表示する順番は２つの方法で定義できます。１つはソートテーブルで実行時のソートの順番を定義する方法であり、もう１つは最もよく使用される方法ですがメインソースのインデックス特性を変更することです。この例では、インデックスは *1* が指定されています。この場合は、顧客 ID を元にソートされます。

式を使用することで動的にインデックスを設定することもできます。ユーザがテーブルの表示順を変更できるようにする場合や、帳票出力時に選択したフィルタを元に処理を最適化する場合に使用します。

インデックス用の式を使用する



1. データビューエディタに移動し、メインソースに位置付けます。*Alt+Enter* を押下します。メインソース特性にカーソルが移動します。
2. インデックス特性に *0* を入力します。これで、右側に *Tab* 移動することができます。ここには、インデックスを式で定義することができます。
3. インデックスの式特性でズームするか *fx* ボタンをクリックします。
4. インデックス式を入力します。式では、ソート順が I の場合インデックス番号を *1* に設定し、それ以外の場合は、2 に設定するようにします。インデックスの位置がデータソース内で移動する可能性を考慮して **INDEX** リテラル（例：'I'INDEX）を使用することで、設定内容が維持されます。
5. OK をクリックすると特性シートに戻り式番号が設定されます。

注： ユーザが表示されたレコードのソート順を変更するには、この他にテーブルコントロールのカラムヘッダをクリックしたり、実行時のオプションとして *Ctrl+K*（オプション→インデックス）を押下するしたり、*Ctrl+T*（オプション→ソート）を押下して独自のソート順を定義する方法があります。

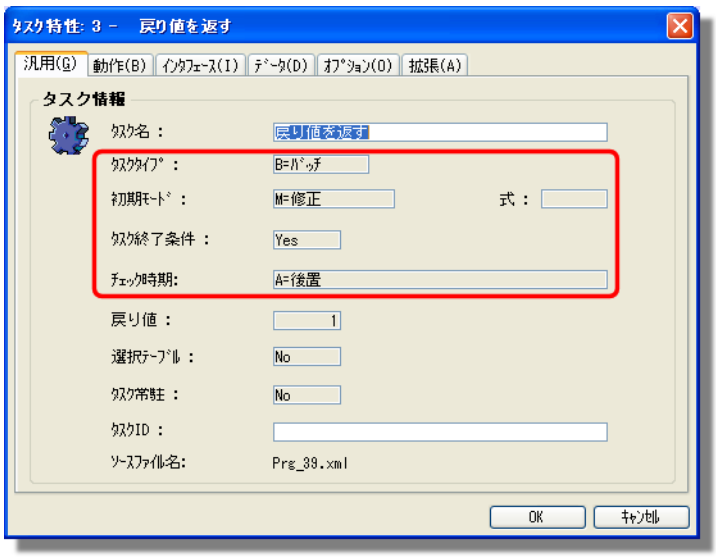
参照： 第 20 章：「タスクで処理されるレコードの順番を動的に設定するには」（434 ページ）

簡単なバッチプログラムを作成するには

ここには、２種類の簡単なバッチプログラムがあります。

- データソース内のすべてのレコードを読み込むプログラム。一般的に、ファイルに出力するような目的で使用されます。このようなプログラムは、**APG**（*Ctrl+G*）を使用して作成することができます。
- 一回のみループするだけのプログラム。通常は、戻り値を返すようになっています。ここではこのようなプログラムの作成方法について説明します。

簡単なバッチタスクを作成する



1. **プログラム**リポジトリで、*F4* を押下し一行追加します。
2. プログラムに名前を入力します。この名前はアプリケーション実行時、Magic は使用しないため任意の名前を使用することができます。
3. プログラム名で**ズーム**（*F5*）します。新規プログラムとなるため、**タスク特性**ダイアログが表示されます。
4. デフォルトの**初期モード**特性は *M= **修正***になっています。これは一般的によく使用されるモードです。照会モードに設定した場合、レコードは更新できなくなります。
5. **タスクタイプ**特性では *B= **バッチ***を選択します。
6. **タスク終了条件**特性を *Yes* に設定します。これは、タスクが永久ループすることを防止するためです。この設定により、1 回だけループするようになります。ループする回数を指定したい場合は、、ここでズームしてタスクが終了するための条件式を定義します。
7. **チェック時期**特性を *A= **後置***に設定します。

これはバッチプログラムなので、デフォルトでは、ウィンドウは表示されません。処理に時間がかかるようであれば、処理中であることを表示するウィンドウを表示する必要があるかもしれません。

この後、このプログラムに必要なロジックを定義することができます。レコードを更新したり、パラメータとしてデータを返したり、ウィンドウに表示させたり、戻り値としてデータを返したりすることができます。

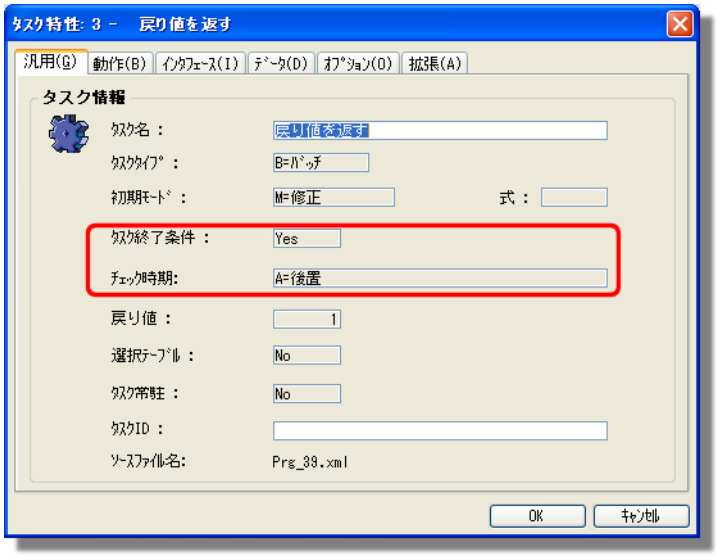
参照： 「起動元のプログラムに戻り値を返すように設定するには」（74 ページ）

永久ループするバッチタスクを止めるには

デフォルトでは、バッチタスクは永久ループするようになっています。

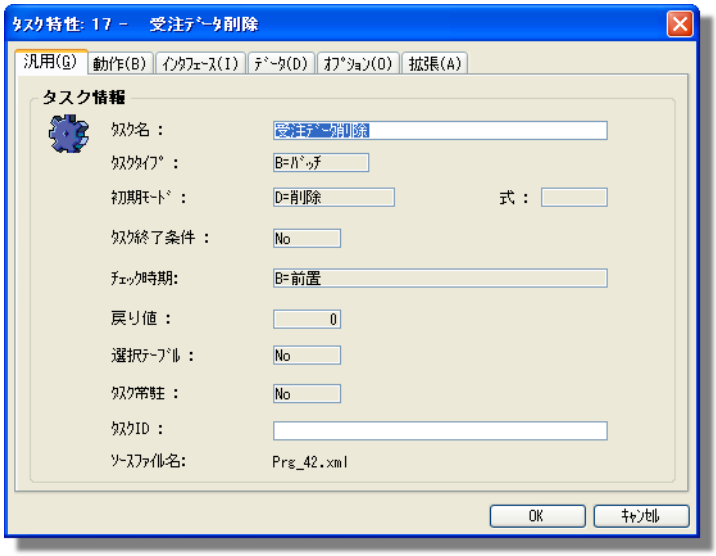
永久ループすることを防止するには、以下のどれかの設定が必要です。

- **データビューエディタ**でメインソースを設定します。メインソースが選択された場合、タスクはメインソースのレコードのアクセスが終了するとタスクは終了します。
- 右図で示されているように、**タスク特性**で**タスク終了条件**特性を設定します。タスクを 1 回だけループさせるには以下のように設定します。
  タスク終了条件：*Yes*
  チェック時期：*A= 後置*
- これ以外の条件でタスクを終わらせたい場合は、**タスク終了条件**特性から**ズーム**して式を定義します。式が True と評価された場合、タスクは終了します。



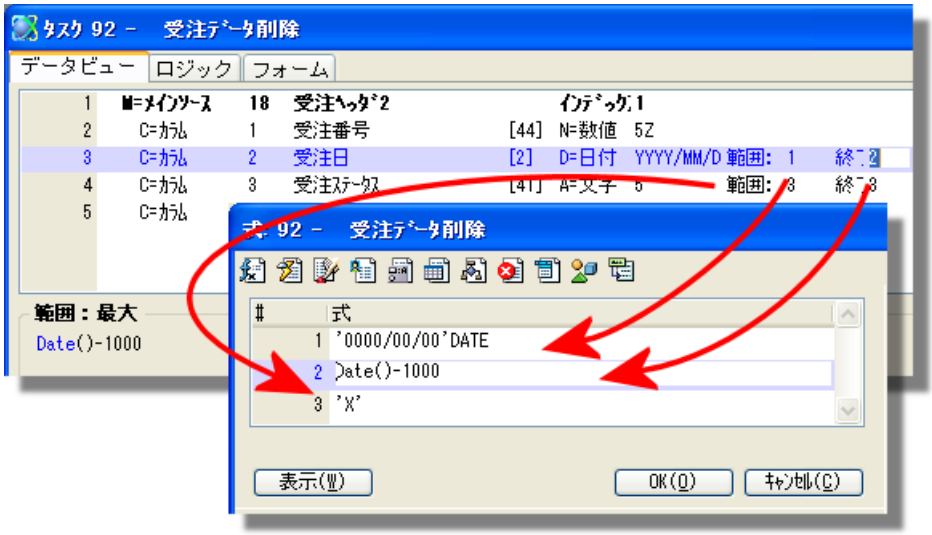
データソースからレコードをまとめて削除するには

データをまとめて削除するプログラムが必要になる場合があります。データベースを削除したり、レコードを保存用テーブルに移動するような場合に使用されます。



削除用のバッチタスクを作成する

1. 必要な場所（サブタスクか個別のプログラムかに関わらず）でタスクを作成します。
2. **タスク特性**ダイアログを開きます。（*Ctrl+P*）
3. 以下の特性を設定します。
   タスクタイプ：***B= バッチ***
   初期モード：***D= 削除***
4. OK をクリックします。



5. 次に**データビューエディタ**を開きます。レコードを削除したいデータソースをメインソースとして選択します。この場合に、受注ヘッダを選択します。
6. 削除するレコードを選択するために必要なカラムを定義します。この場合に、必要なカラムは注文状況のみですが、デバッグのために他のカラムも定義します。
7. **範囲（最大 / 最小）**特性で**ズーム**し、削除するレコードの範囲を定義します。この場合は、0 から 1000 日以前の日付を持つすべてのレコードが削除されます（削除対象のレコードには、受注ステータスに「X」が設定されています）。値が設定されていない場合、デフォルト値として '0000/00/00'DATE が設定されていることと同じ意味を持つため、最小値の設定は必要ありません。

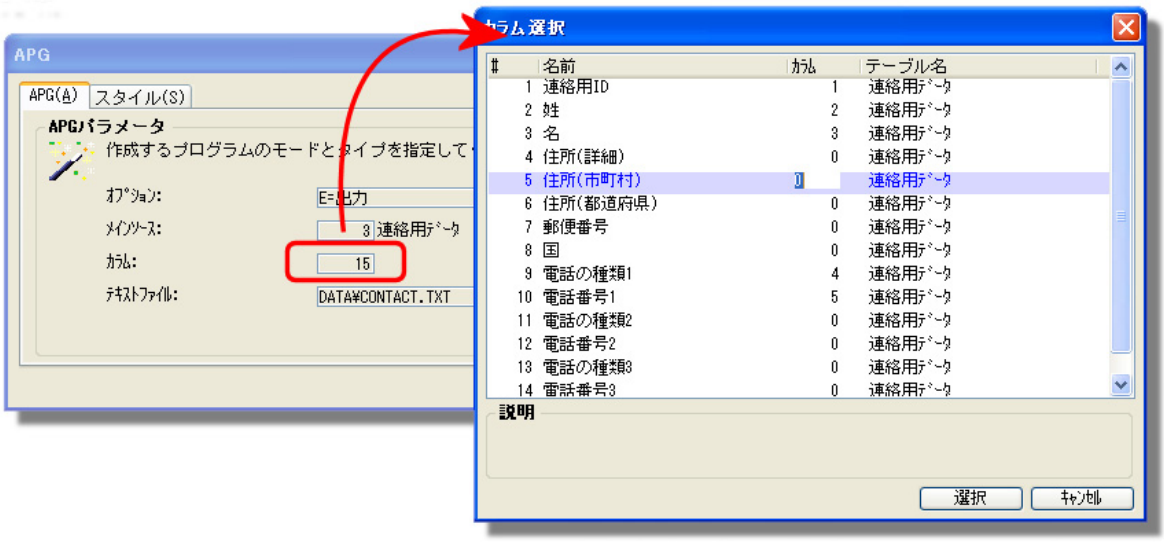
このタスクを実行すると、範囲内のレコードはデータソースから削除されます。

データソースの内容をテキストファイルに出力したり、テキストファイルをデータソースに入力するには

非常によく使用されるタスクの１つに、データソースの内容をテキストファイルに出力したり、テキストファイルの内容をデータソースに読み込むものがあります。Magic はこれらの処理を簡単に行うための機能があります。

データソースの内容をテキストファイルに出力する

1. *F4*（編集→行作成）を押下して、プログラムリポジトリに一行追加します。
2. *Ctrl+G*（オプション→ APG）を押下します。



3. APG ダイアログが表示されます。以下のオプションを設定します。

   オプション：*E= 出力*

   メインソース：出力したいデータソース。ズームすることで一覧から選択できます。

   カラム：ズームして出力するカラムを選択することができます。デフォルトでは、すべてのカラムを出力するようになっています。この例では、ほとんどのカラムの値は 0 のままなので、姓、名、電話タイプ 1、および電話番号 1 だけを選択します。

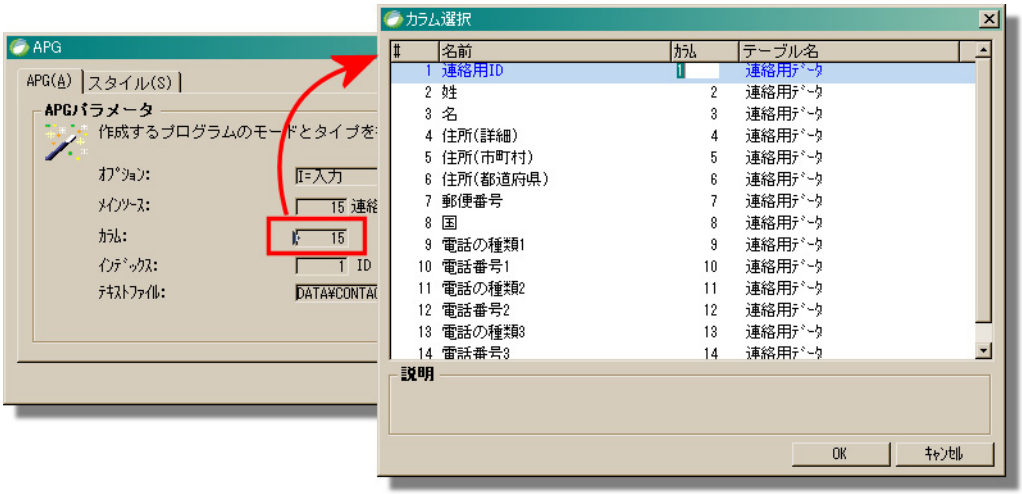
   テキストファイル：出力するテキストファイルの名前を指定することができます。
4. OK をクリックします。

作成されたプログラムを実行すると、メインソースからすべてのレコードが出力されます。このプログラムを修正することで、出力する書式を変更したり、範囲を指定して出力されるレコードを制限したりすることができます。

テキストファイルの内容を読み込む

1. *F4*（編集→行作成）を押下して、プログラムリポジトリに一行追加します。

2. *Ctrl+G*（オプション→ APG）を押下します。



3. **APG** ダイアログが表示されます。以下のオプションを設定します。

   オプション：*I= 入力*

   メインソース：入力先のデータソース。**ズーム**することで一覧から選択できます。

   カラム：ズームして入力するカラムを選択することができます。デフォルトでは、すべてのカラムを出力するようになっています。

   インデックス：ズームして入力時に使用するインデックスを選択することができます。入力時の順番は、入力ファイルの並びに依存するため変更する必要はありません。

   テキストファイル：入力するテキストファイルの名前を指定することができます。作成されたプログラムでは、データ項目や論理名を使用して設定することができます。

4. OK を**クリック**します。

作成されたプログラムを実行すると、テキストファイルの内容がメインソースのレコードとしてが入力されます。入力するテキストファイルの書式に合うように入力フォームの内容を修正する必要があります。

APG によって作成されたプログラムはレコードの重複チェックを行いません。このため、空のデータソースで実行されていたり、データの重複入力ができなかったり、DBMS からのエラーメッセージを受け取った場合の扱いは開発者に依存しています。

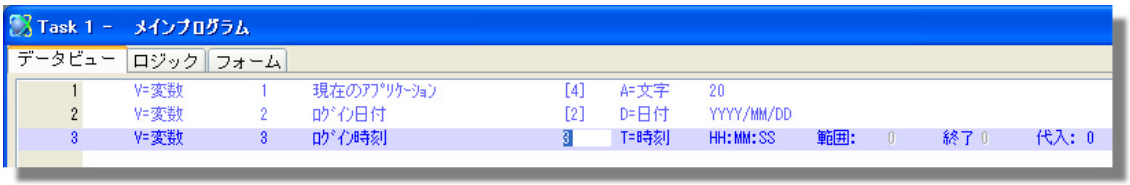
ヒント：もしカラムのレイアウトをデフォルトのままで出力プログラムを作成した場合、入力プログラムもデフォルトのままで作成できます。このため、ある場所から別の場所にデータをコピーするだけの簡単な方法として、APG を使用することができます。

プロジェクト内のどこでもアクセス可能なグローバル変数を定義するには

Magic のタスク内で定義された変数やデータソースは、このタスクおよびそのサブタスクでのみアクセスできます。しかし、プロジェクト内のすべてのタスクで利用可能な変数が必要な場合があります。

これらのグローバル変数は、メインプログラムで定義されることを除いて他の変数と同じように定義することができます。

グローバル変数を定義する



1. プログラムリポジトリを開き（*Shift+F3*）、メインプログラム（**#1**）に移動します。
2. ズーム（*F5*）してプログラムを開きます。
3. データビュータブをクリックします。
4. 他のタスクと同じように、データビューを作成するための操作を行います。

ヒント：メインプログラムでもデータソースをリンクすることができますが、メインプログラムは、アプリケーション実行中オープンされた状態のままになるため、多くのユーザが使用するデータソースをここに定義しないようにしてください。メインプログラム内で変数を定義し、タスク前やタスク後でデータを取り出したり、格納することでデータの内容を保持することができます。

参照： 「プログラムのデータビューを定義するには」（67 ページ）

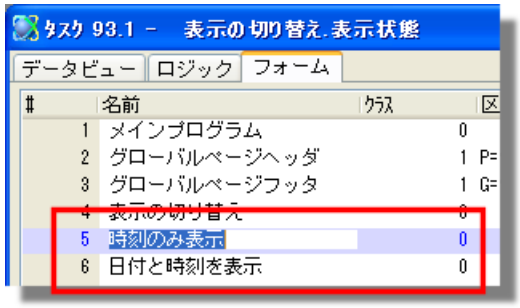
表示するメインフォームを動的に指定するには

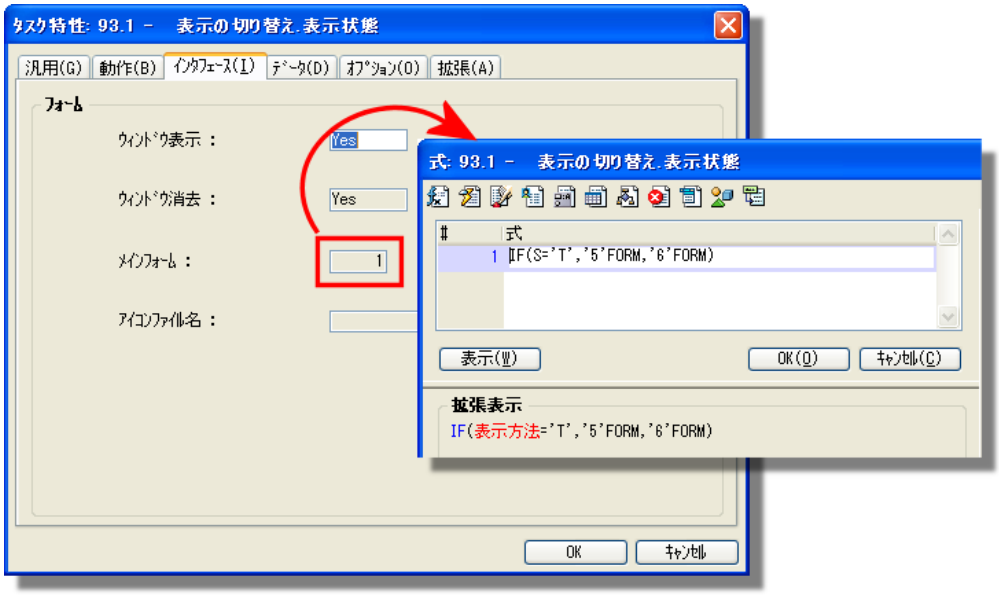
通常各タスクは、表示用の独自のフォームを持っています。しかし、実行時に別のフォームを表示させることができます。

実行時に表示するフォームを指定する

必要条件： フォームはすでに作成されおりて、このタスク（親タスクでない）に定義されているものとします。

1. フォームタブをクリックします。フォームの一覧が表示されているはずです。これらのフォームのいくつかがこのタスクで定義されたものであり、その他は親タスクで定義されたものです。この例では、下位の２つのフォーム（**#3** および **#4**）がこのタスクで定義されたものです。１つはログイン日付と時間を表示するものです。もう一つは、時間を表示するだけのものです。
2. タスク特性ダイアログを開きます（*Ctrl+P* または、オプション→タスク特性）。インタフェースタブをクリックします。
3. メインフォーム特性でズームし実行時に表示させるフォーム番号を表す式を指定します。この場合に、3 または 4 のどちらかを選択するようにしています。



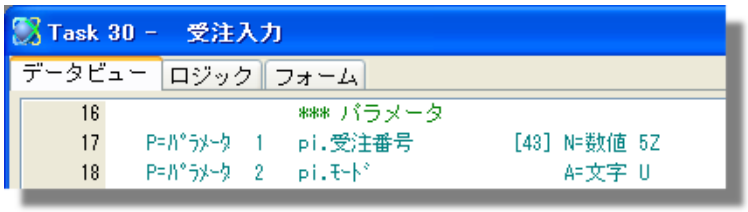


式はここに示されているように、**FORM** リテラルを使用してシングルクォートで囲むように指定する必要があります。これにより、フォームテーブル上のフォームが追加 / 削除された場合もフォーム番号が連動するようになります。

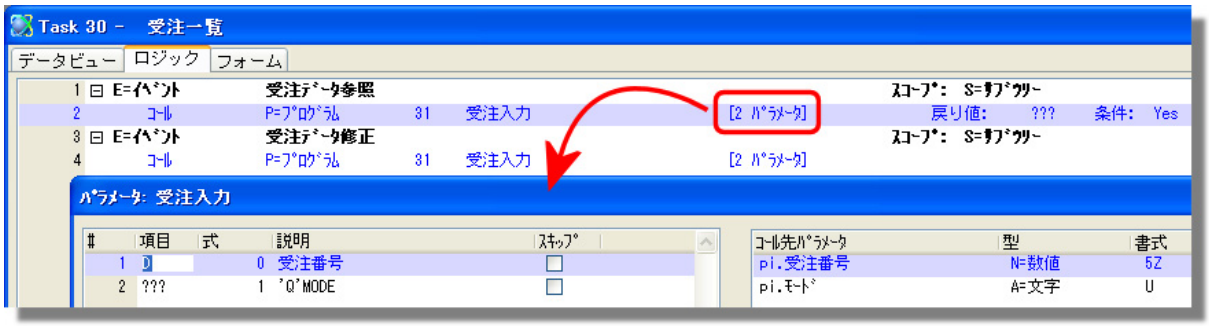
この例において、２つの異なるフォームが、ユーザの選択内容に応じて使用されることになります。

起動元と起動先のパラメータを同期させるには

データビューで**パラメータ**として項目を定義した場合、Magic は起動先プログラムと起動元プログラムの間で自動的に値を同期させようとします。例えば、表示されているように３つのパラメータが定義されいるものとします。



このプログラムを起動している場合、**パラメータ特性**を**クリック**すると以下のように表示されます。



２つのリストが**パラメータ**ダイアログで合っていない場合、**構文チェック**でエラーメッセージが表示されます。渡すか渡さないかを任意に指定できるパラメータの場合、**スキップ**オプションをチェックすることでエラー扱いにならなくなります。

起動されるプログラムのパラメータは、項目名と型、書式が表示されます。

ヒント : *パラメータ項目が定義されていないプログラムにパラメータを渡すこともできます。これは、下位互換を保証するためのものでこのようなプログラムを作成することは推奨しません。これらの変数項目をパラメータ項目に変更し、Ctrl+F（編集→検索と置換→クロスリファレンス）でクロスリファレンスを起動し、このプログラムを起動しているプログラムが正しくパラメータを渡しているかを確認する必要があります。*

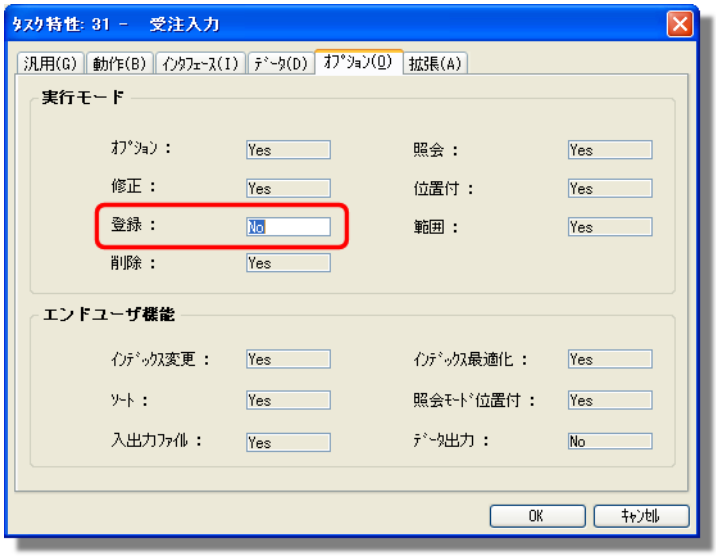
エンドユーザがレコードを追加することを防止するには

Magic のタスクは、デフォルトでユーザがレコードの追加、削除、修正を行うことを可能にしています。

ユーザが新しいレコードを追加することを防止したい場合、以下のようにします。

ユーザが登録モードに切り替えることを防止する

1. **タスク特性**（*Ctrl+P*）を開きます。
2. **オプション**タブを**クリック**します。
3. **登録**特性で、*No* を入力します。
4. ここで**ズーム**（*F5* または、**ダブルクリック**）して式を定義することもできます。実行時に定義された式が False と評価された場合、ユーザは登録モードに切り替えることができなくなります。



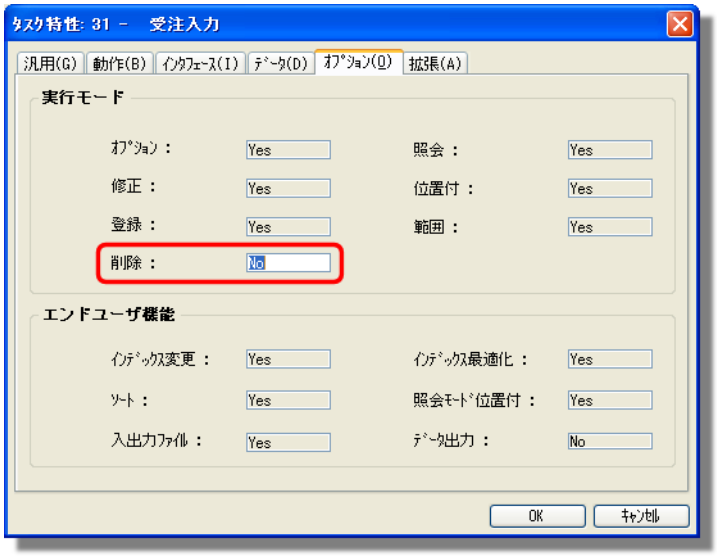
エンドユーザがレコードを削除することを防止するには

Magic のタスクは、デフォルトでユーザがレコードの追加、削除、修正を行うことを可能にしています。

ユーザが既存のレコードを削除することを防止したい場合、以下のようにします。

ユーザが削除モードに切り替えることを防止する

1. **タスク特性**（*Ctrl+P*）を開きます。
2. **オプション**タブを**クリック**します。
3. **削除**特性で、*No* を入力します。
4. ここで**ズーム**（*F5* または、**ダブルクリック**）して式を定義することもできます。実行時に定義された式が False と評価された場合、ユーザはレコードを削除することができなくなります。



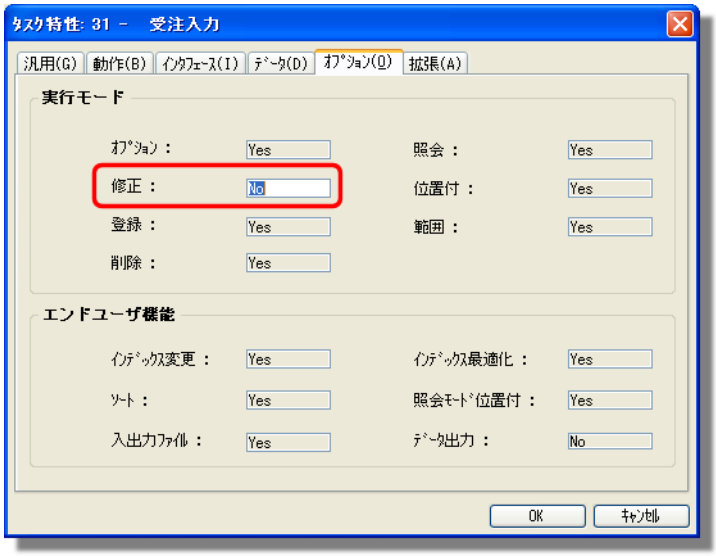
ユーザがレコードを修正することを防止するには

Magic のタスクは、デフォルトでユーザがレコードの追加、削除、修正を行うことを可能にしています。

ユーザが既存のレコードを修正することを防止したい場合、以下のようにします。

ユーザが修正モードに切り替えることを防止する

1. **タスク特性**（*Ctrl+P*）を開きます。
2. **オプション**タブを**クリック**します。
3. **修正**特性で、*No* を入力します。
4. ここで**ズーム**（*F5* または、**ダブルクリック**）して式を定義することもできます。実行時に定義された式が False と評価された場合、ユーザはレコードを修正することができなくなります。

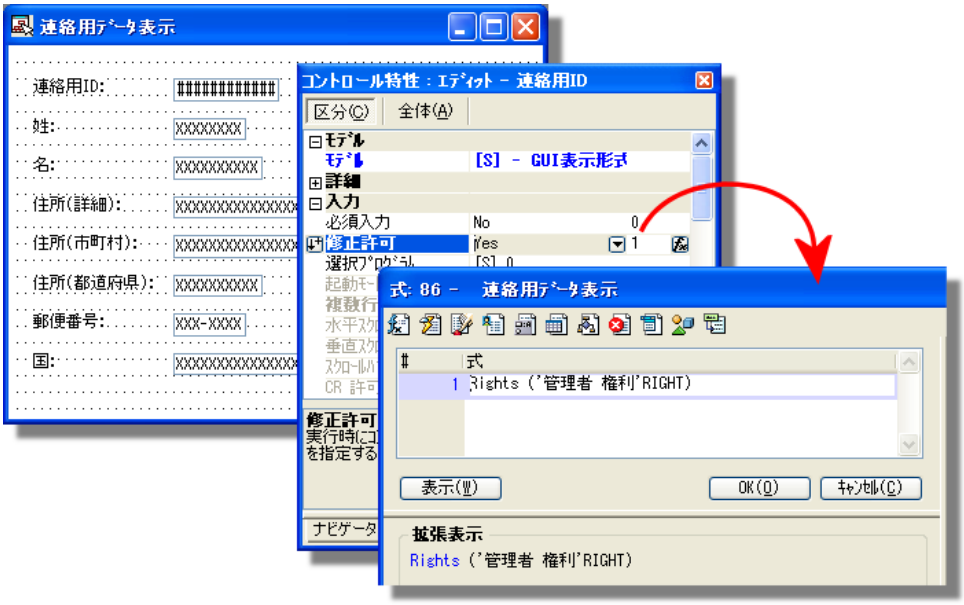


注： これによって、ユーザが照会モードから修正モードに切り換えてレコードを修正することを防止します。しかし、これはタスクがアクセスするすべてのレコードが対象となります。より詳細な条件で制御したい場合（例えば、レコード単位で）、あるいは異なる方法を使用したい場合（例えば、レコードがオープンされる前に有効なモードを設定したり、フィールドレベルでの変更を無効にしたり）は、「ユーザがレコードを修正することを防止するには」（88 ページ）を参照してください。

エンドユーザが特定の項目の内容を変更することを防止するには

タスクが修正モードの場合、デフォルトでは、ユーザはフォーム上のどのデータ項目も修正することができます。しかし、項目毎にデータを更新することを防止することができます。

項目を修正不可に設定する



1. 設定を変更したいコントロールにカーソルを置きます。*Ctrl+* クリックを使用することで複数のコントロールを同時に選択することができます。
2. *Alt+Enter* を押下してコントロール特性を開きます（すでに開いている場合は、特性シートをクリックします）。
   実行中に修正されることがない場合は、修正許可特性にカーソルを置き *No* を設定します。右側の式特性でズームして（または *fx* ボタンをクリックして）式エディタに論理値が返る式を定義することで、動的に切り替えることができます。この例では、**Right** 関数を利用して Admin 権利がある場合のみ修正できるようにしています。

これで、ユーザはこの項目にパークはできますが、定義式に設定された条件に合わない場合、修正できなくなります。

参照： 第 13 章：「マウスを使用した場合のみコントロールへのパークを可能にするには」（253 ページ）

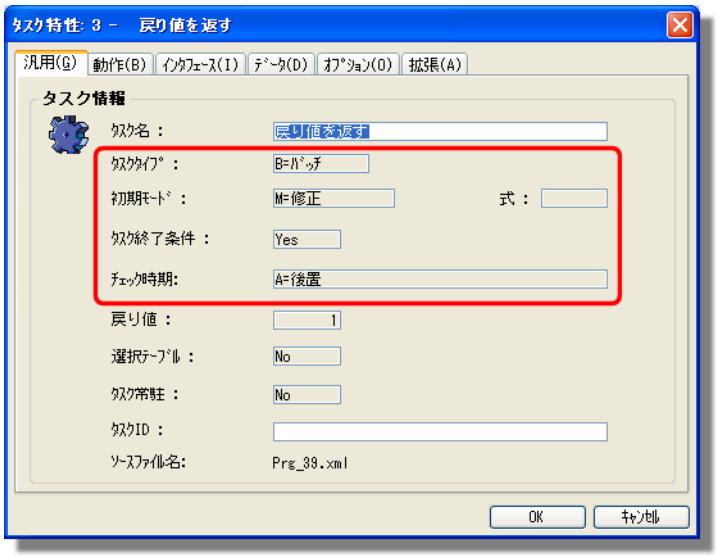
選択プログラムを作成するには

よく作成するプログラムの１つにデータの選択用のプログラムがあります。このタイプのプログラムでは、項目一覧がユーザに対して表示されます。ユーザは、選択したい項目を見つけるために上下にスクロールしたり、キー操作を行うことができます。*Enter* を押下することで項目が選択され、現在パークしている項目の内容が起動元の項目に設定されます。

これらのプログラムは、簡単に作成できます。

選択用プログラムを作成する

1. プログラムリポジトリで、*F4* を押下し一行追加します。
2. プログラムに名前を入力します。この名前はアプリケーション実行時、Magic は使用しないため任意の名前を使用することができます。



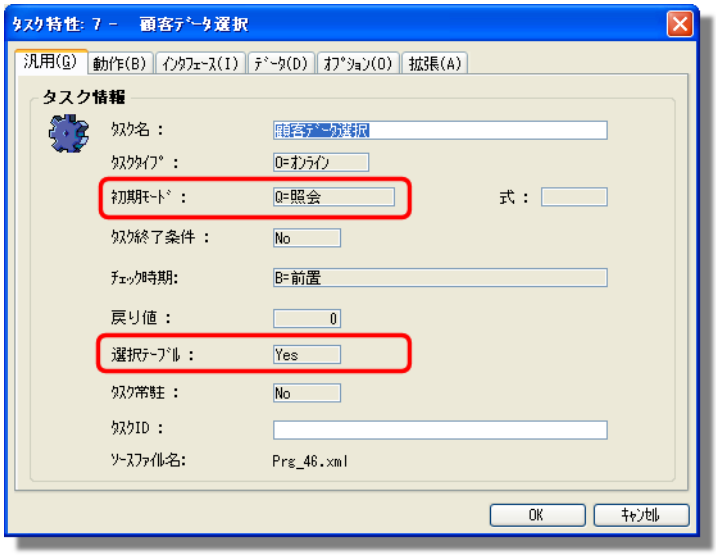
3. 名前カラムでズーム（*F5*）します。新規プログラムとなるため、タスク特性ダイアログが表示されます。ここで以下のパラメータを設定します。

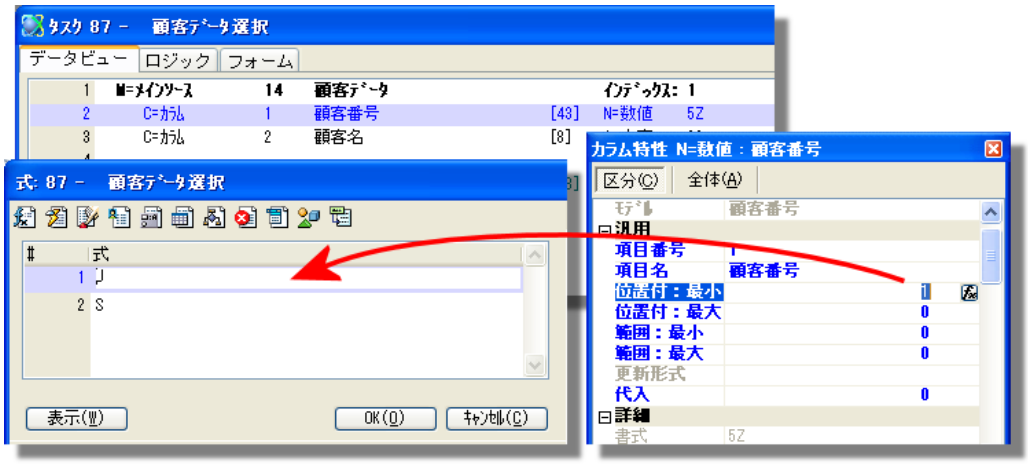
   初期モード：*照会*

   選択テーブル：*Yes*

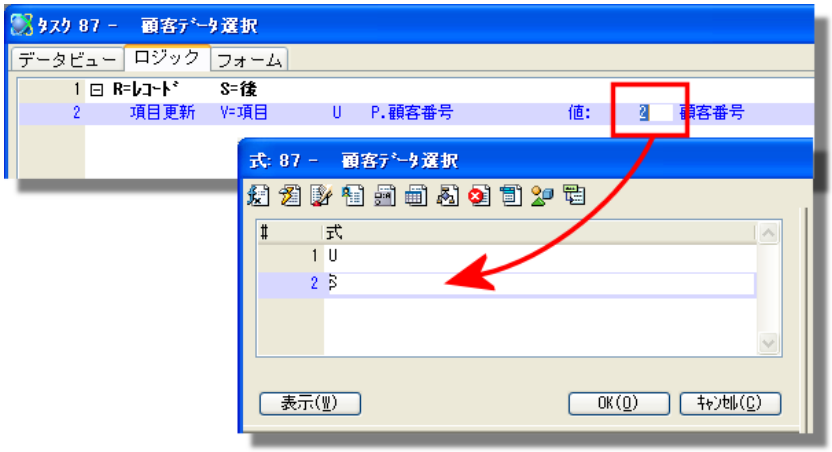
   選択処理ではデータの修正を行わないため、照会モードに設定してください。

   選択テーブル特性は、エンジンの動作を変更します。*Yes* に設定された場合、*Enter* が押下されるとレコード後を実行しタスクが終了します。



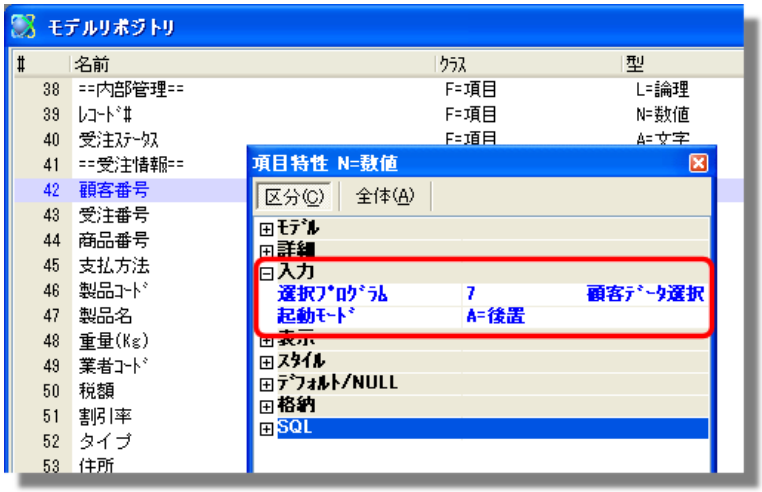


4. **データビューエディタ**では、一覧表示させたいデータソースを指定します。**データソース**特性では、アクセス中のレコードがロックされないように、**アクセス**特性を ***R= 読込***に設定します。
5. また、パラメータ項目を１つ定義します。このパラメータは、起動元に返す値としての**項目モデル**を使用します。この例では、顧客 ID を使用しており、顧客 ID モデル（**#26**）が使用されていますパラメータの名前が、選択する値の名前によく似ていること注意してください。間違った項目を更新しないように項目の名前の付け方を工夫してください。この場合、パラメータであることを表すため、接頭辞「pio」が使用されています。
6. データソースの**位置付：最小**特性にパラメータ項目を設定します。この例では、データソースの顧客 ID カラムを位置付けるためにパラメータの顧客 ID を設定しています。



7. **ロジックエディタ**内では、**レコード後**ロジックユニットを作成します。この**ロジックユニット**では、データソースの値でパラメータを更新します。これは選択された値を起動元に返すための処理です。
8. 最後に、フォームを作成します。これは簡単な**テーブル**コントロールを使用して作成します。***Ctrl+G*** を使用してフォームを作成することができます（「デフォルトのフォームレイアウトを自動的に作成するには」（128 ページ）を参照してください）。

選択プログラムを使用する

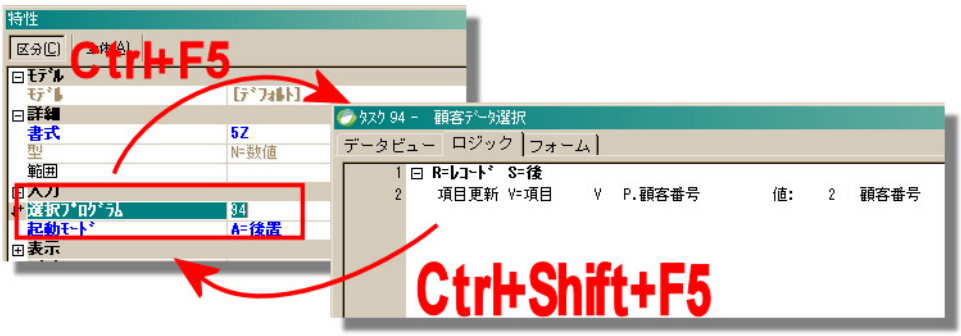


選択プログラムを最も簡単に使用する方法は、モデルやコントロールの**選択プログラム**特性にプログラムを設定することです。この場合、パラメータを明示的に指定する必要はありません。この特性が使用された場合は常に、顧客 ID は暗黙のうちに渡されるようになります。

注： データ数が少ない場合は、データソースを割り当てたコンボボックスを使用することもできます。

指定したプログラムに移動する

選択プログラム特性に設定したプログラムを開くには、通常は、モデルリポジトリを抜け、プログラムリポジトリを開いて該当するプログラムを開く操作が必要です。しかし、**選択プログラム**特性にカーソルが位置付けられている状態で ***Ctrl+F5***（**オプション→オブジェクトに移動**）を押下すると簡単に該当するプログラムを開くことができます。この方法でプログラムを開いた場合、***Ctrl+Shift+F5***（**オプション→オブジェクトから戻る**）を押下すると元の**選択プログラム**特性に戻ります。



注： Ctrl+F5 でプログラムを開いた場合、起動元の場所がスタックされます。このため、プログラムを開いた後、別のオブジェクトを開いた状態で ***Ctrl+Shift+F5*** を押下すると起動元に戻ります。

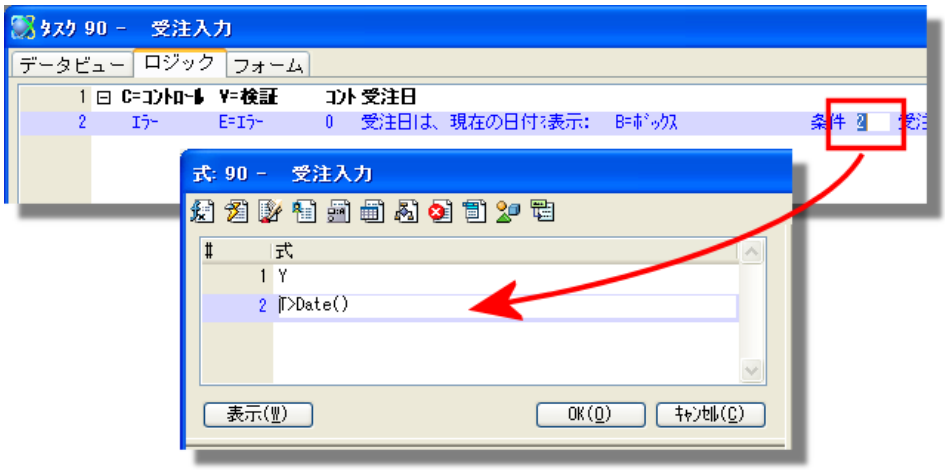
エンドユーザが入力したデータを検証するには

項目モデルを設定した時点で、Magic によるデータの検証処理の大部分が行われることになります。例えば、入力範囲を、1 から 100 の間の数値に制限したり、「A」、「B」、「C」または「X」のみを入力可能にしたい場合は、モデル特性で設定でき、これ以外の検証処理は必要ありません。

しかし、さらに複雑な検証条件が必要な場合、コントロール検証ロジックユニットを作成する必要があります。

コントロール検証ロジックユニットを定義する

1. 最初に、データ内容を検証する対象となるコントロール名特性を確認します。この例では、*受注日*になります。
2. ロジックエディタで、コントロール検証ロジックユニットを作成します。検証対象となるコントロールを選択します。コントロール名特性からズームして一覧から選択することができます。



3. コントロール検証ロジックユニットでは、エラー処理コマンドを定義します。以下の手順で行います。
   - *F4* を押下して 1 行追加します。
   - *E* を入力するか、次のプルダウンリストから*エラー*を選択します。*Tab* を押下します。
   - ズームして表示するエラーメッセージをテキストやデータ項目で定義します。
   - 式を使用しない場合は、*Tab* を押下し、直接エラーメッセージを入力します。
   - ステータス行にエラーメッセージを表示しない場合は、表示カラムで *B= ボックス*（デフォルト）を選択します。
   - 条件カラムでズームして式を指定します。データがエラーの場合は True と評価される式を入力する必要があります。
   - 必要に応じて、エラー処理コマンドの特性を修正して表示内容をカスタマイズすることができます。

受注日がが未来の場合、エラーメッセージが表示され、ユーザがエラー要因を修正するか、レコードの修正内容をすべてキャンセル（*Ctrl+F2* または、編集→キャンセル）しない限り次の項目に移動できなくなります。

ヒント：不適切なプログラミングを行うことで、取り消すことができないエラーメッセージが発生するようになる場合があります。例えば、条件カラムが Yes に設定された場合、エラーメッセージを止めることができません。このようなことが発生した場合、タスクを終了することは不可能になります。デバッグ中にこのような状態が発生したら、開発側のウィンドウに切り替えてデバッグ→停止を選択するか、ツールボックスの赤いボタンをクリック してください。

Tab によるコントロールの移動順序を設定するには

Magic のフォーム上のカーソルは、デフォルトでは上から下に、左から右に移動します。しかし、以下の手順に従うことで、必要に応じてデフォルトの設定を上書きすることができます。

TAB 順序を設定する

1. **自動 TAB 順序**を***解除***します。ここに表示されているようにアイコンがへこんでいる場合、有効な状態です（**描画→順序→自動 TAB 順序**でも確認できます）。
2. 各コントロールの **TAB 順序**特性に数値または数値が返る式を指定します。**TAB 順序**特性が灰色で表示されている場合は、修正できません。ステップ **#1** に戻って解除処理を行います。

   （注 : 式が定義されたコントロールは、*Tab* を割り付けることができないため灰色で表示されます。）
3. コマンドパレットの **TAB 順序を表示**のアイコン（**自動 TAB 順序**アイコンの次）を**クリック**することで *Tab* の順番が表示されます。*Tab* が有効なコントロールは赤く、無効なコントロールは灰色で表示されます。

コントロールの **TAB 順序**を変更した場合、**フォームエディタ**は連続性を維持した状態で保存します。例えば、**TAB 順序**を *1* から *2* に変更した場合、**フォームエディタ**は今まで *2* になっていたコントロールの値を *1* に変更します。

複数のコントロールの TAB 順序を一度に設定する

複数のコントロールの **TAB 順序**を一度に設定することができます。

1. 最初に、前述のように、**自動 TAB 順序**を***解除***します。
2. *Ctrl* を押下しながら設定対象となるコントロールを**クリック**していきます。
3. 選択されたコントロール内での最初の **TAB 順序**を入力します。

この例の場合、この順序でコントロールを選択しました。表示価格、値引き価格、および公開日付。選択された状態で **TAB 順序**に *3* を入力すると以下のように設定されます。表示価格（*3*）、ディスカウント（*4*）公開日付（*5*）。

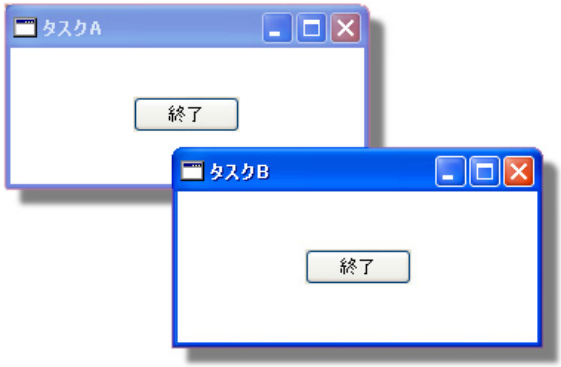
参照 : 第 11 章 :「特定のコントロールにカーソルを移動させるには」（226 ページ）
第 11 章 :「カーソルを一定の方向に移動させた場合のみ処理を実行させるには」（228 ページ）

サブタスクレベルからプログラムを終了するには

一緒に実行しているタスクに関連するスタックを持つことがよくあります。これらのタスクは、時々 1 つのウィンドウのように表示させることがあり、その際、終了ボタンをクリックするとタスクのスタック全体を終了させるように動作するのが一般的です。

このようなプログラムにする最もよい方法は、以下のようにイベントロジックユニットを使用することです。

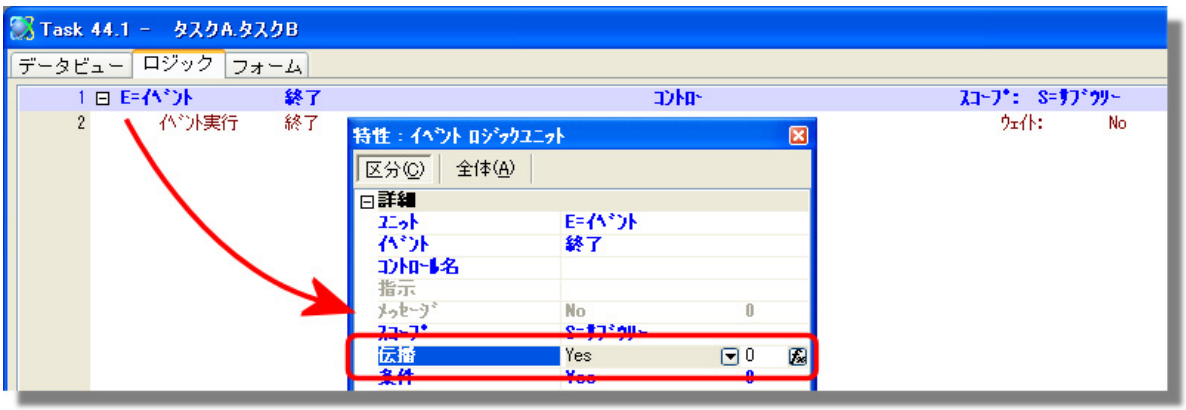
終了イベントを伝播させる



ここに、**タスク A** と**タスク B** があります。タスク A の終了ボタンをクリックすると通常は両方のタスクが閉じます。しかし、タスク B を閉じた場合、フォーカスがタスク A に移るだけです。

タスク B を閉じるとタスク A も閉じるようにするには以下のようにします。

1. 最初に、終了イベントを発行する終了ボタンを作成します。ユーザが  をクリックしたり、*Esc* を押下した場合と同じイベントのため、この方法ですべての状況がカバーされます。
2. 次に、終了イベントによって実行され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は終了イベントで実行されますが、タスク B に伝播されるようにも設定します。
3. ロジックユニット内では再度終了イベントが発行され、これによってタスク A は終了します。



２つのタスクは明確に分かれて表示されていますが、通常タスク B はタスク A のサブフォームとして使用されます。

参照： 第 8 章：「サブフォーム上のコントロールから親のコントロールに自動的に戻るには」（182 ページ）

タスクの編集中に編集内容を保存するには

PC が動作中に変更内容を保存する方が望ましいやり方です。これは複雑なプログラムを編集する時こそ必要です。間違いを犯したりシステムがクラッシュした場合、変更内容が保存されていれば、最後に保存した内容から作業が継続することができます。

プログラムの編集中に保存する

1. ***Ctrl+S***（**オプション→プログラム保存**）を押下することで編集中のプログラムの内容を保存することができます。

これによってどのような変更内容もディスクに保存されます。

タスクにロジックを作成するには

タスクのロジックは**ロジックエディタ**で定義します。**ロジック**タブを**クリック**することで**ロジックエディタ**は開きます。

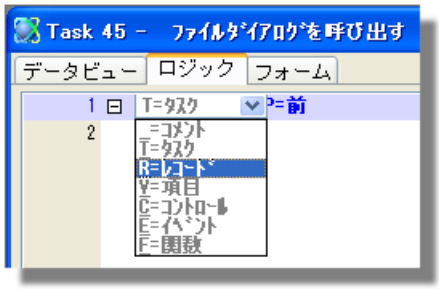
タスクのロジックは**ロジックユニット**と呼ばれるコードのブロック内に保持されます。**ロジックユニット**は、ハンドラを表しているヘッダ行と 0 個以上の*詳細行*（処理コマンド）から構成されています。

ロジックユニットはこれ自身は無手順です。すなわち、**ロジックエディタ**内の位置に依存せず何らかのイベントに対応して実行されます。

ロジックユニット内の***処理コマンド***は手続型で、上から下の順番で実行されます。これらの処理コマンドは、Magic プログラム内で実行される「コード」になります。これらは非常に強力で、他の言語で作成する行数よりはるかに少ない行数で開発することができます。

ヘッダ行を入力する

1. **ロジックエディタ**内のヘッダ行を作成したい行に移動します。
2. ***Ctrl+H*** を押下してヘッダ行を作成します。
3. **ヘッダタイプ**を選択します：***タスク、レコード、項目、コントロール、イベント、***または***関数***があります。
4. **ヘッダタイプ**に依存する**ヘッダ特性**を入力します。



処理コマンドを入力する

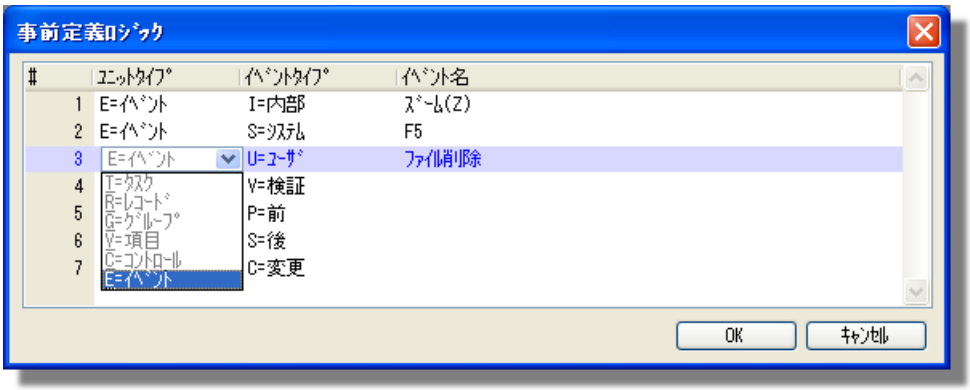
1. **ロジックユニット**内で ***F4*** を押下し、処理コマンドを定義することで必要なロジックを作成します。各処理コマンドの設定内容にはいくつかの違いはありますが、コンテキストヘルプをを参照して作業することができます。処理コマンドについては、「タスクに処理コマンドを設定するには」（100 ページ）を参照してください。

ロジックヘッダを簡単に作成するには

事前定義ロジックテーブルにロジックヘッダのパターンを定義しておくことで、簡単にロジックヘッダを作成することができます。

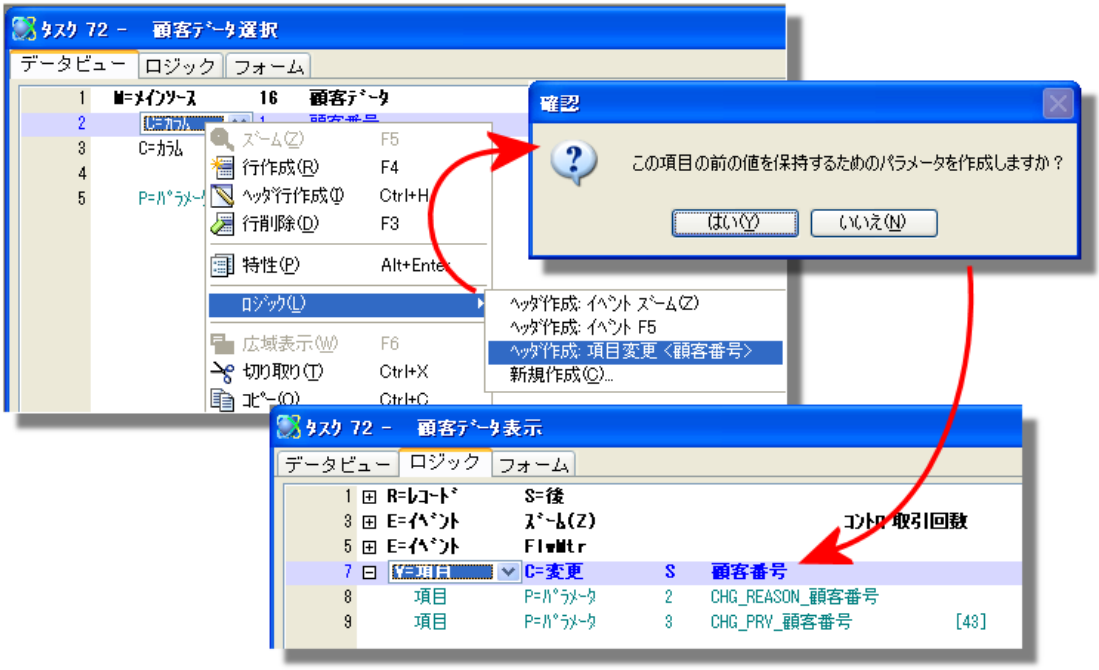
事前定義ロジックを追加する

1. 事前定義ロジックテーブル（オプション→設定→事前定義ロジック）を開きます。
2. よく作成するロジックヘッダのパターン（ユニットタイプ、イベントタイプ、イベント名）を登録します。



データ項目に対応したロジックヘッダを作成する

1. データビューエディタの任意のデータ項目にカーソルを移動します。
2. マウスで右クリックを行いコンテキストメニューを開きます。
3. メニューからロジックをクリックします。ここには、事前定義ロジックテーブルに登録されているヘッダの他に、項目変更や新規作成が表示されます。項目変更を選択します。
4. ロジックエディタに表示か切り替わり、項目変更ロジックユニットのヘッダが追加されます。さらに、ヘッダにパラメータを追加するかどうかを確認するダイアログが表示されます。ここで Yes をクリックするとパラメータ項目が追加されます。



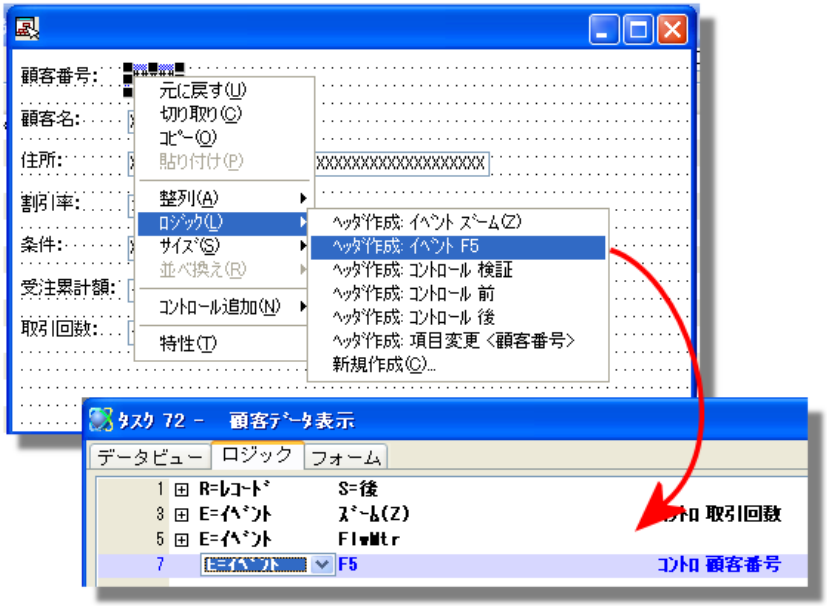
任意ロジックヘッダを作成する

1. ロジックエディタの任意の行にカーソルを移動します。
2. マウスで右クリックを行いコンテキストメニューを開きます。

3. メニューからロジックをクリックします。ここには、事前定義ロジックテーブルに登録されているヘッダの他に、新規作成が表示されます。ヘッダ作成 イベント ズームを選択します。
4. ロジックエディタの最終行にイベントロジックユニットのヘッダ行が追加されます。

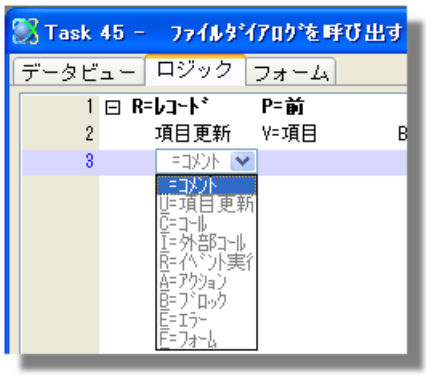
コントロールに対応したロジックヘッダを作成する

1. フォームエディタの任意のコントロールにカーソルを移動します。
2. マウスで右クリックを行いコンテキストメニューを開きます。
3. メニューからロジックをクリックします。ここには、事前定義ロジックテーブルに登録されているヘッダの他に、項目変更や新規作成が表示されます。ヘッダ作成 イベント F5 を選択します。
4. ロジックエディタに表示か切り替わり、イベントロジックユニットのヘッダ行が追加されます。このロジックユニットのコントロール名特性には、フォームで指定したコントロールのコントロール名が設定されます。



タスクに処理コマンドを設定するには

各ロジックユニットには、処理コマンドを入力する行があります。以下のようにして処理コマンドを設定します。



1. 処理コマンドを入力したい行で *F4* を押下します。
2. 処理コマンドのタイプを選択します。ショートカット文字を入力するか、プルダウンリストから選択します。
3. 選択されたタイプにもとづいた特性を設定します。

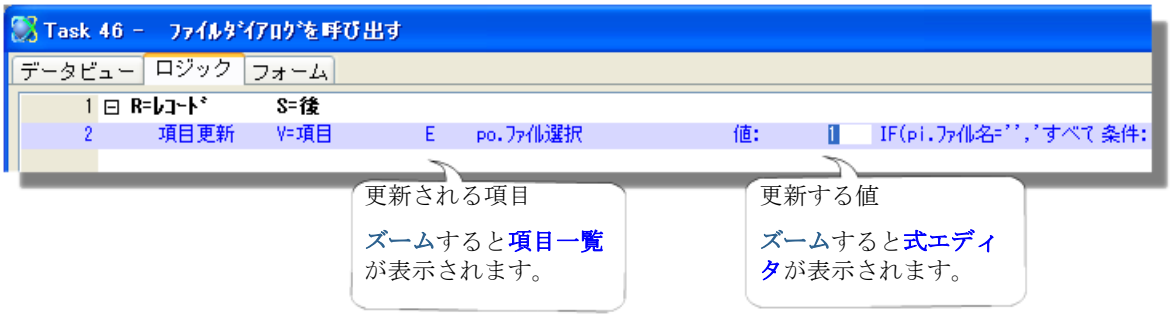
８つのタイプの処理コマンドを入力することができます。以下で各処理コマンドの概要を説明しています。

条件特性を使用する

すべての処理コマンドには、条件特性があります。基本的には「if」の構文に当たります。条件特性が *Yes* の場合、または定義されて式が実行時に True と評価された場合この処理コマンドは実行されます。

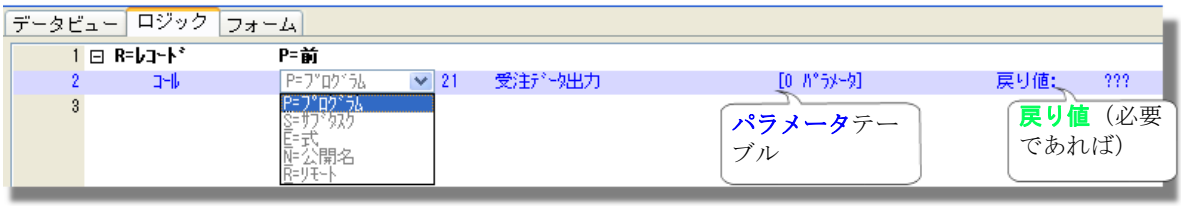
- 条件が *Yes* の場合は、処理コマンドが常に実行されます。
- 条件が *No* の場合は、処理コマンドは決して実行されません。
- 条件が *数値* の場合、式の番号を示しています。条件式を入力するには、ここからズームして式エディタに入り、式を入力します。*Enter* を押下してロジックエディタに戻ります。

項目更新



項目更新処理コマンドは、一般的な言語における代入演算子と同じような処理を行います。ここでは式の評価結果の値を変数項目にコピーしています。

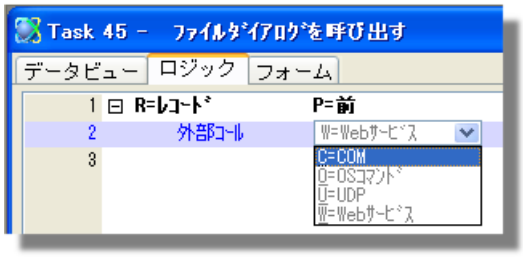
コール



コール処理コマンドは、別の Magic タスクを実行させます。何を起動するかによって設定する特性が異なります。例えば、コール処理プログラムはプログラムリポジトリ内の他のプログラムを番号指定で呼び出します。コールサブタスクは、現在のプログラムのサブタスクのみ呼び出すことができます。コールプログラム名は、公開名を指定してプログラムを呼び出します。

呼び出されるプログラムにパラメータが定義されている場合、パラメータテーブルが使用されます。同様に、戻り値特性は、起動したプログラムからの戻り値を受け取る項目を指定することができます。

外部コール

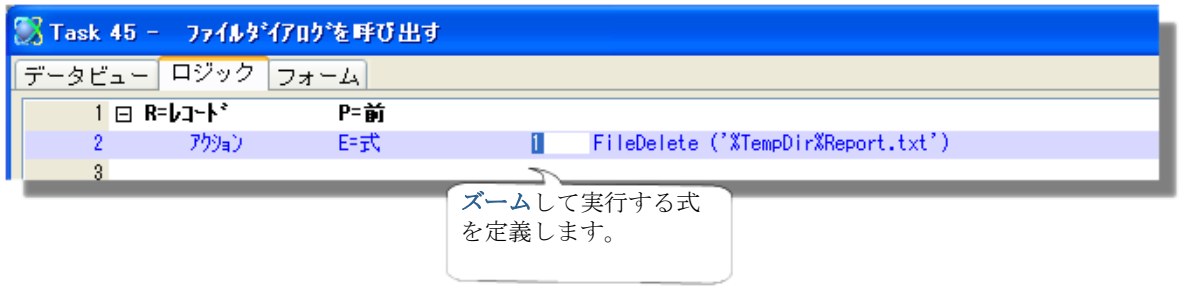


外部コール処理コマンドは、Magic 以外のプログラムを呼び出す場合に使用します。OS のスクリプトや Web サービス、ActiveX オブジェクトなどを呼び出すことができます。これらは各々処理内容に設定内容も異なっています。

イベント実行

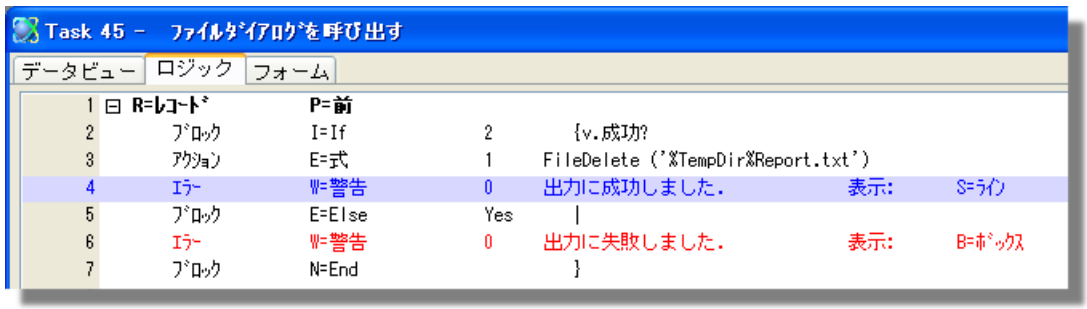
イベント実行処理コマンドは、イベントを発行する処理を行います。ユーザがキー（システムイベント）を押下したり、プッシュボタンによって発行された場合と同じようにイベントが発行されます。

アクション



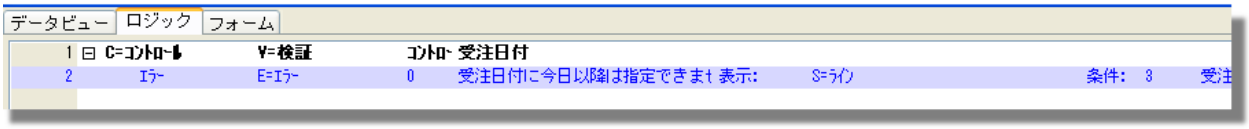
アクション処理コマンドは、データ項目を指定しないで式を実行するために使用されます。この例では、**FileDelete()** 関数を実行しています。

ブロック



ブロック処理コマンドは一般的な言語で行われるようなグループ処理で使用されます。この例では、処理の戻り値にもとづいた if/then/else が定義されています。

エラー

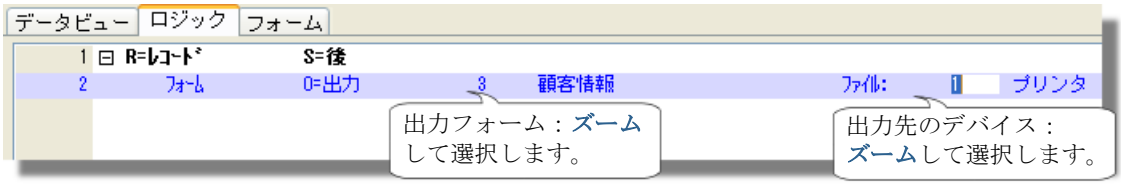


エラー処理コマンドには 3 種類のオプションあります。

- ***警告***は、ユーザにメッセージを表示するだけの機能があります。
- ***エラー***は、ユーザにメッセージを表示し、エラーが解決するまでこれ以降の処理を継続させないようにします。
- ***復帰***は、コントロールロジックユニットとイベントロジックユニットでのみ有効です。エラーが発生した場合、ロジックユニット内を逆のフローで制御が移動します。

エラーはユーザが不正なデータを保存することを防止するため、処理を完全に停止させることができます。このことについての詳細は、「エンドユーザが入力したデータを検証するには」（93 ページ）を参照してください。

フォーム



フォーム処理コマンドは、フォームの入出力を行う場合に使用されます。一般に、フォームを出力する場合、プリンタに帳票形式で、他のプログラムに渡すためにテキスト形式で、インターネット用画面として HTML 形式で出力されます。フォーム入力は、通常テキスト形式で使用されます。

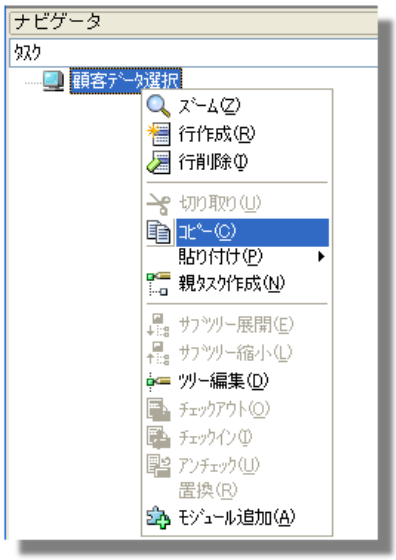
サブタスクとしてタスクをコピーするには

他のプログラムによって呼び出されるだけのプログラムを作成する場合があります。このような場合、そのプログラムをサブタスクとすることで、プログラムの処理の流れを理解することが容易になることがあります。このような内容について説明します。

プログラムをサブタスクとしてコピーする

1. ナビゲータペインでコピーするタスクを選択します。
2. 右クリックでコンテキストメニューを表示し、コピーを選択します。
3. このタスクを貼り付けたいプログラムに移動します。
4. ナビゲータペインで、貼り付け先のタスクの親またはそのサブタスクを選択します。
5. 右クリックでコンテキストメニューを表示します。子タスクとして貼り付ける場合は貼り付け（*Ctrl+V*）、兄弟関係のタスクとして貼り付ける場合は、別のツリーに貼付を選択します。

このような操作を実行すると、プログラムはサブタスクとしてコピーされます。この後、パラメータの設定内容やフォームを変更する必要が出てきます。

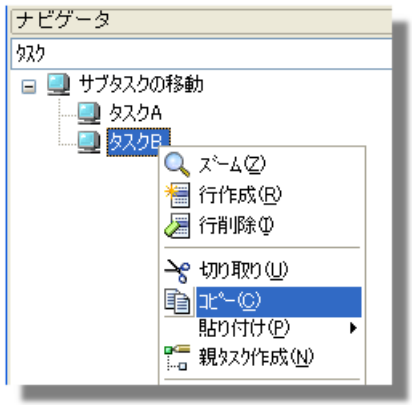


サブタスクをルートタスクにするには

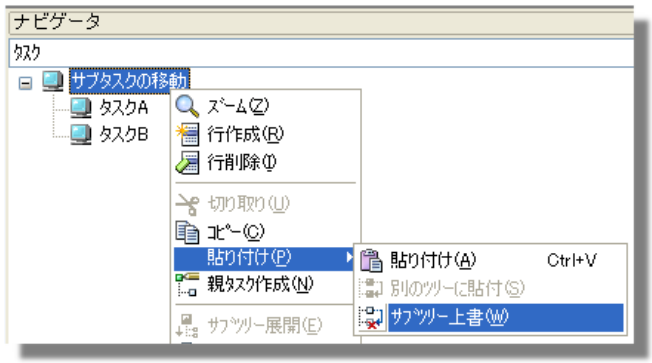
独立したプログラムにしたいサブタスクが存在する場合があります。または、削除したいダミーの開始プログラムを持っているサブタスクがあるかもしれません。どちらの場合も、サブタスクを一番上のルートタスクとして移動する必要があります。ここではこの方法について説明しています。

サブタスクを先頭レベルに移動する

1. ナビゲータペインで、移動したいタスクにカーソルを置きます。
2. 右クリックでコンテキストメニューを開き、コピーを選択します。

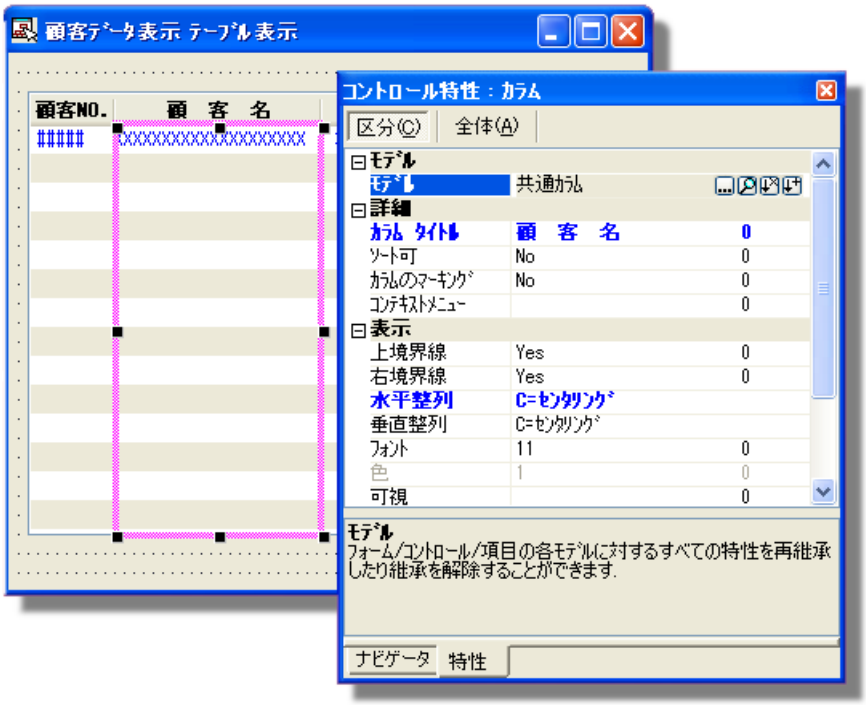


3. 先頭レベルのタスクにカーソルを置きます。
4. 右クリックでコンテキストメニューを開き、貼り付け→サブツリー上書を選択します。
5. 上書き確認ダイアログではいをクリックします。



6. これで、下位レベルのタスクが先頭レベルのタスクに上書きされました。

テーブルコントロールのカラムを選択するには



通常、テーブルコントロールをクリックと、テーブル全体が選択されます。これは、テーブルがグループ化されているためです。

テーブルコントロールのカラムだけを選択したい場合、最初のテーブル行より下にマウスカーソルを置き、*Alt+* クリックを実行します。すでに他のカラムが選択されている状態の場合は、**Tab** によって次のカラムに移動することができます。

ヒント :*Alt+Shift+* クリック *を使用することで複数のカラムを選択することもできます。同時に複数の*カラム特性 *を変更したい場合に便利です。*

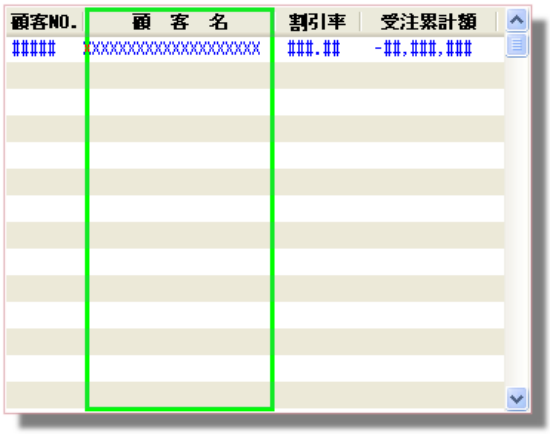
強調表示カラムの左側にコントロールや項目をドロップしてカラムを作成するには

通常、コントロールや項目をテーブルにドロップする場合、ドロップしたカラムの右側にカラムが追加されます。コントロールまたは項目を左側に追加させたい場合は、、*Shift* を押下しながらドロップします。

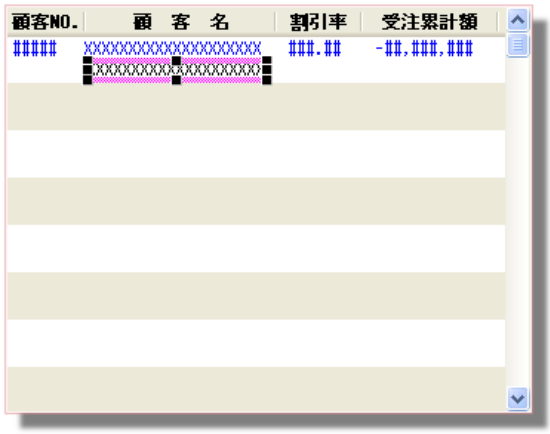
複数のコントロールを同じカラムに配置するには

通常、コントロールをテーブルコントロールに配置すると、ドロップしたコントロールの右側にカラムが追加されます。しかし、*Alt* を押下しながらドロップすることで、選択したカラム内にコントロールが追加されます。

Alt キーを押下しながら項目をドロップした場合の結果



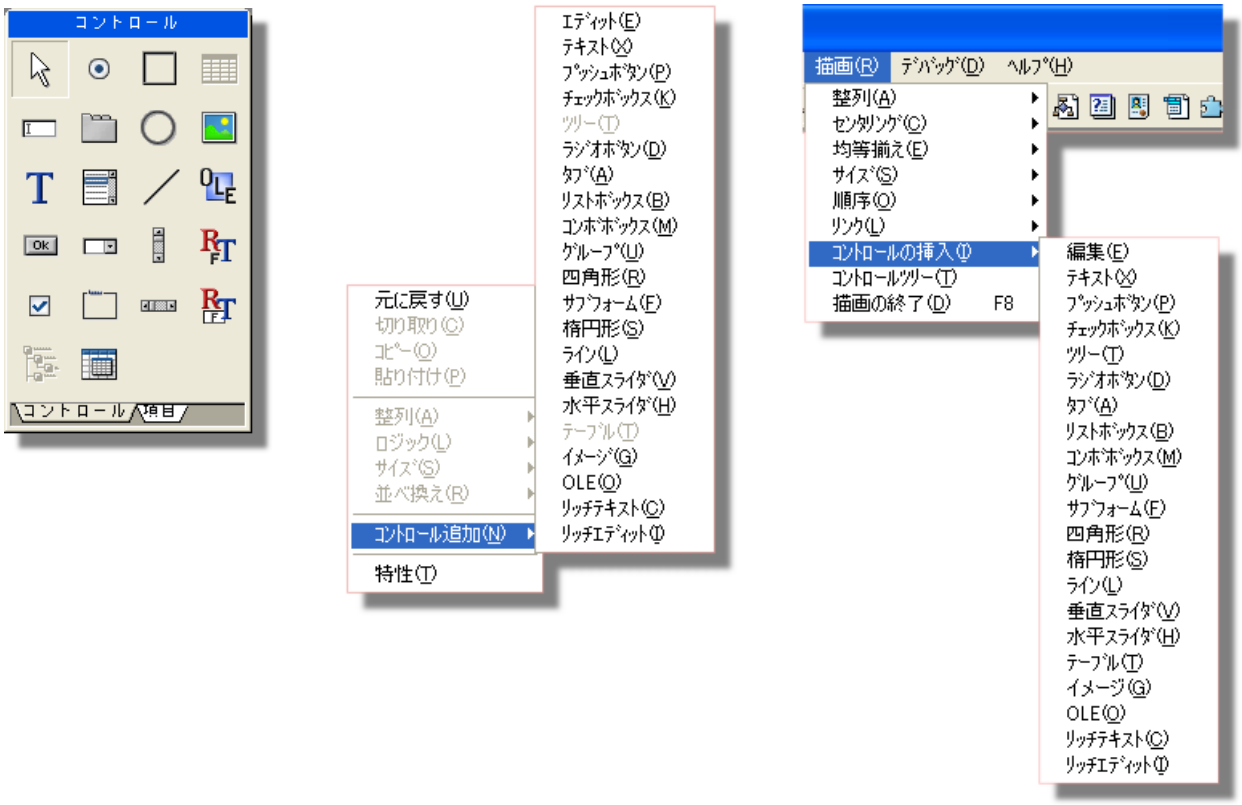
項目をドロップする前



ドロップした後

フォームにコントロールを追加するには

タスク



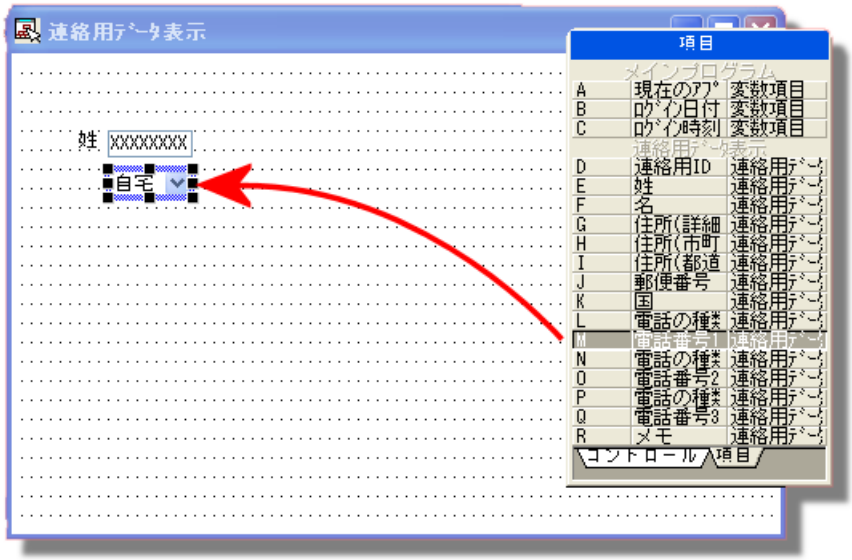
コントロールパレット
配置するコントロールを選択しクリックします。その後、フォーム上の配置したい場所をクリックします。

コンテキストメニュー
配置するコントロールを選択しクリックします。フォームにコントロールが表示されます。

プルダウンメニュー
配置するコントロールを選択しクリックします。フォームにコントロールが表示されます。

上の図で示されるように、コントロールをフォームに追加するには３つの方法があります。各々空の（データ項目が割り当てられていない）コントロールが定義されます。いくつかのコントロール（ラインや、四角形、およびテキストなどのスタティックコントロール）は項目を割り当てることはありません。これ以外のコントロールではに、フォームに配置後、コントロール特性で項目を割り当てることができます。

項目パレットを使用する



コントロールをフォームに配置する別の方法として、**項目**パレットを使用することができます。ここで項目を**クリック**し、フォーム上の配置したい場所で**クリック**するだけです。この方法では、項目に対応するコントロールが配置されます。

データが（**項目モデル**やデータソースまたはデータビュー）で定義される際、使用するスタイルを特性の**スタイル**セクションに設定します。この例では、「電話タイプ」のモデルはコンボボックスを使用するように設定されているため、**項目** N（電話タイプ１）がコンボボックスとして配置されます。

コントロールを連続して複数回ドロップするには

通常、コントロールをドロップすると、カーソルは通常の形に戻ります。この後、別のコントロールを選択する場合は、いったんメニューや**コントロールパレット**に戻る必要があります。

しかし、同じコントロールを複数回**ドロップ**したい場合は、***Ctrl*** を押下した状態でコントロールを**ドロップ**します。.

ヒント : *コントロールのドロップ処理を中断する場合は、**Esc** を押下します。*

子コントロールを選択しないでコンテナコントロールを選択するには

通常、コンテナコントロール（タブやテーブル）をクリックすると、子コントロールとして定義されたコントロールも一緒に選択されます。これは子コントロールの仕様です。

しかし、コンテナコントロールだけを選択したい場合は、*Ctrl* キーを押下しながらコンテナコントロールをクリックします。

再利用のためにフォームの書式を保存するには

フォームは簡単に作成することができます。また、フォームテンプレートを使用して複雑なフォームを保存することで、フォームを規格化したり、再利用することができます。

フォームをテンプレートとして保存する

1. テンプレートとして保存したいフォームを開きます。
2. オプション→テンプレート作成を選択します。
3. ファイルダイアログが表示されたら、ファイル名を入力します。デフォルトの拡張子は *.mft* です。
4. 保存をクリックします。

テンプレートフォームを使用する

1. テンプレートを読み込みたいフォームを開きます。
2. オプション→テンプレート読込を選択します。
3. ファイルダイアログが表示されたら、読み込むテンプレートファイルを選択します。
4. 開くをクリックします。.

テンプレートがフォームに読み込まれます。フォーム上のすべてのコントロールは、オリジナルのフォームとなります。この後、これらのコントロールにタスクの項目を割り当てる作業が必要になります。

注： フォームの規格化を行う別の方法として、フォームとコントロールのモデルを使用することがあります。これらは、見た目が同じようなフォームを定義することを可能にするだけでなく、その規格自体が変更された場合にモデルを変更することで、モデルを使用しているすべてのフォームやコントロールが自動的に変更された規格に合うようになります。

複数のコントロールを同時に選択するには

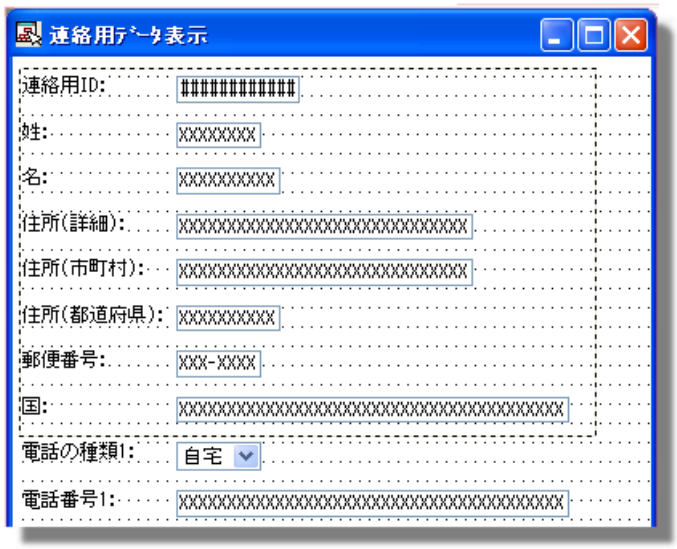
コントロールをクリックすることでそのコントロールを選択することができます。また、同時に複数のコントロールを選択する場合も考えられます。これは、グループでコントロールを移動する場合に便利です。また、コントロールの共通の特性を変更する場合にも役立ちます。例えば、同じようなコントロールのグループを選択し、設定されている同じモデルをすべてに変更したり、すべてのフォントを変更するようなことができます。

コントロールをまとめて選択する方法には、*ラバーバンド*（選択範囲を示す枠）と、*Ctrl+* クリックの２つの方法があります。

ラバーバンドを使用してコントロールを選択する

1. ラバーバンドを使用してコントロールを選択するには、選択するすべてのコントロールの外側のフォーム上をクリックします。
2. すべてのコントロールのまわりを矩形のボックス状にドラッグし、マウスを放します。

エリア内のすべてのコントロールが選択されます。コントロールがラバーバンドに交差した場合も選択されます。



Ctrl+ クリックを使用してコントロールを選択する

Ctrl+ クリックを使用して１つずつコントロールを選択することもできます。

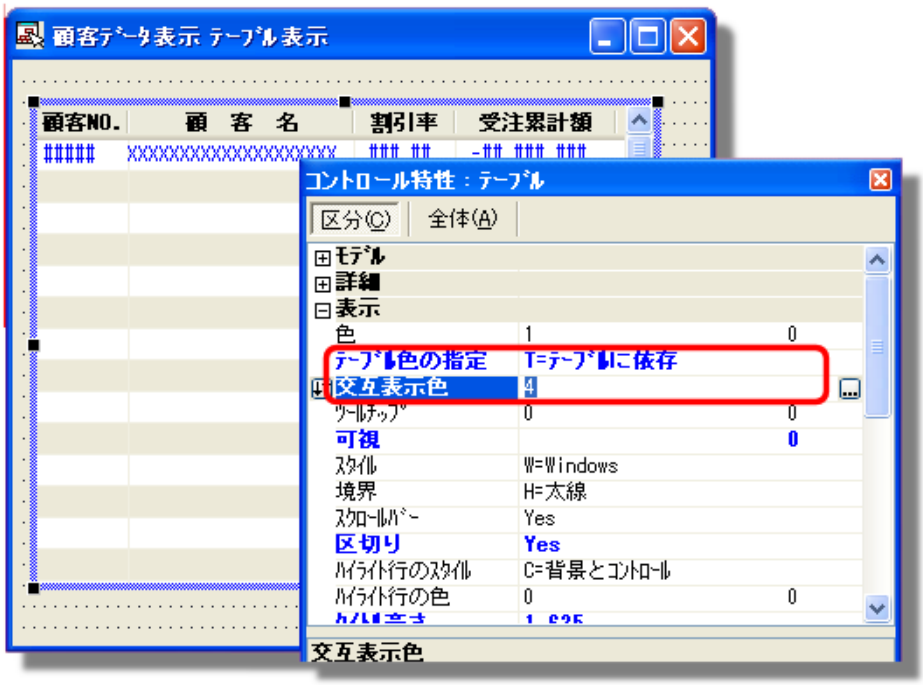
1. *Ctrl* を押下した状態でコントロールをクリックします。
2. *Ctrl* を押下したまま、別のコントロールをクリックします。
3. **#2** の操作をすべてのコントロールに対して繰り返します。
4. 選択を解除したいコントロールがあった場合は、同じ状態でもう一度コントロールをクリックします。

ヒント :*スペースバーを押下することでコントロールを選択解除することができます。*

テーブルコントロールに交互色を表示させるには

サイズの大きなテーブルの場合、行を目で追うのは大変な場合があります。このような場合の 1 つの解決策として、列の色を交互に変更する方法があります。

テーブルで交互色を使用する

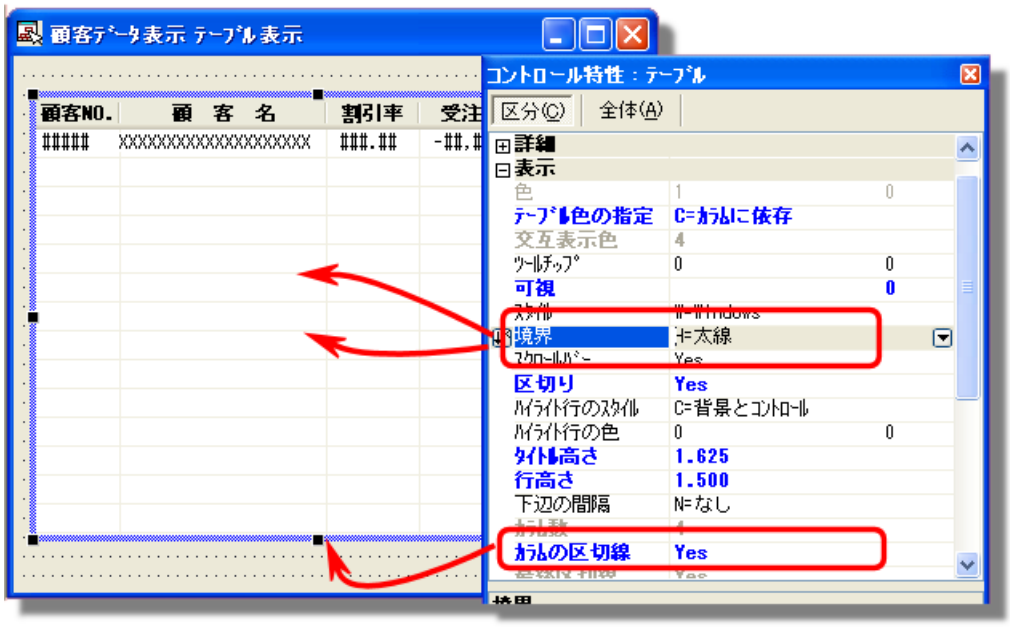


1. *Ctrl+* クリックを行って、テーブルコントロールを選択します。
2. コントロール特性において以下の操作を行います。
   - テーブル色の指定特性を *T= テーブルに依存* に設定します。
   - 交互表示色特性に背景色となる色番号を選択します。番号を直接入力するか、ズームして基本色一覧から選択します。

テーブルは、交互に色が変わった状態で表示されます。基本色は色特性（この例では *1*）で指定された色になり、交互表示色特性（この例では *5*）で指定された色と交互に表示されます。

テーブルコントロールの区切り線の表示／非表示を行うには

タスク



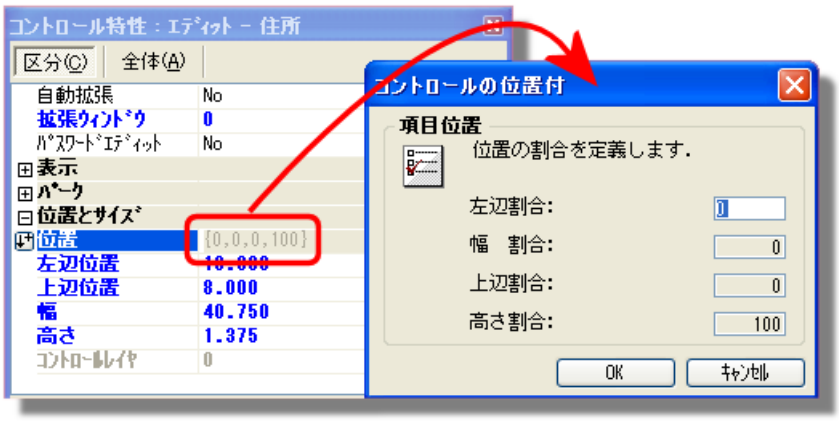
テーブルの**コントロール特性**に値を設定することによって、行とカラムの区切り線を表示させるかどうかを制御することができます。

区切り特性を *Yes* に設定すると、水平の行区切り線が表示されます。*No* に設定すると表示されなくなります。

カラムの区切線特性を *Yes* に設定すると、垂直のカラム区切り線が表示されます。*No* に設定すると表示されなくなります。

ヒント : *アプリケーションのテーブル表示に区切り線が表示されなくても、テーブルの編集時にテーブル上のコントロールを整列させることでより使いやすいものになる場合があります。*

フォーム上のコントロールのサイズを表示テキストに合わせるには



GUI 環境の利点の一つは、ウィンドウがすべてユーザによってサイズ変更できることです。しかしこのことは、すべてのユーザに対して最良のフォームをデザインすることを難しくする要因でもあります。

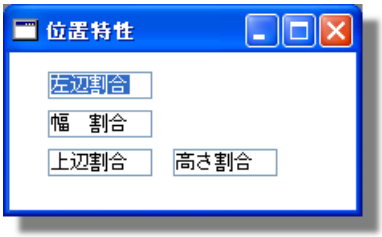
位置特性は、このような問題の解決に役立ちます。位置特性は、フォームのサイズの変更内容に応じて各コントロールのサイズを指定することを可能にするものです。位置特性の 1 つに ***100*** を入力すると、ウィンドウの指定された方向に対して伸びた分の 100% の割合でサイズを変更することを意味しています。***50*** が指定された場合、半分の長さでサイズが変更されます。どのような値が最適化を確認するために、実験することができます。

テーブルコントロールで位置特性を使用すると動作の違いがよくわかります。テーブルの高さ割合が 100% の場合、フォームサイズを変更（拡張）すると、行が追加されます。幅割合が 100% の場合、カラムのサイズが変更され、広くなるためより多くのデータが表示されます。

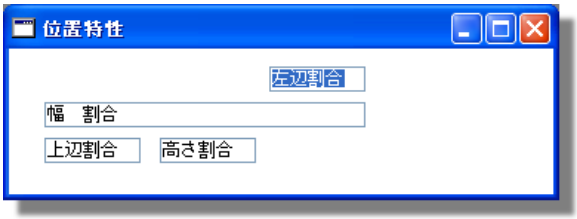
ヒント : *コントロールは元のサイズより小さくはなりません。このため、フォームのサイズが変更されることを考慮して設計する場合、必要最小限のサイズで各コントロールを作成する必要があります。*

以下の図は、位置を 100% に設定した場合の効果を表しています。各エディットコントロールは、特定の方向に対して 100% に設定されており、ウィンドウのサイズを拡張すると、どのような結果になるかが確認できるようになっています。

コントロールの位置を 100% にした場合の効果



拡張する前のウィンドウ



右方向に拡張します。**左辺割合**と**幅割合**が両方とも 100% の場合、コントロールの右側がウィンドウの右端に合わせて動きます。

しかし、**左辺割合**が 100% に設定されているため、コントロールの左端も動きます。このためコントロールのサイズ自体は拡張されません。

下辺を拡張します。**上辺割合**と**高さ割合**の両方が 100% の場合、ウィンドウの下辺に合わせてコントロールの下辺が動きます。

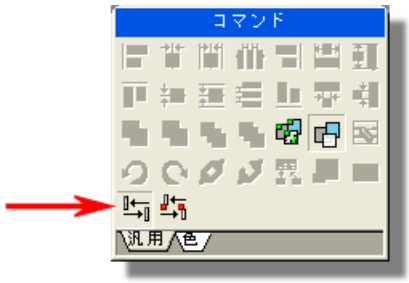
しかし、**上辺位置**が 100% に設定されているためコントロールの上辺も動きます。このためコントロールの高さ自体は拡張されません。

コントロールの TAB 順序を変更するには

Magic のフォーム上のカーソルは、デフォルトでは上から下に、左から右に移動します。しかし、以下の手順に従うことで、必要に応じてデフォルトの設定をオーバライドすることができます。

TAB 順序を設定する

1. **自動 TAB 順序**を***解除***します。ここに表示されているように、アイコンがへこんでいる場合は有効な状態です。(**描画→順序→自動 TAB 順序**でも確認できます)。



2. 各コントロールの **TAB 順序**特性に数値または数値が返る式を指定します。**TAB 順序**特性が灰色で表示されている場合は、修正できません。ステップ **#1** に戻って解除処理を行います。
   (注 : 式が定義されたコントロールは、***Tab*** を割り付けることができないため灰色で表示されます。)

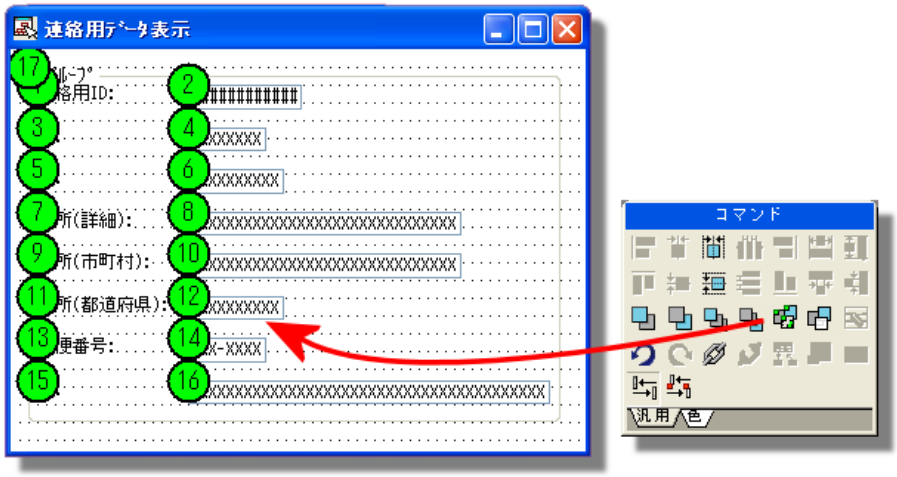
3. **コマンド**パレットの **TAB 順序を表示**のアイコン(**自動 TAB 順序**アイコンの次)を**クリック**することで ***Tab*** の順番が表示されます。***Tab*** が有効なコントロールは赤く、無効なコントロールは灰色で表示されます。

コントロールを他のコントロールより前面に表示させるには

GUI 形式フォームを作成する場合、あるコントロールが別のものと同じ場所に配置されている場合、どのコントロールを全面に表示させるなどの問題が出てきます。これは、*Z オーダ*と呼ばれる機能を使用してプログラムで処理されます。これは、コントロールがフォーム上に表示される順番を参照する値です。下位の Z オーダを持つコントロールは、より高い Z オーダを持つコントロールの背後に表示されることになります。

Magic は、デフォルトで自動的にコントロールの Z オーダを設定します。何らかの理由で Z オーダを変える必要がある場合は、手動で設定する必要があります。

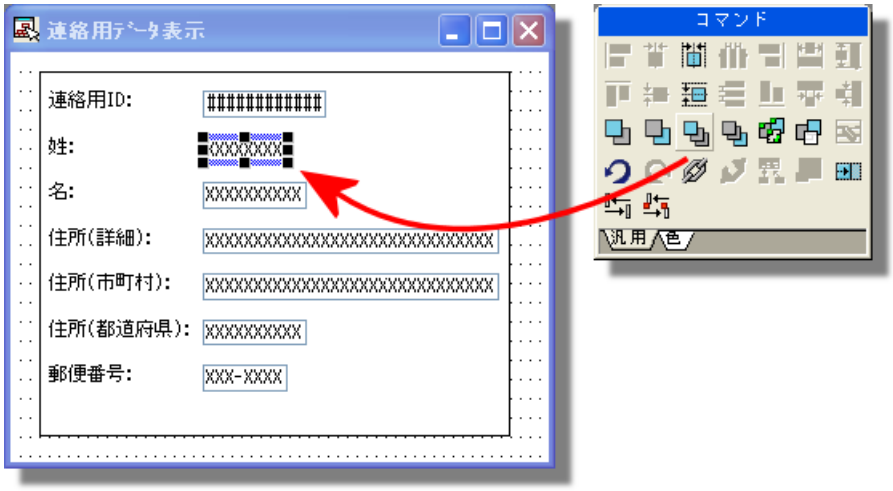
Z オーダを表示する



現在の Z オーダの値を確認するには、**コマンド**パレット（**描画→順序→ Z オーダ表示**）上の **Z オーダ表示**のアイコンを押下します。この例では、複数のコントロールが**グループ**コントロールの背後に配置されています。**グループ**コントロールの **Z オーダ**は *17*、表示されていないコントロールは、*1 ～16* が表示されています。

Z オーダを手動にする

1. Z オーダを確定するために、最初に、**自動 Z オーダ機能**を解除する必要があります。**コマンド**パレットの上の  アイコンを**クリック**するか、**描画→順序→自動 Z オーダ**を選択します。

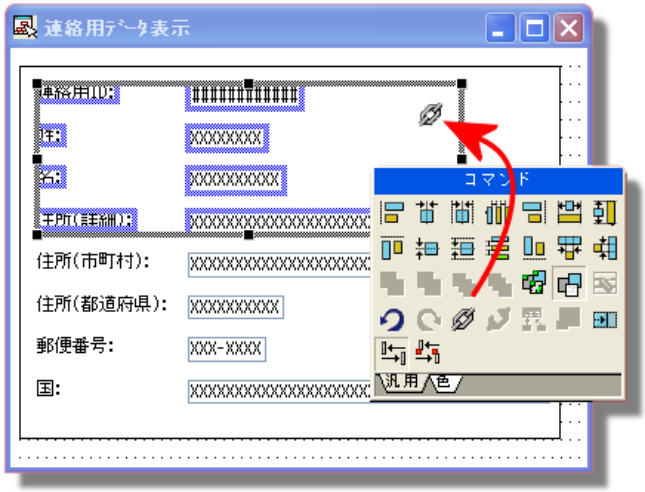


2. これにより、コントロールを選択する場合に、**コマンド**パレット上で利用可能ないくつかのオプションが有効になります。コントロールの表示を制御するために、**前面に移動**や**背面に移動**などが利用できます。

***ヒント :** 表示されていないコントロールを選択するには、Tab を押下し続ける必要があります。表示はされなくても、選択ボックスが表示されることで確認できます。また、**コントロールツリー**（**描画→コントロールツリー**）を表示させることでフォーム上のコントロールを簡単に選択することができます。*

コントロールをリンクする

あるコントロール（例えば、グループコントロール上のテキスト）の一部として別のコントロールを表示させたい場合、一方が他方のコントロールに（親子）リンクさせることで実現できます。リンクした場合、リンクされたコントロールは常に表示されZオーダの内容にかかわらず背後には隠れません。コントロールを他のコントロールにリンクさせるには以下のようにします。



1. *Ctrl* を押下した状態でコントロールをクリックし、リンクするコントロールを（複数）選択します。
2. 親子リンクの アイコンをクリックします。
3. 背景用のコントロール（この例ではグループコントロール）をクリックします。

これで、コントロールはグループコントロールの子コントロールとしてリンクされます。

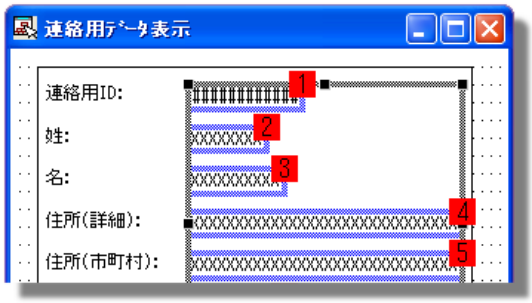
タスクのメインフォームに移動するには

GUI プログラミングでは、実行時に表示されるものだけでもタスク内にたくさんのメインフォームを使用することになります。フォームエディタを開き、該当するメインフォームをスクロールし、*F5* を押下して開くことでメインフォームにアクセスすることができます。

しかし、メインフォームを開く簡単な方法として *Ctrl+M*（オプション→メインフォーム編集）を押下する方法もあります。これによって、タスクのメインフォームを開き、編集を行うことができます。

ヒント : *メインフォームを閉じ、タスクに戻るには、F8（描画→描画の終了）というショートカットが用意されています。*

すべてのコントロールの TAB 順序表示するには



コマンドパレットの上の アイコンをクリックしたり、描画→表示順→ TAB 順序表示を選択することで TAB 順序を表示することができます。

フォームエディタの修正内容を取り消すには

Magic を使用することで迅速にフォームの編集作業を行うことができますが、時として安易にミスを犯してしまうこともあります。このような場合に備えて、フォームエディタには複数レベルで修正内容を取り消す機能があります。

最新の修正内容を取り消すには、*Ctrl+Z*（編集→変更取消）を押下します。ツールバー上またはコマンドパレット上の アイコンをクリックすることで取り消すこともできます。

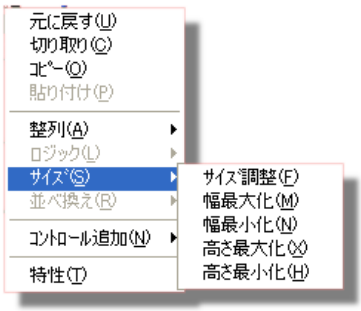
タスク

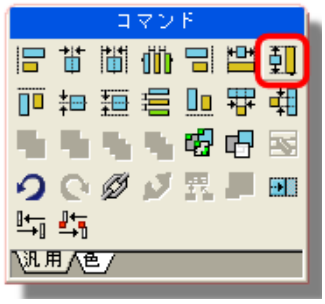
複数のコントロールの幅や高さを同じにするには

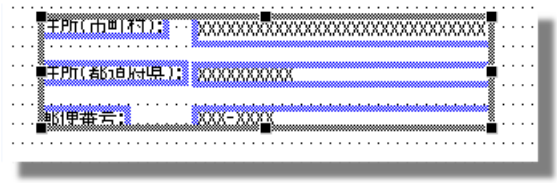
コントロールをフォーム上に整列させるため、高さや幅を同じにする必要があります。

例えば、入力項目に対する項目名表示のテキストが項目と異なる高さになっている場合、これらは上辺位置や下辺位置が同じでも関連性があるものとは認識されない可能性があります。

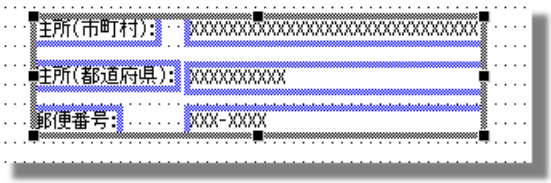
コントロールを同じサイズに調整する簡単な方法は、最大化 / 最小化のコマンドを使用することです。これらはプルダウンメニューや、コマンドパレットで利用することができます。







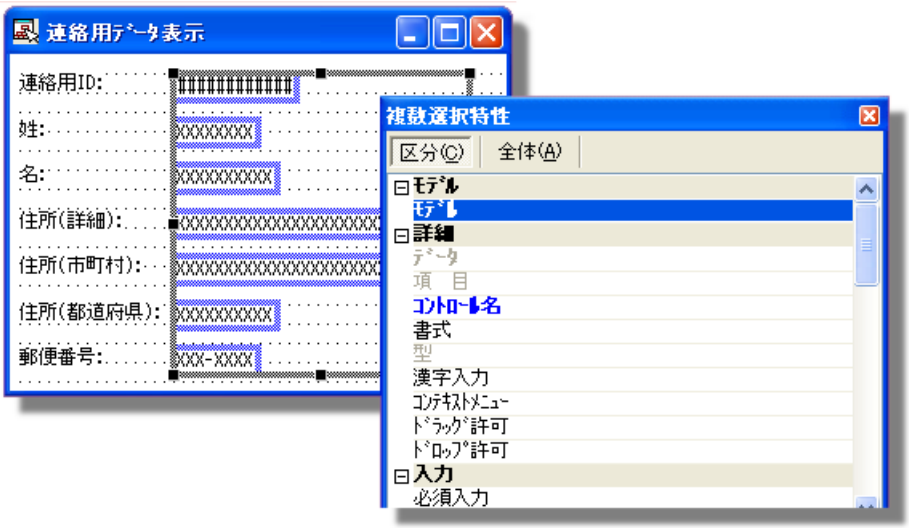
前



後

複数のコントロールの特性を同時に設定するには

通常、同じフォーム上のコントロールは同じように表示させる必要があります。この場合、コントロールには同じ特性を設定する必要があります。コントロールをクリックし、特性シートで特性を変更することで個別に特性を設定することができます。しかし、同時に複数のコントロール特性を変更することで開発工数を減らすことができます。ここではその方法について説明します。



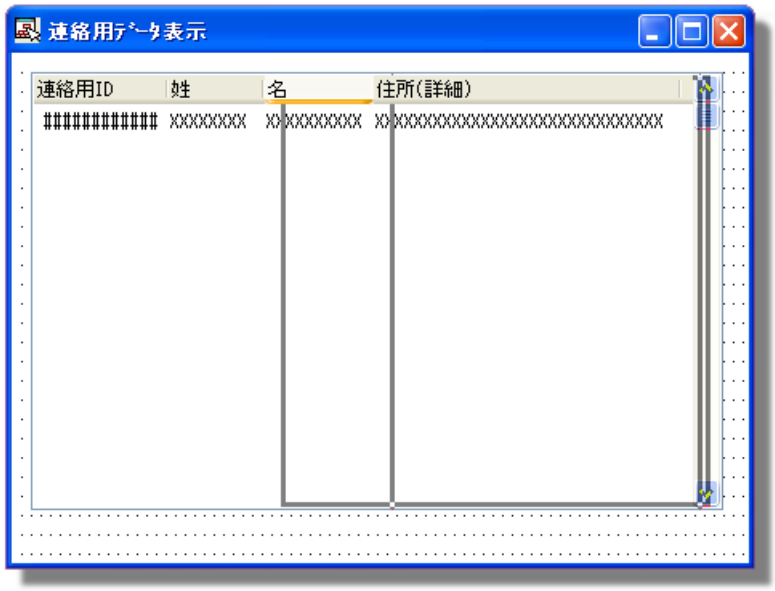
複数のコントロールの特性を変更する

1. 変更したいコントロールを選択します。ラバーバンドや Ctrl+ クリックを使用することで選択できます。
2. 特性シート（*Alt+Enter*）に移動し、特性値を変更します。

複数のコントロールを選択すると、特性シートの表示が変わります。ペインのヘッダには、複数選択特性と表示され、選択したすべてのコントロールで共通の特性のみ設定可能に。異なる種類のコントロールを選択した場合（例：スタティックとエディットなど）、共通する特性が少なくなります。

ヒント：*これは特にモデルをコントロールに定義していて、独自の特性値に設定していないような場合に有効です。同じコントロールであれば、同じモデルを定義することで簡単に特性値を反映させるこができます。*

テーブルコントロールの幅を変更するには



通常、**テーブル**コントロールのカラム上で**ドラッグ**すると**カラム区切り線**が移動しますが、カラム上のコントロールは動きません。カラムの幅を広げる必要があり他のカラムを右側に移動させたい場合は、以下のようにします。

1. テーブルのヘッダ領域上の**カラム区切り線**にカーソルを置きます。カーソルは両側に矢印が表示される形状になります。
2. *Ctrl* を押下しながらカラムを右側に**ドラッグ**します。上図のような灰色のバーが各カラムに表示され、カラムが移動することを示します。

操作を実行すると、カラム上のコントロールも右側に移動されます。

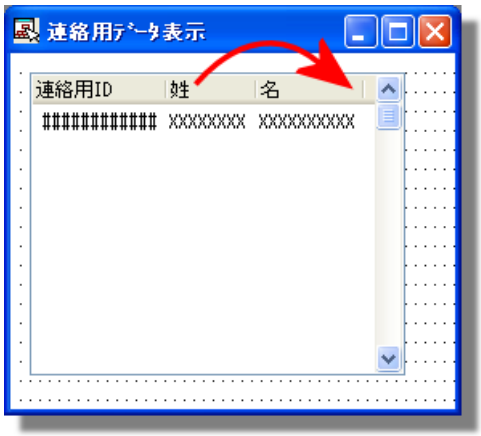
テーブルコントロールのカラムを移動するには

テーブルコントロールのカラムを別の位置に移動する必要がある場合、簡単に行うことができます。

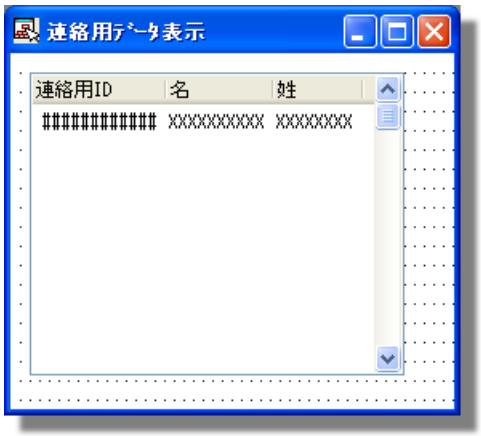
必要条件： テーブルコントロールのスタイル特性が *W=Windows* の場合のみ有効です。

テーブルコントロールのカラムを移動する

1. 移動したいカラムのヘッダ上にカーソルを置きます。
2. マウスボタンを置き、ヘッダを移動したい場所にドラッグします。カラムが表示される場所に若干黒い行が表示されます。
3. マウスボタンを離します。



変更前



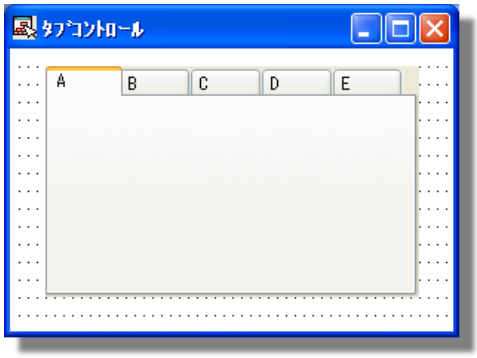
変更後

タブコントロールの編集中にタブの切り替えを行うには

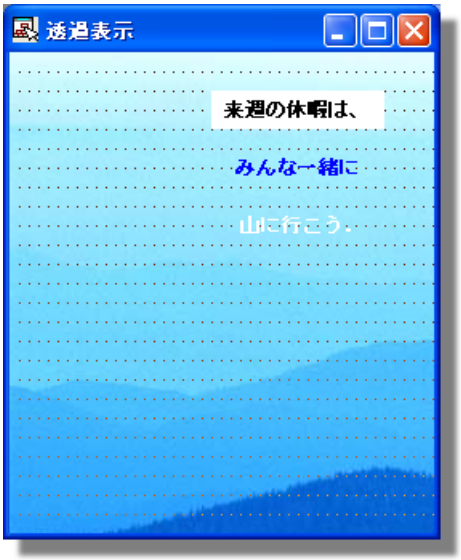
タブコントロールを編集中は、コントロールをクリックするとリンクされたコントロールを含むタブコントロール全体が選択されます。

１つのタブを選択し、そこにどのようなコントロールがリンクされているか確認したい場合は以下の操作を行う必要があります。

1. *Shift* を押下しながら、表示させたいタブコントロールのヘッダをクリックします。この例では、C というヘッダをクリックすると C のタブが開きます。
2. 他の方法として、タブコントロール全体を選択し、*Enter* を押下してます。*Enter* を押下するたびにタブの表示が切り替わります。



コントロールを透過表示するには



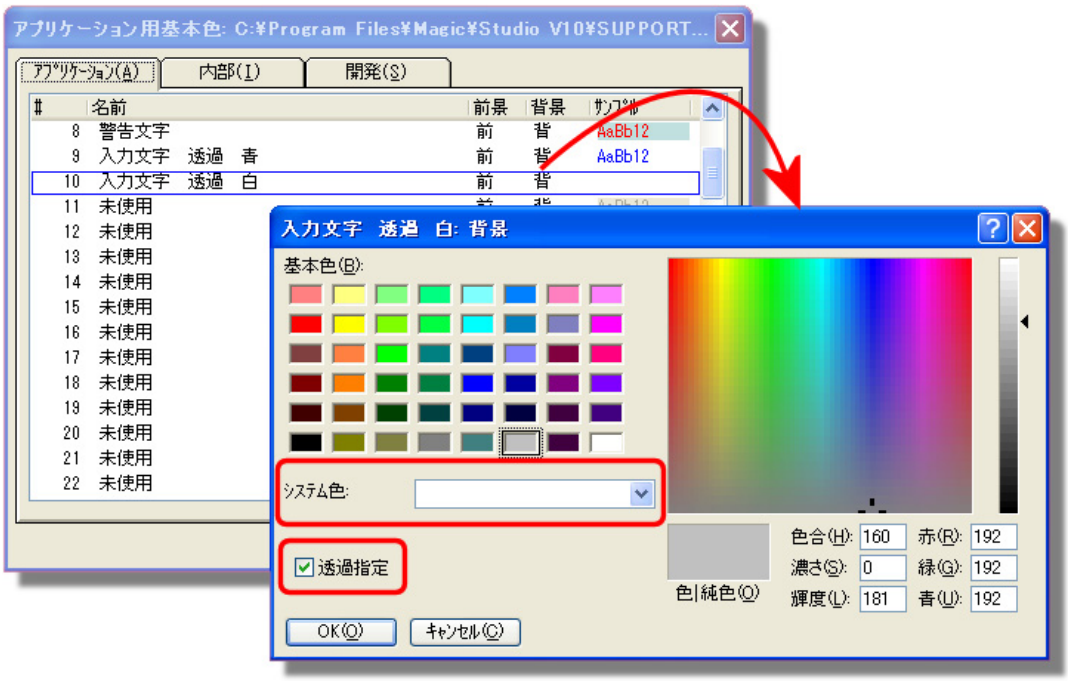
通常、コントロールの背景色は不透明か、Windows から継承された立体色です。例のように背景色を透過にすることで背景の画像を表示させたい場合があります。

最初のテキストコントロール「来週の休暇は」は、白い背景色を持つ平面のテキストコントロールです。次の２つのコントロールは両方とも透過の背景色になっていて、前景色が異なっています。

このような表示にする方法は、基本的に、透過の背景色を持つ色をコントロールに定義することです。以下にどのように行うかを説明します。

背景色を設定する

1. オプション→設定→基本色を選択します。
2. 変更したい色をクリックするか、*F4* を押下して色を追加します。
3. 背景色カラムでズームします。



4. システム色を*空白*に設定します。ドロップダウンリストの先頭で空白行を選択することで設定できます。.
5. 透過のチェックボックスをチェックします。
6. OK をクリックします。
7. 基本色テーブルに戻り、（透過設定を除いた）同じ方法で前景色も設定できます。ここでテキストの表示色を選択します（通常は、黒か白を使用します）。
8. OK をクリックして色指定のダイアログボックスを終了します。

これで平面のコントロールでこの色を定義すると、背景色は透過になります。

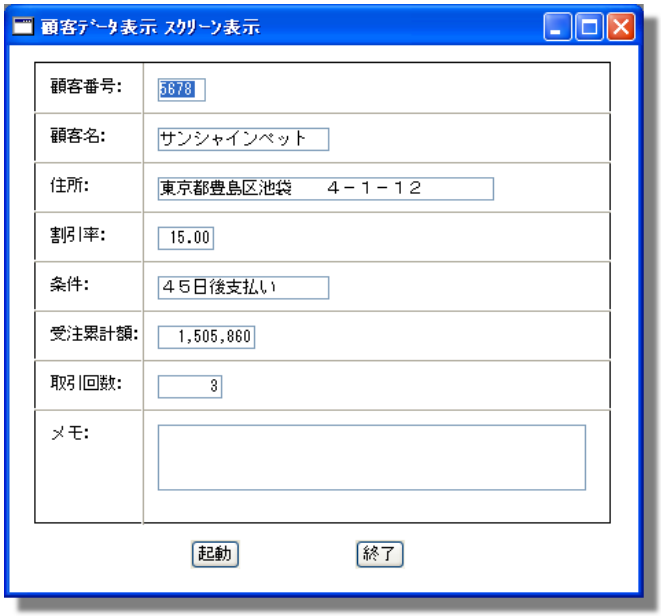
フォームにデフォルトのプッシュボタンを設定するには

フォーム上に複数のプッシュボタンが定義されている場合、ユーザが *Enter* を押下するときに押下されたように動作するデフォルトのプッシュボタンを割り当てる必要が考えられます。

例えば、このフォーム上には、**終了ボタン**がデフォルトとして定義されています。

ウィンドウが開いた時点では、カーソルは顧客番号上にありますが、*Enter* を押下するとタスクは終了します。

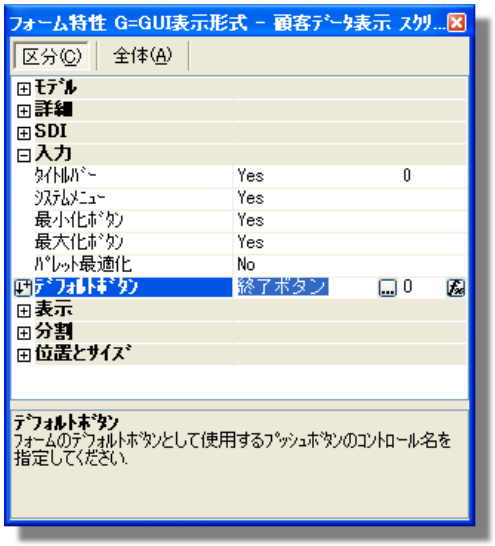
ここでは、どのようにして設定するかを説明します。



デフォルトのプッシュボタンを設定する

1. **プッシュボタン**コントロールを配置し、コントロール名を設定します。
2. **フォーム特性**（コントロールが選択されていなければ *Alt+Enter*）を開きます。
3. 入力セクションの**デフォルトボタン**特性にカーソルを移動します。
4. ここから**ズーム**してプッシュボタンの**コントロール名**を選択します。
5. または、右側の**式**特性で**ズーム**して、**式エディタ**に実行時にコントロール名と表示される式を定義します。

これで、プログラムを実行すると、選択されたボタンはデフォルトボタンになります。

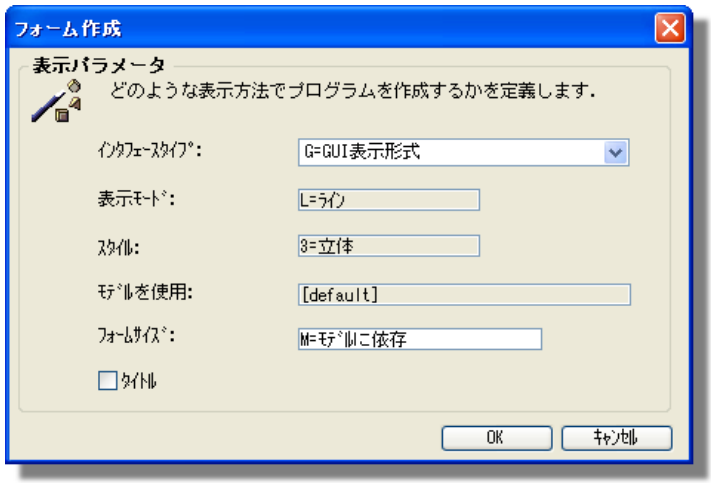


デフォルトのフォームレイアウトを自動的に作成するには

タスクのデータビューをすべてフォームに配置するような場合、簡単な方法があります。これは*フォームジェネレータ*と呼ばれ、データビューとして定義された項目をすべてメインフォームに配置することができます。

フォームジェネレータを使用する

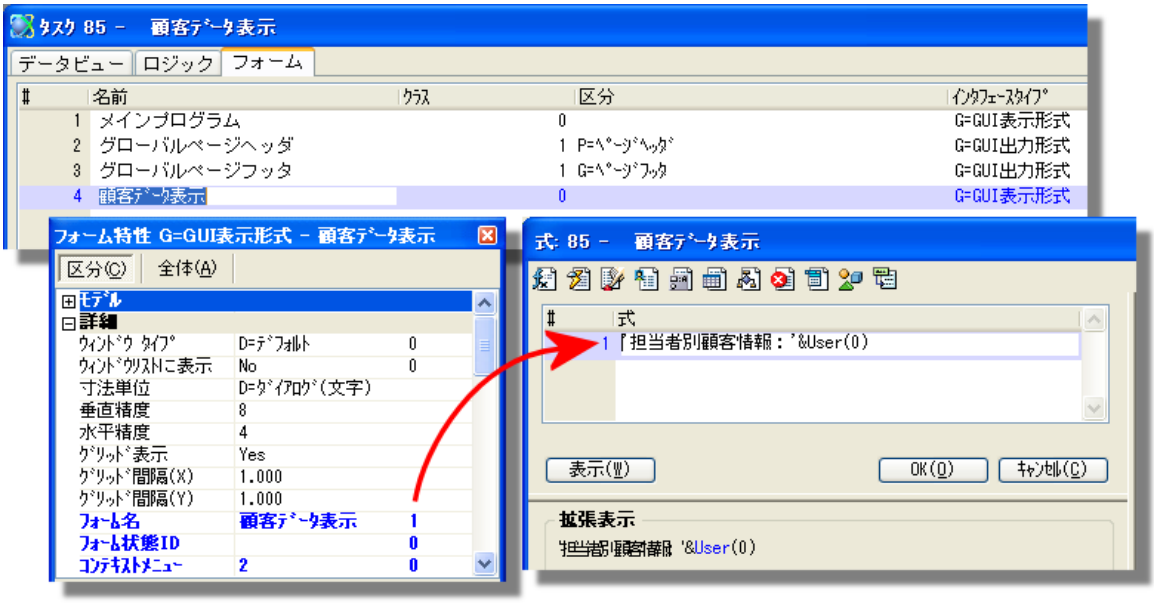
1. フォームを作成するタスクを開きます。
2. フォームタブをクリックします。
3. *Ctrl+G*（オプション→フォーム作成）を押下します。
4. 上書き確認のダイアログボックスが表示されます。はいをクリックします。.
5. フォーム作成ダイアログボックスが表示されます。必要に応じて、オプションを設定します。
   - 表示モード: データをテーブルコントロールで表示する場合はラインを選択します。これ以外はスクリーンを選択します。
   - スタイル：立体または平面を選択します。
   - モデルを使用：フォームモデルを使用する場合はここでモデルを選択します。
   - フォームサイズ：モデルに依存、表示内容に依存、MDI 内に納めるの３つのオプションがあります。
6. OK をクリックします。作成されたフォームにはデータビューがすべて配置されます。



ヒント :Ctrl+G の押下は、フォームエディタ上で行ってください。データビューエディタやロジックエディタで行った場合、APG が実行されプログラムが上書きされてしまいます。

ウィンドウタイトルを動的に表示させるには

一般にタイトルバーを持つウィンドウは、タイトル表示用のテキストを指定します。デフォルトでは、フォーム名がタイトルとして表示されます。しかし、式を使用することで実行時に動的にタイトルを変更することもできます。



フォームタイトルに式を使用する

1. フォームエディタを開きます。
2. フォーム特性（*Alt+Enter*）を開きます。
3. フォーム名特性に移動します。
4. 式特性でズーム（または、***fx*** ボタンをクリック）します。
5. 実行時にタイトルとして評価される式を定義します。この例では、担当者毎のユーザ ID が表示されることになります。

これで、実行時に式で指定された内容がタイトルバーに表示されます。修正前の（固定のフォーム名も残ります。これは開発時のみ表示されます。

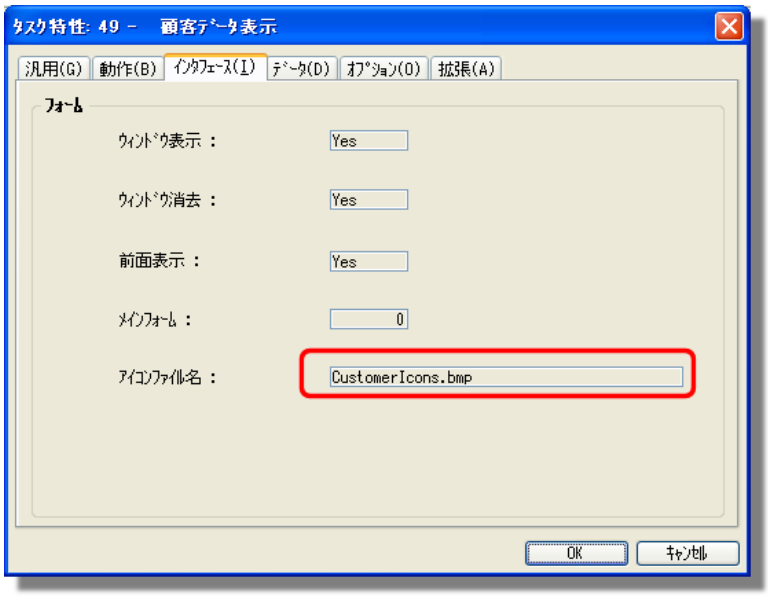
ヒント : *タイトルが長すぎてフォーム名として入力できない場合も、この方法で指定できます。*

フォームのアイコンを設定するには

必要であれば、特定のフォームに表示するアイコンを設定することができます。このアイコンは、フォームの左上に表示され、フォームが最小化された場合もタスクバーに表示されます。この設定により、プロジェクトレベルで設定されたアイコンは上書きされます。

フォームのアイコンを選択する

1. **タスク特性**（*Ctrl+P*）を開きます。
2. **インタフェース**タブを選択します。
3. **アイコンファイル**特性で**ズーム**して、設定したいアイコンファイルを選択します。



ヒント : *アプリケーションがインストールされた場合、ユーザは開発時のセットアップ環境と同じとは限りません。このため、パス名を固定にしないように定義する必要があります。パス名が指定されていない場合、Magic は .edp ファイルが存在する作業フォルダ（%WorkingDir%）を検索します。アイコンファイル名には、相対パスや論理名を使用することもできます。*

タスク

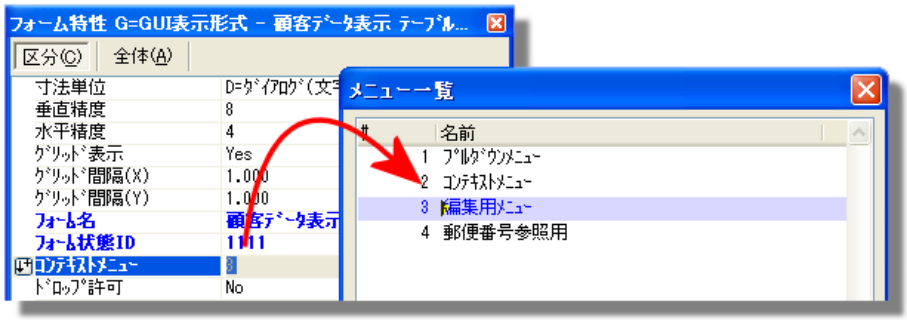
フォームのコントロールにデフォルトのコンテキストメニューを設定するには

メニューリポジトリでは、アプリケーション全体のコンテキストメニューを作成することができます。また、これをオーバライドし、各フォーム毎のデフォルトメニューを作成するもできます。

フォームにデフォルトのコンテキストメニューを設定する

必要条件： メニューがすでに作成されているものとします。

1. フォーム特性（コントロールが選択されていない場合は、*Alt+Enter*）を開きます。

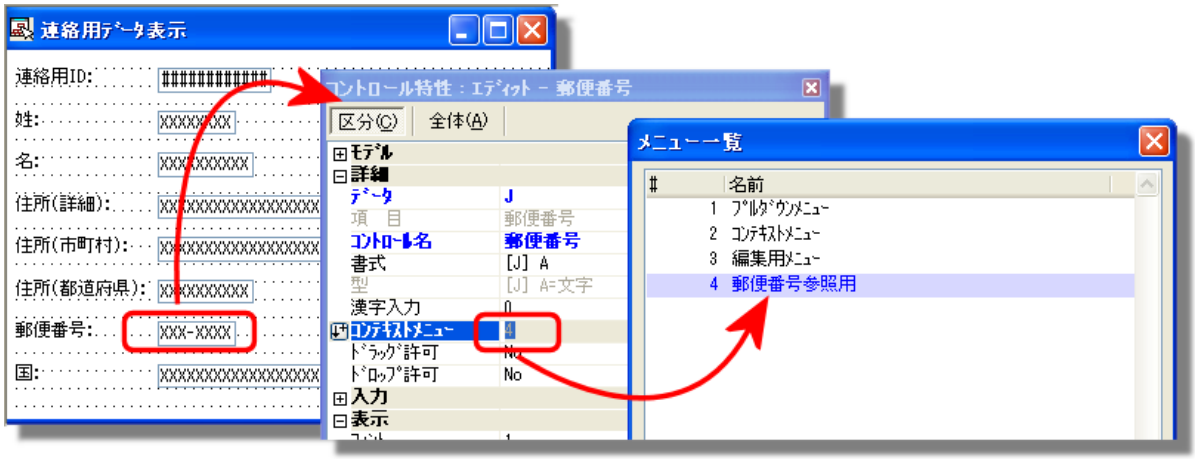


2. コンテキストメニュー特性でズームしてメニューを選択します。
3. 代わりに、右側の式特性でズーム（または *fx* ボタンのクリック）して実行時に式番号として評価される式を定義することもできます。

これで、ユーザがフォーム上でマウスの右クリックを行うと、コンテキストメニューが表示されます。

個々のコントロールにコンテキストメニューを設定するには

メニューリポジトリでは、アプリケーション全体のコンテキストメニューを作成することができます。また、これを上書きし、各フォーム毎のデフォルトメニューを作成するもできます。さらにコントロールレベルのコンテキストメニューを作成することもできます。この例では、郵便番号を検索するための特別なコンテキストメニューを作成していきます。



コントロールレベルのコンテキストメニューを作成する

必要条件：　メニューがすでに作成されているものとします。

1. コントロール特性（コントロールが選択されている場合は、*Alt+Enter*）を開きます。
2. コンテキストメニュー特性でズームしてメニューを選択します。
3. 代わりに、右側の式特性でズーム（または *fx* ボタンのクリック）して実行時に式番号として評価される式を定義することもできます。

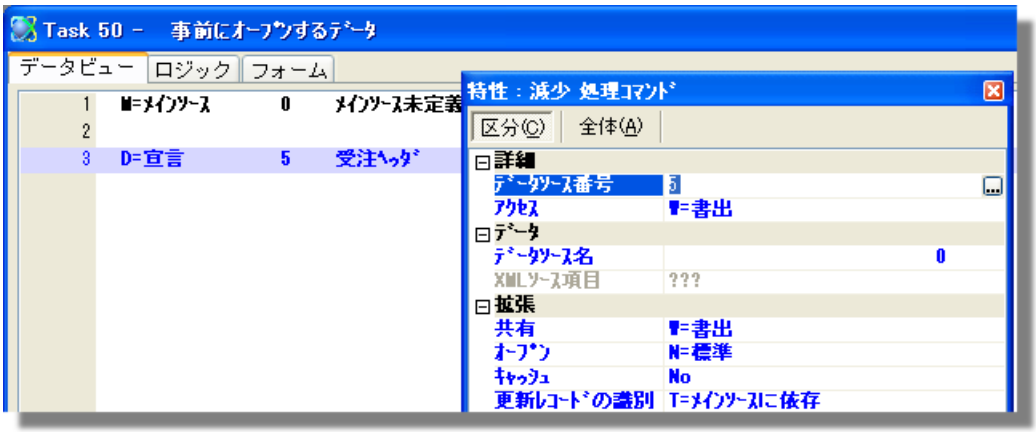
これで、ユーザが郵便番号の入力欄でマウスの右クリックを行うと、コンテキストメニューが表示されます。

タスクがバッチタスクから繰り返し呼ばれる場合のパフォーマンスを改善するには

あるバッチタスクが繰り返し別のバッチタスクによって呼ばれる場合、パフォーマンスが悪くなる場合があります。これは、サブタスクが使用するデータソースを開いたり閉じたりするために必要なオーバヘッドによるものと考えられます。このような場合、以下の方法によってパフォーマンスを改善することができるかもしれません。

バッチのサブタスクのパフォーマンスを改善する

1. 最初に行う最も重要なステップは、サブタスクが使用するデータソースを親タスクであらかじめオープンされているように定義することです。異なるファイルが複数使用する場合、オープンするためのオーバヘッドはファイル数分増えることになります。レコードを読み込むためのオーバヘッドよりファイルをオープンするための方が一般的に大きくなります。

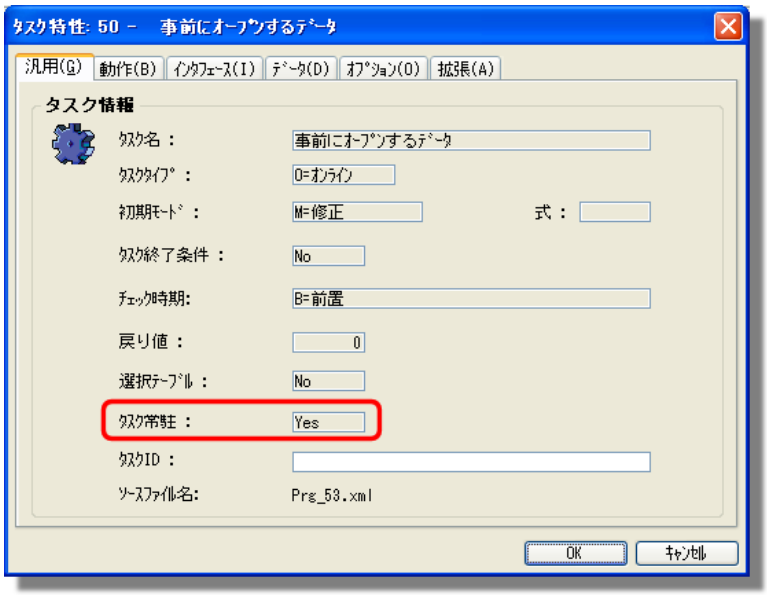


親タスクでデータソースが使用されない場合、データビューエディタで宣言のヘッダ行を作成することで、事前にこのデータソースをオープンしておくことができます。宣言指定のアクセス特性は、サブタスクがデータソースをオープンする場合に使用されるアクセス特性の内容と合っていなければなりません。

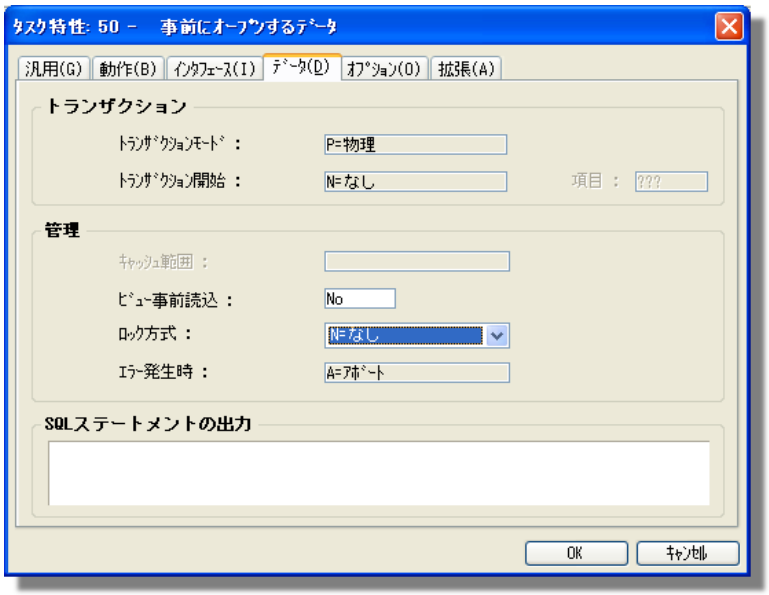
2. 次に、バッチタスクを常駐タスクとして設定します。この設定により起動されるたびに、バッチタスクが再読込されることを防ぎます。

   サブタスクを常駐タスクとして定義するには以下のようにします。

   - タスク特性（*Ctrl+P*）を開きます。
   - 汎用タブを開きます。
   - 常駐タスク特性を *Yes* に設定します。



注： 常駐タスクの使用は現在推奨しておりません。使用される場合も、あまり多用しないようにお願いします。



3. 最後に、親タスクのトランザクションを**遅延モード**にしないように設定します。トランザクションモードが遅延に設定されていると、すべてのサブタスクトランザクションは蓄積され、パフォーマンスを低下させる要因になります。トランザクションモードを確認するには、以下のようにします。

- 親タスクに移動します。
- **タスク特性**（*Ctrl+P*）を開きます。
- **データ**タブを開きます。
- **トランザクションモード**特性を、タスクとして必要な値に設定します。この例では、トランザクション処理を行わない設定になっています。大規模なバッチタスクのほとんどは帳票用のためデータの更新は行われないのでトランザクション処理を行う必要はありません。

第 6 章： 拡張されたロジック

電子メールを送信するには

Magic には電子メールを送受信するための機能が組み込まれています。

電子メールを送信するには、以下の手順でプログラムを作成します。

1. サーバと接続します。
2. 電子メールを送信します。

Magic ではこれらを個別に処理します。

サーバと接続する

MailConnect() 関数を使用してサーバと接続します。この場合、あらかじめメールサーバの名前を知っておく必要があります。それが何なのかわからない場合は、電子メール用のソフトウェアの設定オプションで確認してください。構文は、以下の通りです。

MailConnect(*type*, *server*, *user*, *password***)**

パラメータ：

- *Type* ：接続するサーバのタイプ（1=SMTP, 2=POP3, 3=IMAP）
- *Server* ：サーバのアドレス
- *Userid* ：サーバに接続する際に必要なユーザ ID
- *Password* ：ユーザのパスワード

例えば、juno.olympus.com というアドレスの SMTP サーバに接続する場合は以下のように設定します。

```
MailConnect(1,'juno.olympus.com','','')
```

戻り値は、接続しているサーバの種類に依存します。関数の詳細は『リファレンスヘルプ』を参照してください。

電子メールを送信する

サーバに接続したら、**MailSend()** 関数を使用することで電子メールを送信することができます。構文は以下の通りです。

MailSend(*From*, *To*, *Cc*, *Bcc*, *Subject*, *Message*, *Attachment***)**

パラメータ：

- *From* ：送信元のアドレス。内容はチェックされません
- *To* ：宛先のアドレス。正しくない場合メールは送信されません。
- *Cc* ：CC へのアドレスのリスト
- *Bcc* ：BCC へのアドレスのリスト
- *Subject* ：メールの件名を表すテキスト
- *Message* ：メールの本文となるテキスト
- *Attachment* ：添付ファイルのファイル名

以下は設定例です。

```
MailSend ('jenny@frigates.com', 'fred@aaawidgets.com', '', '', 'May Invoice', 'Attached
is your May invoice', 'F:¥Invoices¥aaa_11_2007.pdf')
```

通常は、ハードコーディングされた文字列ではなくデータ項目を使用することになります。アドレスは、カンマ区切りにすることで複数指定することができます。また添付ファイルも複数指定することができます。

戻り値は数値で、**0** が成功を意味しています。戻り値が **0** でない場合、**MailError()** 関数を使用してエラーメッセージに変換することができます。戻り値を 'BP' という項目に格納する場合は、以下のような式が定義できます。

```
' エラーコードは：'&Str (BP,'5NC')&' '&MailError (BP)
```

この式によってエラーコードとメッセージの両方を表示させることができます。

電子メールにファイルを添付するには

Magic の関数で電子メールを送信する場合、**MailSend()** 関数を使用します。構文は以下の通りです。

MailSend(*From*, *To*, *Cc*, *Bcc*, *Subject*, *Message*, *Attachment***)**

添付ファイルを送信する場合は、**MailSend()** 関数の *Attachment* パラメータを指定する必要があります。カンマ区切りでファイル名を連結することで複数のファイルを送信することができます。

以下は設定例です。

```
MailSend ('jenny@frigates.com', 'fred@aaawidgets.com', '', '', 'May Invoice', 'Attached
is your May invoice', 'F:¥Invoices¥aaa_11_2007.pdf')
```

この例では、F:¥Invoices¥aaa_11_2007.pdf が添付ファイルになります。

参照： 「電子メールを送信するには」（135 ページ）

電子メールを受信するには

電子メールを受信する場合、以下の手順で行います。

1. メールを受信するには、POP3 または IMAP サーバに接続（送信する場合は、SMTP サーバでした）する必要があります。この接続は、通常ユーザ ID とパスワードが必要となるため、以下の例のような式を定義します。

   ```
   MailConnect (3,'juno.olympus.com','FredZ','rabbit16')
   ```

   この例で設定されている *3* は、IMAP サーバに接続することを示すサーバタイプです。その他の３つのパラメータは各々サーバ名、ユーザ ID、およびパスワードです。
2. 戻り値が正の数値なら、キュー内の電子メールの数を示します。この値を保存しておきます。
3. 次に、メッセージを取得する処理を実行する必要があります。メッセージ数と同じ数だけのインデックスを使用してメッセージを取得します。各メッセージ毎に、電子メールの様々な情報を取得する関数を使用することができます。例えば、Y をインデックスとします。この場合、以下の関数を実行できます。

   ```
   MailMsgDate(Y)
   MailMsgFrom(Y)
   MailMsgSubj(Y)
   MailMsgText(Y)
   MailMsgFiles(Y)
   ```

   これらの関数によって取得された情報をすべて保存します。

   メッセージを読み込んだら、以下の関数で削除できます。

   ```
   MailMsgDel(Y)
   ```

   メッセージが削除できない場合、メッセージはメールボックス内に存在していることになります（メールボックスに保存することは、テスト中であれば良い考えです)。
4. 終了する場合は、以下の関数でサーバ接続を切断します。

   ```
   MailDisconnect(2, 'TRUE'LOG)
   ```

   これでセッションが切断され、メールがサーバから削除されます。この例の *2* はサーバタイプで、受信サーバからの切断を意味しています。サーバと切断されメールが削除されると、論理値 'TRUE'LOG が返ります。

サーバのタイプ

メールを受信する場合、２種類のサーバタイプがあります。POP3 サーバに接続する場合は、新規メールのみ受信します。IMAP サーバに接続した場合は、サーバに存在するメールをすべて受信します。

参照： これらの関数の詳細情報は、『リファレンスヘルプ』を参照してください。

添付ファイルを受信するには

電子メールに添付されているファイルを取得するには、以下の手順でロジックを定義します。

1. 最初に **MailMsgFiles** 関数を実行して添付付されているファイルの数を取得する必要があります。構文は以下の通りです。
   MailMsgFiles(*Y*)
   パラメータ：
   - ***Y*** ：メールのインデックス

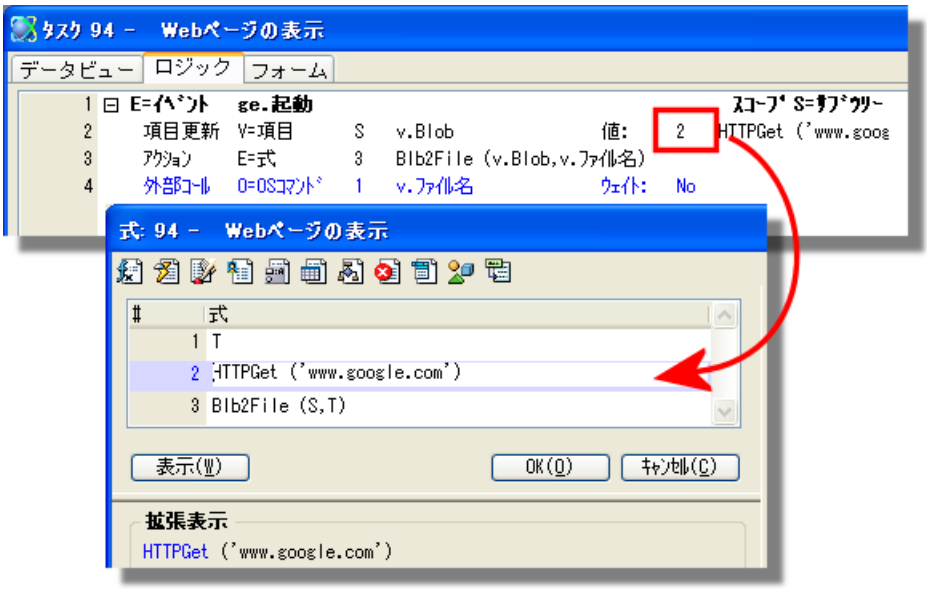
   この関数は、メールに添付されているファイルの数を返します。
2. 次に、各添付ファイル毎に **MailMsgFile** 関数を実行してファイル名を取得します。構文は以下の通りです。
   MailMsgFile(*Y*, *Z*)
   パラメータ：
   - ***Y***：メールのインデックス
   - ***Z***：添付ファイルのインデックス

   この関数は、添付ファイルのファイル名が返ります。
3. 最後に、各添付ファイル毎に **MailFileSave** 関数を実行してファイルを取得します。構文は以下の通りです。
   MailFileSave(*Y*, *Z*, *FileName*, *Flag*)
   パラメータ：
   - ***Y***：メールのインデックス
   - ***Z***：添付ファイルのインデックス
   - ***FileName***：保存する添付ファイル名
   - ***Flag***：ファイルを上書きするかどうかを指定するフラグ

   この関数は、ファイルの保存が成功したかどうかを示す数値が返ります。

Web ページやその他の URL コンテンツを受信するには

HTTPGet() 関数を使用することで URL を指定して、Web ページやその他の URL コンテンツを取得することができます。



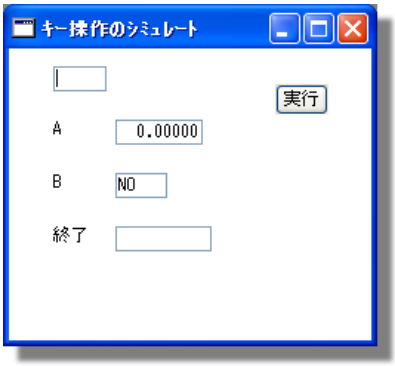
HTTPGet() 関数を使用する

1. **HTTPGet()** 関数を使用して Blob 項目を更新します。構文は以下のとおりです。
   HTTPGet(*URL*, *Arg1*, *Arg2*, . . .)
   パラメータ：
   - *URL*：Web サイトの URL
   - *ArgX*：追加で指定するヘッダ情報
2. **Blb2file()** 関数を使用することで Blob 項目の内容をファイルに返還できます。
3. BLOB の内容を拡張子 **.htm** のファイルに保存すると、Web ブラウザで参照することができます。

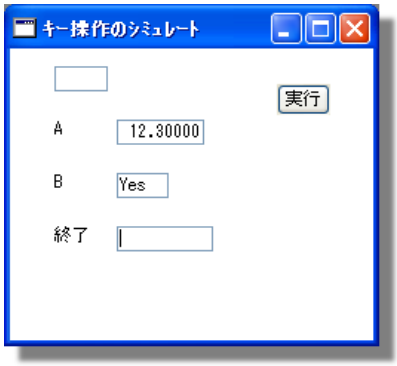
参照： **HTTPGet()** 関数の詳細情報は、『リファレンスヘルプ』を参照してください。

キー操作をシミュレートするには

Magic でキー操作をシミュレートするには２つの方法があります。１つは、**KBPut()** 関数を使用する方法で、もう１つはイベント実行処理コマンドを使用する方法です。**KBPut()** は汎用的な関数で、文字を入力したり、長いマクロ文を作成することができます。他方、イベント実行処理コマンドは、色々な種類のイベントを発行したりするなど、より柔軟性を提供します。ここで、**KBPut()** 関数について説明します。

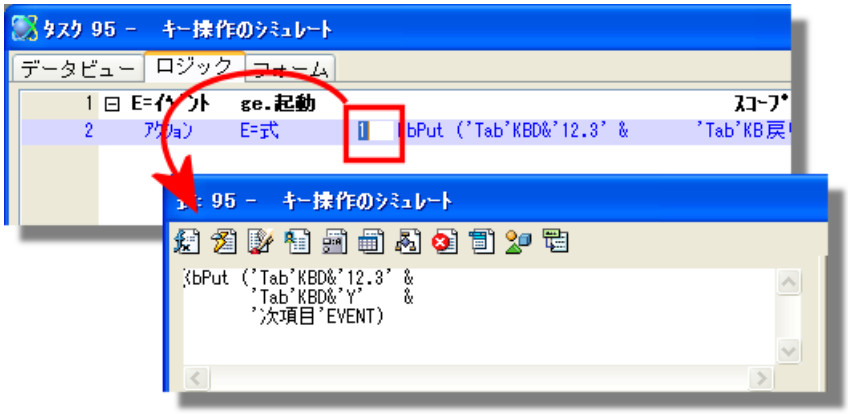


前



後

KbPut() 関数を使用する



KbPut() 関数の構文は以下のとおりです。

KbPut(*String*)

パラメータ：

- *String*：以下に示す書式が指定された文字列です。連結演算子（&）は、文字列を結合する際に使用されます。

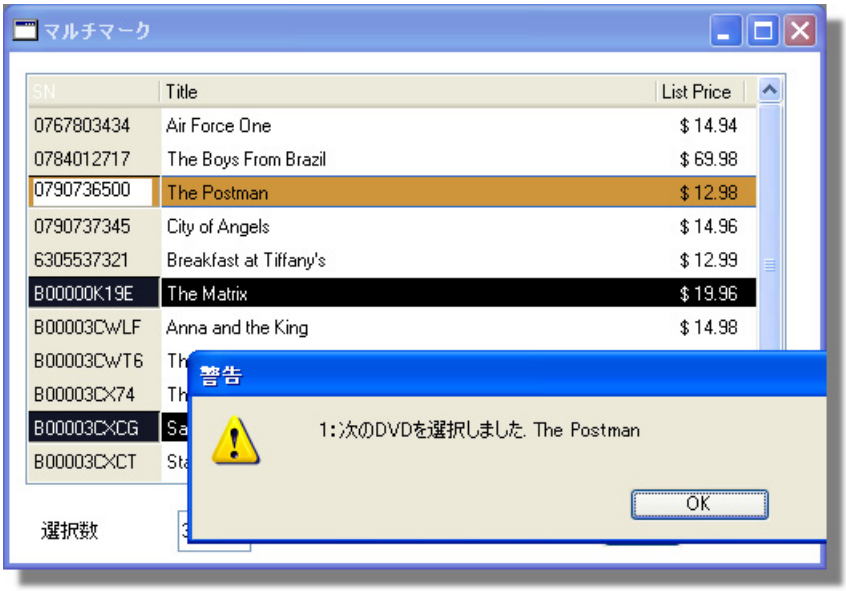
KbPut での文字内容	構文	例	簡単な入力方法
文字 , 数字	クォーテーションで囲まれたテキスト	'123.2'	
キー操作	**KBD** リテラルを使用したキー名	'*Tab*'KBD	コンテキストメニューからキーボードを選択し、キー設定ダイアログから指定します。
イベント	**EVENT** リテラルを使用したイベント名	' 次項目 'EVENT	コンテキストメニューからイベントを選択し、イベントテーブルから指定します。

この例では基本的には、'*Tab*'KBD と ' 次項目 'EVENT は同じ動作になります。キーボード割付テーブルで設定が変更される可能性があるため、EVENT による指定を使用したほうが安全です。

注： **KBPut()** 関数は、ActiveX コントロールや起動された Windows のダイアログには影響しません。

参照： 第 4 章：「Magic エンジンをイベント駆動型で動作させるには」（47 ページ）

複数のレコードをマークしたり、マークしたレコードを処理させるには



ユーザがリストから複数の項目を選択できるようにすることは便利な機能です。Magic にはこのような機能が ***マルチマーキング機能***として組み込まれています。

テーブルをマルチマーキング対応に設定する

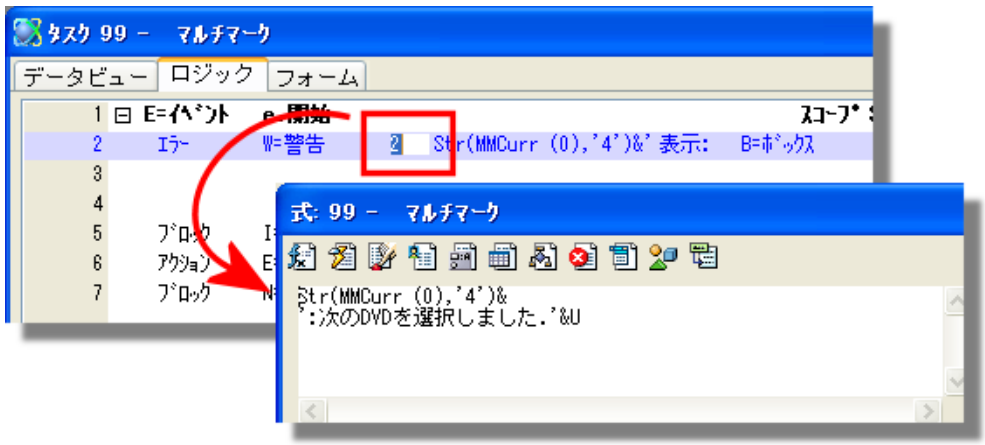
テーブルコントロールの**コントロール特性**で、マルチマークを利用可能になるように設定する必要があります。

1. **テーブルコントロール特性**で、**マルチマーキング**特性を *Yes* に設定します。
2. 少なくとも 1 つのカラムをマーキングカラムに設定します。通常、左側のカラムに設定しますが、すべてのカラムでマークを許可することもできます。マークカラムは、**マーキングカラム**特性を *Yes* に設定します。

   テーブルの**スタイル**特性が *D=Windows 立体*または *W=Windows* の場合、マークカラムは浮き出て表示されます。

   これで、ユーザはこのテーブルから複数のレコードを選択することができるようになります。

マークされたレコードを処理する



レコードが選択されたら、これらを処理する必要があります。このような場合、一般的に別のテーブルに書き込んでから処理することになりますが、この例では、ただ単にユーザにメッセージを表示するだけの処理を説明します。.

1. マークされたレコードを処理する時点で実行するイベントを作成します。この例では、**e. 開始**イベントを実行するプッシュボタンを使用します。

2. イベントに対応するロジックユニットでは、現在のレコードであるかのようにレコードを処理します。この例では、各レコード毎に３つのメッセージが用意され、そのうち１つを選択するようになっています。ブロック処理コマンドによるループはありませんが、これによってループしたような処理が実行されます。

マルチマーク関数を使用して処理する

1. イベント内で **MMStop()** 関数を使用することで、マークされたすべてのレコードが処理される前に処理を終了することができます。**MMStop()** 関数が実行された場合、エンジンは **MMStop()** が実行された時点で処理しているレコード上にパークします。
2. 以下の式を指定したブロック処理コマンドを使用することで、すべてのレコードが処理された後に、ロジックを実行させることができます。

```
MMCurr()=MMCount(0)
```

   MMCurr() 関数は、現在までに処理されたレコード数が返り、**MMCount(0)** 関数はレコードの総数が返ります。
   この例では、最後のレコードが処理された後にマーキングが解除されます。
3. **MMClear()** 関数は、レコードのマークを解除します。

メニューを非表示 / 無効 / チェック表示に設定するには

Magic では、エンドユーザ用のメニューを作成 / 制御することができます。プルダウンメニューやコンテキストメニューを作成することができ、フォームやコントロールにコンテキストメニューを設定することができます。また、実行時に有効にしたり無効にしたりすることができます。ここではこのような処理について説明します。

論理メニュー名を指定する

メニューを個別に有効化 / 無効化するには、ユニークな**論理メニュー名**を指定する必要があります。この設定は、**メニュー**リポジトリで行います。設定したいメニュー項目に対して、**論理メニュー名**を入力します。この名前は実行時は表示されないため、どのような名前も入力できます。この例では、２つの論理メニュー名、EditMenu と TestMenu が定義されています。

論理メニュー名を定義すると、**MnuCheck()** と **MnuEnable()** 関数を使用して有効化 / 無効化やチェック表示の切り替えを実行時に行うことができます。

メニューパス関数を使用する

メインメニューのサブメニューにも、**論理メニュー名**が設定されており、これらは、メニューパスを文字列で取得することができます。例えば、デフォルトのプルダウンメニューの下に、UtilityMenu と名付けられたメニューが定義されており、その下には SetupMenu という名前のメニューが定義されている場合は、パスは以下の通りになります。

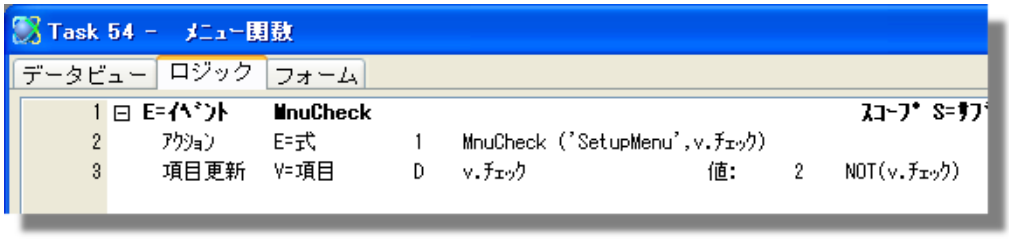
```
UtilityMenu¥SetupMenu
```

この値は、**MnuAdd()** 関数で使用することができます。

メニュー番号を指定する

メニュー名に加え、メニューは **MENU** リテラルによって参照することもできます。上記の例では、Edit menu は **'2'MENU**、Testing Menu は **'3'MENU** と指定できます。**MnuAdd()** 関数で **MENU** リテラルを使用します。

MnuCheck() 関数を使用する

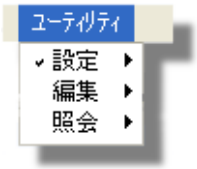


MnuCheck() 関数は、メニュのチェック表示の切り替えを行います。構文は以下の通りです。

MnuCheck(*EntryName*, *Boolean***)**

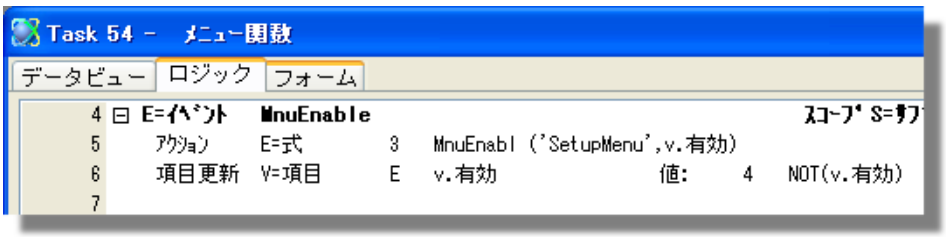
パラメータ：

- *EntryName* ：メニュー定義テーブルに定義されている論理メニュー名です。論理メニュー名は、実行時にメニューとして表示されないので任意の名前が定義できます。
- *Boolean*：True が指定された場合、メニューにチェックが表示されます。False の場合、チェックは表示されません。



この例では、式が実行されるとメニューの設定がチェックされます。

MnuEnabl() 関数を使用する



MnuEnabl() 関数は、メニューの有効 / 無効を切り替えます。構文は以下の通りです。

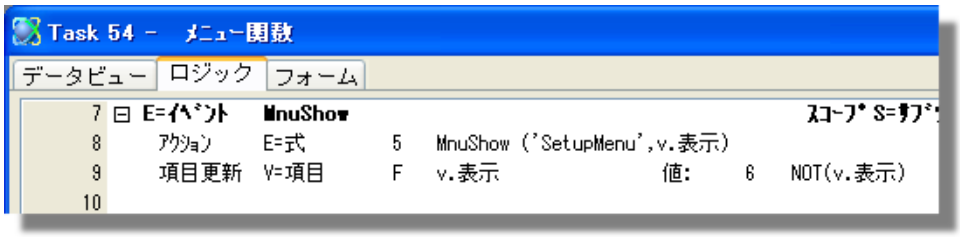
MnuEnabl(*EntryName*, *Boolean*)

パラメータ：

- *EntryName* ：メニュー定義テーブルに定義されている論理メニュー名です。論理メニュー名は、実行時にメニューとして表示されないので任意の名前が定義できます。
- *Boolean*：True が指定された場合、メニューは有効になります。False の場合、無効になります。

この例では、式が実行されるとメニューの設定が無効になります。

MnuShow() 関数を使用する



MnuShow() 関数は、メニューを表示するか否かを指定します。構文は以下の通りです。

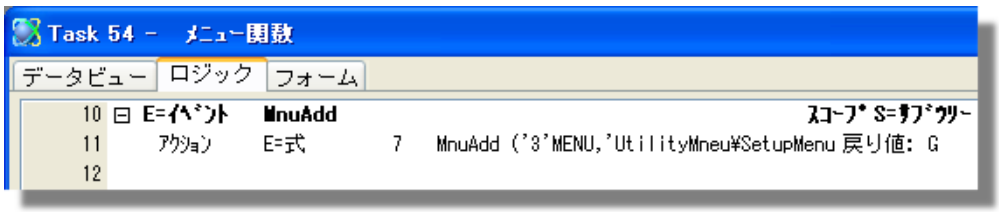
MnuShow(*EntryName*, *Boolean*)

パラメータ：

- *EntryName* ：メニュー定義テーブルに定義されている論理メニュー名です。論理メニュー名は、実行時にメニューとして表示されないので任意の名前が定義できます。
- *Boolean*：True が指定された場合、メニューは表示されます。False の場合、表示されません。

この例では、式が実行されるとメニューの設定は定義はされていますが、表示されていません。

MnuAdd() 関数を使用する



MnuAdd() 関数は、メニュー全体（サブメニューを含む）をメニューに追加することができます。メニューリポジトリにメニューを作成し、メニュー番号を **MENU** リテラルを付加して指定することで追加することができます。

さらに、新しいメニューがどこに位置するかを指定するメニューパスを使用します。構文は以下の通りです。

MnuAdd(*MenuEntry*, *MenuPath*)

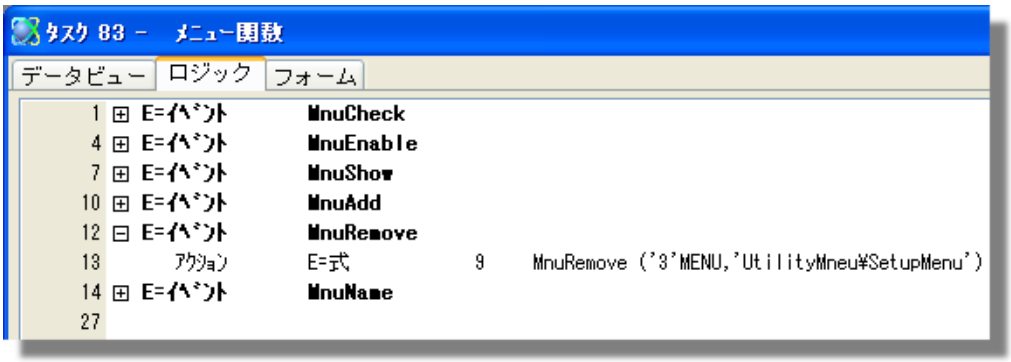
パラメータ：

- *MenuEntry* ：メニュー番号です。この例で指定されている **'3'MENU** は、メニューリポジトリ上の３番目のメニューを示しています。
- *MenuPath*：メニューの追加先のメニューパスです。この例では、'UtilityMenu¥SetupMenu' が指定されており、設定メニューの下にメニューリポジトリの３番目のメニュー内容が追加されます。'UtilityMenu¥SetupMenu¥' と指定した場合、設定メニューのサブメニューとして追加されます。

注： **メニューパスの後尾にバックスラッシュ（¥）が付いた場合と付かない場合で、結果が異なることに注意してください。**

元のメニュー **UtilityMenu¥SetupMenu の場合** **UtilityMenu¥SetupMenu¥ の場合**

MnuRemove 関数を使用する



MnuRemove() 関数は、**MnuAdd()** 関数で追加されたメニューを削除します。構文は以下の通りです。

MnuRemove(*MenuEntry*, *MenuPath*)

パラメータ：

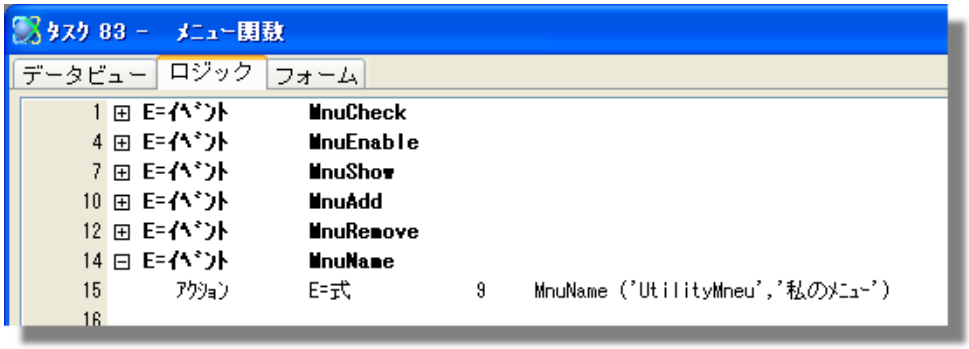
- *MenuEntry* ：メニュー番号です。この例で指定されている **'3'MENU** は、メニューリポジトリ上の３番目のメニューを示しています。

- *MenuPath* : メニューの削除先のメニューパスです。前述の **MnuAdd()** 関数で指定されたメニューパスが指定されています。

MnuReset() 関数を使用する

MnuReset() 関数は、メニューをデフォルト状態にリセットします。メニュー全体を初期化するだけのため、パラメータはありません。

MnuName() 関数を使用する



MnuName() 関数は、メニューの名前を変更することができます。メニューの表示が変わるだけで、動作には影響しません。構文は以下の通りです。

MnuName(*EntryName*, *EntryText***)**

パラメータ：

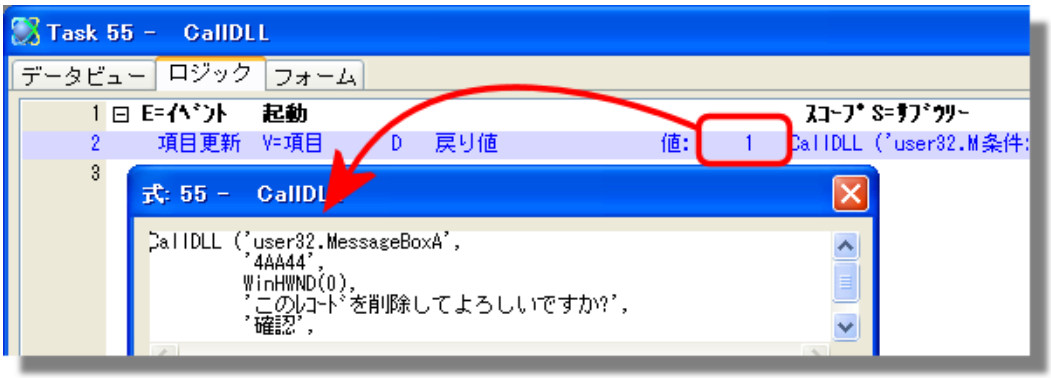
- *EntryName* ：**メニュー定義**テーブルに定義されている**論理メニュー名**です。**論理メニュー名**は、実行時にメニューとして表示されないので任意の名前が定義できます。
- *EntryText* ：実行時に表示されるメニューを表すテキストです。この例では、ユーティリティという名前を私のメニューに変更するように指定されています。

DLL 関数を呼び出すには

Windows 環境や、購入したパッケージには、たくさんの利用可能な DLL があります。独自に作成したものを使用する場合も考えられます。各々の DLL の呼び出し方は異なっていますが、呼び出し先に合わせてパラメータを設定する必要があります。ここでは、Windows API を呼び出してメッセージボックスを表示させる簡単な例を元に説明します。

Magic では、**CallDll()** 関数やコール UDP 処理コマンドを使用して、DLL を呼び出すことができます。ここでは両方の呼び出し方で説明しています。

CallDLL() 関数を使用する



1. コール処理を定義したい行で *F4* を押下します。
2. U を押下して項目更新処理コマンドを作成するか、プルダウンリストから処理コマンドを選択します。
3. 戻り値を格納するデータ項目を選択します。この場合、呼び出す DLL は数値を返すため、戻り値には数値型項目を使用します。数値は、ユーザからの応答を表します。
4. 値カラムでズームして、起動する DLL を定義します。ここでは、**CallDLL()** 関数を使用して呼び出します。構文は以下の通りです。

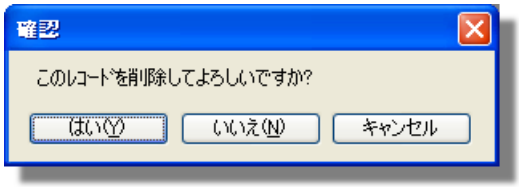
   CALLDLL(*DLLName*, *ArgString*, *Arg1*, *Arg2*. . . **)**

パラメータ：

- *DLLName* ：DLL の名前です。この例では、user32.MessageBoxA になります。
- *ArgString*：パラメータのデータタイプを表す文字列です。各パラメータに対して 1 文字が対応します。最後の 1 文字は、戻り値を表します。
- *ArgX* ：渡されるパラメータです。この例では、４つのパラメータが渡されます。（Window ハンドル値、２つの文字列、およびボックスのスタイルを表す数値）

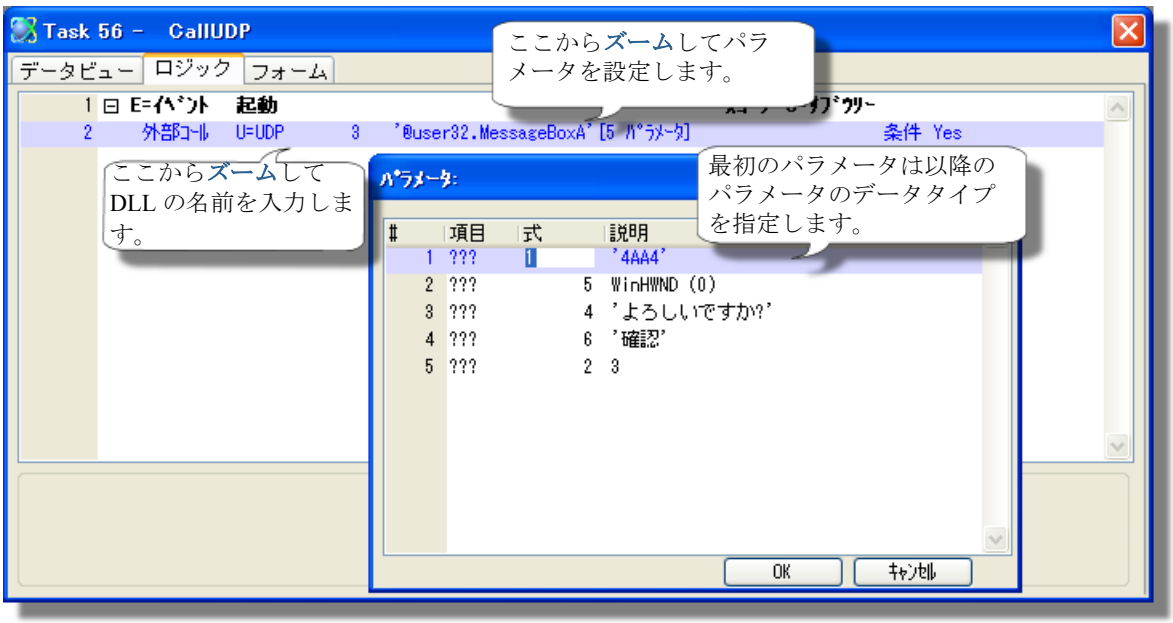
文字	データのタイプ
1	Char
2	Short
4	Long
F	Float
8	Double
D	Double pointer
E	Float pointer
L	Long pointer
A	Null terminated string pointer
V	Void pointer
0	Void

CallDLL 関数を定義し実行すると、Windows のメッセージボックスが表示されます。ユーザがボタンをクリックすると、クリックされたボタンに応じた戻り値（6= はい、7= いいえ、2= キャンセル）が返ります。



ヒント : このように、OS に依存する関数を使用する場合、Magic のタスク内でカプセル化することを推奨します。このようにすると、別の方法で呼び出したりインストール内容が変わったりした場合に変更が簡単になります。

コール UDP 処理コマンドを使用する



コール UDP 処理コマンドを使用して DLL を呼ぶ場合も、**CallDLL()** 関数を使用した場合と同じように利用できます。

1. コール処理を定義したい行で *F4* を押下します。
2. I を押下して外部コール処理コマンドを作成し、次のカラムに移動します。
3. U 押下して **UDP** を選択します。次のカラムに移動します。
4. ズームして式エディタを開き、起動する DLL の名前を入力します。DLL 名の前に @ を指定します。この例では、Windows の user32.MessageBoxA を呼び出すため以下のようになります。

   ```
   @user32.MessageBoxA
   ```

   Enter を押下してロジックエディタに戻り、次のカラムに移動します。
5. ズームしてパラメータテーブルを開きます。ここでは **CallDLL()** 関数の場合と同じ書式で設定します。
   最初のパラメータには、DLL が必要な各パラメータのデータタイプを示す文字を指定します。MessageBoxA 関数の場合は、４つの文字を指定します。*4* は long integer、*A* は Null で終わる文字列を表します。その他のパラメータに対しては必要に応じて項目や式特性からズームして設定します。

構造体のパラメータを DLL に渡すには

Magic は、BLOB 項目を利用したバッファを扱うことができます。バッファ関数を使用することで構造体を構築することができ、構造体にデータを渡したり、受け取ったりすることができます。

構造体を作成する

1. BLOB 項目を定義します。ここには、バイナリデータが格納されます。
   バッファは、バイナリデータとして格納されます。BLOB 自体はデータ長を意識していませんので、データを格納したり取り出す際は、該当データの格納位置とデータ長を明示的に指定する必要があります。
2. 数値データをバッファに追加する場合は、**BufSetNum** 関数を使用します。
   例： BufSetNum ('H'VAR,1,B,3,8)
   BufSetNum 関数のパラメータは以下の通りです。
   - **バッファ項目……全ての バッファ関数で指定します。値が追加されるバッファ項目として** BLOB **型の項目を指定します。**
   - **位置…… 値が追加される位置です。**

   この例では、位置を「1」にしています。構造体の最初であることを示しています。（Magic では、位置指定は 1 から始まります。）
   バッファは、連続したデータの集まりです。そのため、バイナリデータのどこに追加するかを指定することはとても重要です。
   値を連続して指定する場合、値を設定する位置と長さを考慮に入れて位置を指定してください。
   例えば、「位置 =1」、「長さ =2」のデータの後に続くデータの位置は、「3」になります。
   - **値…… 構造体に追加する値です。関数のタイプに合っていなければなりません。例えば、BufSetNum 関数や BufSetBit 関数の場合は、数値であり、BufSetAlpha 関数の場合は、文字データにしなければなりません。**

   ここには、項目や固定値を組み合わせた定義式でも指定できます。
   - **記憶形式……BufSetVariant 関数以外のすべての BuffSet 関数の四番目のパラメータで、数値によって記憶タイプを指定します。**

   例えば、数値を float タイプでバッファに追加するといった指定を行います。有効な記憶タイプは、別表で示しています。
   - **長さ…… 最後のパラメータは、追加するデータの長さです。記憶タイプの長さと合っていなければなりません。この例では、「8」に設定されています。**

参照： **バッファ関数の記憶形式タイプの詳細は、**Magic **の『リファレンスヘルプ』を参照してください。**

Magic からバッファを送る

コールユーザ PRC 処理 コマンドを使用して外部の dll を呼び出す際、バッファを使用してパラメータの受け渡しをすることができます。以下の例では、PC のシステム日付を取得する Windows の API を呼び出しています。

変数項目を準備する

1. オンラインタスクを作成し変数項目を定義します。

シンボル名	名前	型	書式
A	バッファ用変数	B=BLOB	
B	年数格納用変数	N= 数値	4
C	月数格納用変数	N= 数値	2
D	週の何日目かを格納する変数	N= 数値	1
E	日にち格納用変数	N= 数値	2
F	時数数納用変数	N= 数値	2
G	分数格納用変数	N= 数値	2
H	ミリ秒格納用変数	N= 数値	3

API を呼び出すロジックユニットを定義する

1. タスクのロジックエディタを開き、ユーザイベントに対応したロジックユニットを作成します。
2. アクション処理コマンドで構造体の初期化を行います。

```
BufSetNum ('A'VAR,1,0,1,2)
BufSetNum ('A'VAR,3,0,1,2)
BufSetNum ('A'VAR,5,0,1,2)
.....
BufSetNum ('A'VAR,15,0,1,2)
```

3. コールユーザ PRC 処理コマンドを定義し外部 dll を呼び出すようにします。呼び出す関数を定義式で指定します（規約特性は、*STDCALL* に設定します）。

   通常、これらの Dll は、Magic 用にコンパイルされているわけではありません。このため、文字列は以下のように「@」で始まるようにしてください。

   ```
   '@kernel32.GetSystemTime'
   ```

   これは、kernel32.dll 内の GetSystemTimeí 関数を呼び出しています（kernel32.dll は、Windows に標準で添付されているモジュールです）。dll ファイル名にパスを指定することもできます。例えば、

   ```
   '@C:¥Windows¥System32¥kernel32.GetSystemTime'
   ```

   パスが指定されていない場合は、Magic は、最初に Magic の作業フォルダを探し、ない場合は、環境変数の「PATH」で指定されているディレクトリを検索します。

   System32 ディレクトリは、OS で認識できるパスのため、通常指定する必要はありません。
4. コールユーザ PRC 処理コマンドの最初のパラメータとして、「関数マスク」と呼ばれる関数のヘッダを表す文字列を指定します。GetSystemTime 関数の関数ヘッダは、以下の通りです。

   ```
   Public Declare Sub GetSystemTime Lib "kernel32" Alias "GetSystemTime" (lpSystemTimeAs
   SYSTEMTIME)
   ```

   GetSystemTime 関数を呼び出すには、SYSTEMTIME 構造体を必要とし、Sub(Void) と定義されていて戻り値がないことを示しています。このため「関数マスク」は、「T0」となります。

注：　関数の説明は、dll のベンダから取得するか、そのソースファイルを見てください。

「関数マスク」は、以下の文字で表されます。

- 1 …… Char
- 2 …… Short
- 4 …… Long
- F …… Float
- 8 …… Double
- D …… Double pointer
- E …… Float pointer
- L …… Long pointer
- A …… Null で終了する String のアドレス
- V …… Void pointer
- 0 …… Void
- T …… Structure

5. コールユーザ PRC 処理コマンドの二番目のパラメータとして、GetSystemTime 関数に渡すパラメータを指定します。この例では、構造体 SYSTEMTIME をもつ BLOB 項目を渡します。

   パラメータの値は、「参照渡し」で渡され、（外部）関数はその内容を必要に応じて更新します。つまりパラメータは以下のようになります。

   - ??? 式 #　‘T0’
   - A　0　（バッファ用変数）
6. コールユーザ PRC 処理コマンドの後に、バッファから値を取り出す処理を定義します。項目更新処理コマンドで取得したデータを変数項目に格納します。
   - 年数格納用変数（B）……BufGetNum ('A'VAR,1,1,2)
   - 月数格納用変数（C）……BufGetNum ('A'VAR,3,1,2)

   ...
   - ミリ秒格納用変数（H）……BufGetNum ('A'VAR,15,1,2)

注：　SYSTEMTIME *構造体は次のように定義されています。*

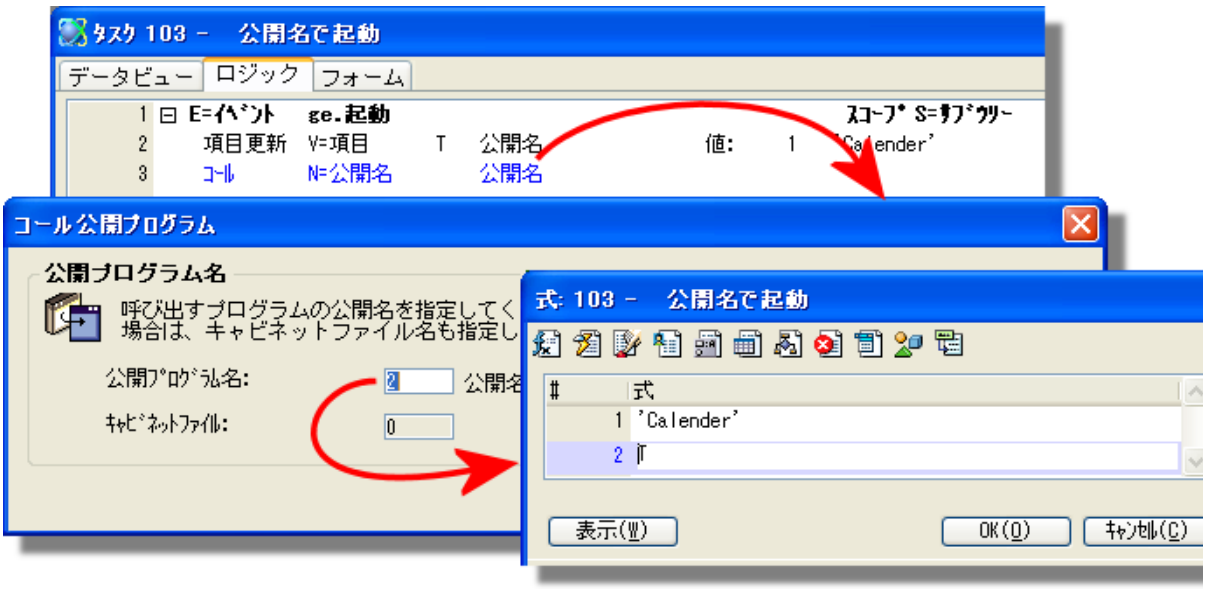
```
Public Type SYSTEMTIME
    wYear As Integer
    wMonth As Integer
    wDayOfWeek As Integer
    wDay As Integer
    wHour As Integer
    wMinute As Integer
    wSecond As Integer
    wMilliseconds As Integer
End Type
```

公開名でプログラムを呼び出すには

プログラムに公開名が定義されている場合、この公開名を指定してプログラムを呼び出すことができます。通常プログラムを呼び出す場合は、このような方法は必要ありません。

しかし、データソース内に起動するプログラム名が格納されている場合は便利です。また、実行時にアクセス可能なキャビネットファイル名を指定して呼び出すような場合にも利用できます。

公開名でプログラムを起動する



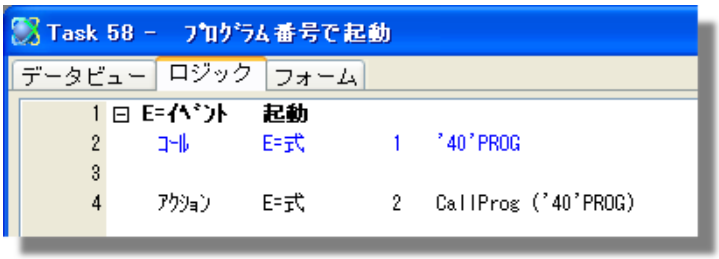
1. *F4* を押下して処理コマンド行を追加します。
2. *C* を押下してコール 処理コマンドを選択します。
3. *N* を入力するかプルダウンリストからコールタイプとして公開名を選択します。次のカラムに移動します。
4. ズームしてコール公開プログラムダイアログボックスを開きます。
5. 公開プログラム名でズームして式エディタを開き公開名を定義するか実行時に公開名として評価される項目を定義します。
   起動したいプログラムが別のキャビネットファイルに定義されている場合は、キャビネットファイルでズームして同じようにキャビネットファイル名を指定します。

   これで実行時に項目に設定されたプログラム名が呼び出されます。

プログラム番号を指定して動的にプログラムを呼び出すには

プログラム番号を指定して呼び出すことができます。これには２つの方法があります。コール式処理コマンドを使用する方法と **CallProg()** 関数を使用する方法です。

CallProg() 関数は、式を使用してプログラムを呼び出す場合や戻り値を他の関数のパラメータとして使用する場合に便利です。



コール式処理コマンドを使用する

1. *F4* を押下して処理コマンド行を追加します。
2. *C* を押下してコール 処理コマンドを選択します。
3. *E* を入力するかプルダウンリストからコールタイプとして*式*を選択します。次のカラムに移動します。
4. ズームして式を定義します。式は、プログラム番号を **PROG** リテラルで指定したものです。例えば、プログラム **#40** を指定する場合、'40'PROG と入力します。

CallProg() 関数を使用する

1. 式エディタで以下のように関数を入力します。
   CallProg(*'n'PROG***)**
   n は起動するプログラム番号です。
   これは、初期値としてフォーム上に表示したり、他の式の一部として使用することができます。プログラムに戻り値がない場合は、アクション処理コマンドでも使用できます。

プログラムの公開名を使用して、式からプログラムを呼び出すために **ProgIdx()** 関数と組み合わせて **CallProg()** 関数を使用することができます。

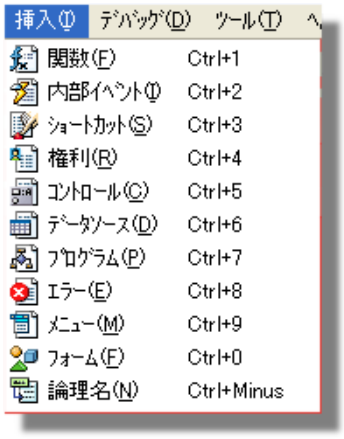
呼び出すプログラムの名前と番号を保存したい場合は、この方法が便利です。例えば、ユーザが独自のメニューシステムを作成するためのテーブルを作成することができます。

注： **PROG** リテラルを使用すると、プログラムリポジトリ内でプログラムが移動してもプログラム番号が同期をとることができます。**CallProg(***61***)** と定義した場合、この関数は実行できますが、プログラム **#61** が **#65** に移動しても、関数は今まで通りプログラム **#61** を呼び出してしまします。

ヒント：この方法は* ProgIdx() *関数と組み合わせ使用する場合に便利です。この関数は、公開名を指定することでプログラム番号が返ります。

プログラムを設定する

プログラム番号を選択する場合、あらかじめ記憶しておく必要はありません。式に追加する項目を選択するために、挿入メニュー（またはコンテキストメニュー）を使用することができます。

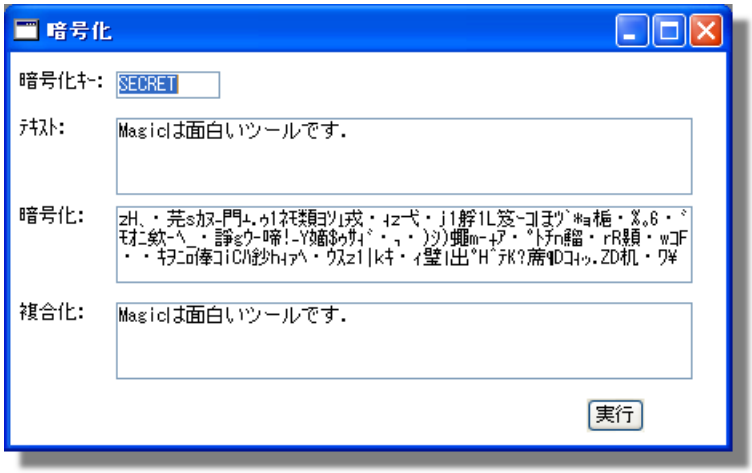


上記の例のプログラム番号を選択するには以下のようにします。

1. *Ctrl+7* を押下して（または、挿入→プログラム、右クリック→プログラム、  アイコン）、プログラム一覧を表示させます。
2. 必要なプログラムを探します。位置付（*Ctrl+L*）機能を利用したり、プログラム名の先頭の文字を入力したり、スクロールすることでプログラムを検索することができます。
3. 選択ボタンをクリックします。

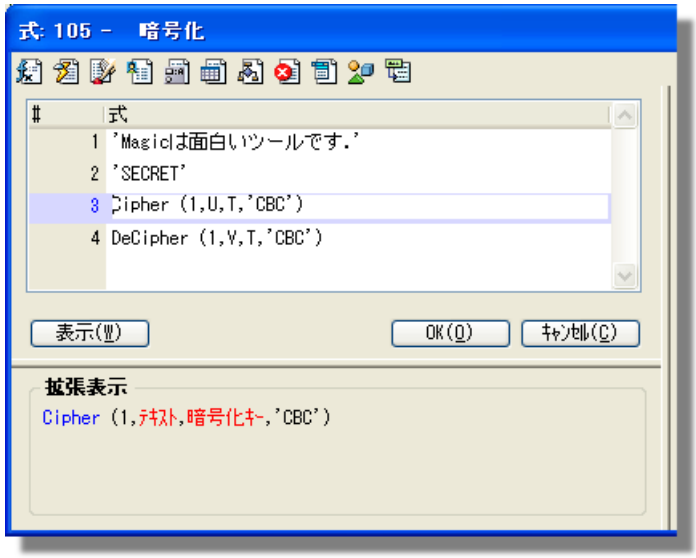
プログラム番号が式に設定されます。

データの暗号化と復号化をするには



Magic でデータを暗号化したり、復号化したりするには **Cipher()** と **DeCipher()** 関数を使用することができます。これらの関数は標準的な暗号化アルゴリズムをサポートしているため、他のアプリケーションで暗号化されたデータを解読することができます。サポートするアルゴリズムは次の通りです：Blowfish, CAST, DES, IDEA, RC2, RC4, RC5, DES3。

Cipher() 関数を使用する



Cipher() 関数の構文は以下の通りです。

Cipher(*Cipher ID*, *Buffer*, *Key* [, *Mode*, *IV*]**)**

パラメータ：

- *Cipher ID*：使用する暗号化アルゴリズムを指定しています。この例では 1 が指定されています。これは、**Blowfish** を指定しています（「サポートする暗号化方法」（156 ページ）を参照してください）。
- *Buffer*：暗号化されるデータを含んだ文字型または、BLOB 型項目を指定します。
- *Key*：キーを含んだ文字型または、BLOB 型のデータを指定します。必要なキーの長さは使用するアルゴリズムに依存します。この例では、キーは SECRET という文字列です。
- *Mode*：どのモードを使用するかを指定するオプションのパラメータです。使用できるモードはアルゴリズムに依存します。
- *IV*：初期ベクトルを含んだ BLOB データです。このパラメータもオプションです。

Cipher() 関数は、暗号化された文字列を含んだ値が返ります。

Decipher() 関数を使用する

Decipher() 関数の構文は、**Cipher()** 関数と同じです。

Decipher(*Cipher ID*, *Buffer*, *Key [, Mode, IV]***)**

パラメータ：

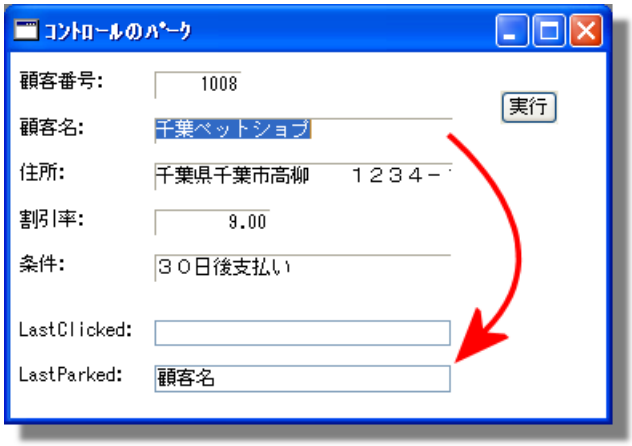
- *Cipher ID*：使用する暗号化アルゴリズムを指定しています。この例では 1 が指定されています。これは、**Blowfish** を指定しています（第 6 章：「サポートする暗号化方法」（156 ページ）を参照してください）。
- *Buffer*：暗号化されるデータを含んだ文字型または、BLOB 型項目を指定します。
- *Key*：キーを含んだ文字型または、BLOB データを指定します。必要なキーの長さは使用するアルゴリズムに依存します。この例では、キーは SECRET という文字列です。
- *Mode*：どのモードを使用するかを指定するオプションのパラメータです。使用できるモードはアルゴリズムに依存します。
- *IV*：初期ベクトルを含んだ BLOB データです。このパラメータもオプションです。

Decipher() 関数は、復号化された文字列を含んだ値が返ります。

サポートする暗号化方法

暗号名	暗号 ID	サポートするモードと IV の長さ	キーの数とキー長	対称
BLOWFISH	1	ECB - NA CBC - 8 CFB - 8 OFB - 8	最小値：1 最大値：56 推奨値：16	対称
CAST	2	ECB - NA CBC - 8 CFB - 8 OFB - 8	最小値：5 最大値：16 推奨値：8	対称
DES	3	ECB - NA CBC - 8 CFB - 8 OFB - 8	キー数：1 サポート値：8 推奨値：8	対称
RC2	5	ECB - NA CBC - 8 CFB - 8 OFB - 8	最小値：5 最大値：16 推奨値：8	対称
RC4	6		最小値：1 最大値：NR 推奨値：16	対称
RC5	7	ECB - NA CBC - 8 CFB - 8 OFB - 8	最小値：1 サポート値：255 推奨値：16	対称
DES3	8	ECB3 - NA CBC3 - 8	キー数：2 最大値：16 or 24 推奨値：24	非対称
RSA	9		最小値：48 最大値：2048 推奨値：128	非対称

ユーザがパークしたコントロール名を取得するには



処理を行う際に、ユーザがどこで最後にパークしたかを知る必要があるかもしれません。**Lastpark()** 関数を使用することで、これを実現することができます。これは、最後にクリックしたコントロールではありません。上の例では、Tab によってある入力項目から別の入力項目に移動した場合を示しています。最後にパークしていた項目は顧客名になります。しかし、どの項目もクリックしていないため **LastClicked()** 関数からは空白が返ります。

LastPark() 関数は、コントロールを含んでいる現在のタスクや子タスクからコントロール名を取得します。しかし、以下のような動作になります。

- 現在パークしている項目のコントロール名は返りません。
- 親タスク内のトリガによって実行されたロジックユニットから起動されたタスクでは値が返りません。

HandledCtrl() 関数を使用して現在のタスク、または親タスクからパークしていたコントロール名を取得することができます。現在の項目の情報を取得するには **This()** 関数を使用することができます。

LastPark() 関数を使用する

LastPark() 関数の構文は以下のとおりです。

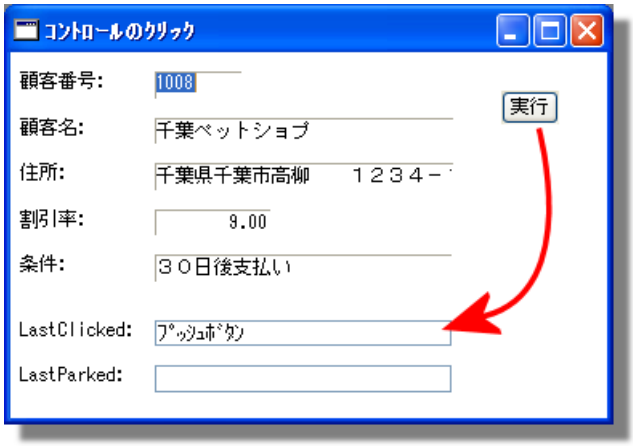
LastPark(*Generation***)**

パラメータ：

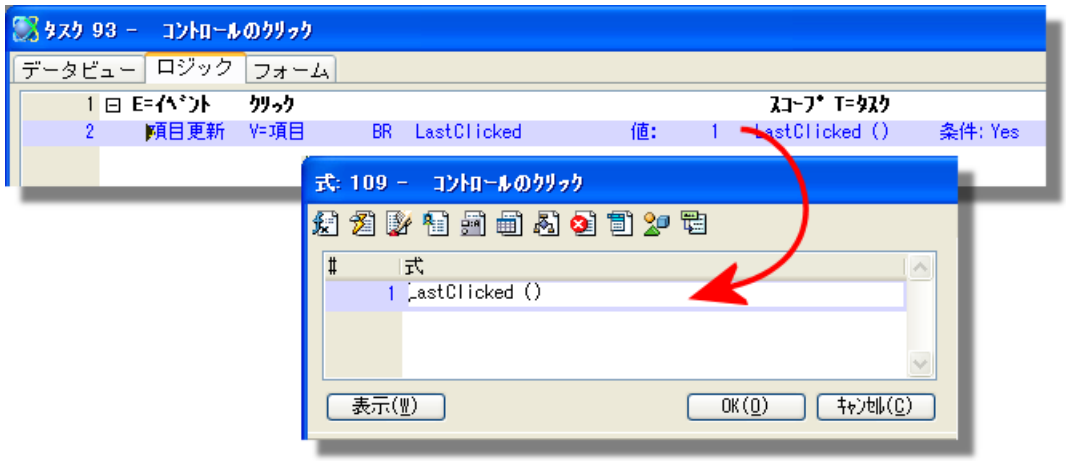
- *Generation* ：タスクツリー上のタスクの位置を示す番号です。**0** は自タスク、**1** は親タスクといったように指定します。

注： 現在のタスクの 1 つの項目に対して何らかの処理を行う場合、そのコントロールに対するためにコントロールイベントを使用することで、これを行うことができます。

ユーザがクリックしたコントロール名を取得するには



処理を行う際に、ユーザどこで最後にクリックしたかを知る必要があるかもしれません。プッシュボタンのようにカーソルがパークできないコントロールも含まれるため、最後にパークした場所とは異なります。

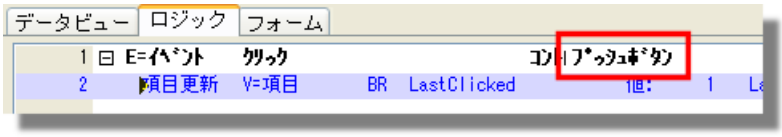


LastClicked() 関数を使用する

必要条件： コントロールには、すでにコントロール名が定義されているものとします。定義されていない場合は何も返りません。

LastClicked() 関数を使用することでユーザが最後にクリックしたコントロール名を取得します。パラメータは指定しません。戻り値として、コントロール名を表す文字列が返ります。この値は、コントロール名を確認するために、他の関数に渡したり、式で使用することができます。

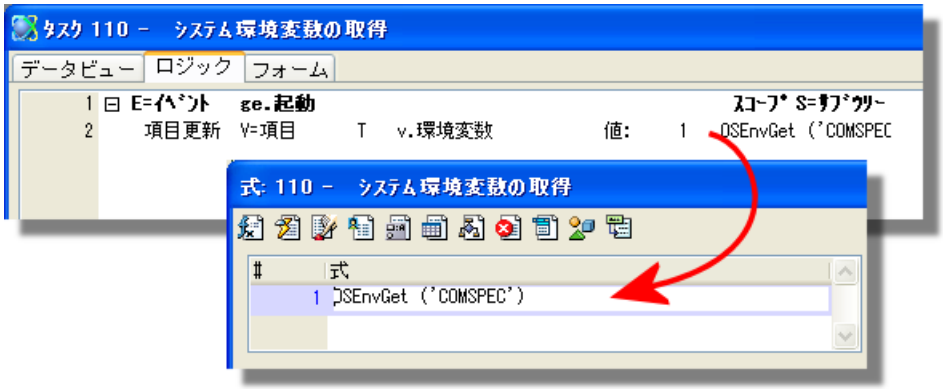
注： ユーザがクリックしたコントロール名にもとづいた処理を実行する場合、クリックイベントのロジックユニットに対してフィールド（コントロール）を設定することでも同じようなことができます。例えば、上記の例において、イベントロジックユニットを定義した場合は、以下のように定義します。



設定されたコントロールがクリックされた場合だけ、このロジックユニットは実行されます。

システムの環境変数から値を取得するには

OS の環境変数には、システムのユーザ ID、OS のタイプ、バージョン、コンピュータ名、およびコマンドシェルのパスなどのような利用可能な多くの情報があります。**OSEnvGet** 関数を使用して環境変数を取り出すことにより Magic プログラムでこれらを利用することができます。



OSEnvGet 関数を使用する

OSEnvGet 関数の構文は以下の通りです。

OSEnvGet(*Variable*)

パラメータ：

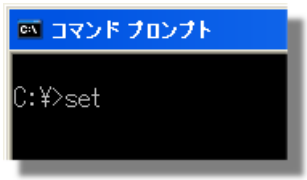
- *Variable*：環境変数の名前を表すテキストです。

この例では、コマンドシェルへのパスを格納するため、項目更新処理コマンドを使用しています。使用すると便利な環境変数名には以下のようなものがあります：

- COMSPEC …… シェル のパス名
- OS …… OS の種類
- USERNAME …… 現在のユーザのユーザ名
- USERDOMAIN …… 現在のユーザのログオンドメイン
- LOGONSERVER …… ドメインコントローラにログオンしたときに表示される名前

しかし、ユーザ独自の環境変数を定義することもできます。ユーザ独自に環境変数を定義することで、便利なこともあります。

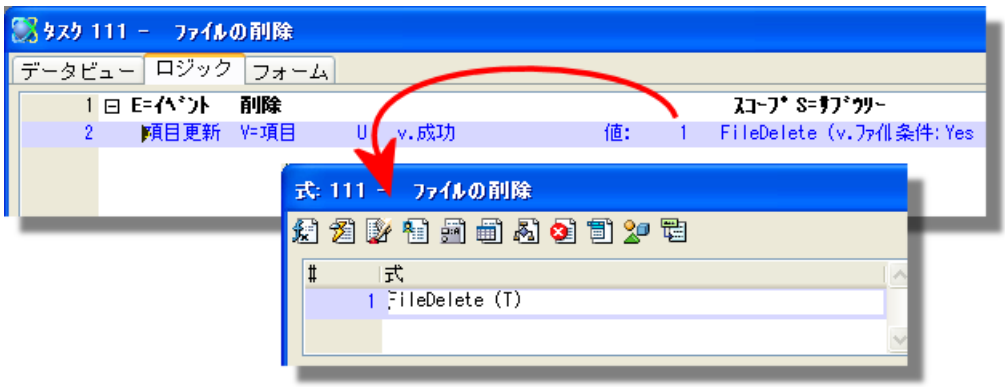
ヒント :*現在の環境変数を確認するには、コマンドプロンプトを起動し、SET コマンドを実行することで表示されます。*



ディスク上のファイルを削除するには

ディスク上のファイルを削除したい場合が考えられます。例えば、一時ファイルを作成した後に、一時ディレクトリからこのファイルを消去させたい場合があります。または、ファイルを再作成する場合、処理を実行する前に古いファイルを削除させる必要があります。

OS コマンドを使用するよりも Magic の関数を使用した方がより便利です。これは、OS コマンドを使用すると、OS のバージョンに依存する場合があるからです。Magic の関数を利用した方がより確実です。また、Magic の関数は、戻り値を簡単に確認することができます。



FileDelete() 関数を使用する

FileDelete() 関数を使用することでディスク上のファイルを削除することができます。構文は以下の通りです 。

FileDelete(*FileSpec*)

パラメータ：

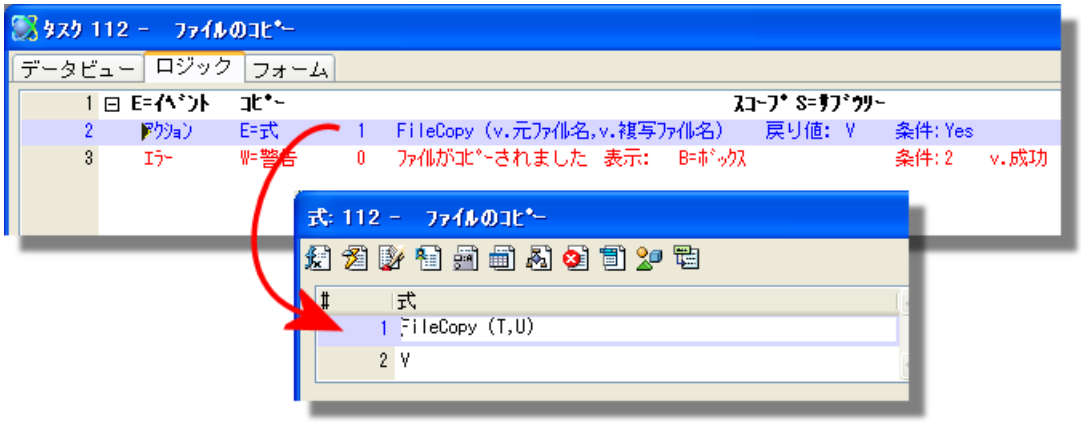
- *FileSpec*：削除するファイル名を表す文字列です。削除が成功した場合、True が返ります。

注： 削除ができなかったり、ファイルが存在しない場合は、False が返ります。しかし **FileExist()** 関数と組み合わせて使用することで、ファイルが削除されたかどうかを確認できるためより確実になります。

参照： 「FileExist() 関数を使用する」（162 ページ）

ディスク上のファイルをコピーするには

ディスク上のファイルのコピーしたい場合があります。例えば、バックアップや保存のためのコピーを作成するような場合が考えられます。



OS コマンドを使用するよりも Magic の関数を使用した方がより便利です。これは、OS コマンドを使用すると、OS のバージョンに依存する場合があるからです。Magic の関数を利用した方がより確実です。また、Magic の関数は、戻り値を簡単に確認することができます。

FileCopy() 関数を使用する

FileCopy() 関数を使用してディスク上のファイルをコピーすることができます。構文は以下の通りです。

FileCopy(*origin*, *target*)

パラメータ：

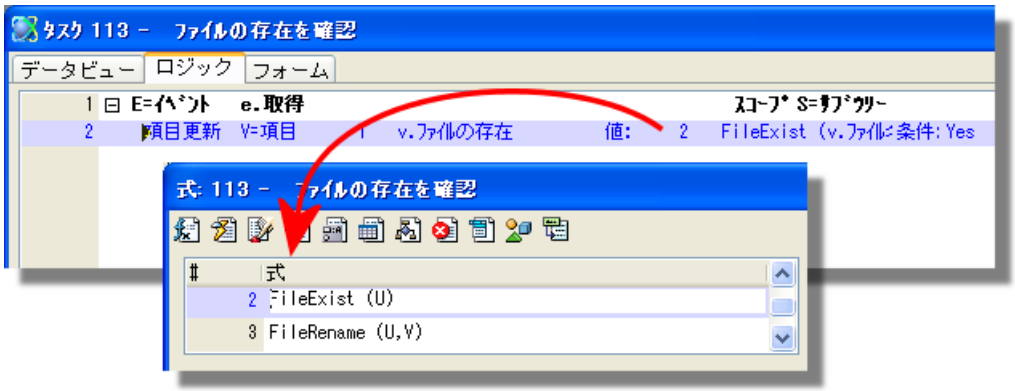
- *origin*：コピー元のファイルを表す文字列です。
- *target*：コピー先のファイルを表す文字列です。コピーが成功した場合、True が返ります。

注： **FileCopy()** 関数は、同じ名前のファイルが存在している場合上書きします。このような場合に警告メッセージを表示させるには、事前に **FileExist()** 関数を実行してチェックする必要があります。

参照： 「FileExist() 関数を使用する」（162 ページ）

ディスク上のファイルの存在をチェックするには

必要な共通関数の 1 つに、ディスク上にファイルが存在しているかどうかをチェックする関数があります。例えば、ユーザが Excel からデータを読み捨てている場合や、データが見つからなかったり、処理が何らかの理由で成功しなかった場合に、メッセージを表示させたい場合があるとします。



OS コマンドを使用するよりも Magic の関数を使用した方がより便利です。これは、OS コマンドを使用すると、OS のバージョンに依存する場合があるからです。Magic の関数を利用した方がより確実です。また、Magic の関数は、戻り値を簡単に確認することができます。

FileExist() 関数を使用する

FileExist() 関数を使用してディスク上のファイルをチェックすることができます。構文は以下の通りです。

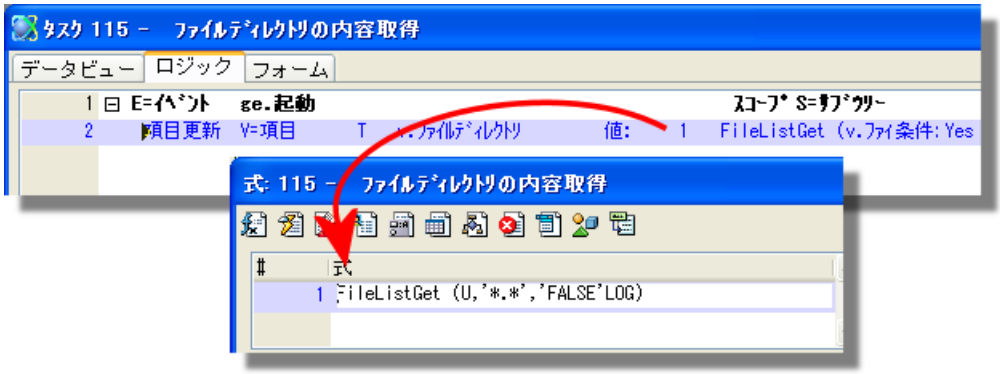
FileExist(*FileSpec*)

パラメータ：

- *FileSpec*：チェックするファイル名を表す文字列です。ファイルが見つかった場合は、True が返ります。

参照： 『リファレンスヘルプ』の入出力関数に関するトピック

ファイルディレクトリの内容を取得するには



プログラムでファイルディレクトリの内容を読み込む必要があるかもしれません。別のプログラムから、現在のプログラムで処理するためにファイルをドロップした場合（例えば、電子メールや Fax 用プログラム、または Web から送られたファイルなど）、その内容を確認する必要があります。

FileListGet() 関数を使用する

FileListGet() 関数を使用してディレクトリの内容を取得することができます。構文は以下の通りです。

FileListGet(*DirectoryName*, *Filter*, *SubDirSearch***)**

パラメータ：

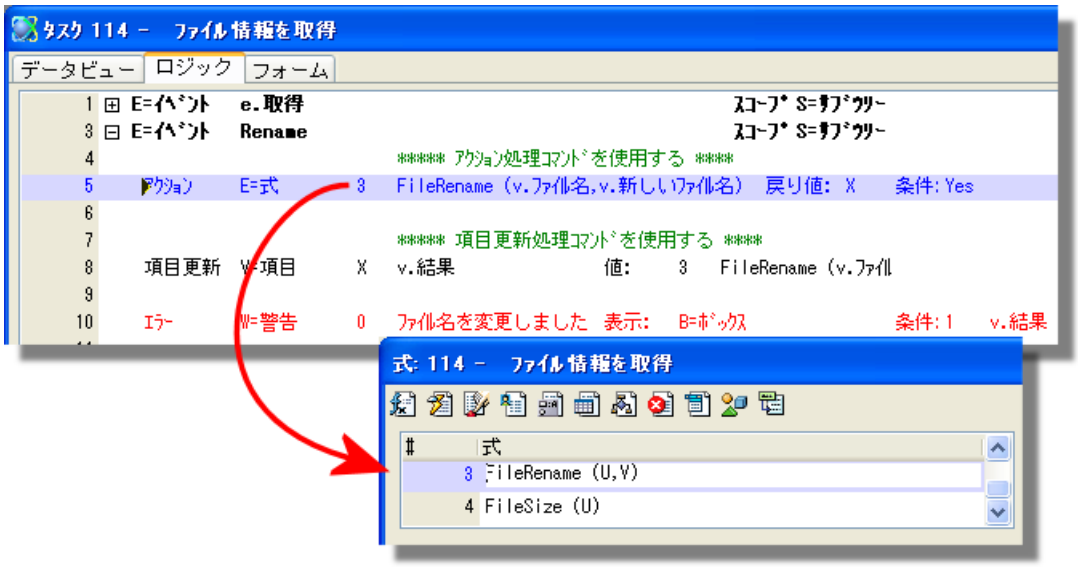
- *DirectoryName* ：検索するディレクトリ名を表す文字列
- *Filter*：フィルタ用マスク文字
- *SubDirSearch*：サブディレクトリも検索するかどうかを指定する論理値

この関数は、文字列をセルとしたベクトル値が返ります。

参照： 第 15 章：「COM オブジェクトに配列を渡す」（340 ページ）
『リファレンスヘルプ』の入出力関数に関するトピック

ディスク上のファイルをリネームするには

ディスク上のファイルの名前を変更する必要があるかもしれません。例えば、新しファイルを作成する前に既存のファイルをバックアップのためにリネームする場合が考えられます。



OS コマンドを使用するよりも Magic の関数を使用した方がより便利です。OS コマンドは、OS のバージョンに依存する場合があるからです。Magic の関数を利用した方がより確実です。また、Magic の関数は、戻り値を簡単に確認することができます。

FileRename() 関数を使用する

FileRename() 関数を使用してファイルの名前を変更することができます。構文は以下の通りです。

FileRename (*origin*, *target*)

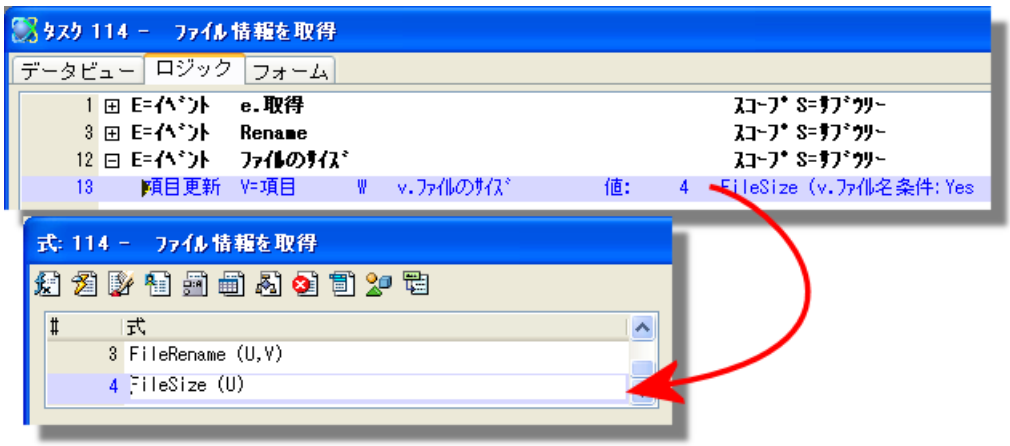
パラメータ：

- ***origin***：元のファイル名を表す文字列です。
- ***target***：変更後のファイル名を表す文字列です。リネームが成功した場合、True が返ります。

上の図では、２つの例が示されています。最初は 7 行目です。**アクション**処理コマンドの**戻り値**カラムを使用して戻り値を取得しています。２番目は、10 行目です。**項目更新**処理コマンドを使用して同じ処理を定義しています。どちらも正しく動作します。

参照： 『リファレンスヘルプ』の入出力関数に関するトピック

ディスク上のファイルのサイズを取得するには



ディスク上のファイルのサイズを知る必要があるかもしれません。データを作成したり、電子メールに添付する際に大きすぎないかどうかをチェックする場合などが考えられます。

FileInfo() 関数を使用する

FileInfo() 関数を使用してファイルのサイズを取得することができます。構文は以下の通りです。

FileInfo (*FileSpec,InfoType*)

パラメータ：

- *FileSpec*：チェックするファイル名を表す文字列です。
- *InfoType*：チェックする情報のタイプをすうちで指定します。ファイルサイズの場合は、*5* を指定します。

FileSize() 関数を使用する

FileSize() 関数を使用してファイルのサイズを取得することもできます。構文は以下の通りです。

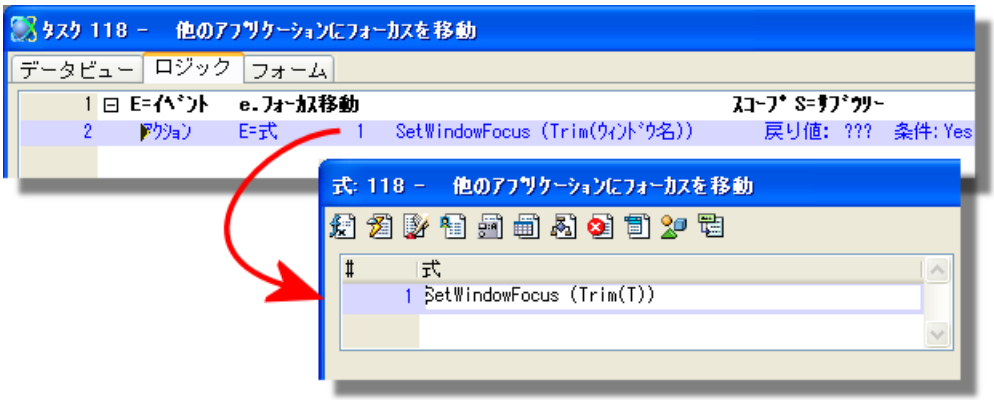
FileSize (*FileSpec*)

パラメータ：

- *FileSpec*：チェックするファイル名を表す文字列です。

参照： 『リファレンスヘルプ』の入出力関数に関するトピック

他の Windows アプリケーションにフォーカスを移すには



他のアプリケーションと連携したシステムを運用する場合、そのアプリケーションにフォーカスを移す必要があるかもしれません。

SetWondowFocus() 関数を使用する

SetWindowFocus() 関数を使用することでウィンドウ名を指定したアプリケーションにフォーカスを移すことができます。構文は以下の通りです。

SetWindowFocus (*Window Name*)

パラメータ：

- ***Window Name***：フォーカスを移したいアプリケーションのウィンドウ名

例えば、Sample.doc という文書ファイルを開いた Micorsoft Word のウィンドウ名は、「Sample.doc - Microsoft Word」になります。ウィンドウ名は、アプリケーションウィンドウの左上に表示されています。また、タスクバーや Windows タスクマネージャでも確認できます。

参照： 『リファレンスヘルプ』の入出力関数に関するトピック

第 7 章： 実行

アプリケーションを実行するには

開発しているプロジェクトが完成した場合、どのようにアプリケーションを実行させるかを検討することになります。これは使用するエンドユーザに依存する場合があります。アプリケーション毎にユニークなアイコンとアプリケーション名を定義する必要があります。また、アプリケーションが売れ筋の製品の場合、スプラッシュスクリーンを表示させたり、独自のインストール処理を必要とすることになるかもしれません。

以下は、アプリケーションの実行に関係する手順と、その詳細情報が記述されている箇所について記述しています。

#	内容	記述箇所
1.	**アプリケーションのアイコンの設定** アプリケーションを識別しやすくするためのシンボルを定義します。	「ロゴファイルを設定する」（173 ページ）
2.	**キャビネットファイルの作成** キャビネットファイルは修正できないプロジェクトの内容がパッケージ化されたファイルです。	「キャビネットファイルを作成するには」（168 ページ）
3.	**クライアントに Magic Client をインストール** *Magic Client* は、アプリケーションを実行するための製品です。	『インストールガイド』
4.	**エンドユーザ用のショートカットやメニューの作成** エンドユーザがアプリケーションを実行しやすくします。アプリケーションによっていろいろな方法があります。	「アプリケーション用のショートカットを作成するには」（169 ページ）
5.	**スプラッシュイメージの設定** アプリケーションアイコンのように作成されたアプリケーションの商標などを表示するものです。	「起動アプリケーション用のスプラッシュイメージを表示するには」（173 ページ）
6.	**自動インストールファイルの作成** アプリケーション用の自動インストールファイルを作成します。	『アプリケーションインストールユーティリティ開発者ガイド』

キャビネットファイルを作成するには

Magic Studio で作業を行っている場合、Magic のプロジェクトファイル（***.edp**）にアクセスしていることになります。この **.edp** はソースファイルのコレクションを参照しています。これは一般的に XML フォーマットので、¥SOURCE サブディレクトリ内に保存されています。

しかし、プロジェクトが完成した場合、インストール用にそれをパッケージ化することになります。パッケージ化されたファイルは、キャビネットファイル（***.ecf**）と呼ばれます。

Magic のインストールによって、**.edp** と **.ecf** のどちらのファイルもクリックすることで実行することができるように設定されます。これらのファイルの違いは以下の通りです。

.edp	**.ecf**
eDeveloper Project	eDeveloper Cabinet File
Magic Studio でオープンします。	アプリケーションを実行します。
開発者のみ	主にユーザ、または開発時にコンポーネントとして使用されます。
XML フォーマットのソースファイルが必要	XML ソースがパッケージ化されています。

プロジェクトを実行する場合、**.ecf** ファイルとその他のサポートファイル（フォントや基本色などの定義ファイル、コンポーネント）のみ必要です。

また、実行するには、ユーザは Magic の実行版（**Magic Client** か **Magic Enterprise Server**）がインストールされており、実行できる状態になっていることが必要です。

キャビネットファイルの作成は非常に簡単です。

キャビネットファイルを作成する

1. プロジェクトを開きます（**ファイル→プロジェクトを開く**）。
2. プルダウンメニューから**ファイル→キャビネット作成**を選択します。
3. キャビネットファイル名の入力要求が表示されます。ファイル名を入力するか、ファイル名を選択し保存を**クリック**します。
4. ファイルがすでに存在している場合、**上書き確認**のダイアログボックスが表示されます。上書きする場合は、はいを**クリック**します。

ダイアログボックスが閉じると、**.ecf** ファイルは作成（更新）され、使用できるようになります。

参照： 「アプリケーション用のショートカットを作成するには」（169 ページ）

アプリケーション用のショートカットを作成するには

アプリケーションを実行する最も一般的な方法は、Windows のショートカットをクリックして起動する方法です。これらのショートカットは、簡単に作成できます。ショートカットは、実行するプロジェクト（**.edp**）ファイルに対して作成するか、Magic Studio を起動するショートカットにデフォルトプロジェクトを指定するようにする作成する方法の２通りあります。

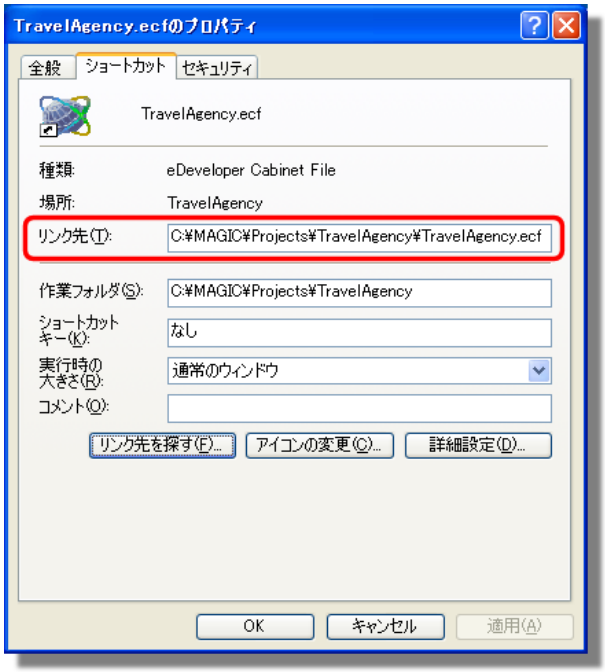
注： ショートカットの作成は必須ではありません。ユーザが Windows エクスプローラを使用して **.ecf** ファイルをクリックすることで、Magic が自動的に起動されます。

アプリケーションファイルのショートカットを作成する

1. ショートカットを作成したいフォルダ内で、コンテキストメニューを表示させ新規作成→ショートカットを選択します。ショートカット作成ウィザードが起動されます。
2. 項目の場所を入力する要求が表示されます。ここでキャビネットファイルのファイル名とパスを入力します。参照ボタンをクリックして選択することもできます。次へをクリックします。
3. ショートカットの名前を入力する要求が表示されます。ここに任意の名前を入力します。
4. 完了をクリックしてダイアログを閉じます。

これでアプリケーションを起動するためのショートカットが作成されます。デフォルトでは、Magic のアイコンが表示されますが、独自なアイコンを定義することもできます。「独自のアイコンを使用する」（170 ページ）

ショートカットのプロパティを開くと、右の例ように表示されます。**.ecf** は、リンクに設定されています。



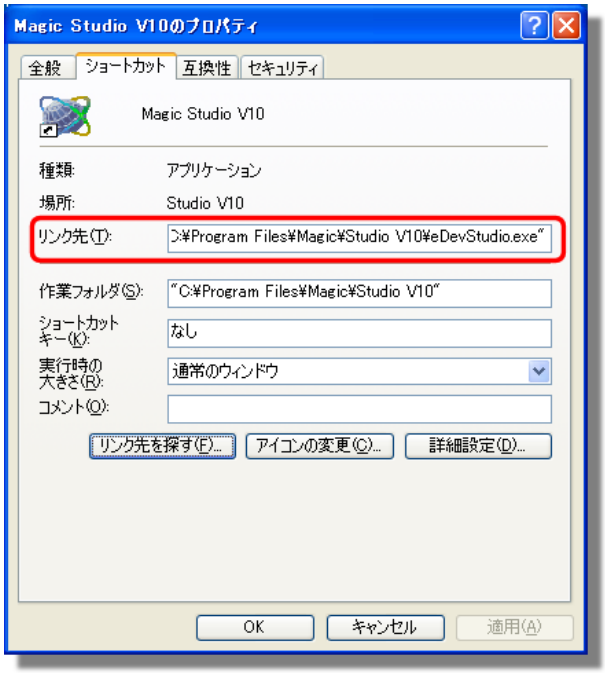
Magic エンジンのショートカットを作成する

1. ショートカットを作成したいフォルダ内で、コンテキストメニューを表示させ新規作成→ショートカットを選択します。ショートカット作成ウィザードが起動されます。
2. 項目の場所を入力する要求が表示されます。ここで Magic の実行モジュールのファイル名とパスを入力します。デフォルトは以下のようになります。

   ```
   "C:¥Program Files¥MSE¥eDeveloper
   10.1¥eDevRTE.exe"
   ```

3. ショートカットの名前を入力する要求が表示されます。ここに任意の名前を入力します。
4. 完了をクリックしてダイアログを閉じます。

ここでは Magic の実行エンジンを起動するだけの設定を説明しています。アプリケーションを自動的に起動させる場合は、「実行エンジンがキャビネットファイルを自動的に読み込むように設定するには」（171 ページ）を参照してください。



独自のアイコンを使用する

上記のどちらの方法でショートカットを作成しても、デフォルトアイコンは Magic アイコンになります。しかし、独自のカスタムアイコンを使用することもできます。

必要条件： 最初にカスタムアイコン用のファイルを作成する必要があります。

1. ショートカットのコンテキストメニューからから**プロパティ**を選択します。
2. **アイコンの変更**ボタンを**クリック**します。**アイコンの変更**ダイアログが表示されます。最上段の入力欄には、「このファイル内のアイコンを検索」と表示されています。.
3. **参照**ボタンを**クリック**して使用するアイコンファイルを選択します。選択すると、表示されていたアイコンがデフォルト Magic アイコンから選択されたアイコン表示に変わります。

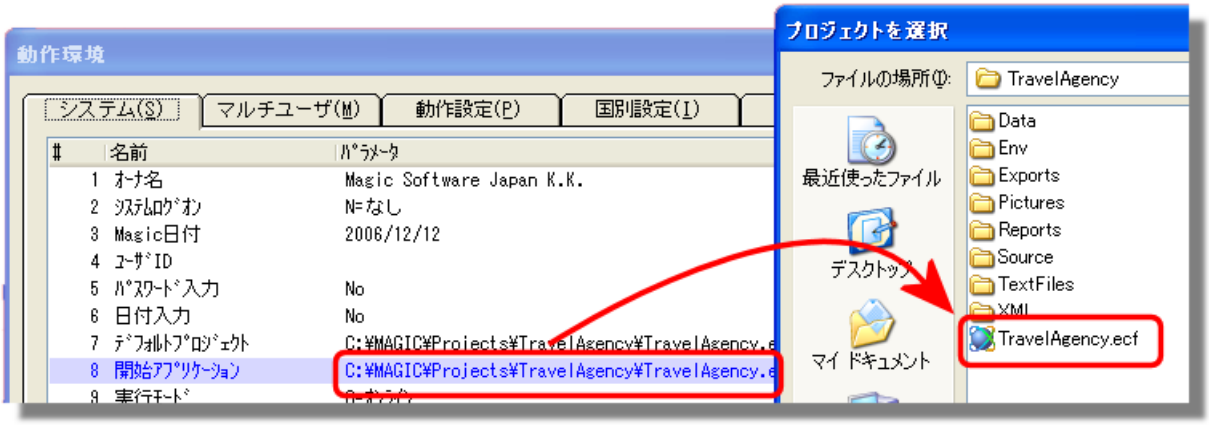
カスタムアイコン用のファイルがない場合は、Windows が持っているアイコンから選択したり、汎用のアイコンファイルを購入して使用することもできます。

実行エンジンがキャビネットファイルを自動的に読み込むように設定するには

動作環境に指定することで、実行エンジンが起動時に自動的にキャビネットファイルを読み込むようにすることができます。ここにキャビネットファイルを指定すると、Magic が起動時にデフォルトで指定されたプロジェクトが実行されます。

２つのデフォルトが存在するので注意してください。１つは**デフォルトプロジェクト**です。**Magic Studio** が起動されたときに自動的にオープンされるプロジェクトファイル（***.edp**）です。もう１つは**開始アプリケーション**です。これは実行エンジンが起動時に読み込まれるキャビネットファイル（***ecf**）です。

動作環境でデフォルトプロジェクトを設定する

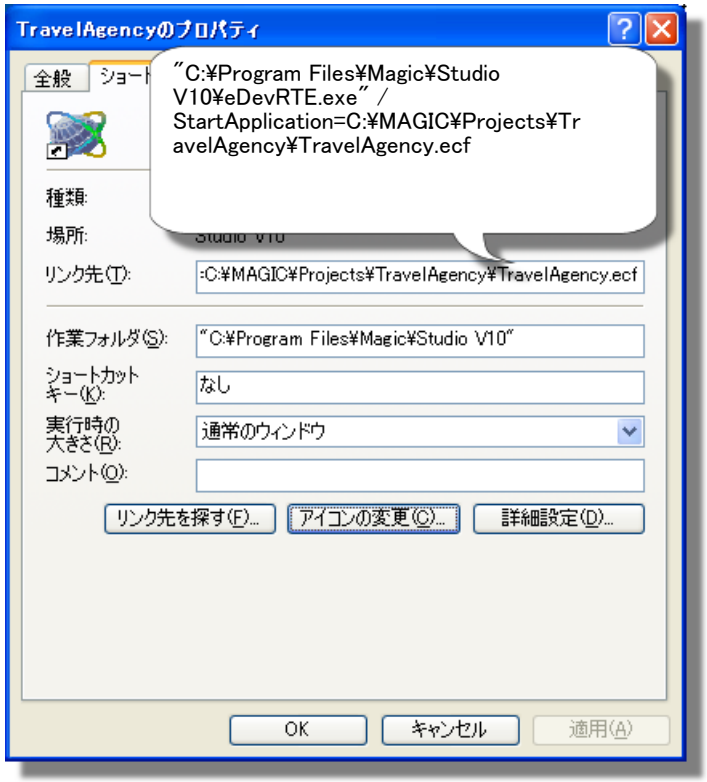


1. プロジェクトを開きます（**ファイル→プロジェクトを開く**）。
2. **動作環境**ダイアログを開き（**オプション→設定→動作環境→システム**）、**開始アプリケーション**を選択します。
3. ここから**ズーム**（***F5*** または、**編集→ズーム**）して**プロジェクトを選択**ダイアログを開きます。ここで **.ecf** ファイルを選択します。直接ファイル名を入力することもできます。

これで、実行エンジンが起動すると指定されたキャビネットファイルがオープンされます。

注： 設定された**開始アプリケーション**は Magic.ini ファイルに保存されます（Magic.ini ファイルは、デフォルトでは Magic のインストールディレクトリ内にあります）。自動的に起動させたいアプリケーションが 1 つしかない場合は、この設定で十分ですが、異なる複数のアプリケーションも同じように起動させたいような場合は、ショートカットを作成して対応する必要があります。詳細は、「アプリケーションファイルのショートカットを作成する」（169 ページ）を参照してください。

ショートカットにキャビネットファイルを設定する



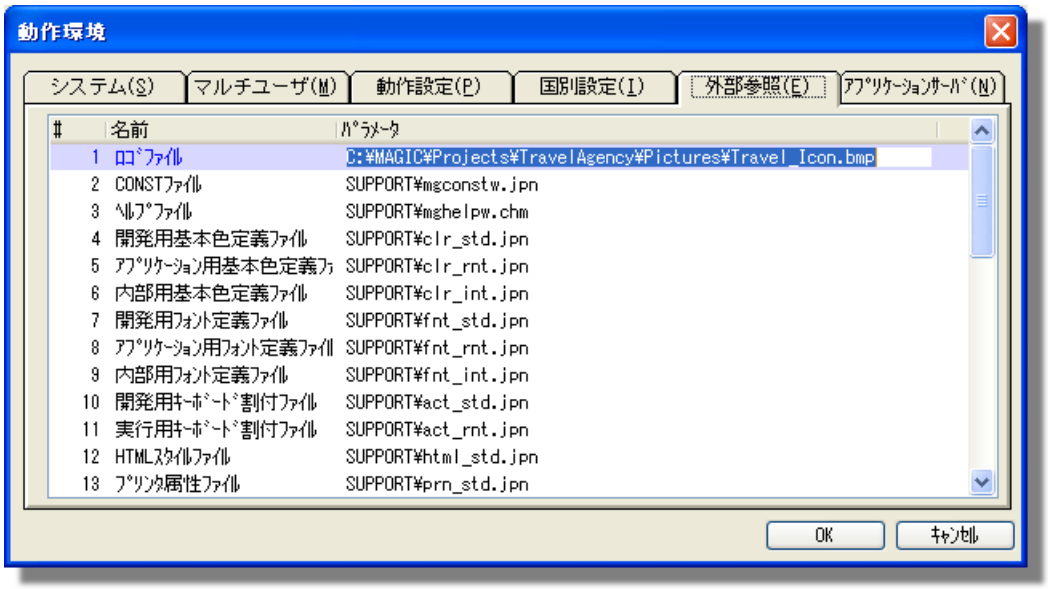
Windows のショートカットの**リンク先**にキャビネットファイルを指定することもできます。

1. **リンク先**で Magic 実行エンジンのパスとファイル名を入力します（パス名にスペースがある場合は、ダブルクォーテーションで囲む必要があります）。
2. 空白とパラメータを指定するためのスラッシュ（/）、パラメータ名（**StartApplication=**）を付加します。
3. キャビネットファイルのパスとファイル名を入力します。

起動アプリケーション用のスプラッシュイメージを表示するには

ロゴファイルを指定することで、起動時にアプリケーション用のスプラッシュイメージを表示させることができます。この設定は、Magic.ini ファイルに保存されるため、どの Magic アプリケーションが起動された場合でも表示されます。スプラッシュスクリーンの利点の一つは、アプリケーションが起動されるまでに、ユーザに対して一時的にビジュアルな演出ができることです。

ロゴファイルを設定する



1. ますロゴファイルを作成する必要があります。これはどのようなイメージファイルでも構いませんが、表示される時にリサイズされないように正しいサイズで作成する必要があります。
2. プロジェクトを開きます（**ファイル→プロジェクトを開く**）。
3. **動作環境**ダイアログを開き（**オプション→設定→動作環境→外部参照**）、**ロゴファイル**を選択します。
4. ロゴファイルの名前を入力するか、**ズーム**（*F5* または**編集→ズーム**）してファイルを選択します。

次回、アプリケーションが起動されると、スプラッシュスクリーンが表示されます。

［このページは意図的に空白にしています。］

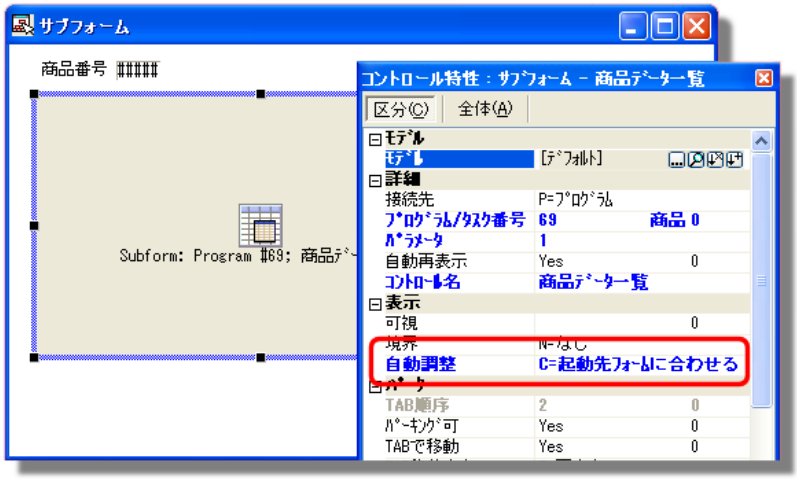
第 8 章： サブフォーム

サブフォームコントロールを起動されるプログラムフォームの寸法に合わせるには

サブフォームを定義する際に、起動されるプログラムのフォームサイズがわからない場合があります。推測して設定することもできますが、自動調整オプションを使用することで、コントロールのサイズを自動的にフォームサイズに合わせることができます。

自動調整を設定する

1. サブフォームコントロールを選択します。
2. ズームしてコントロール特性を開きます。
3. 自動調整特性で *C= 起動先フォームに合わせる* を選択します。



注： *起動先フォームに合わせる* に合わせるを使用すると、サブフォームのサイズは実行時の実際のサイズには影響しません。このため位置特性にも影響しません。

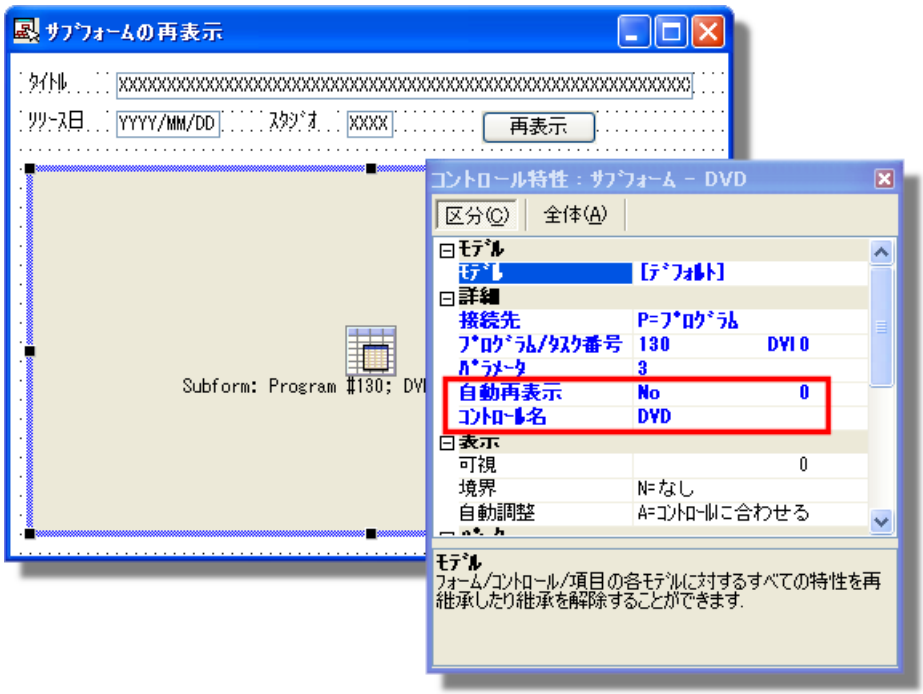
参照： 「自動調整オプションをどのように選択するか」（186 ページ）

サブフォームの再表示を手動にするには

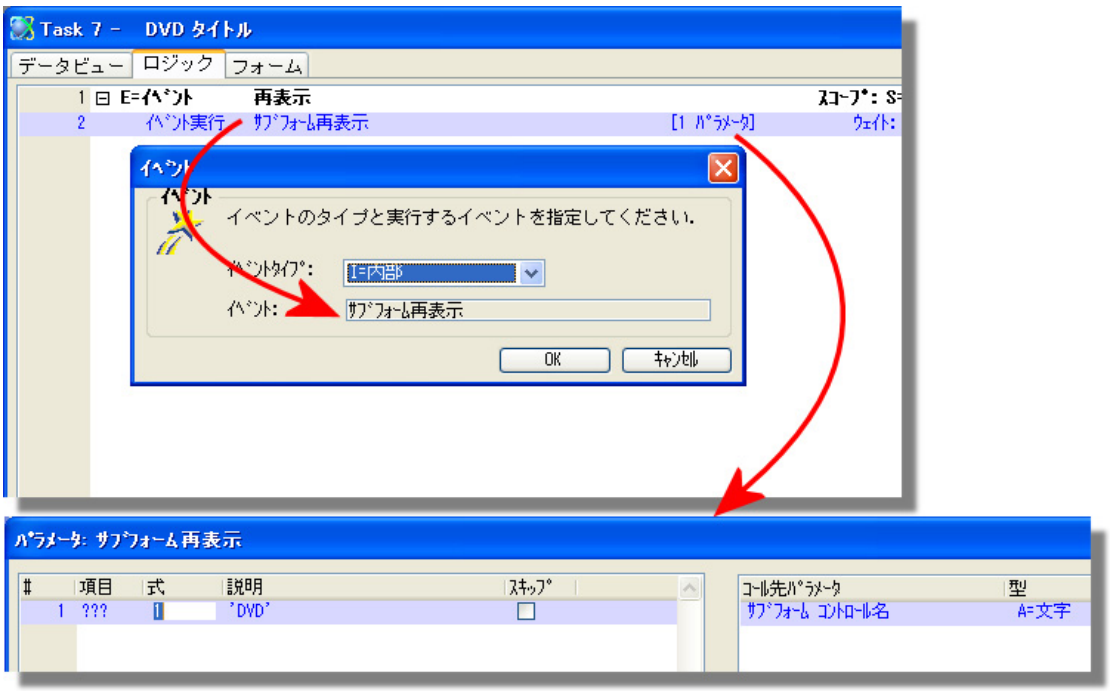
通常、子タスクに渡されるパラメータの値が変わった場合、サブフォームは自動的に再表示します。例えば、親タスクで DVD 一覧を表示し、サブフォームでその詳細を表示している場合、一覧表示をスクロールすると、詳細表示も自動的に変更します。

これは通常に要求される機能で、この処理のために開発者が何かを行う必要はありません。しかし、レコード長が大きかったり、検索が複雑で再表示に非常に時間がかかる場合があると、ユーザにとって面倒なことになるかもしれません。このような場合、自動的な再表示機能を無効にし、ユーザが検索（または再表示）のボタンをクリックした場合だけ実行させるようにした方が都合がいい場合もあります。ここでは、どのようにして無効にするかについて説明しています。

サブフォームの再表示を手動化する



1. サブフォームコントロールの**自動再表示**特性を *No* に設定します。
2. サブフォームコントロールの**コントロール名**特性を確認します。この例では、*DVD* となっています。

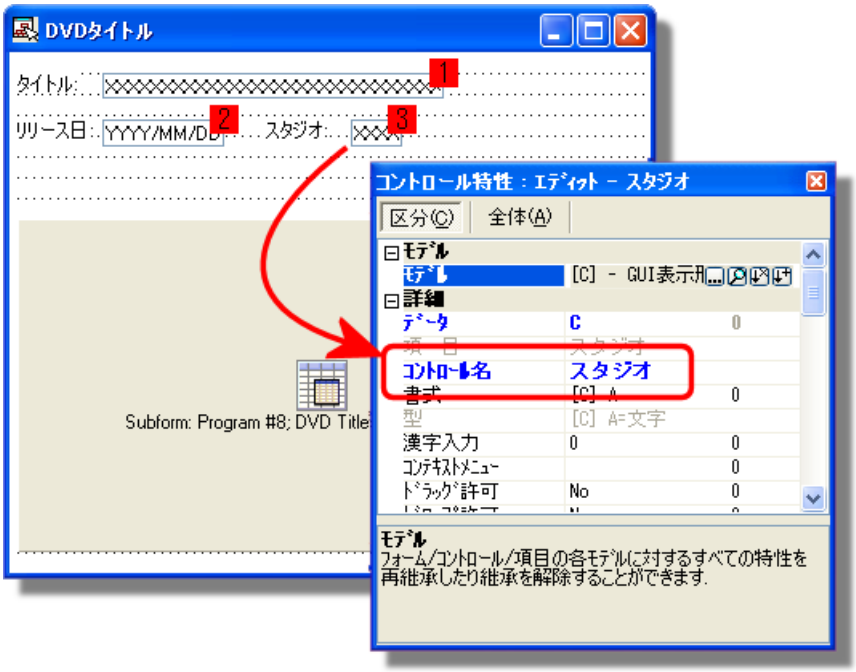


3. サブフォームを再表示したい場合に、実行されるユーザイベントを作成します。この例では、**e. 再表示**のユーザイベントを実行する**プッシュボタン**が定義されています。
4. **ロジックエディタ**で、イベントに対する**ロジックユニット**を作成します。
5. この**ロジックユニット**内で、**サブフォーム再表示**イベントを実行するように定義します。
6. **サブフォーム再表示**イベントへのパラメータとして、再表示したい**サブフォーム**のコントロール名を渡します。これで、ユーザがプッシュボタンを**クリック**すると、サブフォームは再表示されます。

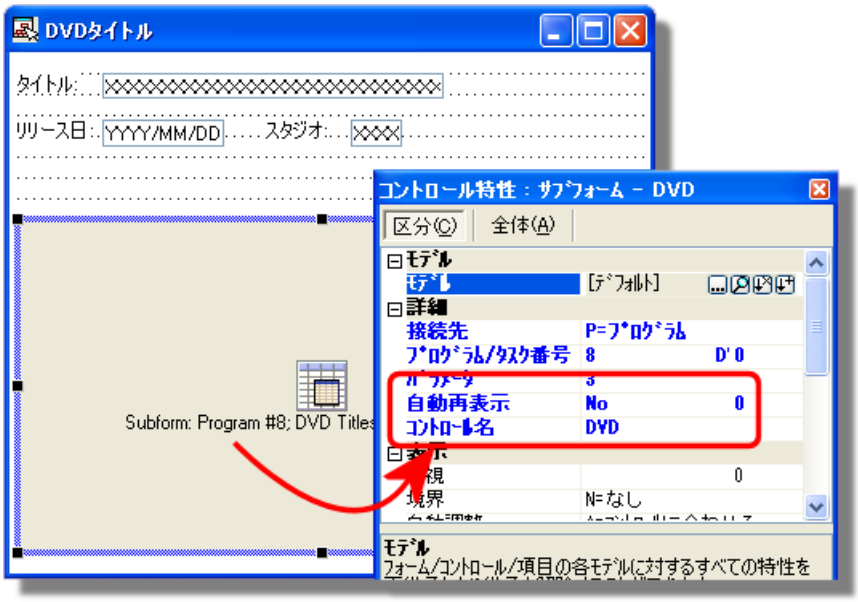
複数のパラメータをサブフォームに渡した場合、最後のパラメータを修正した時のみ再表示するには

通常、サブフォームに渡されるパラメータの値が変更されると再表示されます。しかし、リストに対する検索条件などのように複数のパラメータが指定されている場合、ユーザが条件値のどれかを変更するたびに、サブフォームは再表示されます。これは、ユーザの要求とは合っていないかもしれません。また、複雑な検索である場合、処理速度を落とす可能性もあります。

サブフォームの自動再表示特性を無効にする



1. 親タスク上で、コマンドパレットの TAB 順序を表示アイコン を押下して、TAB 順序が正しく設定されていることを確認します。
2. 最後の項目にコントロール名が設定されていることを確認します（この例では Studio になっています）。



3. サブフォームのコントロール特性で、自動再表示特性を *No* に設定します。

4. サブフォームコントロールのコントロール名を確認します。この例では、*DVD* となっています。

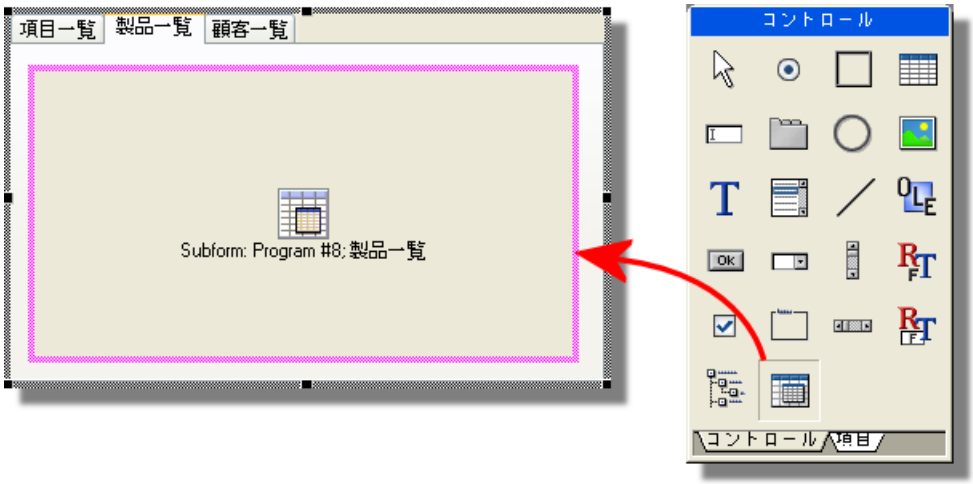
5. ロジックエディタで、最後の検索条件コントロールに対するコントロール後ロジックユニットを作成します。
6. イベント実行処理コマンドを作成し、サブフォーム再表示イベントを定義します。
7. このイベントへのパラメータとして、サブフォームのコントロール名（この例では **DVD Title**）を渡します。
   これで、ユーザが最後のコントロールを通過すると、サブタスクは再表示されます。

ヒント : *サブフォームの* **TAB で移動** *特性と組み合わせた場合、ユーザは条件に入った後で、サブフォームに正しく移るはずです。また、コントロール後が定義されたコントロールでは、* **CtrlGoTo()** *を持つアクション処理コマンドを追加することができます。これによってユーザがサブフォーム内のコントロールに正しくフォーカスを移すことができます。*

タブコントロールに配置したサブフォームの表示を制御するには

サブフォームを**タブ**コントロールに配置した場合、タブが選択された場合でのみ表示されます。

サブフォームをタブに配置する



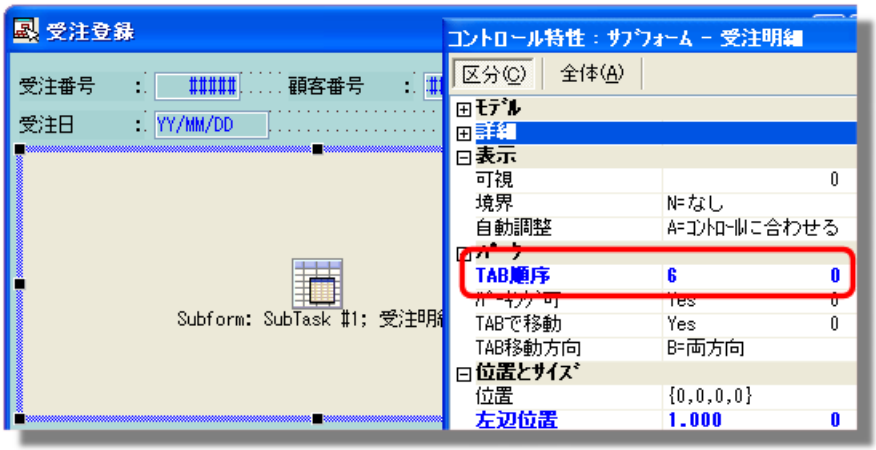
1. ***Ctrl+*** **クリック**で**タブ**コントロールを選択します。この場合、タブにリンクされているコントロールは選択されません。
2. 引き続き ***Enter*** キーを押下します。押下されるたびにタブが表示が切り替わります。
3. サブフォームを配置したいタブに切り替わったら、**サブフォーム**コントロールを**クリック**し、タブに**ドロップ**します。上図で示されるように、ピンク色で囲まれたように表示されます。
4. 通常の方法で、**サブフォーム**コントロールを設定します。

参照 : 第 3 章 :「コントロールモデルを使用して、フォームにコントロールを自動配置するには」（42 ページ）
第 5 章 :「コントロールをリンクする」（118 ページ）

親フォームのコントロールからサブフォームに Tab 入力できるようにするには

通常、Magic では、**TAB 順序**を左上か右下に順番に設定します。しかし、手動で **TAB 順序**を設定することもできます。

サブフォームコントロールの TAB 順序を設定する

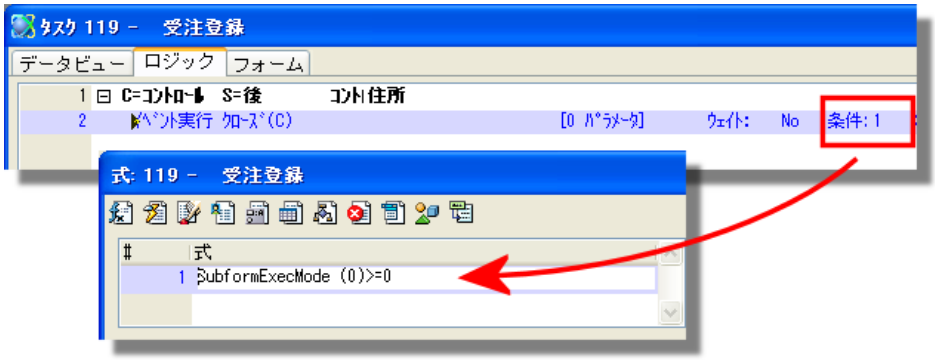


1. **コマンド**パレットの**自動 TAB 順序**アイコン  を押下して（**描画→表示順→ TAB 順序**）、**自動 TAB 順序**モードを解除します。（解除されていない場合）
2. **サブフォーム**以外のコントロールの **TAB 順序**を設定します。
3. **サブフォーム**コントロールの **TAB 順序**を、他のコントロールより大きい値に設定します。この例では、**TAB 順序**は受注日（5）からサブフォーム（6）に移ることになります。
4. **サブフォーム**コントロールの**パーキング可**特性と **TAB で移動**特性がデフォルト値（*Yes*）に設定されていることを確認します。

サブフォーム上のコントロールから親のコントロールに自動的に戻るには

ユーザがサブフォームに移動した場合に、そこから抜けられるようにする必要があります。コントロールイベントを使用することでこのようなことを簡単に実現することができます。しかし、サブフォームプログラムは **Tab** 移動で終了することのない１つのプログラムとして受け取られます。このため、終了する必要がある場合 **SubformExecMode()** 関数を使用して制御します。

サブフォームを終了するイベントを設定する



1. どのコントロールが最後にパークされるのかを決定し、そのコントロールにコントロール名が定義されていることを確認します。この例では、住所という名前が設定されています。
2. コントロール後ロジックユニットを作成し、そのコントロール名を指定します。
3. イベント実行処理コマンドを作成し、条件特性に以下の式を設定します。

   ```
   SubformExecMode(0)>=0
   ```

   タスクがサブフォームとして呼ばれていない場合、**SubformExecMode()** 関数は -1 を返します。戻り値がこれ以外の場合、サブフォームがどのように起動されたかの情報が取得できます。『リファレンスヘルプ』を参照してください。

最初にサブフォームタスクが起動されたときのみタスク前 / 後を実行させるには

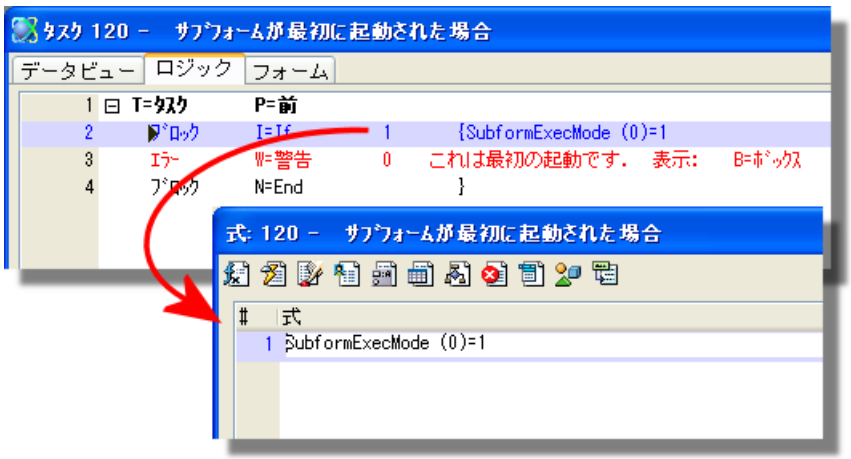
タスク開始時は常にタスク前が実行され、タスク終了時は常にタスク後が実行されます。しかしサブフォームを使用する場合、親タスクから最初に起動された場合に、内容を表示するために、タスク前／後を一度を実行します。しかし、親タスクが再表示した場合、再びユーザがサブタスクに入った場合も実行されます。このため、サブフォームタスクが最初に起動されたときのみ実行するように定義する場合は、**SubformExecMode()** 関数を使用する必要があります。

サブフォームタスクが最初に起動された場合、**SubformExecMode(0)** は 1 を返します。このため以下の条件式を使用します。

```
SubformExecMode(0) = 1
```

ブロック処理コマンドの条件にこの式を設定すると、サブフォームが最初に起動された場合のみ、ブロック内の処理が実行されます。

サブフォームが最初に起動された場合のみ実行するブロックを定義する



1. 修正中のロジックユニットで詳細行を追加します（*F4* または、編集→行作成）。
2. B を入力するかプルダウンメニューを使用して、ブロック処理コマンドを選択します。
3. *ブロック If* と *ブロック End* の２行が追加されます。*If* の後のカラムに移動します。
4. ズーム（*F5* または、ダブルクリック）して式エディタを開き、以下の式を入力します。

   ```
   SubformExecMode(0)=1
   ```

5. *Enter* を押下して式を確定しロジックエディタに戻ります。

これで、サブフォームが最初に起動されたときのみブロックは実行されます。

ユーザがサブフォームに入ると常にタスク前 / 後が実行されるようにするには

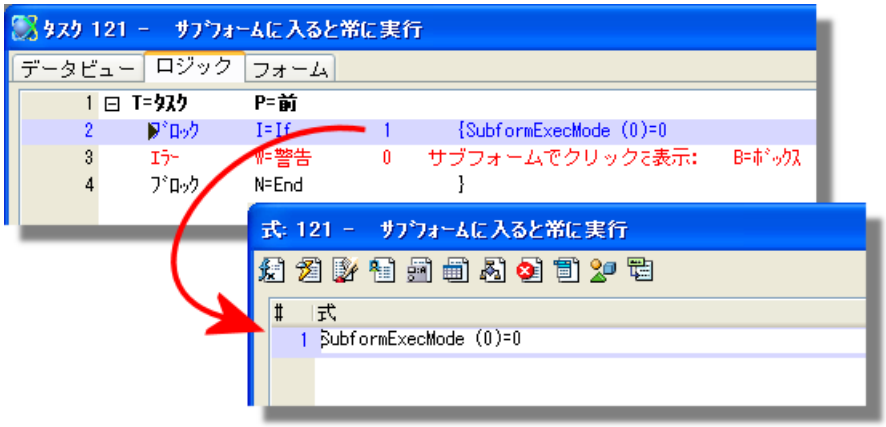
タスク開始時は常にタスク前が実行され、タスク終了時は常にタスク後が実行されます。しかし、サブフォームでは、どのように起動されたかによって処理内容を制御するためにブロック処理コマンドを利用することができます。この場合、**SubformExecMode()** 関数を使用します。

ユーザの操作（*Tab* 移動やマウスクリックなど）でサブフォームタスクに入った場合、**SubformExecMode(*0*)** 関数は「0」を返します。このため以下の条件式で判別できます。

```
SubformExecMode(0) = 0
```

ブロック処理コマンドの条件にこの式を設定すると、ユーザの操作によってサブフォームが起動された場合のみ、ブロック内の処理が実行されます。

ユーザがサブフォームに入るときにのみ実行するブロックを定義する



1. 修正中のロジックユニットで詳細行を追加します（*F4* または、編集→行作成）。
2. *B* を入力するかプルダウンメニューを使用して、ブロック処理コマンドを選択します。
3. ***ブロック If*** と ***ブロック End*** の２行が追加されます。*If* の後のカラムに移動します。
4. ズーム（*F5* または、ダブルクリック）して式エディタを開き、以下の式を入力します。
   ```
   SubformExecMode(0)=0
   ```
5. *Enter* を押下して式を確定しロジックエディタに戻ります。

これで、ユーザがサブフォームに入った場合のみブロックは実行されます。

サブフォームが再表示されたときにサブタスクのタスク前 / 後を実行させるには

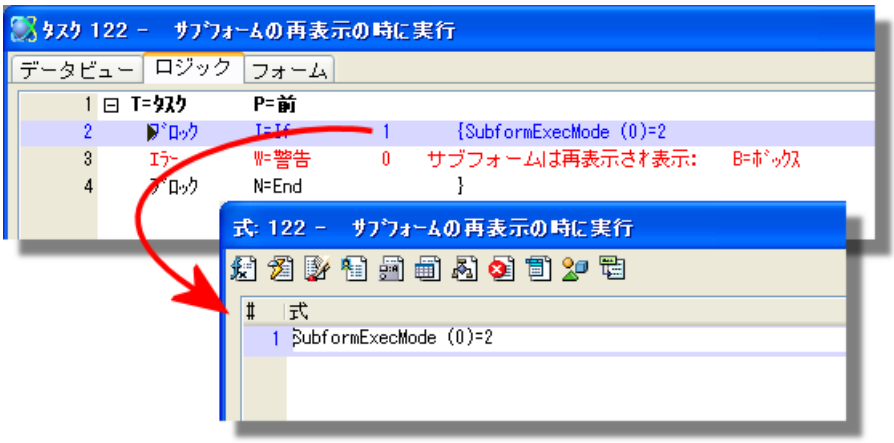
タスク開始時は常にタスク前が実行され、タスク終了時は常にタスク後が実行されます。しかし、サブフォームでは、どのように起動されたかによって処理内容を制御するためにブロック処理コマンドを利用することができます。この場合、**SubformExecMode()** 関数を使用します。

自動再表示や手動のサブフォームを再表示イベントなどで再表示処理を行うことによりサブフォームタスクを実行させた場合、**SubformExecMode(0)** は 2 を返します。このため以下の条件式で判別できます。

```
SubformExecMode(0) = 2
```

ブロック処理コマンドの条件にこの式を設定すると、サブフォームが再表示した場合のみ、ブロック内の処理が実行されます。

サブフォームが再表示されたときにのみ実行するブロックを定義する



1. 修正中のロジックユニットで詳細行を追加します（*F4* または、編集→行作成）。
2. *B* を入力するかプルダウンメニューを使用して、ブロック処理コマンドを選択します。
3. ***ブロック If*** と ***ブロック End*** の２行が追加されます。*If* の後のカラムに移動します。
4. ズーム（*F5* または、*ダブルクリック*）して式エディタを開き、以下の式を入力します。

   ```
   SubformExecMode(0)=2
   ```

5. *Enter* を押下して式を確定しロジックエディタに戻ります。

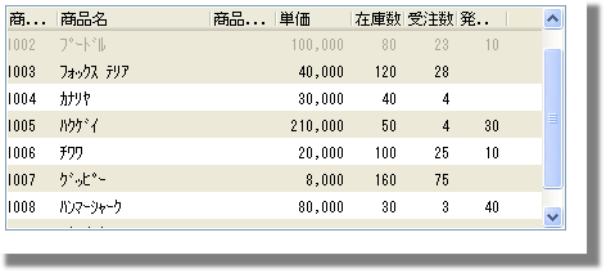
これで、サブフォームが再表示された場合のみブロックは実行されます。

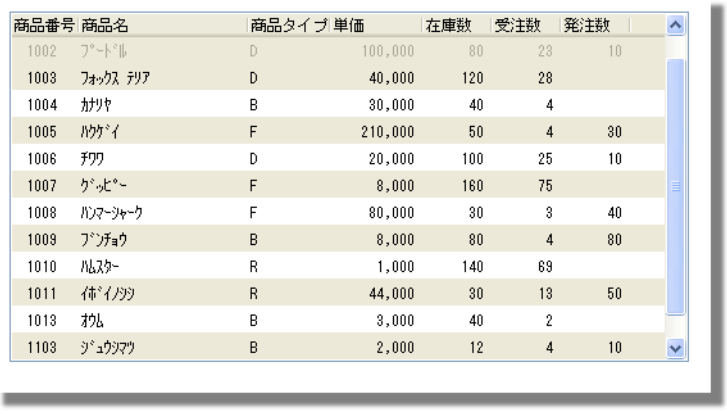
自動調整オプションをどのように選択するか

サブフォームコントロールの自動調整特性は設定内容によって動作が異なります。これにより、どれを選択すればいいか決めかねる場合があります。以下の例は、３つのオプションを選択した場合どのように動作するかを説明したものです例です。３つの場合のサブフォームコントロールはまったく同じサイズになっています。

自動調整オプションの結果

N= なし	
	***なし***：サブフォームタスクのフォームは、サブフォームコントロールのサイズ内に収まる内容のみ表示されます。呼び出されたフォームが大きすぎる場合、切り取られ、スクロールバーが表示されます。例えば、サブフォームコントロールの幅が 80 ユニットに設定され、呼び出されたフォームの幅が 88 ユニットに設定されている場合、実行時は 80 ユニットのサイズでのみ表示されます。 サブフォームコントロールに位置特性が設定され、フォームのサイズが変更される場合、サブフォームタスクはより広い範囲を表示することができますが、サブフォームタスク内のコントロールは変更されません。

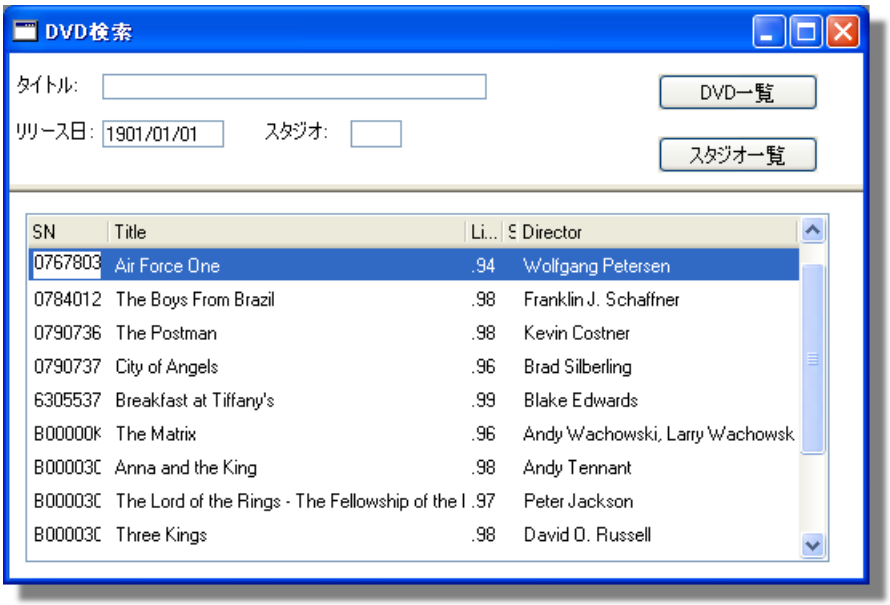
A= コントロールに合わせる	
	***コントロールに合わせる***：サブフォームタスクはコントロールのサイズに適合されます。ユーザがサブフォームタスクのサイズを変更したように表示されます。従って、サブフォームタスクで位置特性が設定されている場合、コントロールのサイズはそれに応じて伸びたり、縮んだりします。

C= 起動先フォームに合わせる	
	***起動先フォーム合わせる***：サブフォームコントロールは、サブフォームタスクのフォームサイズに合わせて大きさが変更されます。この例では、サブフォームコントロールが上記の２つの例と同じサイズですが、サブフォームコントロールは、サブタスクフォーム全体に収まるように拡大されて表示しています。この例では、サブフォームコントロールの位置特性は全く影響しません。

第 9 章： 分割

分割機能を利用して２つのフォームを表示するには

表示画面を設計する場合、画面上のコントロールをどのように配置するかは、重要な問題になります。例えば、別の部分をより多く表示させたいために***邪魔にならないところ***にフォームの一部を移動したいと考える場合があります。このようにフォームの一部を移動させる手段を与える簡単な方法として、***フォームの分割機能***があります。



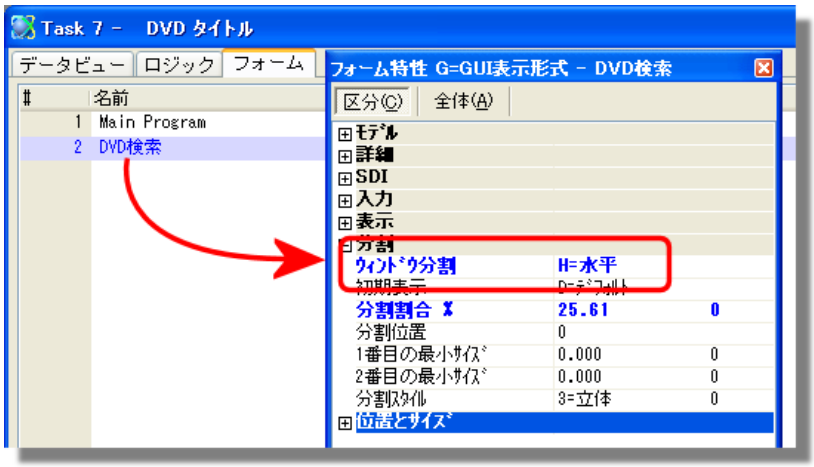
分割フォームは、１つのフォームのように動作する２個別のフォームです。上の例では、**分割線**をフォームの中央で移動させるにより上段と下段の表示エリアの表示比率エリアを変更することができます。

また、フォームの下段に表示されているタスクを変更することができます。複数のタスクを呼び出すことができます。この例では、ユーザが**スタジオ一覧**ボタンを**クリック**すると、フォームの下段では DVD の一覧を表示する代わりに、スタジオの一覧が表示されます。

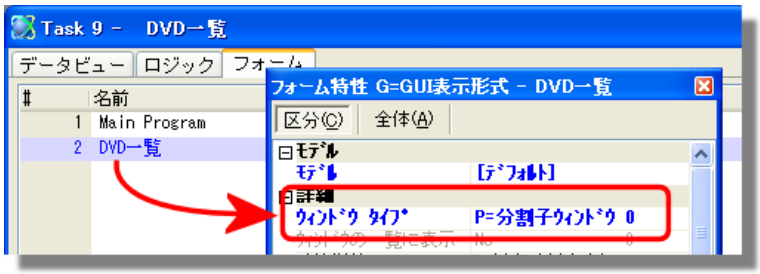
注： メインフォームの一部を個別のフォームに表示させるという点において、**分割フォーム**と**サブフォーム**には機能的に共通する部分があります。しかし、サブフォームはメインフォーム内に位置された四角形の領域に表示されるのに対して、分割フォームは、分割線によってメインフォームを２つに分割されて表示されるという違いがあります。また、分割フォームはエンドユーザによってサイズを変更させることができますが、サブフォームはメインフォームのサイズに比例して変更することのみ可能です。

分割フォームを設定する

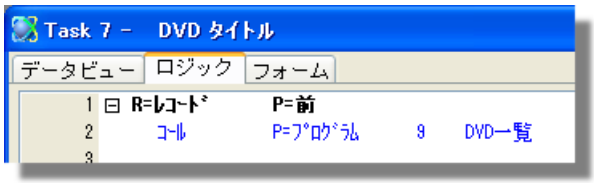
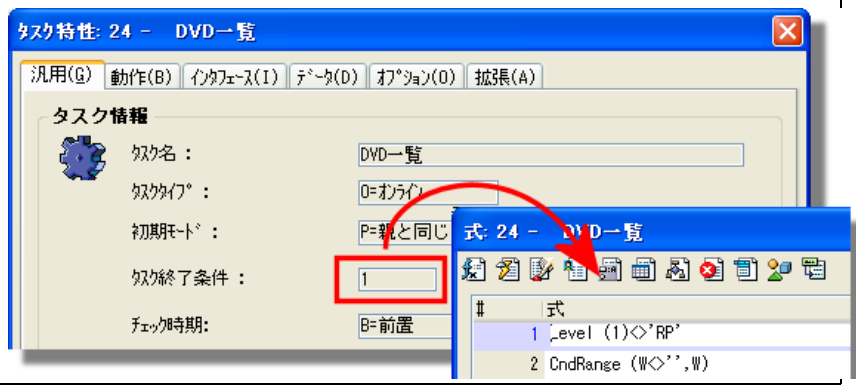
1. 親タスクのフォーム特性のウィンドウ分割特性を選択し、*V= 垂直* または *H= 水平* に設定します（この特性を有効にするため、フォームをいったん閉じる必要があります）。

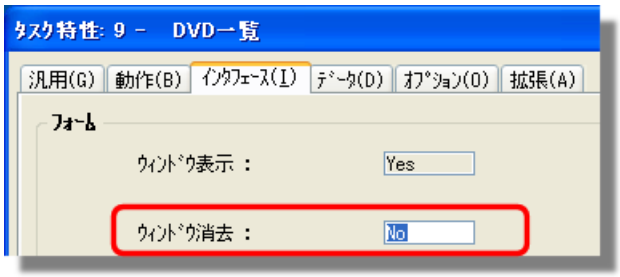


2. 子タスクを開き、フォーム特性のウィンドウタイプ特性を選択し、*D= 分割子ウィンド* を設定します（ウィンドウタイプは式で指定することもできます。この場合、別の目的で再利用することができます）。



3. 親タスクで、明示的に子タスクを呼び出す必要があります。例えば、ボタンがクリックされると、イベントロジックユニットが起動されるようにします。親タスクが起動されると同時にサブタスクの初期設定が必要な場合、レコード前でタスクを起動させます。以下は、このような処理をどのように実現するかを説明しています。

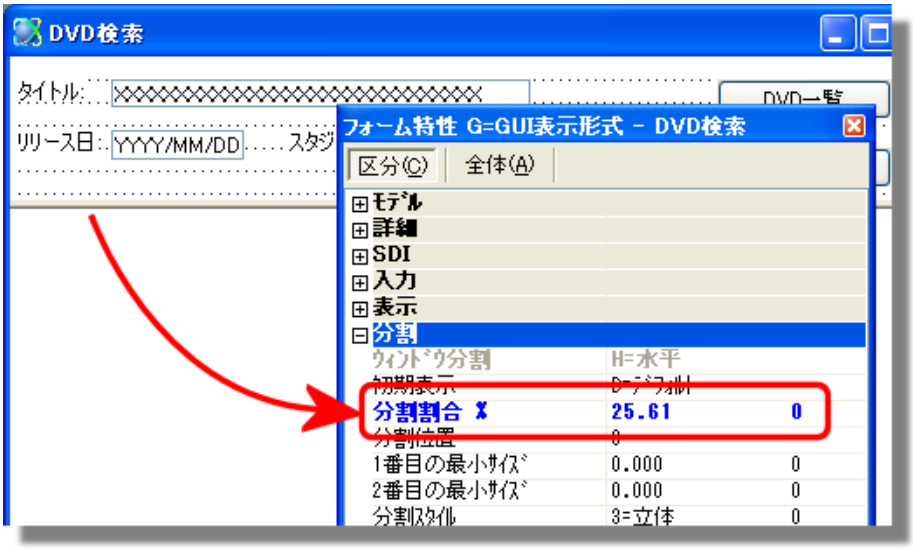
親タスクのレコード前で子タスクを呼び出すように定義します。	
子タスクのタスク特性（タスク→特性）を開き、タスク終了条件特性に条件式として *Level(1)='RP'* を設定します。 これによって、親タスク（のレコード前）から最初に呼び出された場合のみ終了するようになります。	

また、子タスクの**タスク特性**（**タスク→インタフェース**）の**ウィンドウ消去**特性を ***No*** に設定します。	

4. 分割フォームで**位置**特性を使用することで、分割ウィンドウの移動によるタスクのサイズを決めることができます。

参照 : 第 8 章 :「サブフォーム」（175 ページ）
「分割画面の初期設定を行うには」（190 ページ）

分割画面の初期設定を行うには

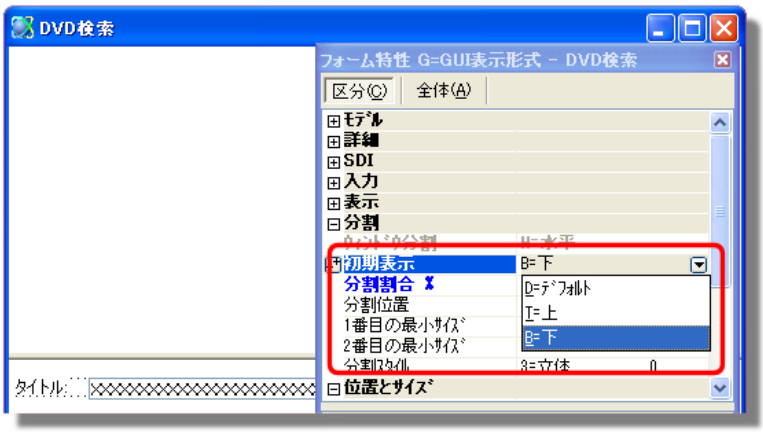


ユーザは分割フォームのサイズを変更することができますが、開発時に分割画面の初期値を設定しておくことができます。これにはいくつかの方法があります。

- フォーム特性の分割割合 % 特性を選択して手動でサイズ（%）を設定する。
- 分割線をマウスでドラッグし、必要な場所に移動する。分割割合 % 特性は、移動内容に応じて更新されます。
- 分割割合 % 特性を式で指定する。これによって実行時に動的に指定できるようになります。

親タスク表示を分割画面の別の側に表示させるように設定するには

分割画面を設定すると、メイン（すなわち親タスク）の表示位置のデフォルトは自動的に決定されます。しかし、この表示位置を任意に変更させることもできます。



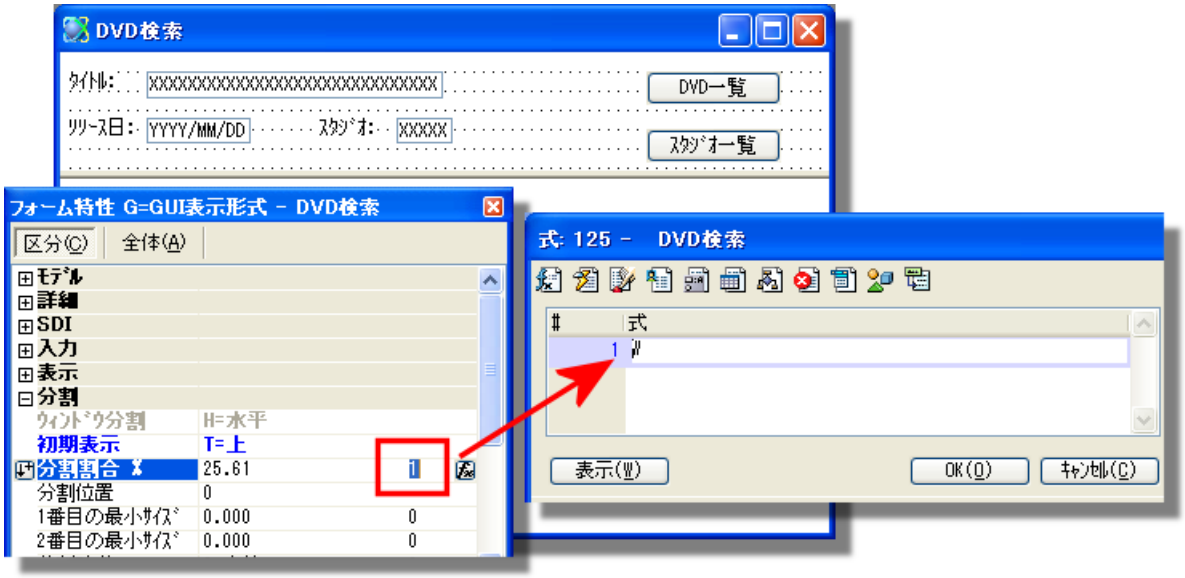
親タスクの表示位置を設定する

1. フォーム特性の初期表示特性を選択します。
2. 親タスクを表示させる位置を選択します。この例では、*B= 下*が選択されたことにより親タスクは下段に表示されるようになります。

分割画面の割合を動的に指定するには

通常フォームを作成する場合、手動で**分割割合**を設定することになりますが、式を使用して動的に設定することもできます。この例では、データ項目を使用して**分割割合**を変更しています。

割合を動的に設定する



1. **フォーム特性**の**分割割合 %** 特性を選択します。
2. **式**特性で**ズーム**（または *fx* ボタンを**クリック**）して**式エディタ**を開き、必要な式を定義します。

注： **分割割合**特性は、タスクに再入力したり、式で使用されているデータ項目が更新されたり、別のレコードに移動した場合に再評価されます。

現在の分割割合の値を取得するには

分割割合はユーザによって手動で変更したり、式によって変更したりすることができます。**SplitterOffset()** 関数を使用することで、現在の**分割割合**の値を取得することができます。

SplitterOffset(*0*) は % で返し、**SplitterOffset(*1*)** は絶対値で返します。絶対値は、使用するフォームの**寸法単位**（ダイアログ、センチ、またはインチ）にもとづいた値です。

エンドユーザによって実行時に設定された分割位置を維持するには

ユーザが Magic のフォームをカスタマイズする方法にはいくつかあります。セッション間でユーザのカスタマイズ内容が維持されることができれば、非常に便利です。

Magic は分割位置などのユーザ設定内容を自動的に保持することができます。この場合、フォーム特性の**フォーム状態 ID** 特性を設定する必要があります。**フォーム状態 ID** 特性には文字列を入力します。直接入力するか、実行時に評価される式を定義します。Magic は、エンドユーザのフォームの設定内容を保存する場合にこの特性を使用します。

この特性は、以下のように動作します。

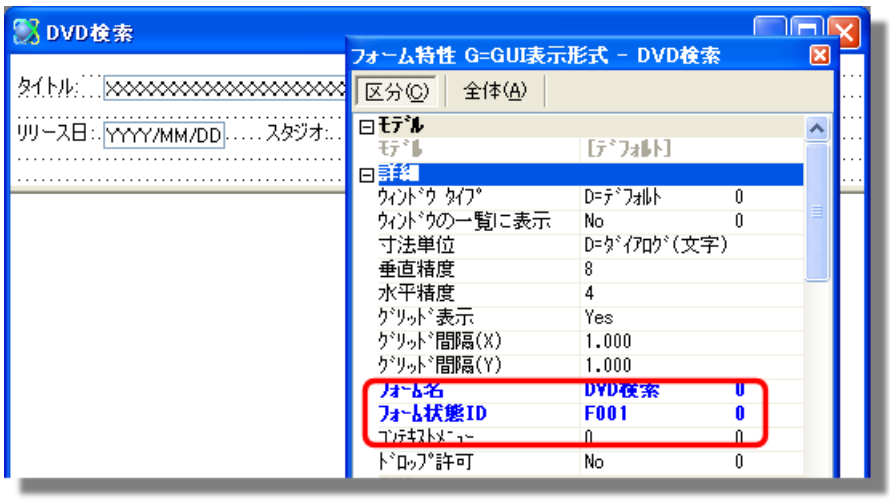
- **フォーム状態 ID** 特性が空白の場合、Magic はユーザ情報を保存しません。
- **フォーム状態 ID** 特性が空白でない場合、この文字列を使用してフォームの情報を保存します。

従って、各フォーム毎にユニークな ID が定義されている場合、各フォームの状態がユーザ毎に保存されることになります。これは、要求された機能を実現する最も簡単な方法となります。

- ２つのフォームが同じ ID を共有している場合、フォーム情報も共有されます。フォーム A とフォーム B が ***F001*** という**フォーム状態 ID** を共有し、ユーザがフォーム A を 75% の分割状態に設定した場合、ユーザがフォーム B を開くと、そのフォームも 75% の分割状態で表示されます。
- １つのフォームに２つ以上の ID を割り当てることもできます。例えば、フォーム C がある目的のために使用され、その場合は ***F001A*** という ID が定義されているものとします。しかし、別の目的で ***F001B*** という ID を使用する場合も発生するということです。これによって、エンドユーザがフォームを使用している目的に応じて２つの異なる分割割合を設定することができます。

注： **フォーム状態 ID** は、フォームの**寸法単位**や**幅**などの状態を保存する場合にも使用できます。

フォーム状態 ID を設定する



1. 親タスクのメインフォームを開きます。
2. フォーム特性の**フォーム状態 ID** 特性を選択し、ユニークな文字列（例えば、上図の場合 F001）を入力します。式で入力する場合は、右側の**式**特性で**ズーム**して、**式エディタ**を開き式を入力します。

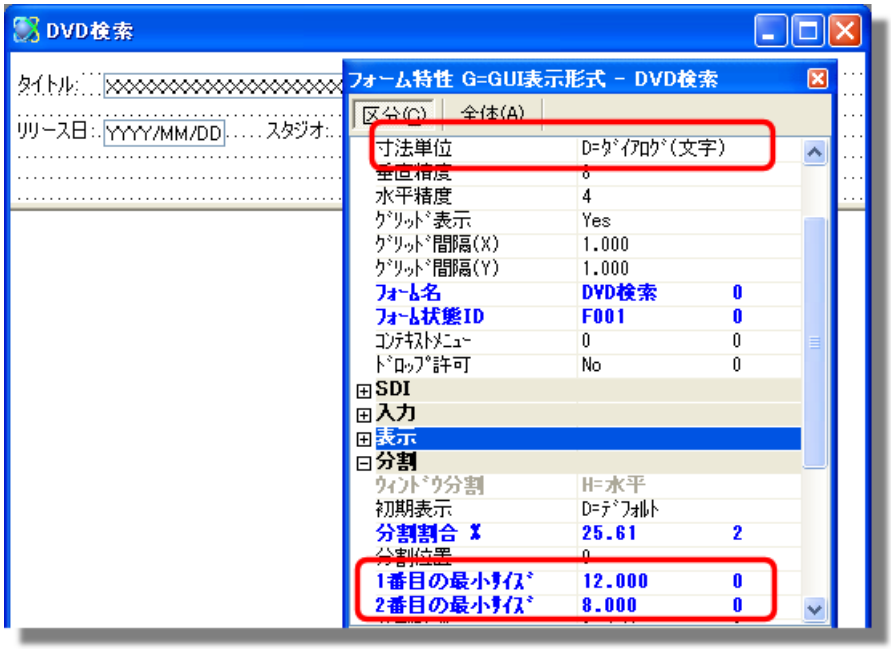
ヒント： *フォーム情報は実行時にのみ保存されます。開発時は保存されません。従って、動作を確認したい場合は、キャビネットファイルを作成し、実行エンジンでアプリケーションを実行してください。*

分割領域を最小サイズに設定するには

デフォルトでは、エンドユーザは 1 つの領域がフォーム全体を表示するようにサイズを変更することができます。あるサイズ以下には変更できないように、一番目（親）のフォームや二番目（子）のフォームの最小サイズを設定することができます。

最小サイズの値は絶対値（% ではありません）で設定します。フォームが使用している**寸法単位**を基に決定されます。この例では、***C= ダイアログ***が設定されています。

最小サイズを設定する



1. 親タスクのメインフォームを開きます。
2. **フォーム特性**を開きます。
3. 親タスクのフォームの最小サイズを設定する場合は、**１番目の最小サイズ**特性を設定します。
4. 子タスクのフォームの最小サイズを設定する場合は、**２番目の最小サイズ**特性を設定します。.

上の例では、**寸法単位**が ***D= ダイアログ***に設定されているため、親フォーム（上段に表示）は 12 文字（半角）分より小さくすることができません。また、子フォーム（下段に表示）は 8 文字（半角）分より小さくすることができません。

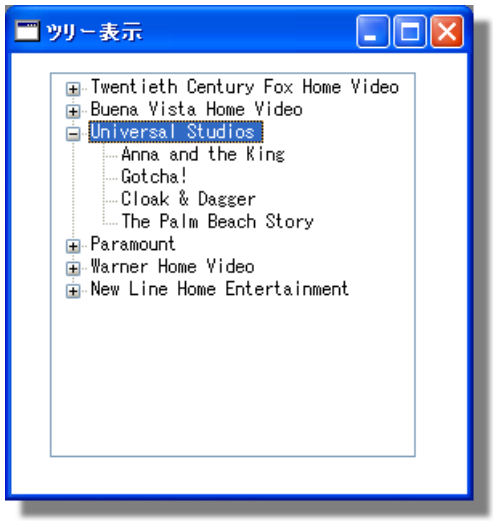
注： 最小サイズを式で設定することもできます。

.

第 10 章 : ツリーコントロール

データをツリー形式で表示するには

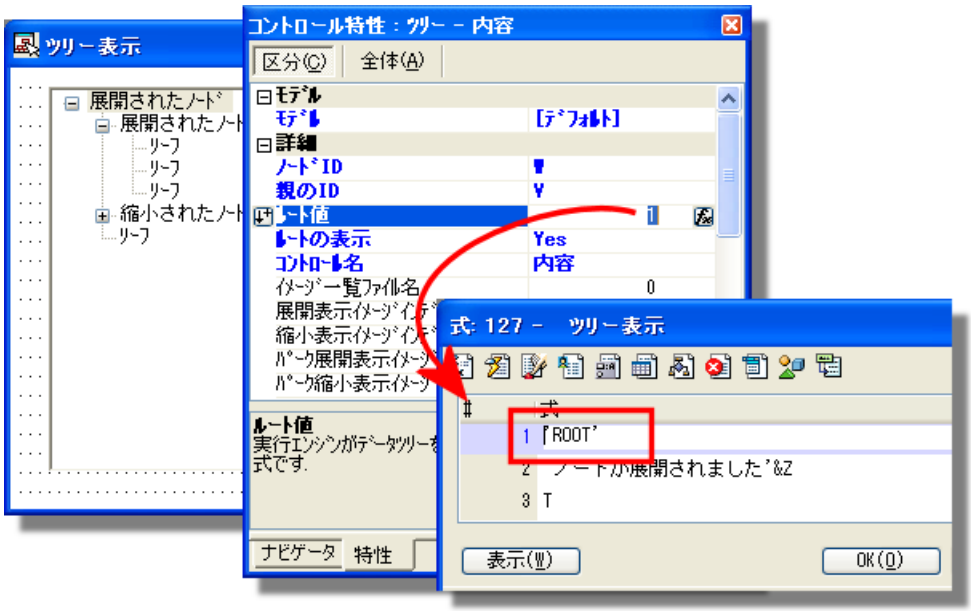
ツリー形式は、データを一覧表示する場合に便利です。この形式は Windows 環境では一般的で、ユーザにも見慣れている表示です。



Magic には、必要に応じてこのような表示を行うためのツリーコントロールがあります。開発者が行わなければいけないことは、階層構造に定義されたデータソース（メモリテーブルで十分です）にデータを格納し、必要なオプションを設定することです。

ここでは、基本的な作業手順を説明しています。

ツリーコントロールを定義する



1. ツリーに対応した書式でデータを保存します。第 10 章：「ツリー表示させるための階層的なデータソースを定義するには」（197 ページ）を参照してください。
2. データビューエディタでメインソースとしてツリー用のデータソースを定義し、タスクに必要なカラム（ノード ID、親 ID など）を定義します。

3. コントロールパレットからツリーコントロールを選択し、フォームにドロップします。サイズはに必要に応じて変更します。
4. ツリーのコントロール特性を開き、以下の特性値を設定します。
   - ノード ID：レコードのキーとなるカラムを指定します。
   - 親 ID：このレコードの親のレコードを示す ID が格納されたカラムを設定します。
   - ルート値：ツリーの最上位のレコードを示す値を指定します（この例では、ROOT です）。
   - 表示項目：ツリーの上で値が表示される項目を指定します。

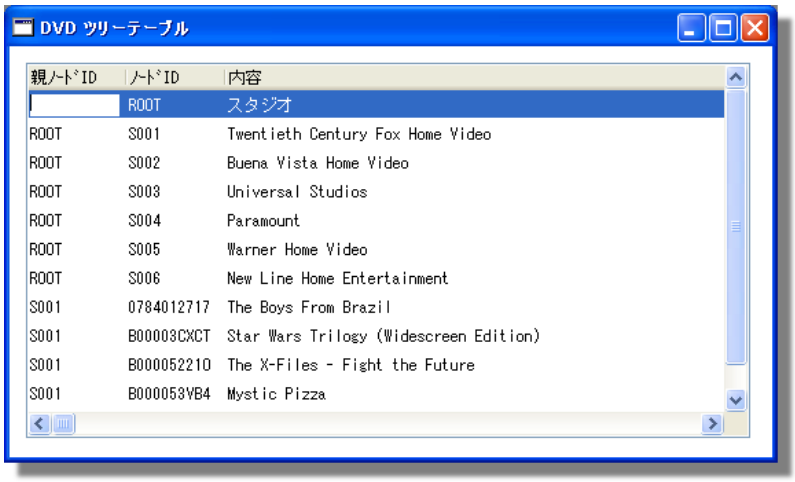
データの書式が正しく設定されていれば、テーブルとして表示されます。実行時に、テーブルの外観やどのように動作するかを制御するためのオプションがあります。これらの内容については以下の説明を参照してください。

参照： 「ツリー表示させるための階層的なデータソースを定義するには」（197 ページ）
「ツリーのノードにアイコンを設定するには」（198 ページ）
「ツリーを表示する際にノードを自動的に展開させるには」（203 ページ）

ツリー表示させるための階層的なデータソースを定義するには

ツリーを表示させるためには、正しく構造化されたデータを定義する必要があります。このためには、一時的にデータを保持するためのメモリテーブルを作成する必要があります。

構造を実現する上で２つの項目（*ノードID* と *親ID*）が必要です。**ノードID** は特定のレコードを識別するもので、レコード毎にのユニークな ID が定義されている必要があります。**親 ID** はこのレコードの親を識別するためのものです。



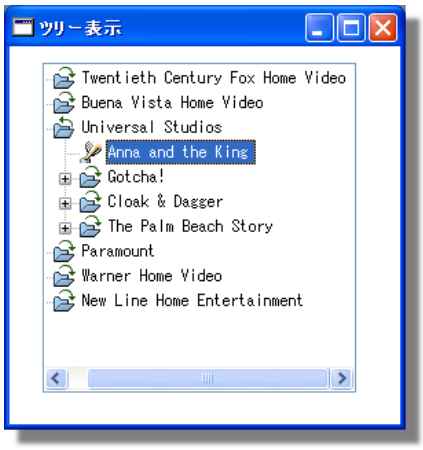
再帰

各レコードの**ノードID** と**親 ID** には異なる値が設定されていなければなりません。異なっていない場合、1 つの枝から何回でも開くことのできるができる再帰的なツリーになってしまいます。例えば、**ノードID**：S002 を持つレコードは、S002 の**親 ID** を持つことができないということです。また、**親 ID** と**ノードID** の両方を空白にできるのは、ルートノードのみになります。

ヒント：*ツリーを複数の異なるデータソースから構成することは可能です。この例では、**スタジオ**テーブルと DVD テーブルのデータを使用しています。これらのデータソースはそれぞれが非常に異なるデータが定義されているため、１つのツリーとして表示させるため、最低限のカラムのみ使用するようにしています。より多くの情報を表示したい場合は、一時テーブルにコピーして使用しないで、オリジナルのデータソースとリンクして使用してください。*

ツリーのノードにアイコンを設定するには

アイコン未設定



アイコン設定済み

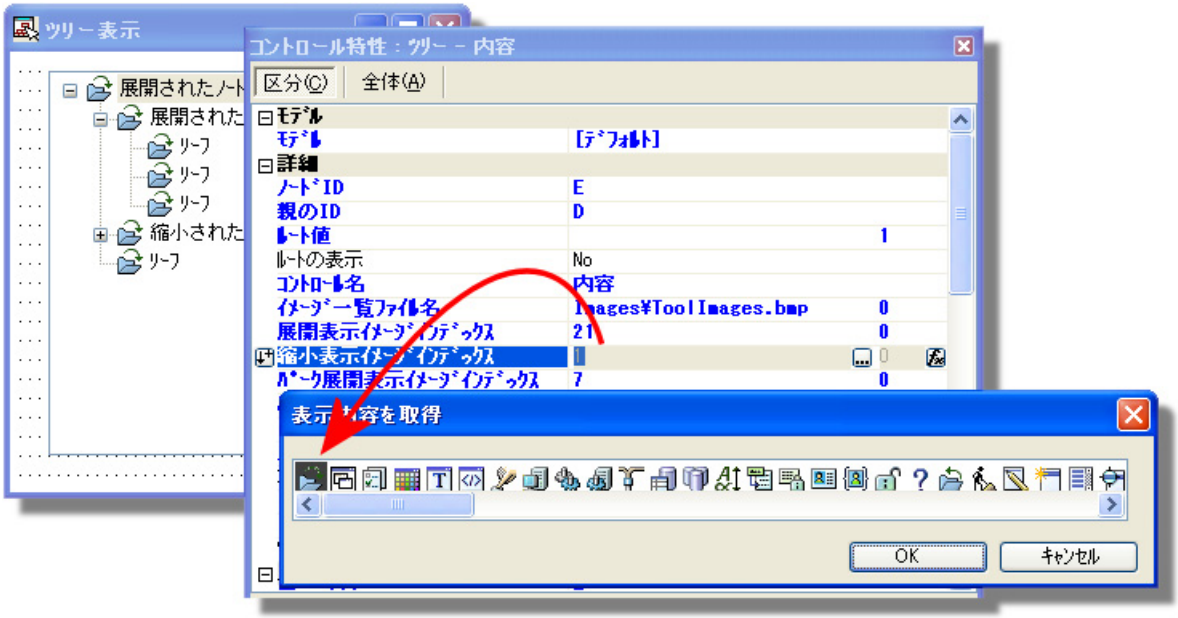
ツリーコントロールにアイコンを表示させることができます。

ツリーには以下の４つの状態があります。

- **展開表示イメージ**：ノードが完全に展開されているか、子ノードが存在しない場合
- **縮小表示イメージ**：ノードが縮小されている場合
- **パーク展開表示イメージ**：カーソルが展開されたノード上でパークされている場合
- **パーク縮小表示イメージ**：カーソルが縮小したノード上でパークされている場合

これらのアイコンは各々、番号を指定することによって設定されます（これはアイコンファイルにおけるインデックスです）。インデックスが 0 に設定されている場合、イメージは表示されません。

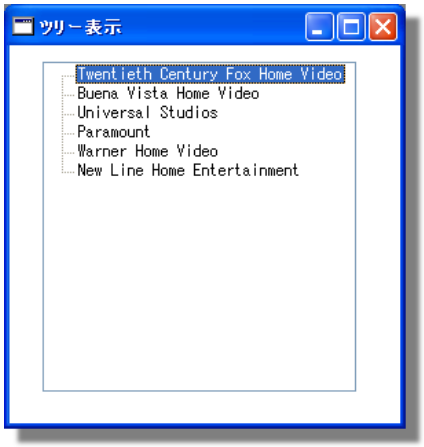
アイコンを使用する



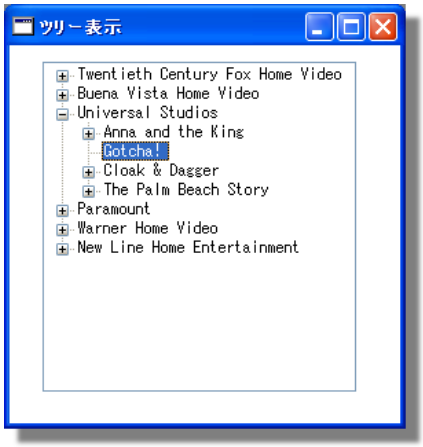
必要条件： アイコンファイルを事前に用意しておく必要があります。アイコンファイルは、必要なアイコンが同じサイズで水平方向に並んだ状態で定義されているビットマップファイルです。どのアイコンを使用するかは、番号で指定されます。これらは、汎用的なイメージファイルを購入したり、独自に作成することで入手してください。

1. ツリーコントロールを選択します。
2. コントロール特性の**イメージ一覧ファイル名**特性を選択します。
3. ズームしてイメージファイルを選択するか、直接ファイル名を入力します。
4. ４つの状態の**イメージインデックス**特性でズームしてアイコンを選択します。

展開 / 縮小ボタンを表示 / 非表示するには



ボタンの表示 = *No*



ボタンの表示 = *Yes*

ツリーの各ノードは、デフォルトで隣に小さなプラス（+）またはマイナス（−）のボタンが表示されています。+をクリックするとノードが展開表示され、−をクリックすると縮小表示されます。

ボタンの表示特性を使用することで、これらのボタンの表示／非表示を切り替えることができます。

展開 / 縮小ボタンの表示 / 非表示を切り替える

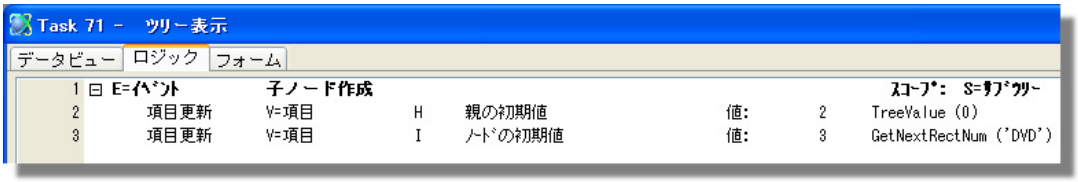
1. ツリーコントロールを選択します。
2. コントロール特性の**ボタンの表示**特性を選択します。
3. ボタンを表示する場合は *Yes*、そうでない場合は *No* を設定します。

注： **ボタンの表示**特性を *Yes* に設定すると、ツリーが最初に表示された場合、ツリーのリーフ（サブノードがないノード）は、+がデフォルトで表示されます。このような状態を回避するには、**ノードの事前読み込み**特性を *Yes* に設定してください。「子ノードを持つノードのみ展開ボタンを表示させるには」（205 ページ）も参照してください。

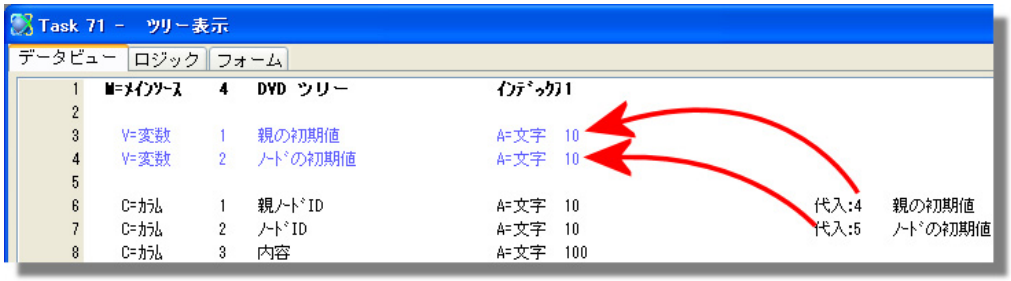
実行時にノードを追加するには

ツリーが表示されている場合、その機能の多くはテーブルコントロールの動作と同じです。しかし、ツリーではデータは平面的に表示されません。従って、ユーザがレコードを追加した場合、そのレコードは兄弟関係のノードか、子ノードになります。どちらの場合も、新しいノードを作成する際の親ノードと現在のノードの初期設定が正しく行われなければなりません。

子ノードを作成する



1. 子ノード作成イベントを実行するプッシュボタンを定義します。プッシュボタンのラベルは、子ノード作成（アプリケーションに合わせて任意に設定可）に設定します。子ノード作成はプルダウンメニューにはデフォルトで存在しないので、実行時のオプションメニューに表示させたい場合は、追加する必要があります。



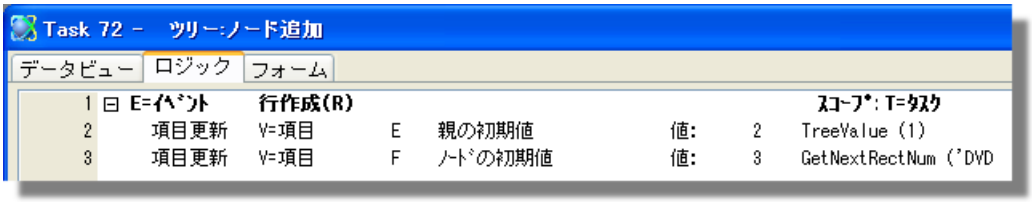
2. データビューエディタ内で、レコードが作成された場合に**親 ID** と**ノード ID** の初期値を設定するための変数項目を作成します。
3. 子ノード作成イベントに対するロジックユニットを作成し、伝播特性を *Yes* に設定します。
4. ロジックユニット内で、新規レコードが子ノードになるように、適切に項目を更新します。これは、**親 ID** の項目には現在の項目の**ノード ID** の値が設定される必要があることを意味しています。ID は、**TreeValue(*1*)** 関数を使用することで自動的に取得することができます。
   また、ユニークな値を**ノード ID** として初期設定する必要があります。この例では、自動的にユニークなキーを取得するために作成した**ユーザ定義関数**を使用しています。

   これで、新規ノードは適切に初期設定され、ユーザによって必要なデータを入力できるようになります。

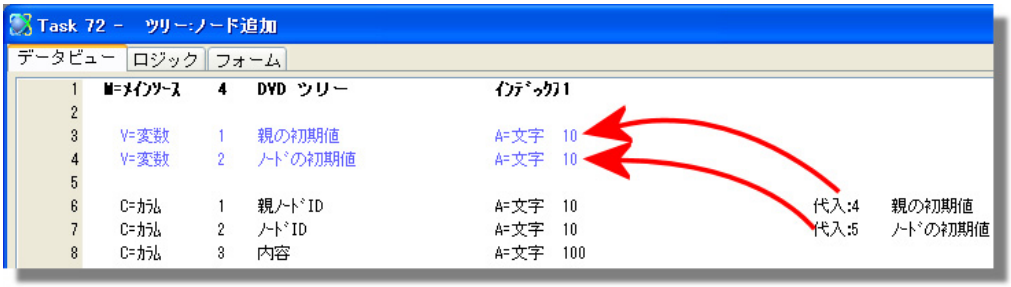
実行時に兄弟関係のノードを追加するには

ツリーが表示されている場合、その機能の多くはテーブルコントロールの動作と同じです。しかし、ツリーではデータは平面的に表示されません。従って、ユーザがレコードを追加した場合、そのレコードは兄弟関係のノードか、子ノードになります。どちらの場合も、新しいノードを作成する際の親ノードと現在のノードの初期設定が正しく行われなければなりません。

兄弟関係のノードを作成する



1. 行作成イベントを実行するプッシュボタンを定義します。プッシュボタンのラベルは、兄弟ノード作成（アプリケーションに合わせて任意に設定可）に設定します。*F4* の押下（または、編集→行作成）でイベントを発生させることもできますが、エンドユーザは気がつかない可能性があります。



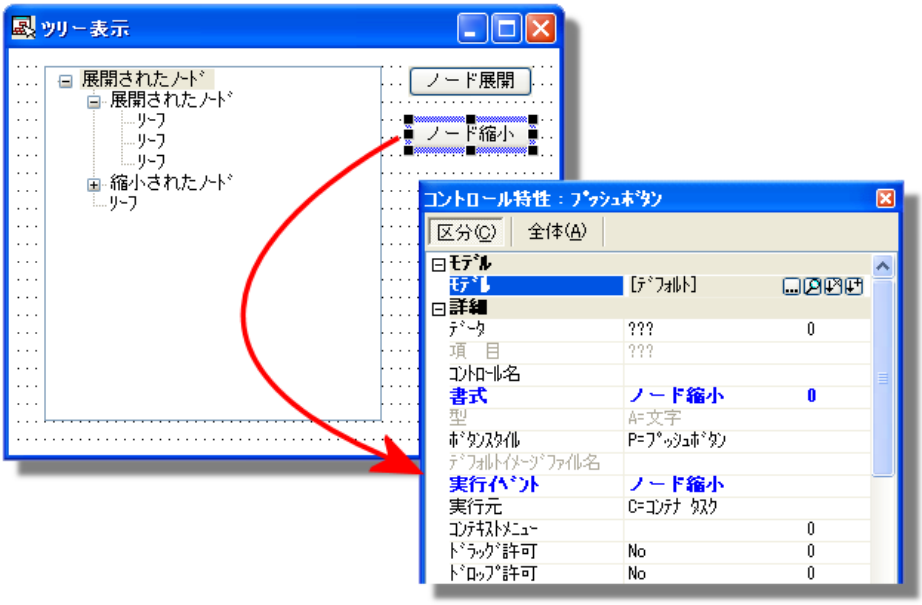
2. データビューエディタ内で、レコードが作成された場合に**親 ID** と**ノード ID** の初期値を設定するための変数項目を作成します。
3. 行作成イベントに対するロジックユニットを作成し、伝播特性を *Yes* に設定します。
4. ロジックユニット内で、新規レコードが兄弟ノードになるように、適切に項目を更新します。これは、**親 ID** の項目には現在の項目の**ノード ID** の値が設定される必要があることを意味しています。ID は、**TreeValue(*1*)** 関数を使用することで自動的に取得することができます。
   また、ユニークな値を**ノード ID** として初期設定する必要があります。この例では、自動的にユニークなキーを取得するために作成した**ユーザ定義関数**を使用しています。

   これで、新規ノードは適切に初期設定され、ユーザによって必要なデータを入力できるようになります。

実行時にツリーノードを明示的に展開 / 縮小させるには

ユーザが、ツリーのノードを展開したり縮小する場合、＋表示や－表示をクリックします。しかし、イベントを実行することで明示的にノードを展開したり縮小する場合は、ノード展開 / ノード縮小イベントを使用します。

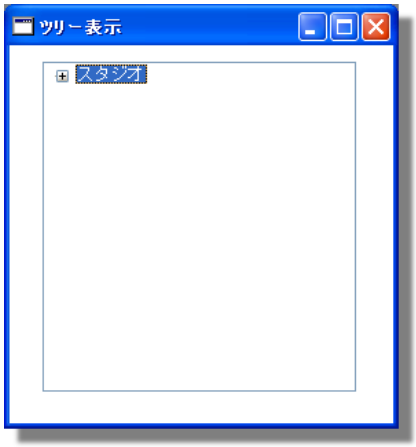
ノード展開 / ノード縮小イベントを実行する



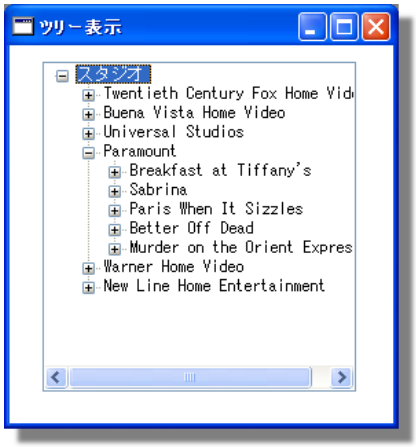
ロジックユニットにイベント実行処理コマンドを定義したり、プッシュボタンにイベントを割り当てることでこれらのイベントを他のイベントと同じように実行させることができます。

どちらのイベントも、ユーザが現在パークしているノードで実行されます。

ツリーを表示する際にノードを自動的に展開させるには



自動展開 = *No*



自動展開 には式が定義され、ルートノードと１つのノードが展開されるように設定されています。

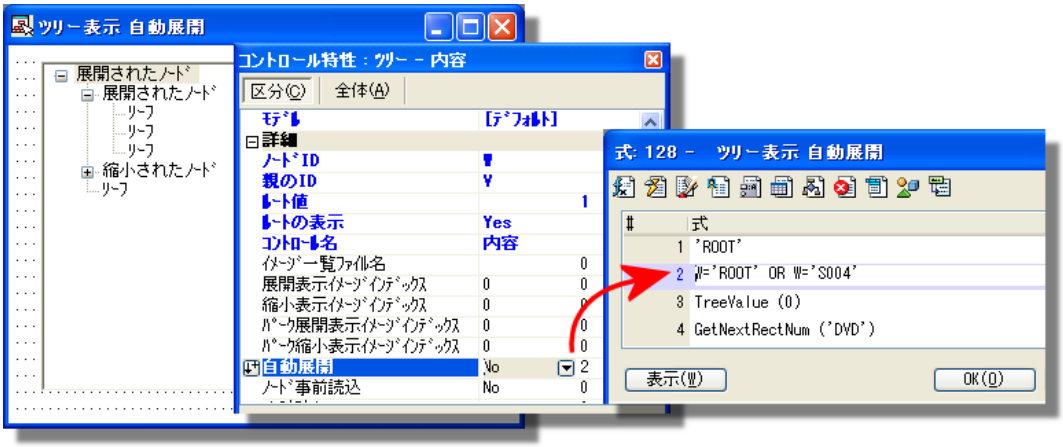
デフォルトでは、ツリーが開く際、上図の左の例のようにノードは展開されないようになっています。ユーザが、ノードを**ダブルクリック**したり、＋表示を**クリック**することで展開することができます。

しかし、ツリーが表示される時点でノードを展開するか否かは、開発時に指定することができます。これは、**自動展開**特性を使用することで実現できます。

自動展開特性が *Yes* に設定されている場合、すべてのノードは展開表示されます。しかし、式で指定することで、実行時に動的に指定することもできます。右の上の例では２つのノード（ROOT と S004）を展開しています。

この機能は、**TreeNodeGoto()** 関数と一緒に使用することで、ツリーを開き、カーソルを特定のノードに位置付けることができます。

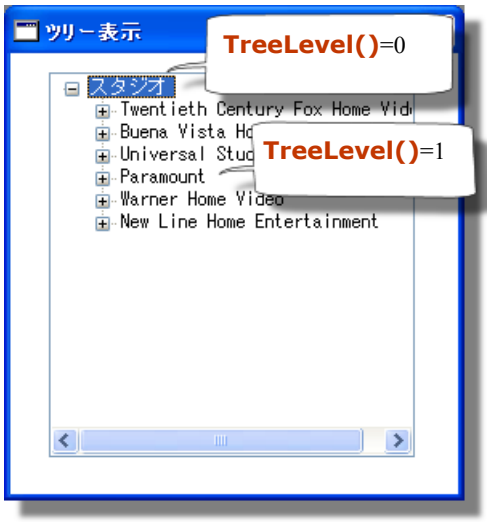
特定のノードに対して自動展開を設定する



1. ツリーコントロールを選択します。
2. コントロール特性の**自動展開**特性を選択し、*Yes* を設定します。この設定では、ツリーが開くとすべてのノードが展開されます。
   指定されたノードのみを展開させるには、式を指定します。この例では、**ノード ID** が ROOT か S004 の場合に展開するように設定されています。これによって上図の例のようにルートノードと１つのノードのみ展開されます。

自動展開をツリーレベルで設定する

TreeLevel() 関数を使用することで、特定のツリーレベルだけを展開させるようにすることができます。



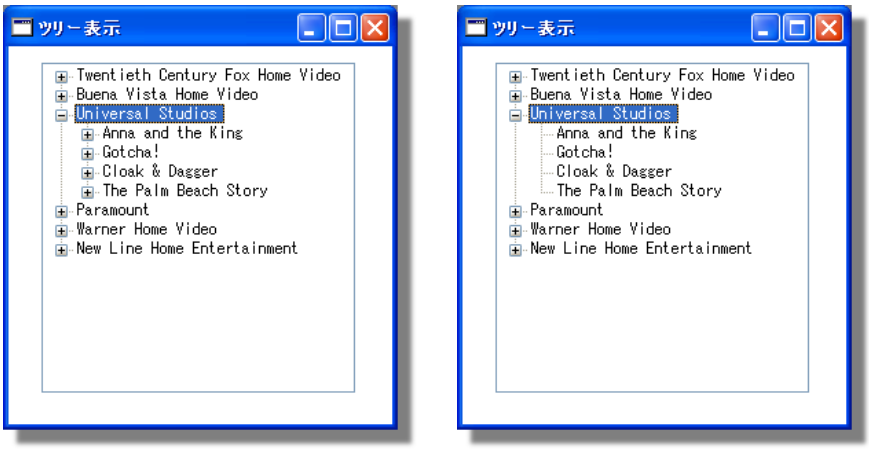
この例では、**自動展開**特性に以下の式を設定することでツリーの一番上のレベルだけを展開しています。

```
TreeLevel()=0
```

指定する式を以下のように変更することで、上位の２つのレベルのみを展開させることができます。

```
TreeLevel()=0 または TreeLevel()=1
```

子ノードを持つノードのみ展開ボタンを表示させるには



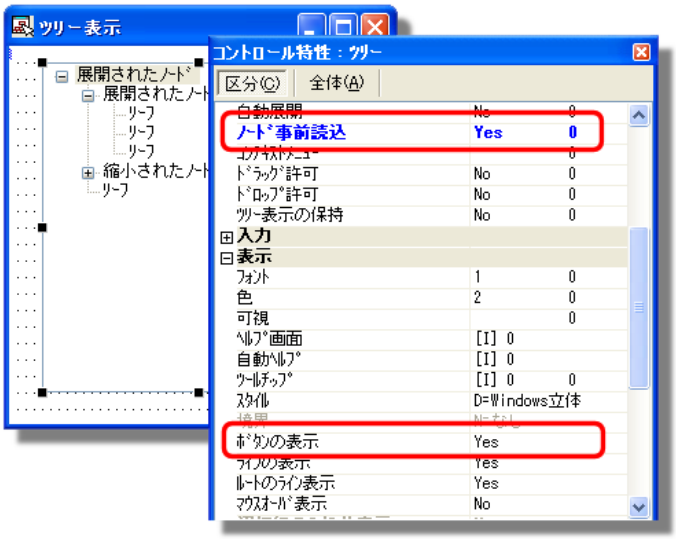
ノード事前読込 = *No* **ノード事前読込** = *Yes*

デフォルトでは、ツリーのノードには＋や－のボタンが表示されます。＋をクリックするとノードが展開され、－をクリックすると縮小されます。最初にツリーが表示された時点では、すべてのノードに＋が表示されます。これは、Magic エンジンが子ノードに該当するレコードをまだ読み込んでいないため、子ノードが存在するかどうかを認識できないためです。ノードをクリックして、子ノードが存在しないことを確認すると＋は表示されなくなります。

Magic エンジンに対して、事前にツリー内のすべてのレコードを事前に読み込むように指定することで、このような動作を変更することができます。これによって、リーフノードは展開ボタンが表示されなくなります。

事前読込を有効にする

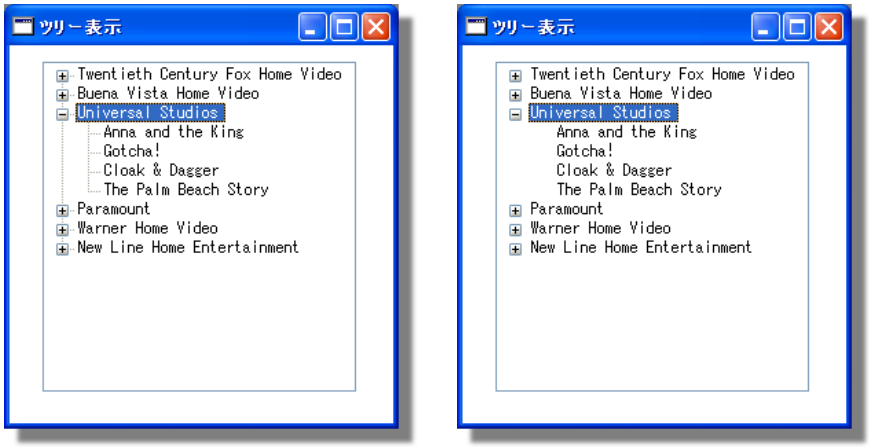
1. ツリーコントロールを選択します。
2. コントロール特性の**ノード事前読込**特性を選択し、*Yes* を選定します。
   これによって、子ノードの存在に応じて展開 / 縮小のボタンが適切に表示されます。



注： レコード数が非常に多い場合、**ノード事前読込**特性を *Yes* に設定すると、ツリーの読込に時間がかかる場合があります。

参照： 「展開 / 縮小ボタンを表示 / 非表示するには」（199 ページ）

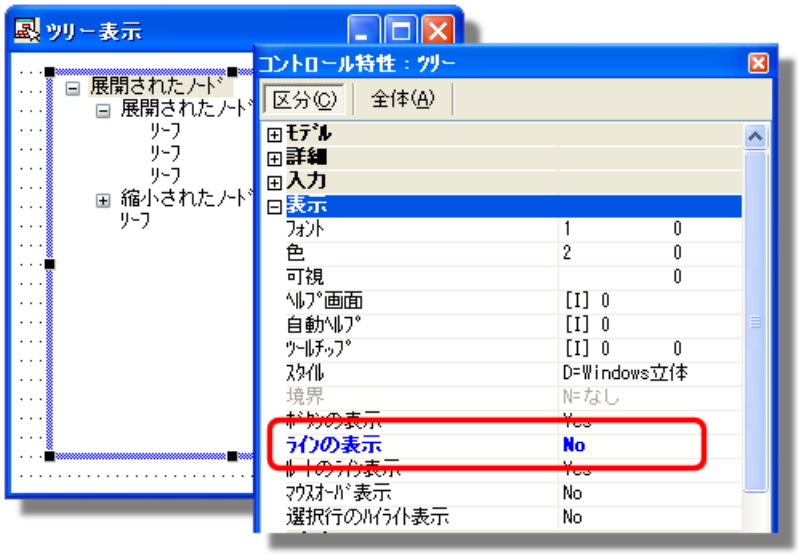
ツリーコントロールの接続線を表示 / 非表示させるには



ラインの表示 = *Yes* **ラインの表示** = *No*

デフォルトでは、ツリー内のノードは点線で接続されて表示しています。**ラインの表示**特性を使用することで、この点線の表示 / 非表示を切り替えることができます。

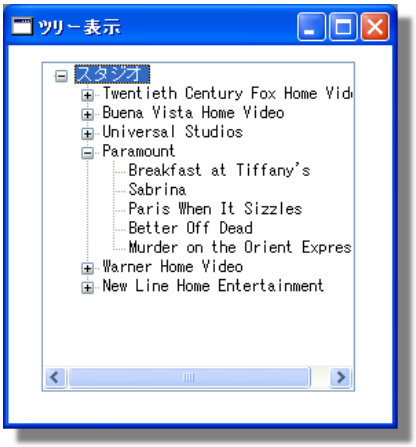
接続線の表示を切り替える



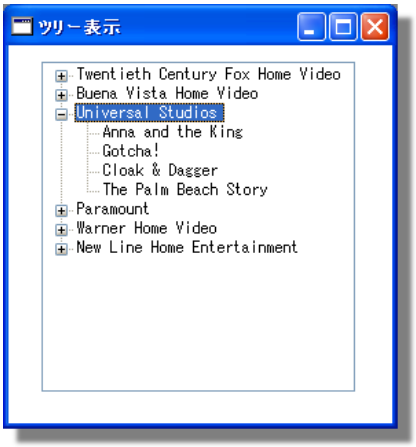
1. **ツリー**コントロールを選択します。
2. **コントロール特性**の**ライン表示**特性を選択します。接続線を表示させたい場合は *Yes* を選定し、表示しない場合は *No* を設定します。

   特性値を変更するとツリー表示に即反映されます。

ルートノードの表示 / 非表示を切り替えるには

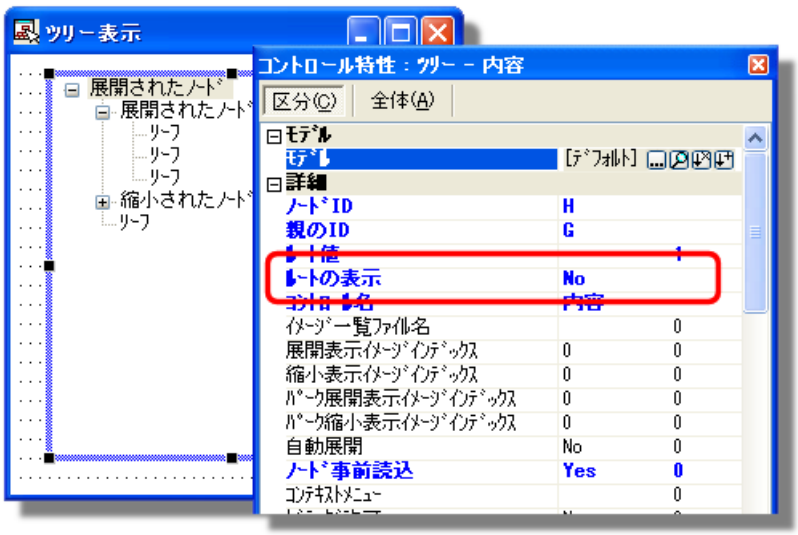


ルート表示 = *Yes*



ルート表示 = *No*

ルートノードの表示 / 非表示を切り替える

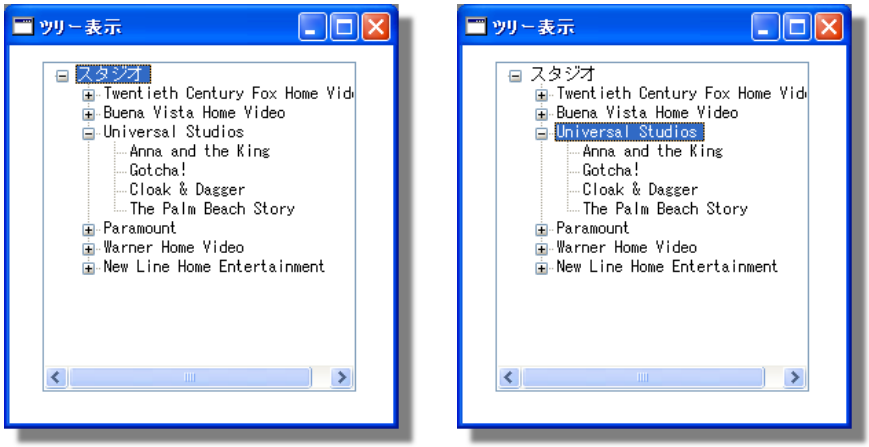


1. ツリーコントロールを選択します。
2. コントロール特性の**ルートの表示**特性を選択します。ルートを表示させたい場合は *Yes* を選定し、表示しない場合は *No* を設定します。

   変更内容は、実行時に反映されます。

注： 表示の有無にかかわらずルートノード用のレコードは必要です。ルートが存在しない場合は、エラーメッセージが表示されます。

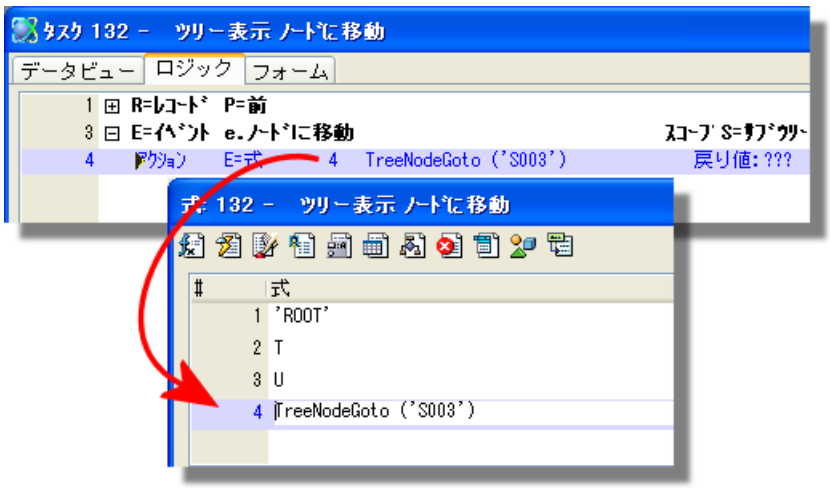
特定のノードに移動するには



カーソルのデフォルトの位置 **カーソルが指定されたノードに移動**

デフォルトでは、左図に示されているように、ツリー開くとカーソルを一番先頭のノード上にパークします。しかし、指定されたノードにカーソルを移動させることもできます。これは、**TreeNodeGoto()** 関数を使用することで実現できます。

TreeNodeGoto() 関数を使用する



1. ロジックエディタを開き、ツリーの展開処理を行うヘッダ行を作成し、必要なイベントを定義します。
2. アクション処理コマンドを定義し、以下の式を設定します。

TreeNodeGoto (*Node*)

パラメータ：

Node：移動先のノード ID を表す値です。この例では、Universal Studio は S003 の ID が設定されており、このノードに移動するように設定されています。指定されたノードがどのレベルに存在しているかは、知る必要がありません。

ヒント :タスク前で TreeNodeGoto() *関数を使用することで、指定したノード上にパークさせることができます。レコード前でも使用できますが、* IsFirstRecordCycle (0) *または*ウェイト *=No の条件を設定する必要があります。条件を指定しないとループする可能性があります。*

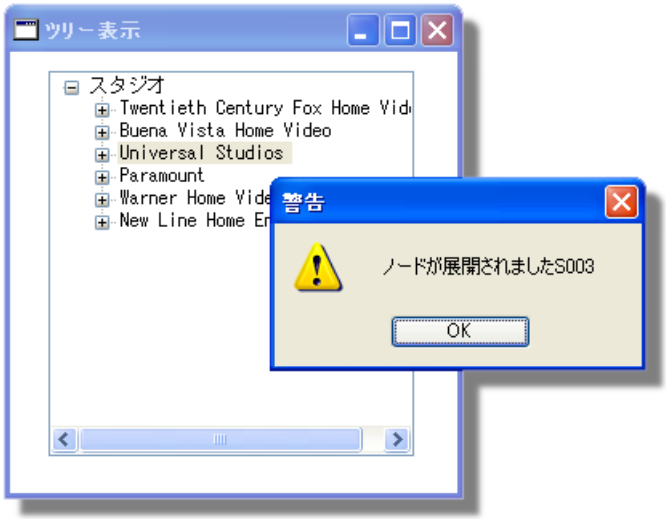
参照 : 「実行時にツリーノードを明示的に展開 / 縮小させるには」（202 ページ）

エンドユーザが行ったノード展開 / 縮小の操作に対応する処理を実行させるには

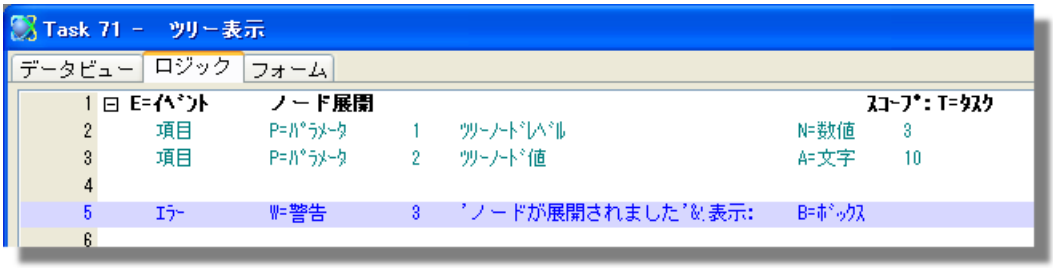
ノードが展開されたり縮小された場合に、何らかの処理を実行させる必要が発生します。例えば、画面に表示されている小計を更新する処理などが考えられます。また、実行中に子ノードを作成する必要があるかもしれません。

ツリーを定義するためにメモリテーブルを使用することで、各ノード上にカスタマイズされたデータを表示することができます。

これらの処理を実現するために、ノード展開 / ノード縮小イベントを使用してロジックユニットを作成します。

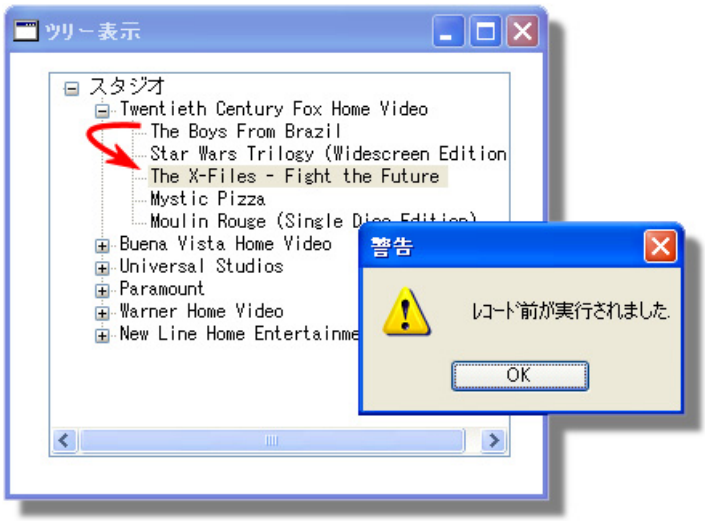


ノード展開 / 縮小イベントを取り込む



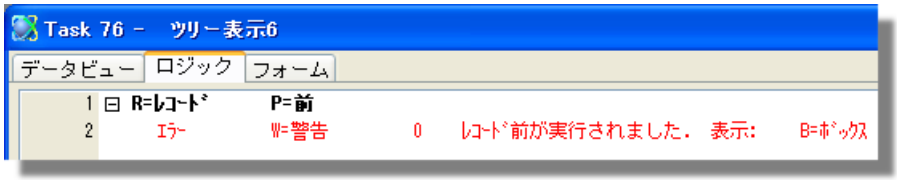
1. *Ctrl+H* を押下してロジックユニットのヘッダ行を作成します。
2. *E* を入力して*イベント*を選択し、内部イベントのノード展開またはノード縮小を選択します。
3. ダイアログボックスで「イベントに対応したパラメータを作成しますか？」というメッセージが表示されます。**Tree Node Level** と **Tree Node Value** の２つのパラメータ項目を作成する場合は、はいをクリックします。
4. これで、ユーザがノードを展開したり縮小した場合、実行するロジックユニットが定義できました。ロジックユニット内に定義されたパラメータ項目を使用してノードレベルと値を確認することができます。
5. イベント特性で、伝播特性を *Yes* に設定します。設定しないと、このロジックユニット内でイベントは止められ、ノードの展開 / 縮小が行われません。

ユーザのノードから別のノードに移動する操作に対応するには



ユーザが、あるノードから別のノードにフォーカスを移した場合に、何らかの処理を実行させたい場合があります。ツリー内の各ノードが同じデータソースのレコードであれば、容易に実現できます。通常のレコードイベント（レコード前やレコード後）が、レコードの移動の際に実行されるからです。レコードが異なるフォーマットで表示されているにすぎません。

ノードに入る動作を取り込む

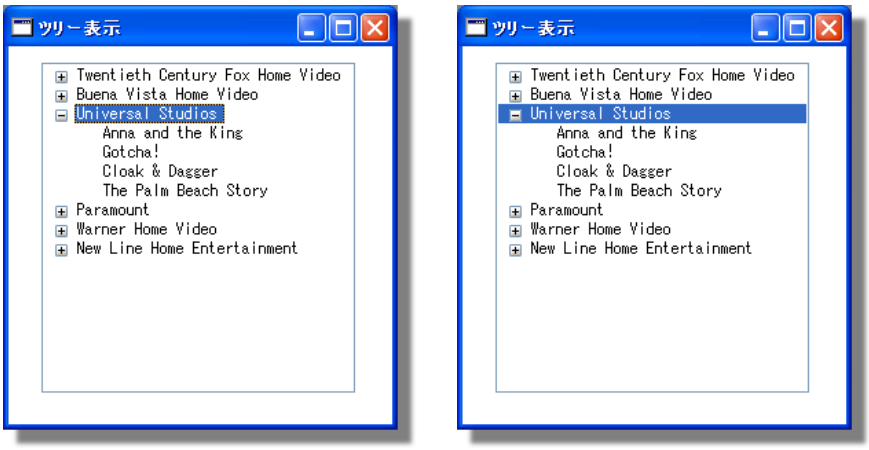


カーソルがノードに入る前にその動作を取り込むには、レコード前に処理コマンドを定義する必要があります。

ノードから抜け出る動作を取り込む

データビューが変更された場合のみ、レコード後が実行されます。レコードが変更されなくても、ノードから抜け出る動作を取り込む必要がある場合は、タスク特性の強制レコード後特性（動作タブ）を *Yes* に設定します。

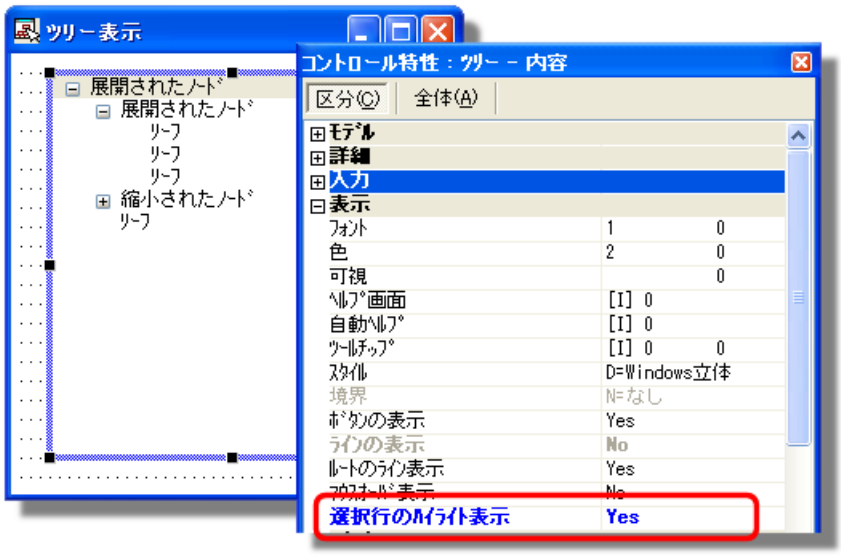
現在のツリーノードの行を強調表示させるには



選択行のハイライト表示 = *No* **選択行のハイライト表示** = *Yes*

ツリー内でノードを選択すると、そのノードは強調表示されます。デフォルトでは、ノードテキストのみが強調表示されます。しかし、ツリー全体を横切る強調行として表示させることもできます。これは、**選択行のハイライト表示**特性によって制御することができます。

行ハイライトの指定を切り替える



1. ツリーコントロールを選択します。
2. コントロール特性の**選択行のハイライト表示**特性を選択し、強調行を表示させる場合は *Yes* を設定します。表示させない場合は、*No* を設定します。

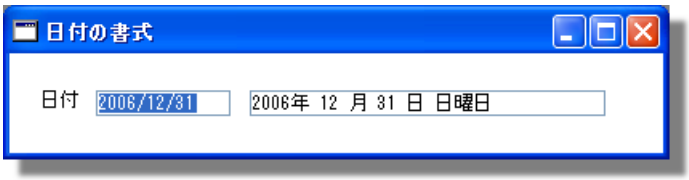
   特性値を変更すると動作に即反映されます。

注： **ラインの表示**特性が *No* の場合のみこの特性は有効になります。

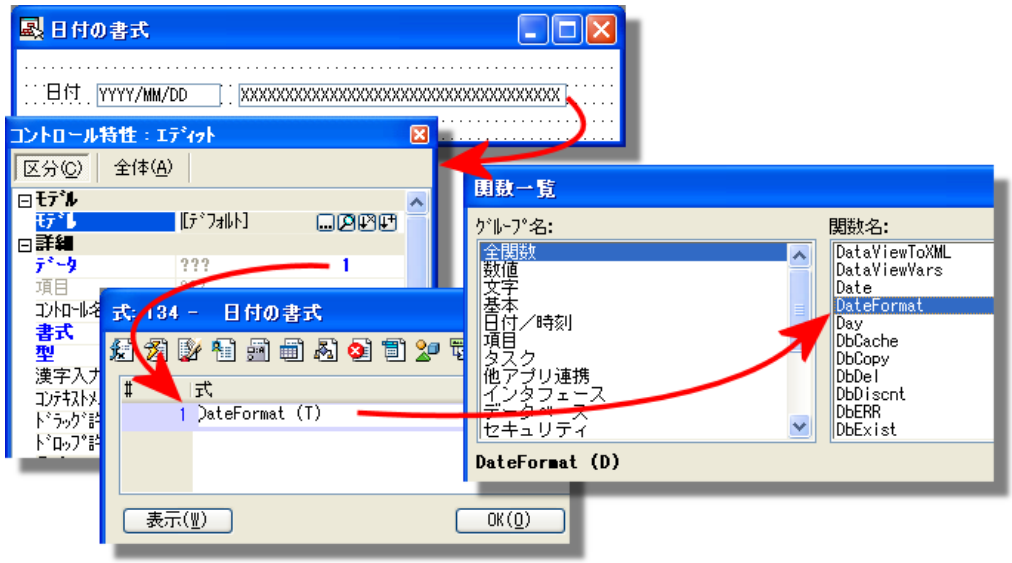
［このページは意図的に空白にしています。］

第 11 章 : いろいろなロジック

ユーザ定義関数を作成するには



ユーザ定義関数は特殊なタイプのロジックユニットとして扱われます。ユーザ定義関数には、他のロジックユニットで利用されているものと同じパラメータ項目や変数項目、および処理コマンドを使用します。主な違いの 1 つに、戻り値があるということです。関数が式やフォームで使用される場合、関数から返る値を（いったんデータ項目に格納することなく）直接使用することができます。



上図の例では、帳票用として複雑な書式で定義された文字列として日付データを変換する関数について示されています。変換された文字列は、式の中でこの関数を使用することで直接フォームに定義することとができます。

ユーザ定義関数は、Magic に組み込まれている関数と同じように関数一覧に表示されます。これにより、通常の内部関数と全く同じように利用することができます。これはまた、内部関数と同じ名前のユーザ定義関数を作成した場合、内部関数をオーバライドすることになります。このような関数の再帰的な利用はできないため、関数名の定義方法については注意する必要があります。

以下、ユーザ定義関数の作成方法について説明しています。

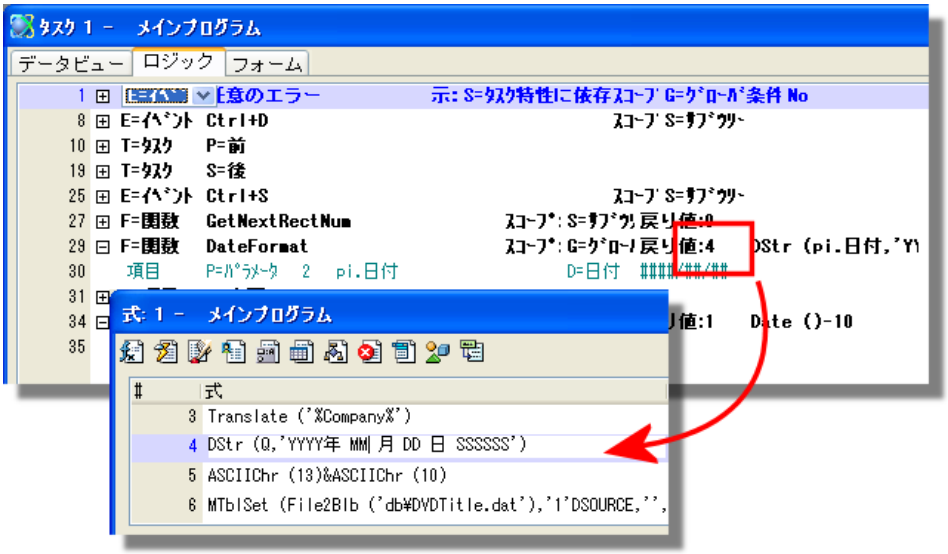
関数の有効範囲を定義する

ユーザ定義関数の有効範囲は、それが定義されていル場所と関数のスコープ特性の値に依存します。

- 関数があるタスク内に定義されており、**スコープ**特性が ***T= タスク***と設定された場合、その関数はそのタスク内でのみ参照できます。
- 関数が親タスク内で定義されており、**スコープ**特性が ***S= サブタスク***と設定された場合、その関数はすべてのサブタスク内で参照できます。従って、**メインプログラム**内で作成された関数は、アプリケーション全体で参照することができます。
- 関数がコンポーネントの一部の場合、複数のプロジェクト間で共有することができます。

参照： 「現在のタスクでのみ有効な関数を作成するには」（217 ページ）
「プロジェクト全体で有効な関数を作成するには」（218 ページ）

ユーザ定義関数を作成する



1. タスクの**ロジックエディタ**を開きます。
2. ***Ctrl+H*** を押下してヘッダ行を作成します。
3. **F** を入力して***関数***を選択します。カーソルが右側に移動し、関数名の入力カラムに入ります。
4. この関数の名前を入力します。どのような名前も使用できますが、Magic の内部関数と同じ名前を指定しないようにしてください。
5. 必要であれば、入出力用のパラメータ項目を定義します（詳細は、「ユーザ定義関数のパラメータを設定するには」（215 ページ）を参照してください）。
6. **戻り値**を定義します（詳細は、「ユーザ定義関数の戻り値を設定するには」（216 ページ）を参照してください）。

これで、**ユーザ定義関数**が作成されました。この関数は、内部関数と一緒に**関数一覧**に表示されるようになります。

ユーザ定義関数のパラメータを設定するには

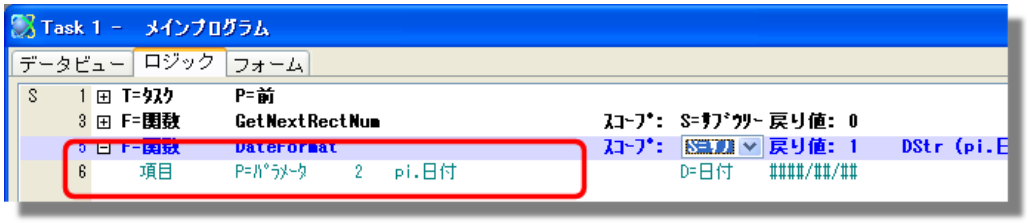
ユーザ定義関数のパラメータはタスクやイベントロジックユニットに定義されたパラメータと同じように動作します。これらは、パラメータとして定義された項目です。

関数用にパラメータを作成するには、必要なパラメータ項目を適切な順番で作成します。パラメータを作成すると、関数一覧にはデータ型に対応する文字でパラメータが表示されます。

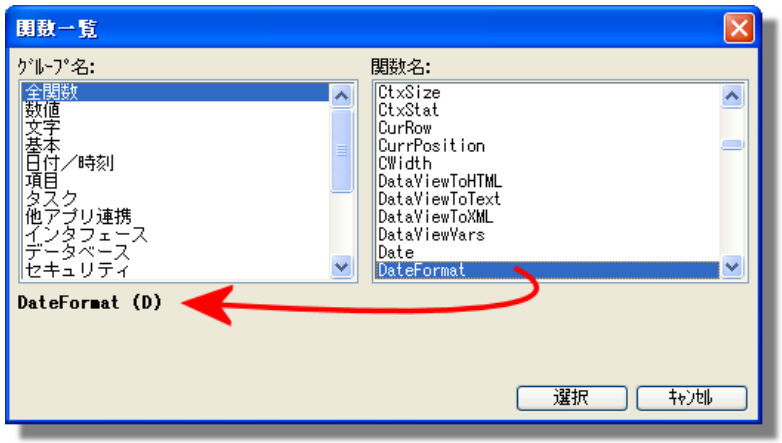
関数で使用されるパラメータは、入力パラメータです。関数が式で使用されるため、データは常に参照型で渡され、関数によって変更することはできません。データを返したい場合は、戻り値特性を使用する必要があります（「ユーザ定義関数の戻り値を設定するには」（216 ページ）を参照してください）。

ヒント：パラメータ項目として定義しない（変数項目のまま）場合も同じように動作しますが、関数一覧にパラメータは表示されません。

関数のパラメータを定義する



1. 関数ロジックユニットに移動します。
2. *F4*（編集→行作成）を押下します。空白行がカーソル位置の下に作成され、カーソルは追加された行の左側の入力カラムに移動します。
3. *V* を入力します。***項目***という文字が表示されカーソルは右側に移動します。
4. *P* を入力します。***パラメータ***という文字が表示され、カーソルが右側に移動します。
5. パラメータの名前を入力し、カーソルを右側に移動します。
6. この項目のモデルを選択するか、型や書式などの特性を設定します。
7. 複数のパラメータがある場合は、上記と同じような操作を繰り返して作成します。



これで、作成した関数はパラメータを受け取ることができるようになります。関数が関数一覧に表示されると、定義されたパラメータのデータ型に対応する文字が表示されます。

ユーザ定義関数の戻り値を設定するには

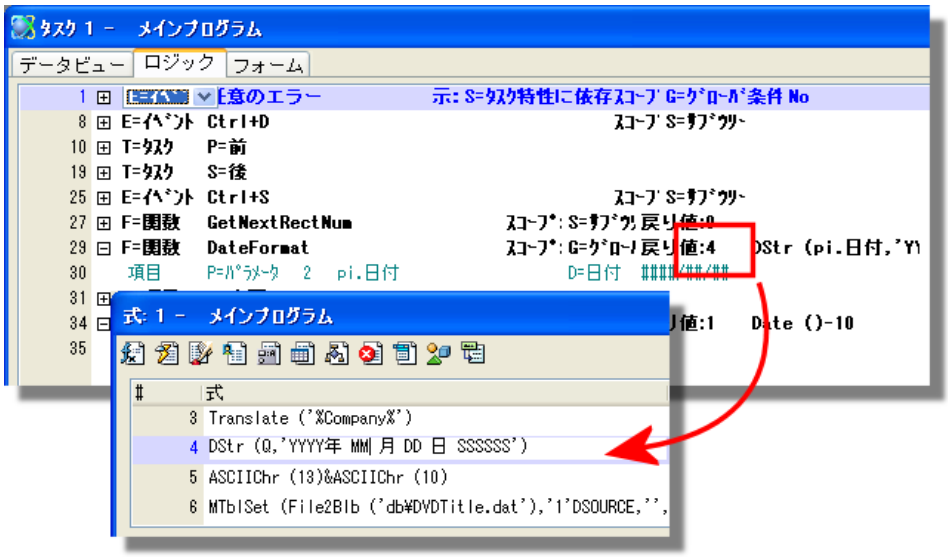
ユーザ定義関数の主な利点の一つとして、戻り値を定義することができることです。戻り値は、変数項目に一時的に格納することなく、フォーム上やパラメータで直接使用することができます。これにより、プログラミングの効率化が高まります。

また、関数は常に式の中で使用されるため、入力パラメータは入力のみに使用されます。関数内でこのパラメータの値を変更しても、起動元には返りません。データを返したい場合は、戻り値を使用します。

戻り値の使用は任意です。必ずしも値を返す必要があるわけではありません。

ユーザ定義関数では、どのようなデータ型の戻り値も返すことができます。正しく関数を使用するか否かは開発者に委ねられています。しかし、例えば、数値項目に文字項目を戻すように設定した場合、**構文チェック**ユーティリティを実行するとエラーを返します。

ユーザ定義関数に戻り値を設定する



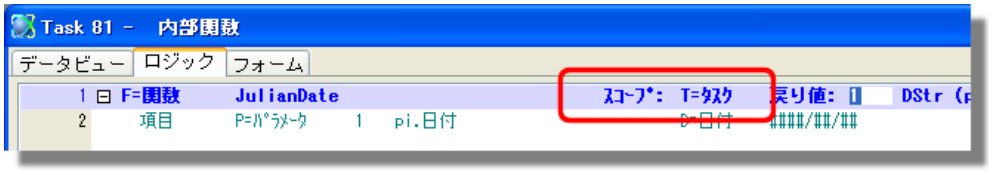
1. **関数**ロジックユニットのヘッダ行に移動します。
2. **戻り値**特性に移動します。
3. ズーム（*F5* または、ダブルクリック）して**式エディタ**を開きます。
4. 戻り値として返したい値として評価される式を入力します。この例では、日付を文字列に変換するために **Trim()** と **Dstr()** の２つの関数を使用しています。

これで、関数を実行すると、式で指定された値が返ります。

現在のタスクでのみ有効な関数を作成するには

作成された関数の有効範囲は**スコープ**特性で決まります。**スコープ**特性が ***T= タスク***と設定されている場合、関数はこのタスクでのみ参照できます。

ローカル関数を作成する



1. 関数のヘッダ行に移動します。
2. **スコープ**特性に移動します。
3. ***T= タスク***を選択します。

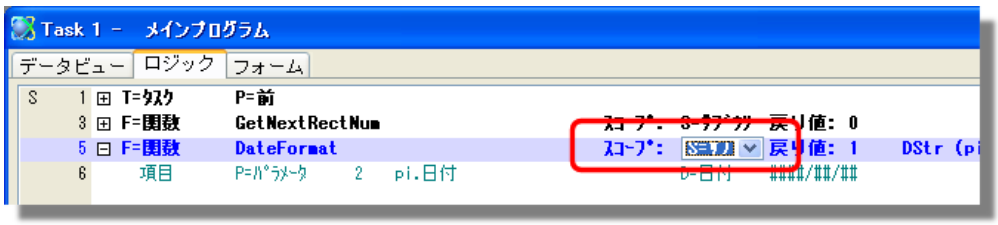
これでこの関数は、このタスク内でのみ有効になります。

参照： 関数を作成する基本的な作業方法については、「ユーザ定義関数を作成するには」（213 ページ）を参照してください。

プロジェクト全体で有効な関数を作成するには

作成された関数の有効範囲はスコープ特性で決まります。スコープ特性が *S= サブタスク*と設定されている場合、関数はこのタスクとサブタスクで参照できます。これは、スコープ特性を *S= サブタスク*と設定されたユーザ定義関数がメインプログラムに定義された場合、この関数はプロジェクト全体で使用することができることを意味しています。

グローバル関数を作成する



1. メインプログラムでユーザ定義関数を作成します。
2. スコープ特性に移動します。
3. *S= サブツリー*を選択します。

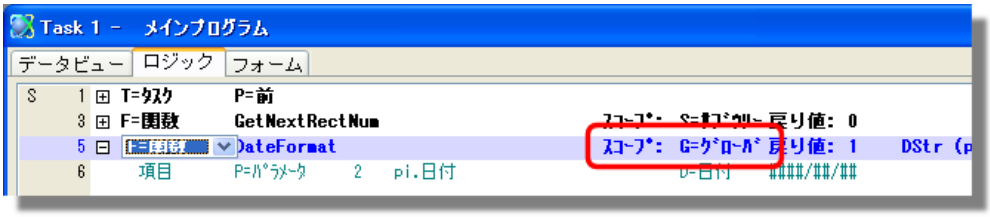
これでこの関数は、プロジェクト全体で有効になります。

参照： 関数を作成する基本的な作業方法については、「ユーザ定義関数を作成するには」（213 ページ）を参照してください。

複数のプロジェクト間で関数を共有するには

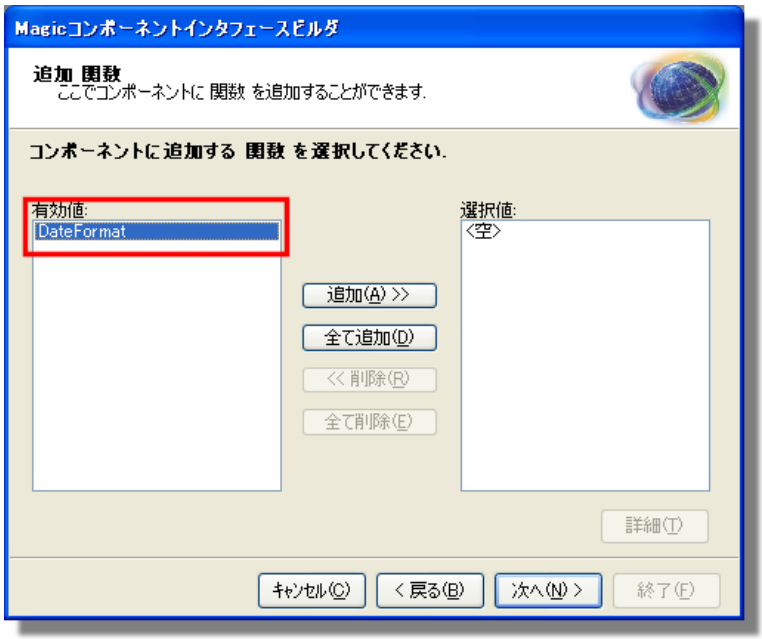
作成された関数の有効範囲はスコープ特性で決まります。スコープ特性が *G= グローバル*と設定されている場合、関数はコンポーネントとして公開することができます。これは、スコープ特性を *G= グルーバル*と設定されたユーザ定義関数がメインプログラムに定義された場合、この関数は他のプロジェクトでも使用することができることを意味しています。

グローバル関数を作成する



1. メインプログラムでユーザ定義関数を作成します。
2. スコープ特性に移動します。
3. *G= グローバル*を選択します。

これで、以下に示すように、コンポーネントの作成時に、Magic コンポーネントインタフェースビルダで選択できるようになります。

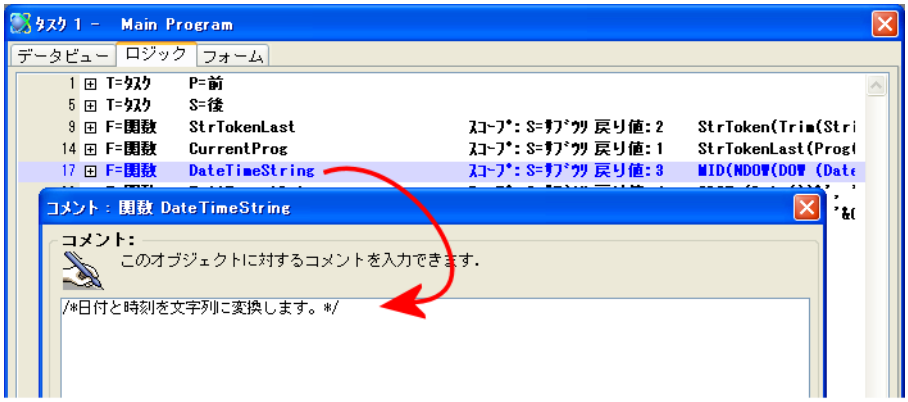


参照： 「プロジェクト間で Magic のオブジェクトを再利用するには」（357 ページ）

ユーザ定義関数のヘルプを定義するには

作成された関数のヘルプテキストを**関数一覧**で表示させることができます。**関数**ロジックユニットに**コメント**を追加することで表示されます。このようにすることで、この関数の意味や利用方法などを利用者に正確に伝えることができます。

ユーザ定義関数にコメントを定義する



1. 任意のタスクで**ユーザ定義関数**を作成します（**メインプログラム**で作成することで、プロジェクト内で有効な関数になります）。
2. ***F12***（**オプション→コメント入力**）を押下して**コメント**ダイアログを開きます。
3. この関数の内容に関するテキストを入力します。テキストは、以下のように「/*」と「*/」で囲みます。
   /* 日付と時刻を文字列に変換します。*/

これで、**関数一覧**でこの関数にカーソルが位置付けらると、下辺に入力したヘルプテキストが表示されます。

一連の処理を繰り返し実行させるには

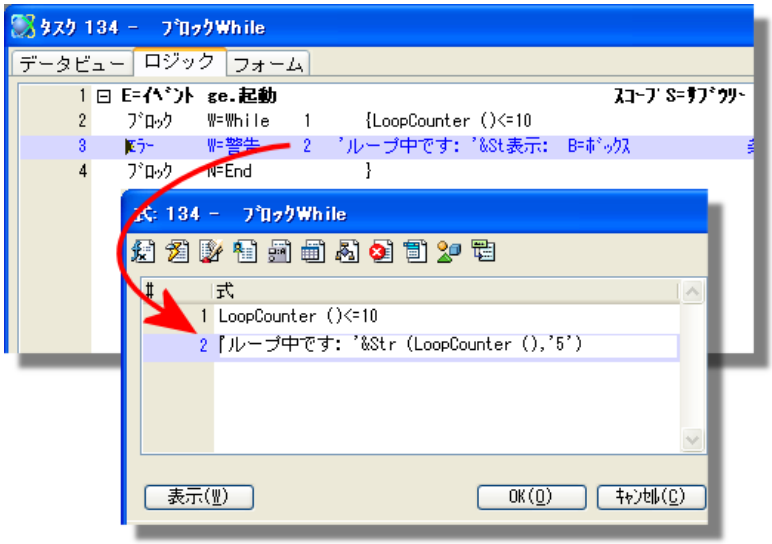
一連の処理コマンドを何度も繰り返して実行させたい場合、一般的なプログラミング言語ではある種類のループメカニズムを使用しています。Magic におけるループメカニズムは、**ブロック While** 処理コマンドで実現できます。これは、どの**ロジックユニット**にも定義することができます。

ブロック While 処理コマンドは、指定された条件が True の場合、ブロック内の処理を実行します。この例では、処理を 10 回実行するように定義しています。

LoopCounter() 関数を使用する

ブロック While 処理コマンドでは、**LoopCounter()** と呼ばれる独自の関数を使用することができます。この関数は、処理が繰り返された回数を戻すため、ループ処理のための特別なカウンタを作成する必要がありません。

ブロック While 処理コマンドを定義する



1. 使用したい**ロジックユニット**に移動します。
2. ***F4***（**編集→行作成**）を押下します。空白行が追加され、カーソルは追加行の左側の入力カラムに移動します。
3. ***B*** を入力します。**ブロック**処理コマンドが選択され、***ブロック If*** と ***ブロック End*** の 2 つの行が表示されます。カーソルは、***If*** の位置に位置付けられます。
4. ***W*** を入力します。***ブロック While*** が選択されます。
5. 右に ***Tab*** 移動し、**ズーム**（***F5*** または、**ダブルクリック**）します。**式エディタ**が開きます。
6. **式エディタ**には、ループを抜ける場合に False が返る条件式を設定します。この例では、以下の式を設定しています。

```
Loopcounter() <= 10
```

繰返し番号が 11 になると False になるため、ループは 10 回実行されることになります。

データ項目を更新するには

Magic でデータ項目を更新するには、いくつかの方法があります。

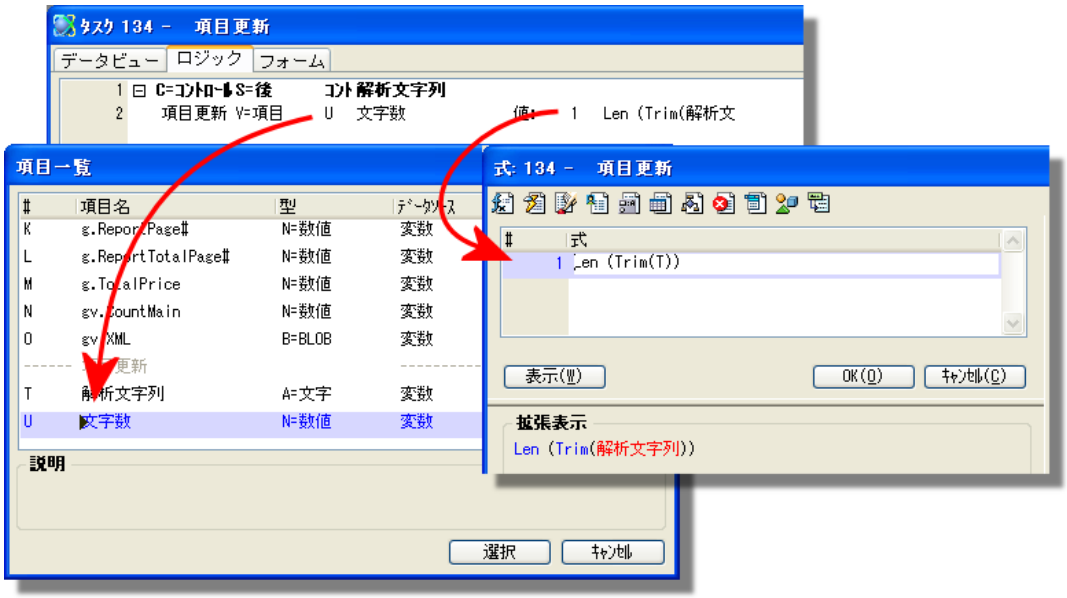
- デフォルトの値を、項目モデルやデータソースまたはタスク内での変数定義であらかじめ割り当てる。
- タスク内でデータビューを定義する際に、代入特性を使用する。
- フォーム上の項目の場合、ユーザによって入力することができます。
- 項目更新処理コマンドの実行する。
- VarSet() 関数のような関数を使用する。
- フォーム入力処理コマンドによってデータ入力をする。

項目更新処理コマンドは、ロジックユニットで項目の値を更新する通常の方法です。ロジックユニット内で実行される範囲では、項目更新処理コマンドは手続型です。

オンラインプログラムでは、代入特性の方が項目更新処理コマンドよりよく使用されます。代入特性は非手続き型のため、代入式で使用されている項目の値が変更されると自動的に代入値も更新されます。このような動作によって、いつ更新すべきか気にする必要がなくなります。

この例では、文字型項目内の文字数を文字数項目に格納しています。コントロール後ロジックユニットに更新処理が定義されているため、ユーザが入力欄から抜けるとすぐに文字数項目が更新されます。

項目更新処理コマンドを使用する

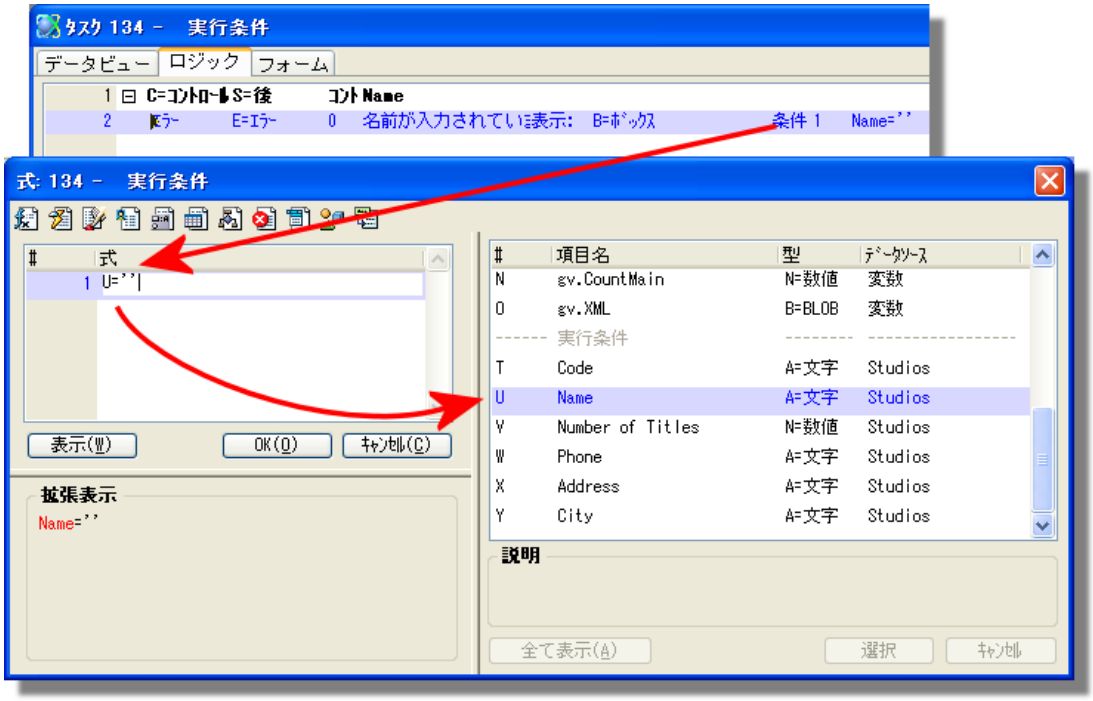


1. 使用したいロジックユニットに移動します。
2. *F4*（編集→行作成）を押下します。空白行が追加され、カーソルは追加行の左側の入力カラムに移動します。
3. *U* を入力します。項目更新処理コマンドが選択され、カーソルは右に移動します。
4. ズーム（*F5* または、ダブルクリック）して項目一覧を開きます。更新したい項目を選択し *Enter* を押下するか選択ボタンをクリックします。一覧から選択しないで項目番号を直接入力することもできます。
5. 右側に *Tab* 移動して値カラムにカーソルを置きます。
6. 値カラムでズーム（*F5* または、ダブルクリック）して式エディタを開きます。
7. 更新する値として評価される式を入力します。他の項目の値で更新するだけであれば、その項目の項目番号を指定します。カウンタのリセット処理を行うのであれば、0 という数値を入力することもあります。また、上記の例のような文字列の文字数を返す式なども指定できます。
8. *Enter* を押下するか、OK をクリックすることで値カラムに戻ります。

参照： 第 20 章：「タスクのデータ項目の値を設定するには」（451 ページ）

項目の値を処理コマンドの実行条件に設定するには

Magic には処理コマンドの実行を制御するための様々な方法があります。基本的には、論理値を条件式として指定することで可能です。この例では、ユーザが入力項目を抜けた時に、入力項目が空白の場合にエラーメッセージを表示させるように式を指定しています。



処理コマンドの条件式を入力する

1. 条件の設定が必要な処理コマンドの行に移動します。この例では、エラー処理コマンドになります。
2. 条件カラムに移動します。
3. ズーム（*F5* または、ダブルクリック）して、式エディタを開きます。
4. *F4* を押下して、1 行追加します。
5. 再びズーム（*F5* または、ダブルクリック）して右側に表示されている項目一覧にカーソルを移動します。以下のようにして項目を選択します。
   - 選択したい項目にカーソルを移動して、*Enter* を押下するか、選択ボタンをクリックします。
   - 項目番号を直接入力します。（この例では、B）
6. 必要に応じて、式を追加します。

処理コマンドを実行させる必要がある場合は、True になるように式を定義します。条件カラムは、If 文と同じと考えてください。条件が True の場合、処理コマンドは実行されます。この例では、*B=''* と設定されています。これは、項目 B が空白の場合 True と判断され、エラーメッセージを表示するようになります。

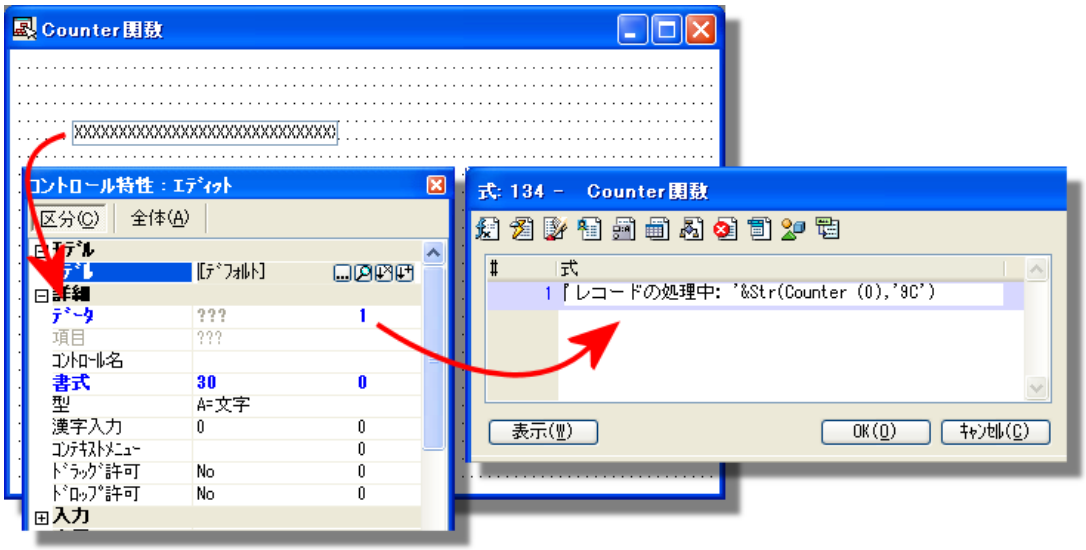
参照： 第 21 章：「式」（461 ページ）

バッチプログラムで処理されたレコードの連番を取得するには

バッチプログラムでは、処理中のレコード番号が必要な場合があります。ユーザに情報用メッセージを表示したり、連号を作成したり、またはデバッグ中に処理をキャンセルするためにこの番号を使用することがあります。

このような処理を実現するために、Magic では **Counter()** 関数を使用します。この関数は、タスクの世代情報という 1 つのパラメータ指定します。**Counter(*0*)** は、現在のバッチタスクが処理しているレコード数を返し、**Counter(*1*)** は親タスクが処理しているレコード数を返します。

Counter() 関数を使用する



処理中のレコード数を取得するには、式エディタで以下のように入力します。

```
Counter (0)
```

上の式は現在のタスクのレコード数を取得します。**Counter(*1*)** は親タスクのレコード数を返し、**Counter(*2*)** はその上位のタスクのレコード数が返ります。

この例では、ユーザにメッセージを表示するために **Counter(*0*)** で返された値を使用しています。戻り値は数値のため、**Str()** 関数を使用して文字列に変換しています。

オンラインタスクで最初のレコードの場合のみ実行する条件を設定するには

オンラインタスクでは、ユーザが最初にタスクを起動した時点で、処理を実行させたい場合があります。タスク起動時は、データソースをオープンし、データの初期設定を行うため、このような処理を実行させる適切なロジックユニットはレコード前になります。しかし、ユーザが表形式でデータを参照している場合、ユーザがレコード間を移動させるたびに処理が実行される必要はないかもしれません。

このような場合、**IsFirstRecordCycle()** 関数を使用して対応できます。この関数は、レコード前が最初に実行されたときのみ True を返します。

IsFirstRecordCycle() 関数を使用する

1. **IsFirstRecordCycle()** 関数を使用したい場所で式を定義します。
2. 以下のように入力します。

```
IsFirstRecordCycle(0)
```

上の式は、現在のタスクの最初のレコードサイクルを確認するものです。親タスクを確認は、**IsFirstRecordCycle(*1*)** を使用します。

この例では、タスクが起動された時点でのみ **CtrlGoTo** 関数が実行されるようになっています。

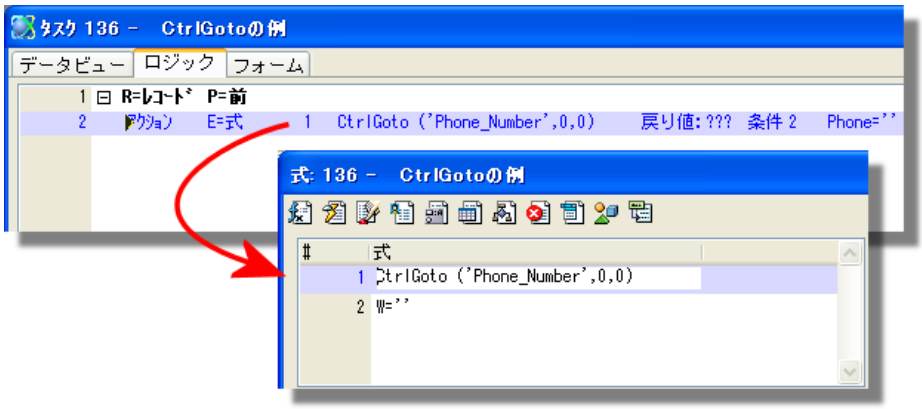
特定のコントロールにカーソルを移動させるには

カーソルを特定のコントロールに移動させたい場合があります。例えば、エラーメッセージが表示された時に、エラー対応する表示項目に移動するようにしたい場合があるかもしれません。

CtrlGoto() 関数を使用することでカーソルを指定したコントロールに移動させることができます。

CtrlGoto('Control name', row, generation)	
'Control name'	移動先のコントロールのコントロール名
row	テーブル上のデータの場合、行数を指定します。
generation	タスクの世代番号。現在のタスクが 0、親タスクが 1 になります。

CtrlGoto() 関数を使用する



1. カーソルの移動処理を実行させたいロジックユニットに移動します。この場合、ユーザが最初にレコードに入った時点で、カーソルを電話番号に移動させたいため、レコード前に定義しています。
2. *F4* を押下して 1 行追加します。
3. *A* を入力してアクション処理コマンドを指定します。カーソルは式カラムに移動します。
4. ズーム（*F5* または、ダブルクリック）して式エディタを開きます。
5. *F4* を押下して 1 行追加します。以下の関数名を入力します。

   ```
   CtrlGoto(
   ```

   Ct と入力して *Ctrl+Space* を押下することでオートコンプリート機能が使用できます。
6. コンテキストメニューを開きコントロールを選択することでコントロール一覧が表示されます。ここから移動したいコントロール名を選択できます。直接コントロール名を入力することもできます。
7. ほとんどは、現在のタスクのコントロールへの移動で、テーブル内のコントロールでもないので、以降の２つのパラメータは 0,0 と入力するだけになります。

   しかし、テーブルを使用している場合、２番目のパラメータは移動先の行番号となります。例えば、**CtrlGoto(*'Phone_Number',3,0*)** では、カーソルをテーブル内の３行目の Phone_Number という名前のコントロールに移動します。

   親タスクのコントロールに移動するのであれば、３番目のパラメータを 1 に、その上のタスクに移動するのであれば 2 を指定します。**CtrlGoto(*'Phone_Number',0,1*)** は、カーソルを親タスクの Phone_Number という名前のコントロールに移動します。

コントロールに入ったり抜けたりした場合に実行するロジックを作成するには

ユーザが項目に入ったり、抜けた時に、処理を実行させたい場合がります。例えば、ユーザが項目に入った時点で何らかの有益な情報を表示させたり、自動的に選択一覧を表示させたりすることなどが考えられます。ユーザが項目を抜けた時点で、何らかの計算処理を実行したり、ユーザの入力内容を検証するような処理も必要になります。

この種のロジックは、**コントロール前**と**コントロール後**の**ロジックユニット**に定義されます。

コントロール前を使用する

1. *Ctrl+H* を押下して**ロジックユニット**のヘッダ行を作成します。
2. *C* を入力して***コントロール***を選択します。次のカラムに移動します。
3. ドロップダウンリストから *P* を入力して、***前***を選択します。次のカラムに移動します。
4. *ズーム*（*F5* または、*ダブルクリック*）して**コントロール一覧**から対象となるコントロールを選択します。コントロールがまだ定義されていなかったり、（コンポーネントを使用するなどで）参照できない状態の場合、コントロール名を入力することもできます。
5. 作成された**ロジックユニット**内で実行したい処理コマンドを入力します。

これで、ユーザが指定されたコントロールに入った場合、定義された処理が実行されます。

コントロール後を使用する

1. *Ctrl+H* を押下して**ロジックユニット**のヘッダ行を作成します。
2. *C* を入力して***コントロール***を選択します。次のカラムに移動します。
3. ドロップダウンリスから *S* を入力して、***後***を選択します。次のカラムに移動します。
4. *ズーム*（*F5* または、*ダブルクリック*）して**コントロール一覧**から対象となるコントロールを選択します。コントロールがまだ定義されていなかったり、（コンポーネントを使用するなどで）参照できない状態の場合、コントロール名を入力することもできます。
5. 作成された**ロジックユニット**内で実行したい処理コマンドを入力します。

これで、ユーザが指定されたコントロールを抜けた場合、定義された処理が実行されます。

注：　コントロール後と**コントロール検証**は同じような動作に見えますが、実は全く異なります。ユーザがコントロールを抜けた場合、**コントロール後**は必ず実行されますが、**コントロール検証**はコントロールにパークしたか否かにかかわらず、ユーザが項目を通過した時点で実行されます。例えば、レコードが保存される前に、項目が空白のままになっているかどうかを確認する場合は、**コントロール検証**を使用します。

カーソルを一定の方向に移動させた場合のみ処理を実行させるには

ユーザが Windows 環境でどこでもクリックする傾向がある場合、画面上でカーソルを上下に移動してもそれに関係なく、項目に対応する処理を実行させる必要があります。しかし、キーボード操作が中心のアプリケーションとして、カーソルの移動方向にもとづいて、処理が実行されるかどうかを指定することができます。**Flow()** 関数を使用することでこのような処理が可能です。

Flow() 関数のパラメータ	方向	モード
N	順方向	ステップモード
F	順方向	高速モード
P	逆方向	ステップモード
R	逆方向	高速モード
S	選択終了	
C	取消終了	

ユーザがエディタ上の前方（上）から項目に入った場合、**Flow(*'N'*)** は True を返します。後方（下）から入った場合、**Flow(*'P'*)** は True を返します。

順番にカーソルを移動したり、特定のコントロールをスキップした場合のみ処理を実行させるには

カーソルを動かしたり、クリックすることによって、特定のコントロールにフォーカスを置くとすぐに実行する必要のある処理を定義することができます。これは、そのコントロールのコントロール前に処理コマンドを定義することで可能となります。

コントロール前を使用する

1. *Ctrl+H* を押下してロジックユニットのヘッダ行を作成します。
2. *C* を入力して*コントロール*を選択します。次のカラムに移動します。
3. ドロップダウンリスから **P** を入力して、*前*を選択します。次のカラムに移動します。
4. ズーム（*F5* または、*ダブルクリック*）してコントロール一覧から対象となるコントロールを選択します。コントロールがまだ定義されていなかったり、（コンポーネントを使用するなどで）参照できない状態の場合、コントロール名を入力することもできます。
5. 作成されたロジックユニット内で実行したい処理コマンドを入力します。

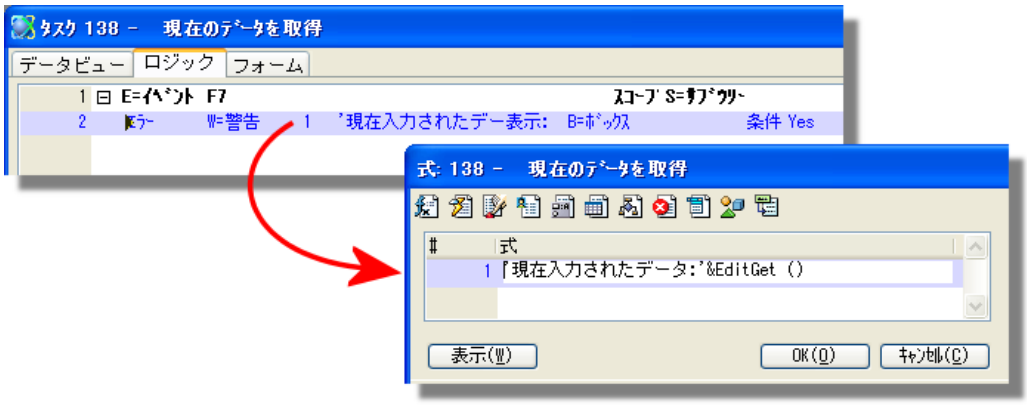
これで、ユーザが指定されたコントロールに最初にフォーカスを移した場合のみ、定義された処理が実行されます。

コントロール上（エディット / リッチエディット / 複数選択リストボックス）にフォーカスが残っている状態で新規入力データを検索するには

ユーザがデータをフィールドに入力した場合、ユーザがフィールドを抜けるまで、変更されたフィールドはデータ項目に保存されず、ハンドラでも処理できません。ユーザがフィールドを抜けた後、コントロール後で入力内容の検証が通常必要になります。

しかし、ユーザがフィールドを抜ける前に現在入力されたデータ内容を検索したい場合もあります。例えば、フィールド上にフォーカスがある状態で、計算処理など該当するフィールドと関連する何らかの処理を実行するために、ショートカットキーを押下するような場合がこれにあたります。

EditGet() 関数を使用する



フィールドの現在の値を取り出す場合は、以下の式を使用します。

```
EditGet()
```

ed を入力し *Ctrl+Space* を押下すると、オートコンプリートのポップアップメニューが表示されるのでここから関数を選択できます。

EditGet() 関数は、イベントロジックユニット内で起動されると、入力されたデータを返します。

EditGet() 関数はどのような編集フィールドに対しても動作します。また、式エディタではどのようなデータ型が返るのかが判別できません。上図の例において、数値フィールドで押下された *F7* に対してロジックユニットが実行された場合、結果は無効になります。

このような問題を回避するため、コントロール名や **HandledCtrl()** 関数と組み合わせて使用するようにしてください。

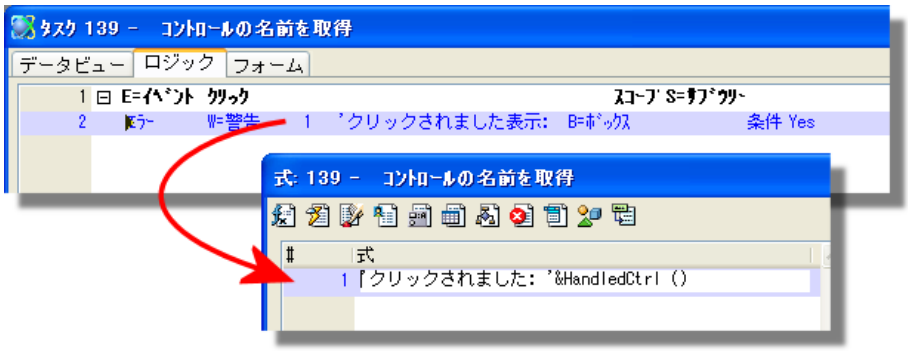
コントロール修正イベントをトリガとした場合、フィールドの修正を行った時点でロジックユニットが実行されるため、ここで **EditGet()** 関数を使用すると入力内容をチェックすることができます。

参照： 「イベントが発行されたコントロールを識別するには」（231 ページ）
「特定のコントロールにパークしている時にのみ実行されるイベントロジックユニットを定義するには」（232 ページ）

イベントが発行されたコントロールを識別するには

イベントロジックユニットは、それを処理するタスクより高いレベルで存在することができます。このよい例として、メインプログラム内でのイベントロジックユニットが挙げられます。これはプロジェクト内のどのタスクでも利用することができます。ユーザがどのコントロールでパークしているかを知るためには、**HandledCtrl()** 関数を使用する必要があります。

HandledCtrl() 関数を使用する



1. 式エディタを開きます。
2. 式を入力します。

```
HandledCtrl()
```

Ha を入力し *Ctrl+Space* を押下すると、オートコンプリートのポップアップメニューが表示されるのでここから関数を選択できます。

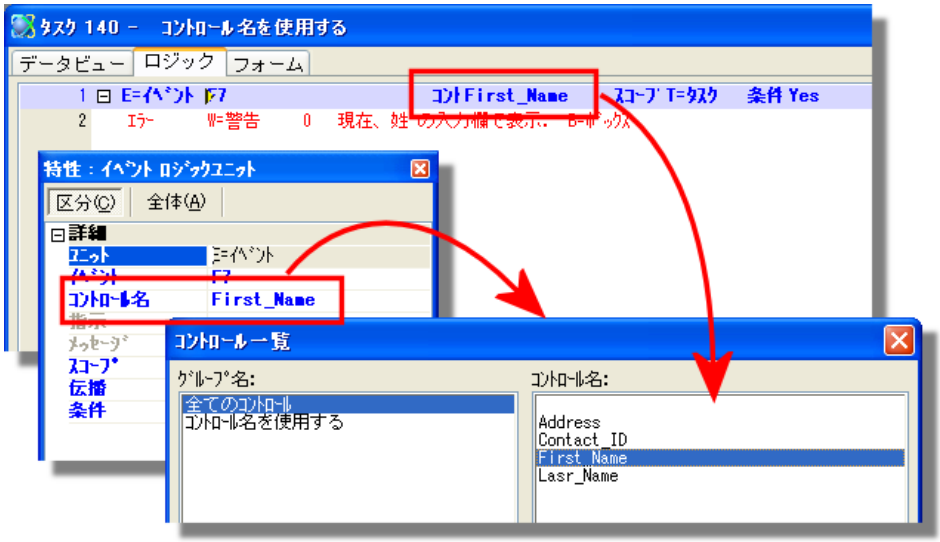
これで、イベントロジックユニット内で決定をするためにコントロール名を使用することができます。例えば、特定のヘルプファイルを呼び出すために、コントロール名の文字に対応した汎用的なヘルプシステムを持つことができます。

参照： 「特定のコントロールにパークしている時にのみ実行されるイベントロジックユニットを定義するには」（232 ページ）

特定のコントロールにパークしている時にのみ実行されるイベントロジックユニットを定義するには

イベントロジックユニットを使用することで、イベントを簡単に受け取ることができます。特定のコントロールにパークされている場合のみ、イベントが実行される必要があるのであれば、イベントのコントロール名特性を使用することができます。

イベントのコントロール名特性を使用する



イベントロジックユニットのコントロール名特性から、ズーム（F5 またはダブルクリック）してイベントが実行されるコントロール名を選択します。

この例では、システムイベントの F7 に対応して First_Name という名前のコントロールを選択しています。従って、ユーザが *F7* を押下すると、ユーザが First_Name という名前のコントロールにパークしている場合のみロジックユニットが実行されます。

ヒント：*複数のコントロールに対して 1 つのイベントハンドラハンドルを定義する必要がある場合は、イベントハンドラで* **HandledCtrl()** *関数を使用します。詳細は、「イベントが発行されたコントロールを識別するには」（231 ページ）を参照してください。*

第 12 章 : 日付と時刻

現在の日付を取得するには

Magic で現在の日付を取得するには、**Date()** 関数を使用します。**Date()** 関数は、Magic が現在実行中の PC のシステム日付を返します。

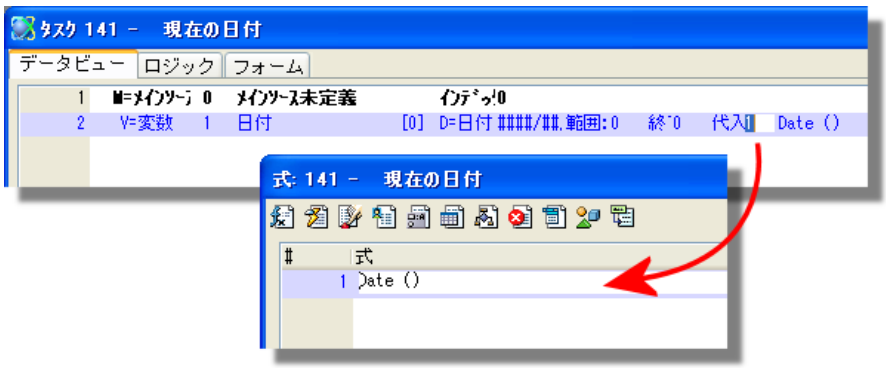
注 : この関数は、**MDate()** 関数とよく似ています。この関数は、システム日付ではなくユーザが設定可能なログイン日付を返します。この関数が使用される場合もあります。例えば、経理システムのようにユーザに対して登録日付の変更を可能にさせる必要がある場合に使用します。

日付の格納形式

Magic での日付は、0001/01/01 以降からの日数を表す数値で格納されています。変換はすべて自動的に行われるため、開発者が意識する必要はありませんが、内部処理上は数値として扱われるということです。従って、5 日を追加する場合、もとの日付から 5 日を進めることになります。月末の日付やうるう年のような問題に関して考慮する必要はありません。

日付がデータソース内に格納される場合、記憶形式はデータソース定義で設定された特性によって決定されます。データがレポートライタなどのような他のツールと共有している場合、日付フィールドの定義が、他のツールで処理が可能であることが重要になります。

Date() 関数を使用する



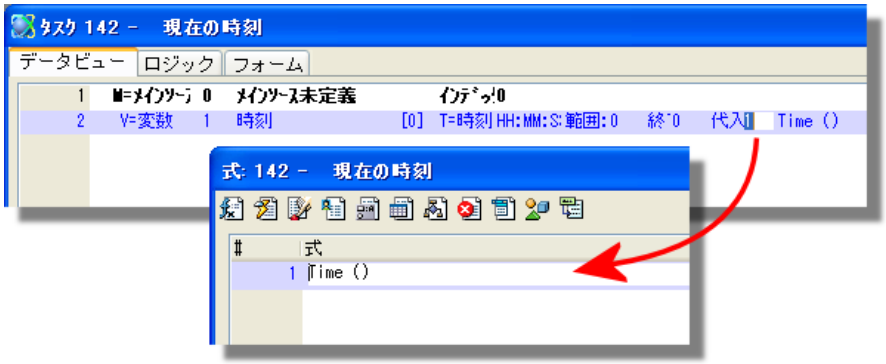
ここで示されるように、**Date()** 関数は**代入**カラムでよく使用されます。これは、タスクが起動された時に現在の日付を表示させたり、レコードの新規作成時にタイムスタンプを設定する場合に便利です。

この場合、式には、**Date()** と入力するだけです。実行時、関数は現在の日付を返します。

現在の時刻を取得するには

Magic で現在の時間を取得するには、**Time()** 関数を使用します。**Time()** 関数は、Magic が実行している PC のシステム時間を返します。

Time() 関数を使用する



ここで示されるように、**Time()** 関数は**代入**カラムでよく使用されます。これは、タスクが起動された時に現在の時刻を表示させたり、レコードの新規作成時にタイムスタンプを設定する場合に便利です。

この場合、式には、**Time()** と入力するだけです。実行時、関数は現在の時刻を返します。

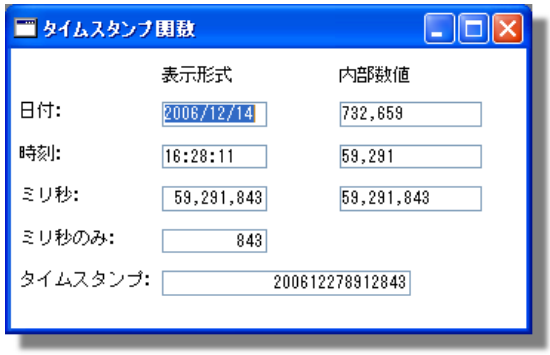
現在の時間をミリ秒単位で取得するには

Time() 関数は、0 時以降の秒数を返します。秒数で格納されてはいますが、通常は **HH : MM : SS** の書式で表示されます。

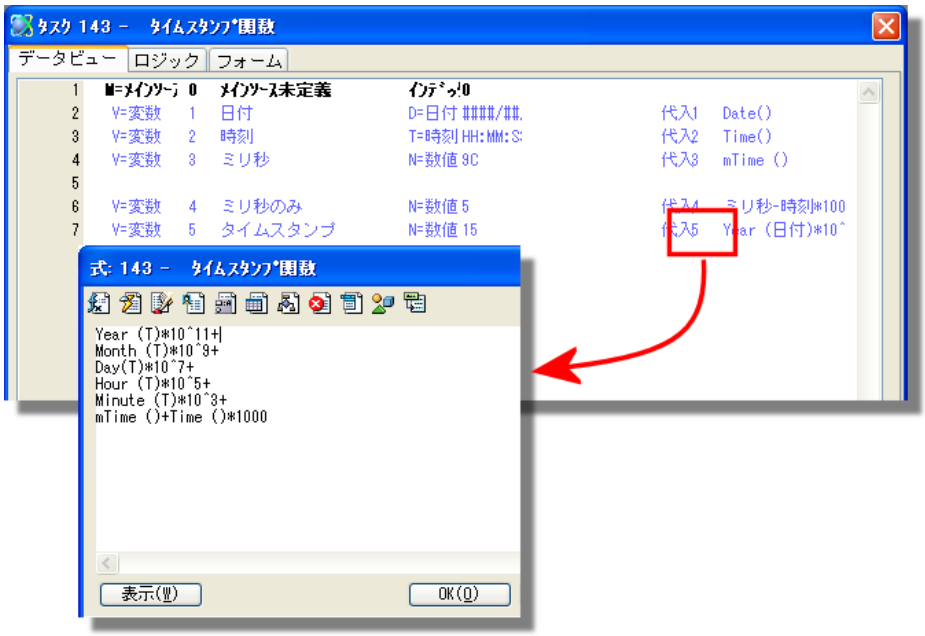
しかし、タイムスタンプを作成する上で細かい単位が必要な場合があります。**mTime()** 関数は、このような場合に使用します。この関数は、0 時以降のミリ秒数を返します。

ミリ秒数を表示させるための自動化された機能はありません。ミリ秒を取得するには以下の式を使用します。

```
mTime() - Time() * 1000
```



mTime() 関数を使用してタイムスタンプを作成する



ここに示された式を使用して数値データとしての 日付 / 時刻のタイムスタンプを作成することができます。この場合、18 桁の数値コードをを作成します。

```
YYYYMMDDHHMMSSmmm
```

文字列でタイムスタンプを作成することもできますが、数値で作成した方がバイト数が少なくすみます。

日付データの計算を行うには

前述のように、Magic における日付データは 0 年以降の日付を表す数値が格納されています。従って、日付を加算するには、日数を追加するだけで可能になります。例えば：

```
Date() + 5
```

この式では、現在から 5 日後の日付が返ります。

しかし、年数や月数および日数をもとに加算処理を行う場合は、**AddDate()** 関数を使用します。この関数は、以下の４つのパラメータを使用します。

AddDate(Date, Years, Months, Days)	
Date	加算対象の日付データ
Years	加算する年数を表す数値
Months	加算する月数を表す数値
Days	加算する日数を表す数値

例えば、以下のような式を作成した場合：

```
AddDate(B,0,-1,0)
```

指定された日付から 1ヶ月が減算されます。従って、例えば 2008 年 7 月 6 日をもとに計算を行った場合、**AddDate()** 関数は 2008 年 6 月 6 日を返します。

時刻データの計算を行うには

Magic における時刻データは 0 時以降の日付を表す数値が格納されています。従って、時刻を加算するには、秒数を追加するだけで可能になります。例えば：

```
Time() + 5
```

この式では、現在から 5 秒後の時刻が返ります。

しかし、時数や分数および秒数をもとに加算処理を行う場合は、**AddTime()** 関数を使用します。この関数は、以下の４つのパラメータを使用します。

AddTime(Time, Hours, Minutes, Seconds)	
Time	加算対象の時刻データ
Hours	加算する時数を表す数値
Minutes	加算する分数を表す数値
Seconds	加算する秒数を表す数値

例えば、以下のような式を作成した場合：

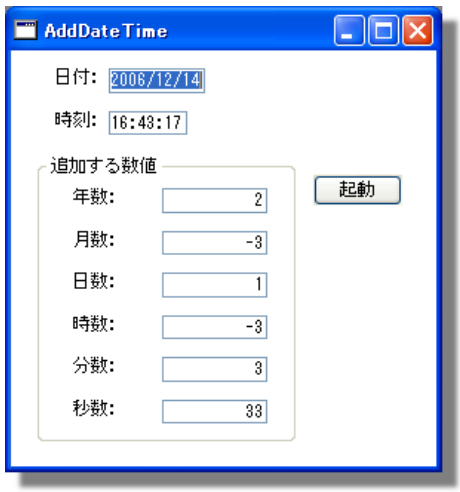
```
AddTime(B,0,-1,0)
```

指定された日付から 1 時間が減算されます。従って、例えば 14:08:03 をもとに計算を行った場合、**AddTime()** 関数は 13:08:03 を返します。

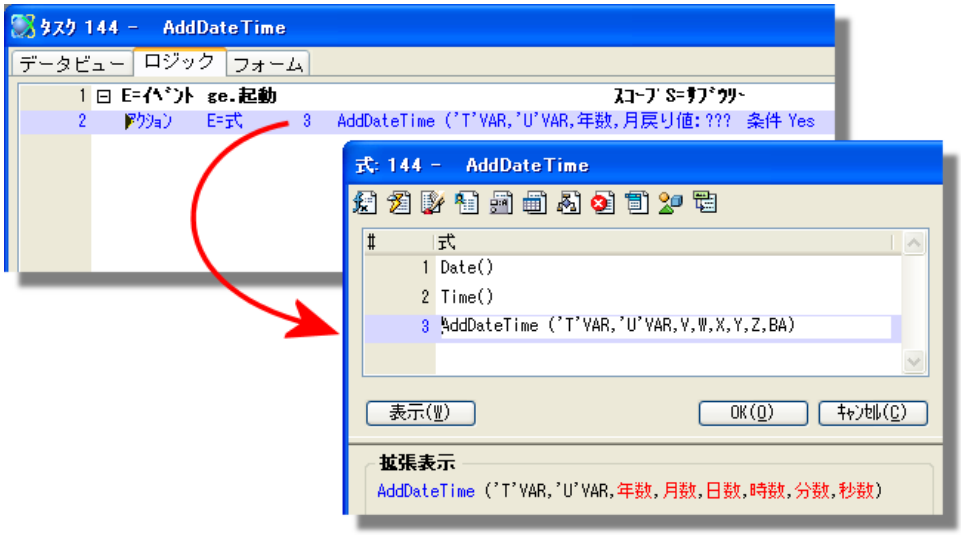
日付と時刻を組み合わせたデータを加算するには

日付と時刻の項目の値に対する加算処理は、さまざまな処理で使用されます。例えば、22:00:00 に業務を開始した作業者に対する作業時間を計算する場合を考えてみます。開始時間から 8 時間を追加した場合、それに応じて日付も加算する必要があります。

このような処理に対して、Magic には 1 回で日付と時間を扱うことができる関数があります。**AddDateTime()** 関数は、日付と時間（年、月、日、時、分、秒）を指定することで、まとめて加算 / 減算の処理を行うことができます。



AddDateTime() 関数を使用する



AddDateTime() 関数は以下の構文で指定します。

AddDateTime(*DateVariable*, *TimeVariable*, *Years, Months*, *Days, Hours*, *Minutes*, *Seconds***)**

パラメータ：

- *DateVariable* ：更新対象の日付データ
- *TimeVariable*：更新対象の時刻データ
- その他のパラメータは、日付や時刻の各部分を表す数値です。
  正の数値を指定すると加算、負の数値を指定すると減算になります。

項目の指定方法について

上記の例では、最初の２つのパラメータの項目で **VAR** リテラルを使用しています。つまり、クォーテーションで囲まれた項目番号の後に VAR という文字が続いています（'A'VAR と 'B'VAR）。これは、Magic がアドレス参照で項目を指定するための書式です。この指定は、式の中で項目を更新する場合に指定する必要があります。通常、Magic は **項目更新**処理コマンドで項目を更新しますが、この場合、同時に２つの項目を更新しているため、このような参照指定が必要になります。

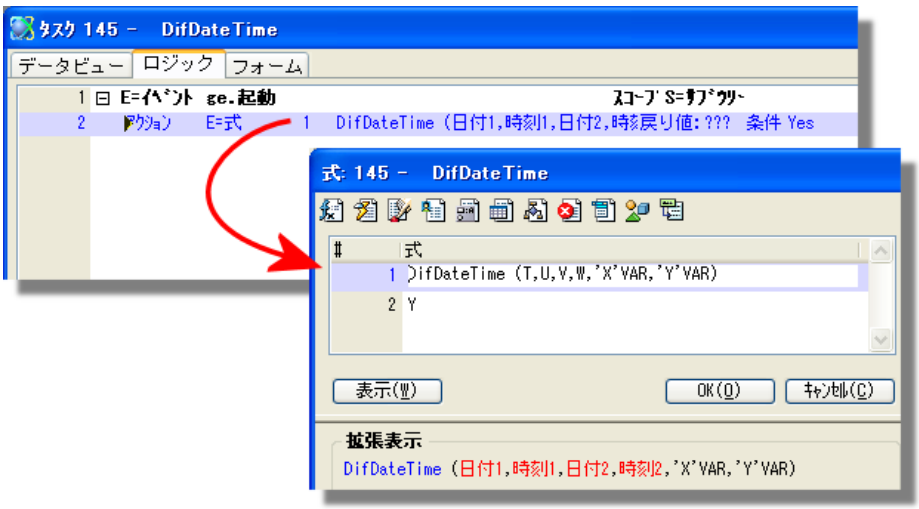
２つの日付 / 時刻データの差分を計算するには

2 つの日付 / 時刻データの差分を計算する処理は、さまざまな処理で使用されます。

このような処理に対して、Magic には 1 回で日付 / 時間の差分を計算することができる関数があります。**DifDateTime()** 関数は、以下の例のように２つの日付 / 時刻の組み合わせデータの差分を計算することができます。

DifDateTime() 関数は、差分の日数と秒数を返します。この秒数を時間データとして扱い、**Hour()**、**Minute()** そして **Second()** の各関数を使用して、時数 / 分数 / 秒数に変換することができます（「指定された時刻の時 / 分 / 秒の各部分を取得するには」（247 ページ）を参照してください）。

DifDateTime() 関数を使用する



DifDateTime() 関数は以下の構文で指定します。

DifDateTime(*Date1*, *Time1*, *Date2*, *Time2*, *DaysVariable*, *SecondsVariable***)**

パラメータ：

- *Date1* と *Time1*：最初の日付 / 時刻の組み合わせデータ
- *Date2* と *Time2*：２番目の日付 / 時刻の組み合わせデータ
- *DaysVariable*：差分の日数データを表す数値
- *SecondsVariable*：差分の秒数データを表す数値

項目の指定方法について

上記の例では、最初の２つのパラメータの項目で **VAR** リテラルを使用しています。つまり、クォーテーションで囲まれた項目番号の後に VAR という文字が続いています（'E'VAR と 'F'VAR）。これは、Magic がアドレス参照で項目を指定するための書式です。この指定は、式の中で項目を更新する場合に指定する必要があります。通常、Magic は **項目更新**処理コマンドで項目を更新しますが、この場合、同時に２つの項目を更新しているため、このような参照指定が必要になります。

指定された日付に対応する月の最初の日付を取得するには

帳票処理のための日付範囲を求めるために、月の最初の日付を取得する必要がある場合があります。**BOM()** 関数を使用することで、このようなことが実現できます。

BOM() 関数を使用する

BOM() 関数の構文は以下の通りです。

BOM(*date***)**

パラメータ：

- *date* ：日付項目（直接日付を指定するか日付データを返す式を指定）

この関数は、月の最初の日付を返します。

参照： 「指定された日付に対応する月の最後の日付を取得するには」（242 ページ）

指定された日付に対応する年の最初の日付を取得するには

帳票処理のための日付範囲を求めるために、年の最初の日付を取得する必要がある場合があります。**BOY()** 関数を使用することで、このようなことが実現できます。

BOY() 関数を使用する

BOY() 関数の構文は以下の通りです。

BOY(*date***)**

パラメータ：

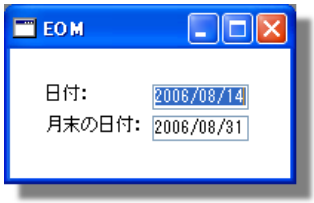
- *date* ：日付項目（直接日付を指定するか日付データを返す式を指定）

年の最初の日付を返します。

参照： 「指定された日付に対応する年の最後の日付を取得するには」（243 ページ）

指定された日付に対応する月の最後の日付を取得するには

帳票処理のための日付範囲を求めるために、月の最後の日付を取得する必要がある場合があります。**EOM()** 関数を使用することで、このようなことが実現できます。



EOM() 関数を使用する利点の一つは、うるう年を正しく処理することです。

EOM() 関数を使用する

EOM() 関数の構文は以下の通りです。

EOM(*date***)**

パラメータ：

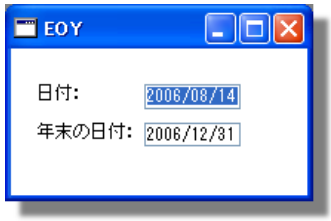
- *date* ：日付項目（直接日付を指定するか日付データを返す式を指定）

月の最初の日付を返します。

参照： 「指定された日付に対応する年の最初の日付を取得するには」（241 ページ）

指定された日付に対応する年の最後の日付を取得するには

帳票処理のための日付範囲を求めるために、年の最後の日付を取得する必要がある場合があります。**EOY()** 関数を使用することで、このようなことが実現できます。



EOY() 関数を使用する

EOY() 関数の構文は以下の通りです。

EOY(*date***)**

パラメータ：

- *date* ：日付項目（直接日付を指定するか日付データを返す式を指定）

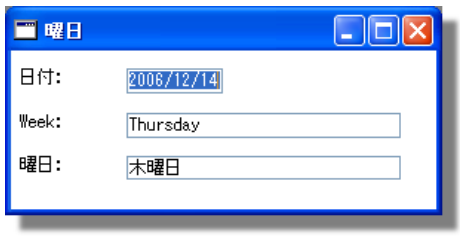
年の最後の日付を返します。

参照： 「指定された日付に対応する年の最初の日付を取得するには」（241 ページ）

指定された日付に対応する曜日名を取得するには

指定された日付が何曜日になるかを取得することは難しいことかもしれません。しかし、**CDOW()** 関数を使用することで、簡単に取得することができます。

日本語で曜日を表示させる場合は、**JCDOW()** 関数を使用します。



CDOW() 関数を使用する

CDOW() 関数の構文は以下の通りです。

CDOW(*date***)**

パラメータ：

- *date* ：日付項目（直接日付を指定するか日付データを返す式を指定）

曜日を表す文字列を返します。

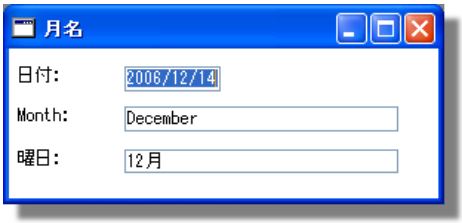
ヒント : *週単位の日付計算を行う必要がある場合は、**DOW()** 関数を使用することができます。この関数は週単位で日数（日曜日を 1 とした、日数）が返ります。プログラム内での演算用に使用するには、こちらの方が便利です。*

参照 : 「日付書式」（249 ページ）

指定された日付に対応する月名を取得するには

指定された日付に対応する月名を取得する場合は、**CMonth()** 関数を使用します。

日本語で月名を表示させる場合は、**JMonth()** 関数と **Month()** 関数を組み合わせたり、**Month()** 関数で月数を求めて表示させるようにします。



CMonth() 関数を使用する

CMonth() 関数の構文は以下の通りです。

CMonth(*date***)**

パラメータ：

- *date* ：日付項目（直接日付を指定するか日付データを返す式を指定）

月名を表す文字列を返します。

ヒント : *月数の計算を行う必要がある場合は、**Month()** 関数を使用することができます。この関数は月数を数値で返します。プログラム内での演算用に使用するには、こちらの方が便利です。*

参照 : 「指定された日付の年 / 月 / 日の各部分を取得するには」（246 ページ）

指定された日付の年 / 月 / 日の各部分を取得するには

プログラム内で日付データを使用する場合、日付データを分離する必要があるかもしれません。例えば、その年の 1ヶ月間の全レコードを要約するような場合が考えれます。

日付データを分離するには、以下の３つの関数があります。

- **Day(*date*)**：その月の日数を表す数値を返します。
- **Month(*date*)**：月数を表す数値を返します。
- **Year(*date*)**：年数を表す数値を返します。

パラメータ：

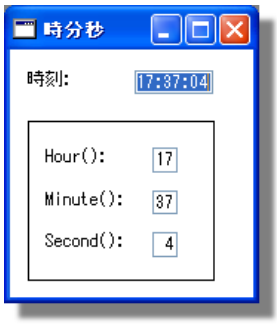
- *date* ：日付項目（直接日付を指定するか日付データを返す式を指定）

ヒント：*月の名前や曜日の名前を表示させたい場合は、***CMonth()** *関数や***CDOW()** *関数を使用します。これらは、文字列で名前を返します。*

参照： 「指定された日付に対応する曜日名を取得するには」（244 ページ）

指定された時刻の時 / 分 / 秒の各部分を取得するには

時刻データを時数、分数、または秒数に分ける場合、計算処理を実行して値を取得することで実現できます。しかし、このような処理を簡単に行うことのできる関数があります。**Hour()**、**Minute()**、および **Second()** 関数を使用することで、分離された時刻データが返ります。



時刻データを分離するには、以下の３つの関数があります。

- **Hour(*time*)**：時数を表す数値を返します。
- **Minute(*time*)**：分数を表す数値を返します。
- **Second(*time*)**：秒数を表す数値を返します。

パラメータ：

- *time*：時刻項目（直接時刻を指定するか時刻データを返す式を指定）

ヒント : ミリ秒で取得したい場合は、**MTime()** を関数を使用します。

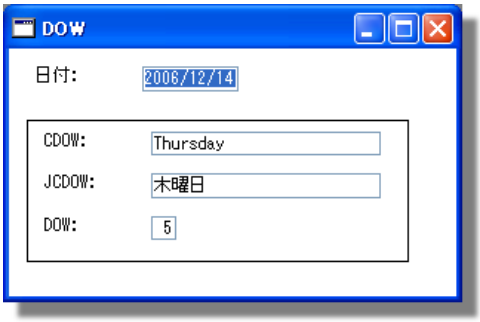
参照 : 「現在の時間をミリ秒単位で取得するには」（235 ページ）

指定された日付の週単位の日数を取得するには

プログラムの内容によっては、週単位で処理を行う必要があるかもしれません。例えば、金曜日に給料が配達される場合、指定された日付が金曜日か否かを確認する必要があるはずです。

値が数値で返るようであれば、、曜日を確実にチェックできます。特に、アプリケーションが他言語で実行されるような場合、このオプションはスペルや大文字／小文字の問題を低減させることになります。

指定された日付から週番号を取得するには、**DOW()** 関数を使用します。



DOW() 関数を使用する

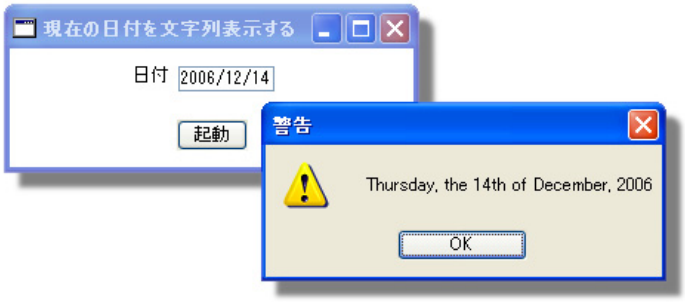
DOW(*date***)** は、週単位での日数を返します。返される値は、1（日曜日）から 7（土曜日）までです。

例えば、2007/08/19 は日曜日なので **DOW(***'2007/08/19'DATE***)** は、1 を返します。

ヒント : *指定された日付けの曜日名（英語）を取得する場合は、**CDOW()** 関数を使用します。日本語の曜日名を取得する場合は、**JCDOW()** 関数を使用します。*

参照 : 「指定された日付に対応する曜日名を取得するには」（244 ページ）

日付データを文字列に変換するには



Magic における日付値は基本的には数値で、0001/01/01 以降の日数として扱われます。フォーム上の日付を表示する場合、コントロールの**書式**特性にもとづいて Magic が自動的に処理するため、変換処理は必要ありません。

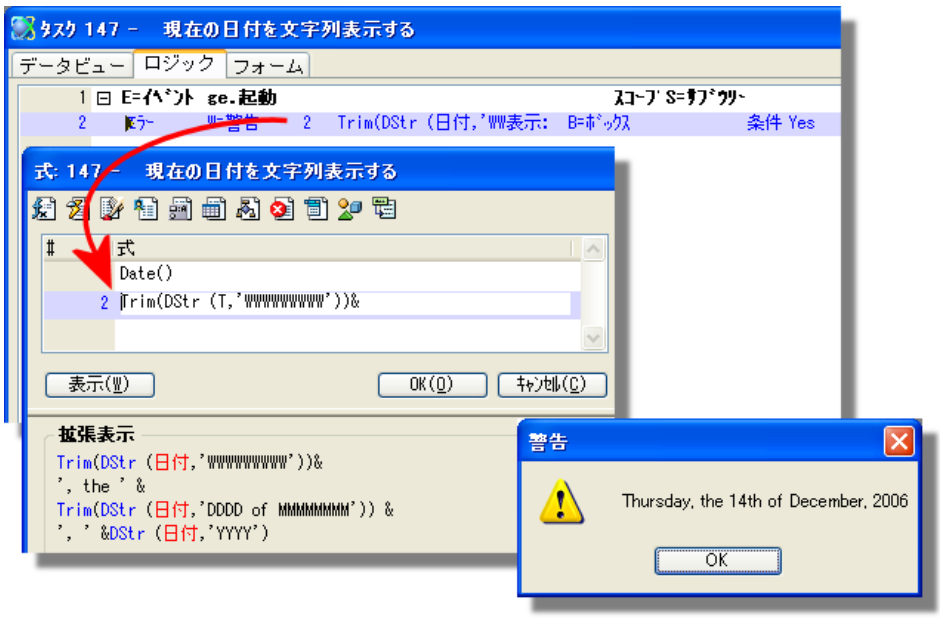
しかし、日付を文字列に含めたい場合（例えばエラー処理コマンドでのメッセージ）、日付を文字列に変換する必要があります。このような場合、**DStr()** 関数を使用して実現します。

日付書式

日付を表示する場合、Magic には複数の書式を指定することができます。日付の書式は、どのように表示されるかを指定します。書式は、日付を解釈する一種のテンプレートです。文字で示されたプレイスホールダは、どのように日付を解釈するかを示すために使用されます。例えば、2007 年 6 月 30 日は以下のように扱われます。

書式	どのように表示されるか	コメント
YY/MM/DD	07/06/30	
YYYY/MM/DD	2007/06/30	
YYYYMMDD	20070630	
YYYY-MM-DD	2007-06-30	
YYYY	2007	
YY	07	
MMMM	June	
DDD	181	ユリウス歴での年内の日数
DDDD	30th	
WWW	Sat	
WWWWWWWWW	Saturday	
##/##/####	Depends	設定→動作環境→国別指定→日付モードの指定に依存します。

DStr() 関数を使用する



DStr() 関数の構文は以下の通りです。

DStr(*date, picture***)**

パラメータ：

date ：日付項目（直接日付を指定するか日付データを返す式を指定）

picure：文字列に変換する場合の書式

この関数は、***picure*** で指定された書式の文字列を返します。

参照： 「文字列で格納された日付を日付データに変換するには」（251 ページ）

文字列で格納された日付を日付データに変換するには

通常 Magic は、日付変換処理を行います。すなわち、フォーム上に日付を表示したり、フォームの日付を読み込んだり、データソースから読み込む場合、変換処理を実行するプログラムを作成する必要がありません。しかし、文字列を日付として取得するためにその文字列を解析する必要がある場合、若干のプログラムを組む必要があります。このような場合、**DVal()** 関数を使用する必要があります。

DVal() 関数は文字列を解析して日付データに変換するために指定呈された書式を使用します。書式と文字列の整合性は、開発者に依存しています。例えば以下のようになります。

パラメータの例	結果
DVal(‘07/06/30’DATE, ‘YY/MM/DD’)	2007,June 30
DVal(‘06/30/07’DATE, ‘DD/MM/YY’)	不正な結果

参照 : 「日付データを文字列に変換するには」（249 ページ）

［このページは意図的に空白にしています。］

第 13 章 : GUI の処理

マウスを使用した場合のみコントロールへのパークを可能にするには

デフォルトでは、オンラインフォーム上のフィールドは、*Tab* によってカーソルをパークさせることができます。しかし、マウスを使用した場合のみアクセス可能にしたい場合は、コントロールの TAB で移動特性を *No* に設定します。

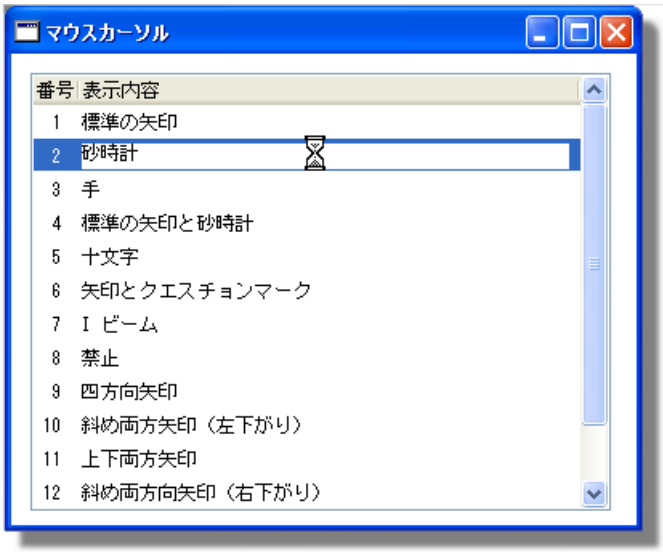
式で指定することで、必要に応じて *Tab* 移動を可能にすることもできます。例えば、表示されている文字列をコピーする場合のみパークさせるというような制御が可能です。

TAB で移動特性を設定する

1. 変更したいコントロールを選択します。必要であれば、*Ctrl+* クリックまたはラバーバンドを使用して、複数のコントロールを選択することで同時に変更することもできます。
2. コントロール特性（*Alt+Enter*、すでに開いている場合は特性シートをクリック）に移動します。
1. TAB で移動特性の値を *No* に設定します。式で指定する場合は、右側に移動し、ズーム（*F5*）して式エディタを開きます。パークさせたい場合に True が返る式を入力します。

   これで、ユーザは、フィールドに *Tab* 移動することはできなくなりますが、マウスでクリックすることは可能です。

マウスカーソルのアイコンを変更するには



通常の Windows アプリケーションと同じように Magic もマウスカーソルを制御しています。しかし、必要に応じてデフォルトの表示内容を変更することができます。

Magic では、14 個の異なるアイコンのセットから選択できます。これらのアイコンは、Windows 環境で標準的に持っているものです。例えば、標準の矢を大きくしたり、異なるシンボルを使用したり、ユーザ毎に独自の Windows 設定を変更することができます。

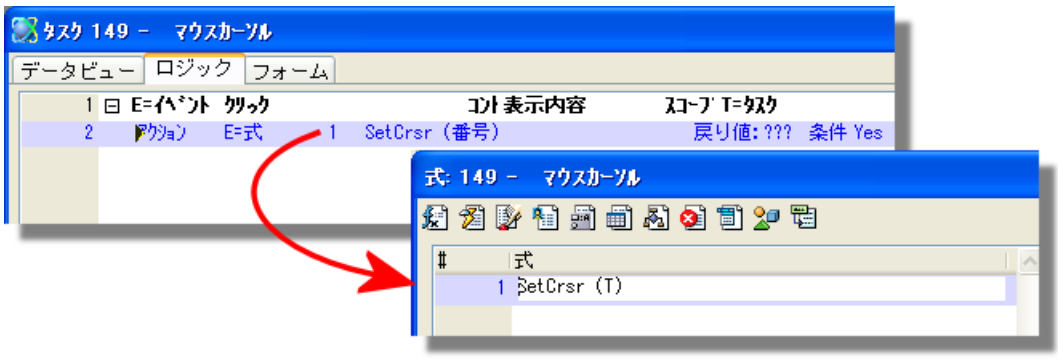
使用するアイコンを変更するには、**SetCrsr()** 関数を使用します。構文は以下のとおりです。

SetCrsr(*number*)

パラメータ：

- ***number***：1 ～ 14 の数値です。

SetCrsr() 関数を使用する



1. カーソル形状の変更処理を実行させたいロジックユニットに移動します。上記の例では、ユーザがカーソルタイプのリストをクリックすることで、カーソルの形状が変化するようになります。
2. アクション処理コマンドを作成します。
3. 以下の式を設定します。

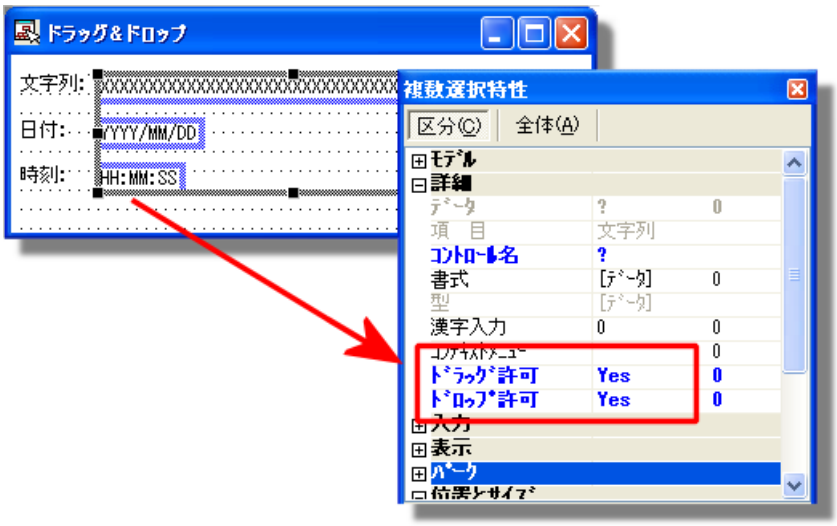
   SetCrsr (number)

   パラメータの ***number*** は変更したいカーソルの形状を現す数値です。1 ～ 14 の数値を直接指定すること可能ですが、この例では、数値型項目を使用しています。

   これで、ロジックユニットが実行されると、カーソルの形状が変化します。

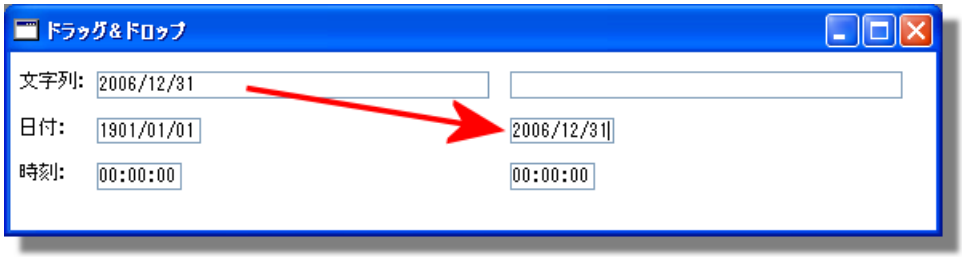
注： アイコンを変更させた後は、デフォルトの形状に戻すようにリセット処理を組み込むようにしてください。

ドラッグ & ドロップでデータをコピーするには



アプリケーションでドラッグ＆ドロップ処理を行うには、**ドラッグ許可**と**ドロップ許可**の各特性を *Yes* に設定する必要があります。

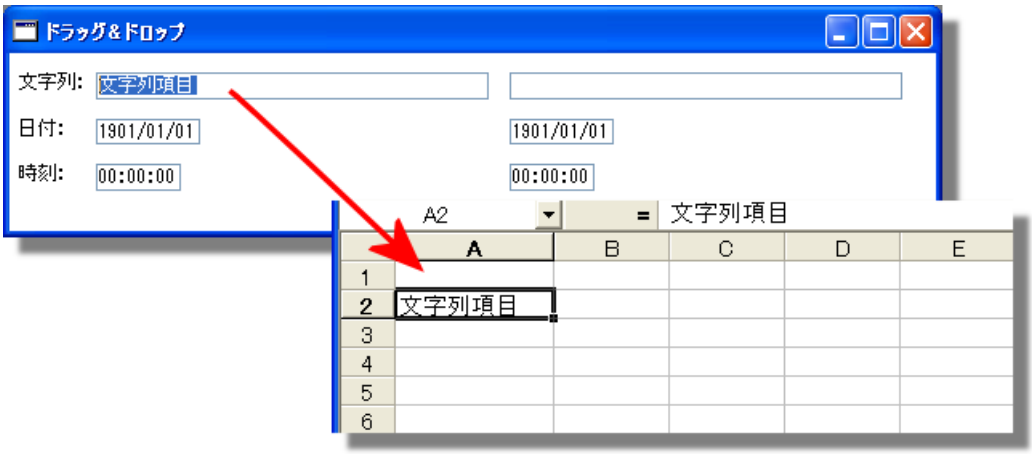
すべて設定されていれば、データをフィールドでドラッグすることがでます。



例えば、この例では、ドラッグ＆ドロップ処理を使用して日付フィールドに文字列のデータをコピーしています。

Magic でより高度なドラッグ＆ドロップの編集処理を行うには、ドラッグ＆ドロップ関数やハンドラを使用する必要があります。ドラッグまたはドロップ処理が発生した場合、**ドラッグ開始**と**ドロップ**に対する**イベント**ロジックユニットによってこれらを処理することができます。**DragSetData** と **DropFormat** 関数を使用することで、ドラッグデータの書式に関するより細かい制御が可能になります。

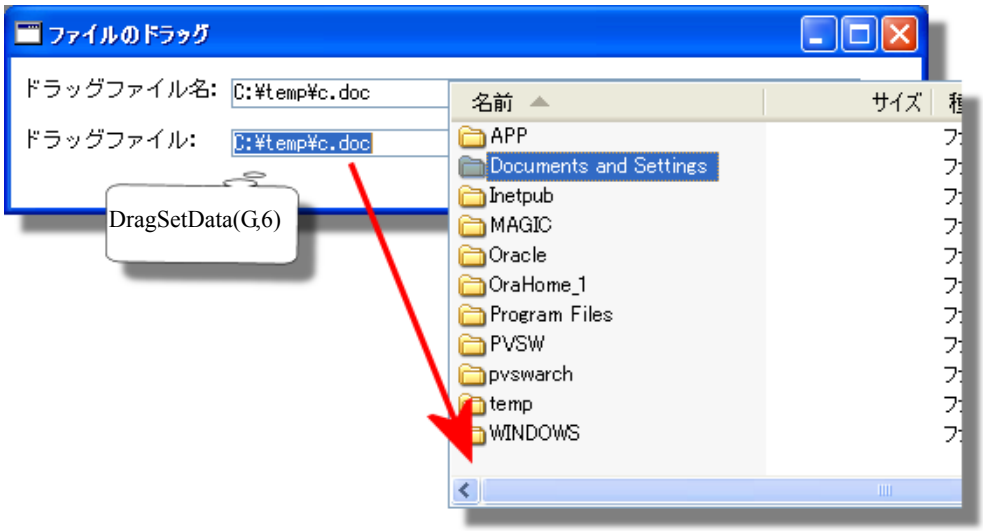
ドラッグ & ドロップを使用して外部アプリケーションにデータを移動するには



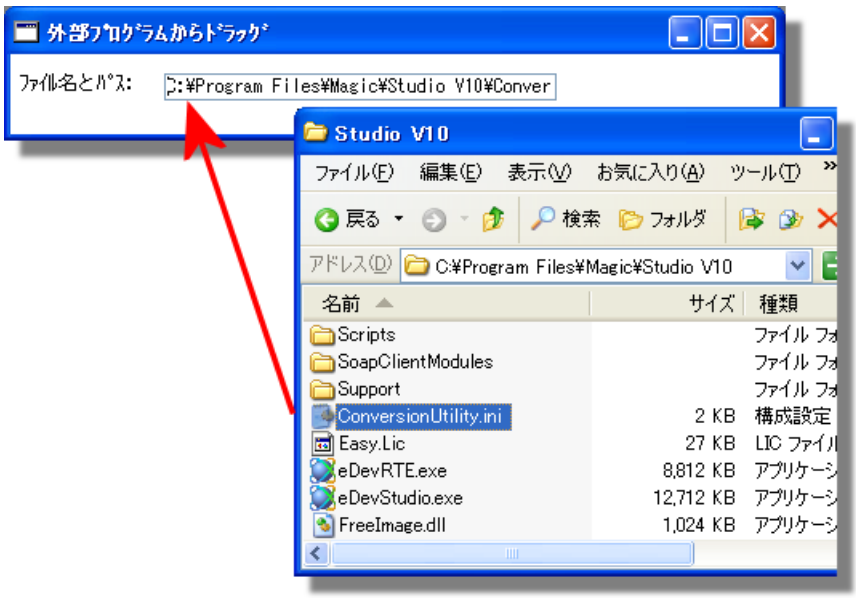
外部アプリケーションへのドラッグ & ドロップを有効にするには、ドラッグしたいコントロールの**ドラッグ許可**特性を ***Yes*** に設定する必要があります。

Magic でより高度なドラッグ&ドロップの編集処理を行うには、ドラッグ&ドロップ関数やハンドラを使用する必要があります。ドラッグまたはドロップ処理が発生した場合、**ドラッグ開始**と**ドロップ**の各**イベント**ロジックユニットがこれらを取得することができます。**DragSetData** と **DropFormat** 関数を使用することで、ドラッグデータの書式に関するより細かい制御が可能になります。

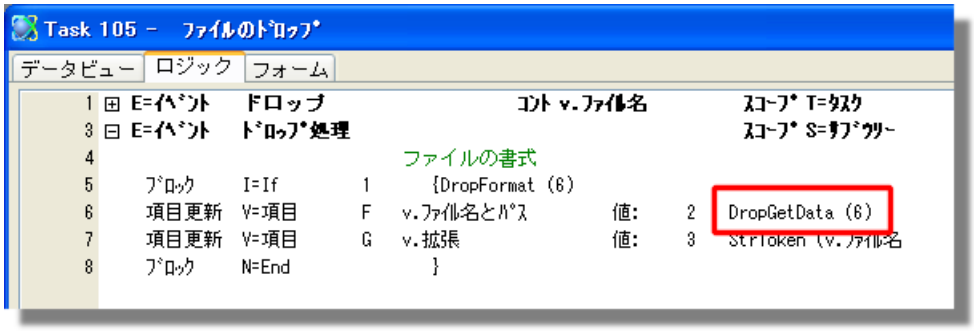
例えば、ファイルを参照ウィンドウ内に**ドラッグ**したい場合、ファイルのテキスト名をドラッグすることはできません。参照プログラムに対して、**ドラッグ**するものがファイル全体であることを指定する必要があります。このような処理を実行するには、**ドラッグ開始**の**イベント**ロジックユニット内で **DropSetData()** 関数を使用します。



ドラッグされたファイルからファイル名を取得するには



Windows からファイルを文字型フィールドにドラッグすると、フルパスのファイル名が自動的に取得されます。ファイル参照ウィンドウ内で２つのファイルを選択した場合、両方のファイル名をカンマ区切りで取得することができます。

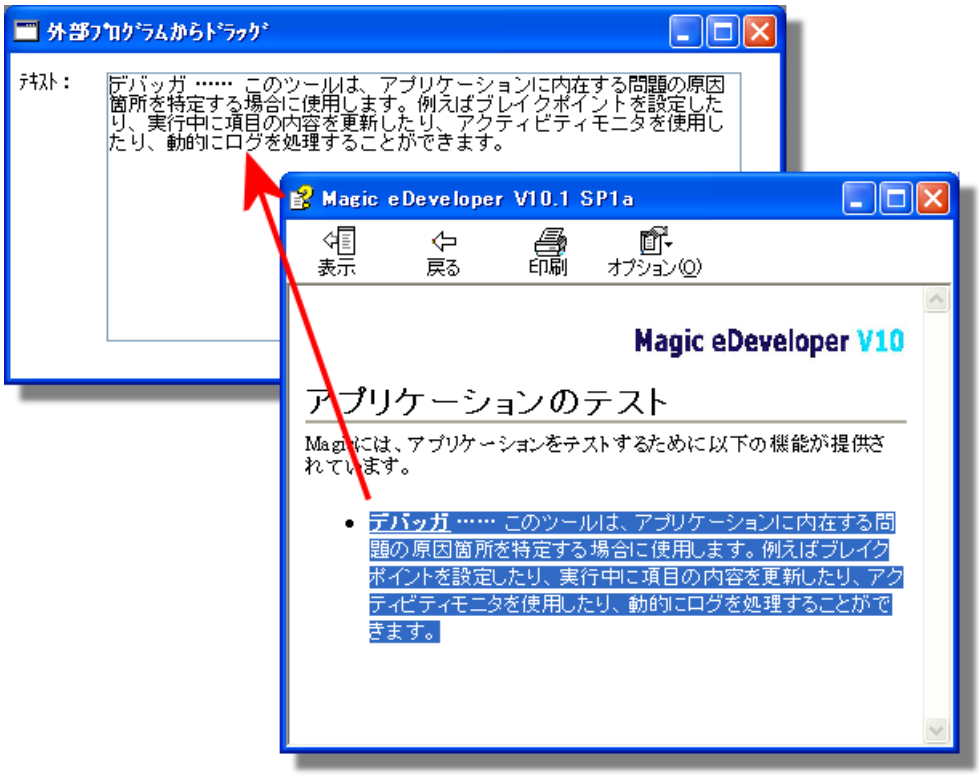


手動でこのファイル名を取り出すには、**DropGetData()** 関数を使用します。

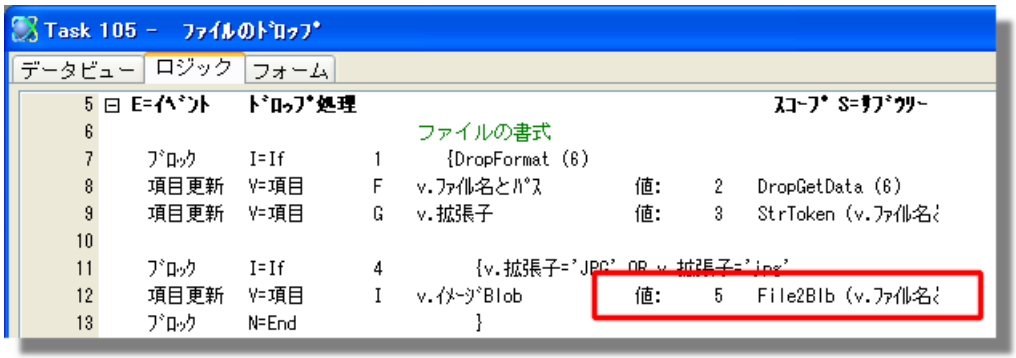
上記のプログラム例では、以下のように関数を使用しています。

- **DropFormat(*6*)**：ファイルがドラッグされていることをチェックしています。
- **DropGetData(*6*)**：項目を更新することでファイルの名前を取得しています。

外部アプリケーションから Magic にデータをドラッグするには

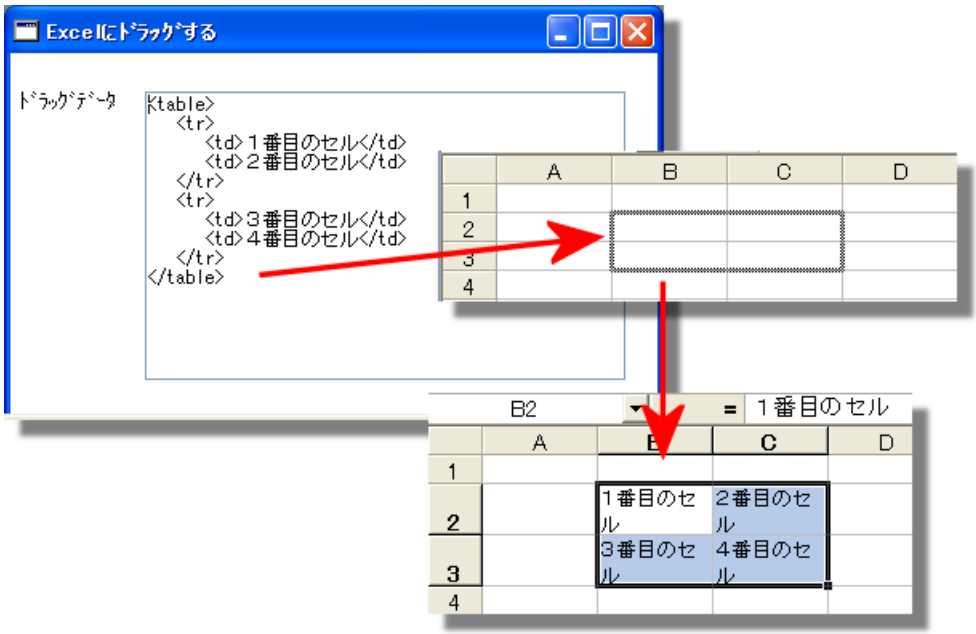


テキストコントロールの**ドロップ許可**特性が *Yes* に設定されている場合、他のアプリケーションから Magic にテキストを**ドラッグ**することができます。この動作の内容の善し悪しは、データを**ドラッグ**するアプリケーションの内部処理に依存します。これは、Windows 内のドラッグ & ドロップ処理と同じように、２つのフィールド間の書式を合わせる必要があるからです。

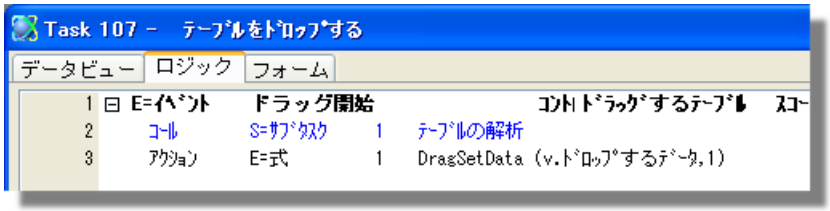


様々な状況において、外部データの書式を処理するために **DropGetData()** 関数を使用する必要があります。例えば、Magic のフィールド内に JPG ファイルを Windows エクスプローラからドラッグすることを想定してください。**DropGetData(*6*)** を使用することでファイル名を取り出すことができます。ファイルの拡張子をチェックし、それが JPG ファイルであると識別された場合、画像を表示させるにはファイルを BLOB に変換する必要があるため、**File2Blb()** 関数を使用します。

Magic から Excel にテーブルのデータをドラッグするには



Magic から Excel にデータをドラッグした場合、Excel はその書式を認識し、それに対応したデータとして処理します。この例では、複数のテーブルデータを HTML 形式に変更することで、テキストを Excel にドラッグしています。Excel はこのテキストを 4 つのセルを持つテーブルとして認識し、それに応じてデータがドロップされています。



ユーザに対し、Magic のデータソースの内容をドラッグすることを許可する場合、Magic のデータを文字列フィールド内に HTML または XML 形式で格納する必要があります。そして、ドラッグするテキストデータを送るために **DragSetData()** 関数を使用します。

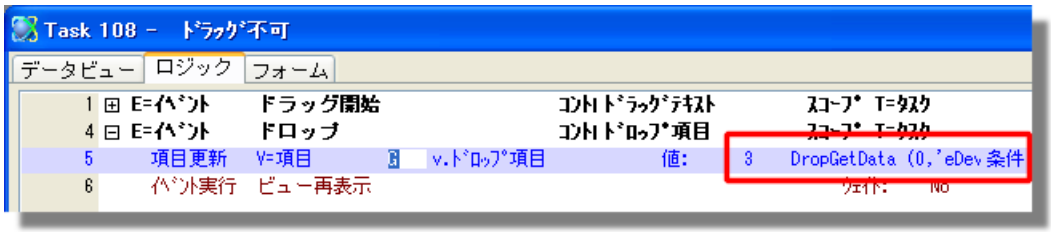
外部アプリケーションがデータを取得できないように Magic 内だけでドラッグ＆ドロップを許可するには

外部アプリケーションは、処理に必要なデータフォーマットと整合性がとれない限り**ドロップ**されたデータを読むことはできません。アプリケーションが指定されたデータフォーマットを認識していない場合、**ドロップ**は許可されません。

従って、例えば、**フォーマット 0** の「**ユーザ定義フォーマット**」を使用した場合、どの外部アプリケーションもデータフォーマットを認識することができず、**ドロップ**も許可されません。

この例において、フォーマットを 0 に設定するために、**DragSetData()** 関数を使用しています。また**伝播**特性を ***No*** に設定しているため、組み込まれているドラッグ機能も働きません。

この設定を行った場合、**ドロップ**ハンドラ内で同じフォーマット指定が使用されるまで、データは Magic の他のフィールドに**ドラッグ**することはできません。



複数のコントロールを一緒にドラッグするには

Magic のデフォルトのドラッグ処理は、１つのコントロールに対してのみ有効ですが、ロジックを作成することでテーブルコントロール上のレコードのような複数のコントロールに対してドラッグすることができます。

以下の手順によって、レコードをマークしてドラッグし、外部アプリケーションや Magic の別のコントロールにドロップすることができます。

複数のコントロールをドラッグできるようにする

1. ラインモード表示のタスクに文字型変数項目（A）を定義します。変数項目のサイズは、ドラッグされる文字列の総桁数分を指定します。この変数項目は、ドラッグバッファ内の文字列を連結させるために使用されます。
2. GUI 表示フォームにテーブルコントロールを配置し、ドラッグ許可特性を *Yes* に設定します。（マルチマーキング特性も *Yes* に設定してください。）
3. タスクにイベントロジックユニットを作成し、内部イベントのドラッグ開始イベントをトリガとして定義します。
4. ロジックユニット内に項目更新処理コマンドを定義し、変数項目（A）を初期化（空白で更新）するようにします。条件式は以下のようになります。
   ```
   MMCurr(0)=1
   ```
5. 再度項目更新処理コマンドを定義し、ドラッグ処理によってコピーしたいテーブル内のカラム（B）で、変数項目（A）を更新します。必要によってデータ変換を行い、既存の文字列に項目の内容を連結させるようにします。２つの値の間には、区切り文字として Chr(9)（=HT コード）を挿入します。この場合以下のような式になります。
   ```
   Trim(B)&ASCIIChr(9)&Trim(A)
   ```
6. アクション処理コマンドを定義し、**DragSetData()** 関数を実行します。パラメータとして、一番目に変数項目（A）を指定し、二番目にデータのタイプとして「1」を指定します。
   ```
   DragSetData(Trim(A),1)
   ```
7. テーブルをマルチマーキングした状態でドラッグし、他のアプリケーションにドロップすると、マークしたレコードの指定されたカラムデータが連結されて貼り付けられます。

「HT コード」で連結させることで、例えば Micrisoft Excel のようなスプレッドシートにドラッグすると、各カラムのデータが異なるセルに挿入させることができます。

データソース間での複数のレコードをドラッグするには

マークされたテーブルのレコードのインデックスをベクトルデータに格納することで、複数のレコードに対するドラッグ & ドロップが可能になります。これらのインデックスは、ドロップ時に相手のデータソースに貼り付けることができます。

以下の手順で、複数のレコードをドラッグできるようになります。

1. ユーザインタフェースを作成します。
2. ２つのロジックユニットを定義します。
3. ２つのリンクコマンドを定義します。

ユーザインタフェースの作成

最初に、同じ構成のデータソースを定義します。

一方のデータをソーステーブルとし、他方をデスティネーションテーブルとします。

この例では、データソースに文字型の２つのカラム（SN と Title）を定義します。データのインデックスセグメントに SN を割り当てます。

２つのテーブルを使用したプログラムを作成する

1. ラインモードでソーステーブルを表示させるプログラムを作成します。メインフォームのテーブルコントロールは、ドラッグ許可を *Yes* に設定します。また、サブフォームコントロールも配置します。
2. このプログラムのサブタスクとしてデスティネーションテーブルをラインモード表示するタスクを作成します。メインフォームのテーブルコントロールにリンクされた各カラムのドロップ許可を *Yes* に設定します。
3. 親タスクのフォームエディタを開き、サブフォームコントロールの接続先特性を *サブタスク*、*プログラム／タスク番号*特性を *1*（#2 で定義したサブタスク）に設定します。
4. 親タスクのデータビューエディタを開き、ベクトル型の変数項目を定義します。変数名を **Index Vector** とします。この場合、事前にモデルリポジトリに SN と同じ書式のモデルを定義しておいてください。ベクトル型の変数項目のセルモデル特性には、このモデルを指定します。

２つのロジックユニットを定義する

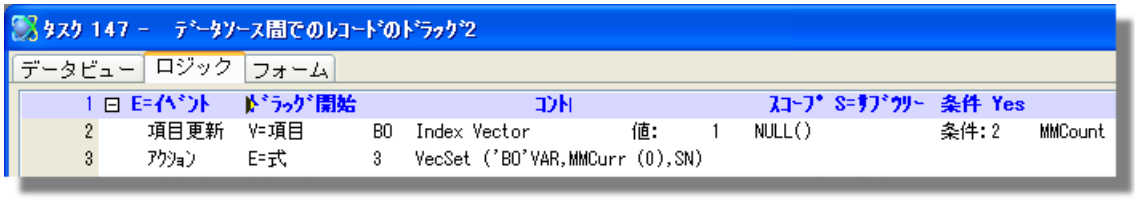
次に２つのイベントロジックユニットを定義します。一方はドラッグ開始イベントで、他方はドロップイベントをトリガにしたものです。

ドラッグ開始イベントのロジックユニットを定義する

1. 親タスクのロジックエディタを開き、イベントロジックユニットを作成します。ドラッグ開始イベントをトリガとして定義します。
2. 最初のレコードでロジックユニットが実行された場合に、変数「Index vector」を **NULL()** 関数で更新します。これは、更新条件を MMCurr (0)=1 とすることで可能です。
3. アクション処理コマンドを使用することで、マークされたレコードに対してベクトルデータに値を設定するようにします。

   VecSet（Index vector 項目 , MMCurr (0), ソーステーブルの Code 項目）

   これにより、マークされたレコードに対する Code 項目の内容を含むベクトルデータが返ります。

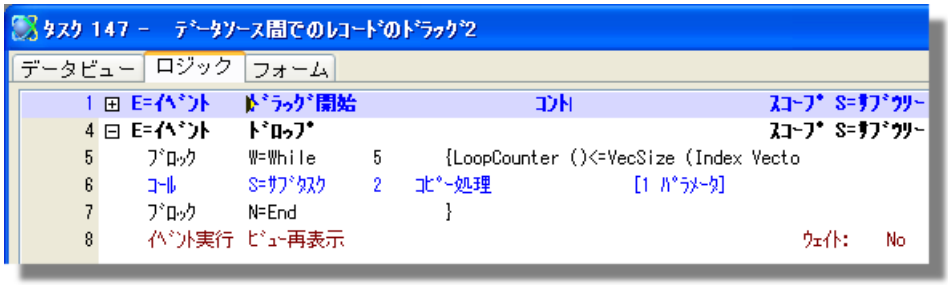


ドロップイベントのロジックユニットを定義する

1. 親タスクのロジックエディタを開き、ドロップイベントをトリガとしたイベントロジックユニットを作成します。
2. ロジックユニット内に、ブロック処理コマンドで *While* を定義します。条件特性に以下の式を指定します。

   LoopCounter ()<=VecSize（Index Vector 項目）

3. ブロック内にコールサブタスク処理コマンドを定義します。
4. サブタスクに変数「Index vector」の現在値をパラメータとして渡します。VecGet(Index vector 項目 , loopcounter()) を使用します。
5. ブロックの外で、イベント実行処理コマンドを使用して、ビュー再表示という内部イベントを発行するようにします。



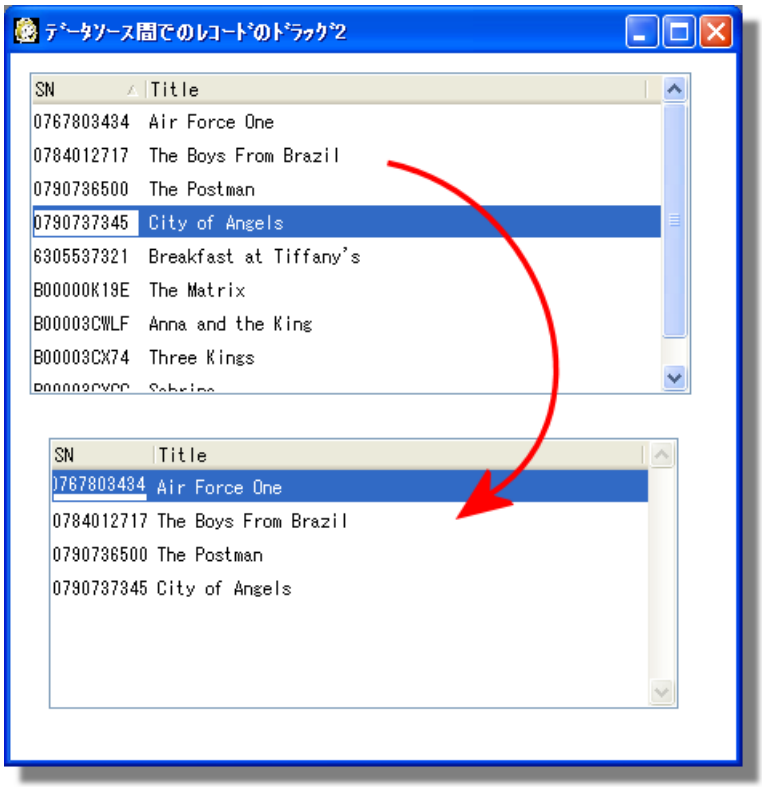
6. 起動されるサブタスクを作成します。このタスクは、バッチタスクでメインソースがありません。タスク終了条件は *Yes* で、チェック時期を *A= 後置* と指定します。
7. 上記のバッチタスク内のデータビューエディタを開き、数値型のパラメータ項目を定義します。ここには、親タスクから渡されるパラメータ値（Code）が渡されます。

２つのリンクコマンドを定義する

次に、バッチタスク内に２つのリンクコマンドを定義します。１つはソーステーブルを、もう１つはデスティネーションテーブルをリンクします。

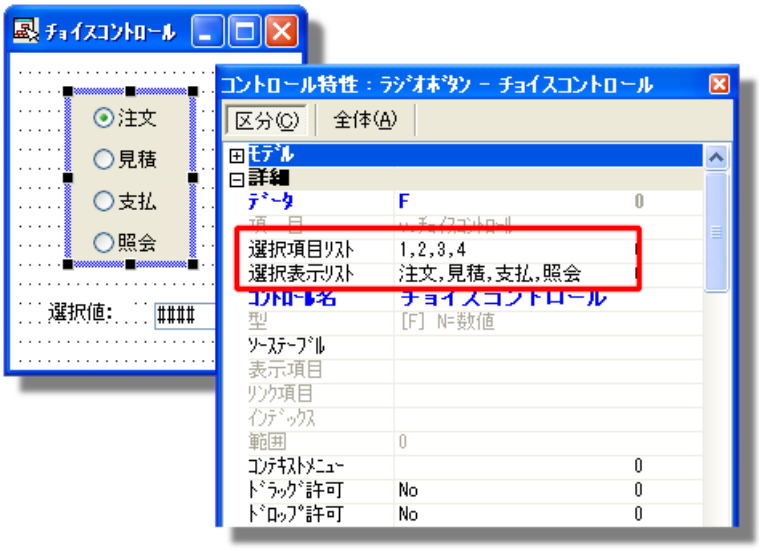
1. ソーステーブルに対する照会リンクコマンドを定義します。パラメータ項目で位置付けします。
2. デスティネーションテーブルに対する書込リンクコマンドを定義します。パラメータ項目で位置付けと代入指定をを行います。
3. レコード後で、デスティネーションテーブルの Title 項目の値をソーステーブルの Title 項目の値で更新するように定義します。

これで、親タスクのレコードをマルチマークした状態で子タスクのテーブル上のカラムにドラッグ & ドロップすることでレコードがコピーされます。



チョイスコントロールで定義されている値と異なるテキストを表示するには

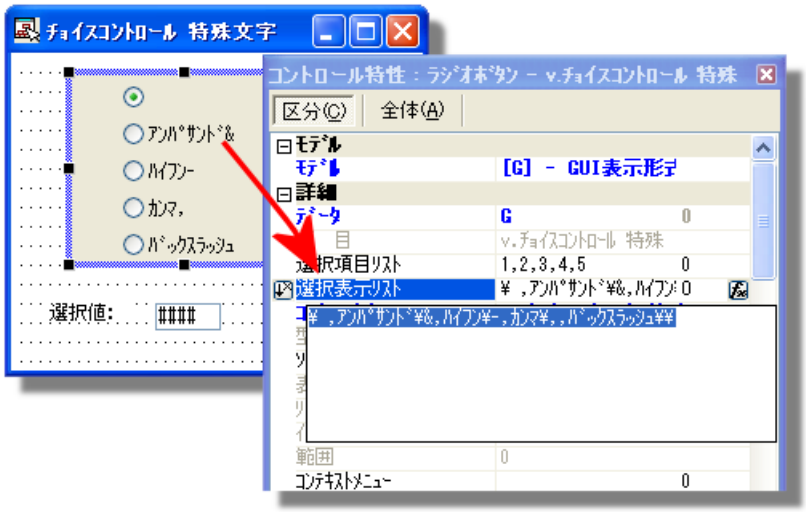
チョイスコントロールを使用する際、定義されているデータと異なる内容を表示させたい場合があります。異なる言語環境で実行するアプリケーションを開発する場合、特に重要な問題となります。この例では、4 つの指示が表示されています。注文、見積、返済、問い合わせです。しかし、プログラム内部では 1、2、3、4 のコードとして格納されています。



ここで示されているように、この設定内容はコントロールにてすべて処理されます。**選択項目リスト**特性にチョイスコントロールの値を順番に設定します。**選択表示リスト**特性に、同じ順番で表示項目を設定します。

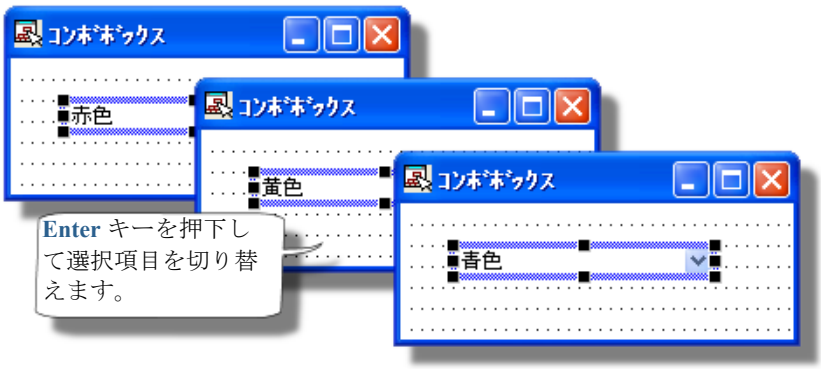
この例では、値が直接設定されていますが、式を使用したりデータソースを使用して設定することもできます。

チョイスコントロールに特殊文字を表示するには



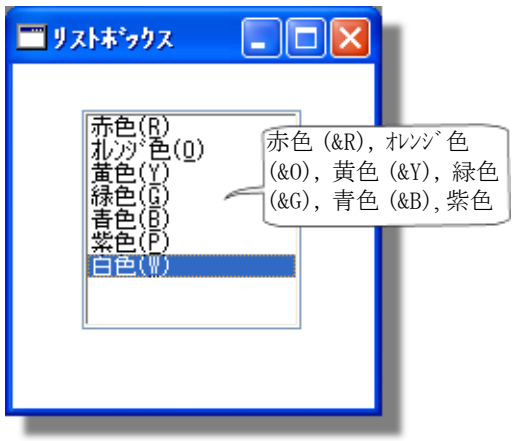
いくつかの文字は、内部的にチョイスコントロールで使用されています。例えば、カンマは、選択項目を区切り文字として使用されています。リスト内でこれらの文字を使用するためには、エスケープ文字としてバックスラッシュ(¥) を前に付加する必要があります。ここで示されているように、空白、アンパサンド、およびカンマがこれに当てはまります。

フォームエディタ上でチョイスコントロールオプションを切り替えるには



フォームエディタ上でチョイスコントロールにパークしている間、*Enter* を押下することで選択項目を切り替えることができます。タブコントロールのオプションが変更する場合、タブにリンクされた異なるコントロールが表示されるため、タブコントロールと組み合わせて定義されている場合は、この機能は便利です。

アクセラレータをチョイスコントロールオプションに設定するには

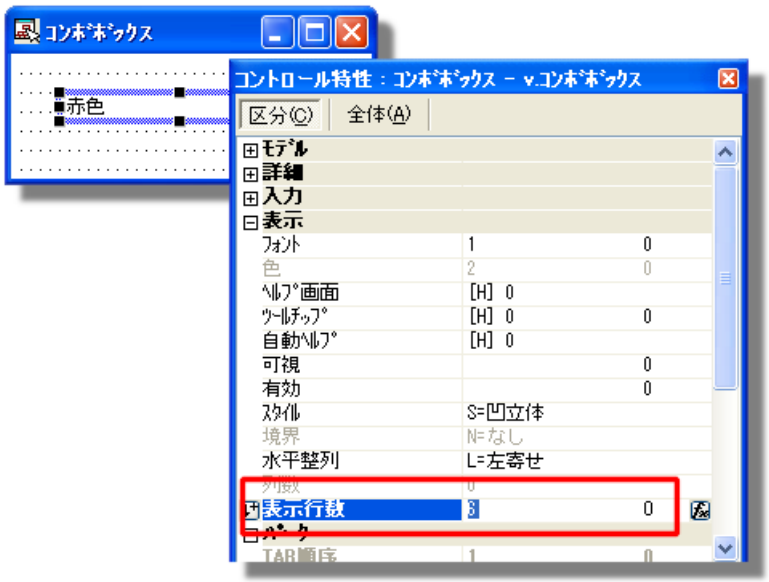


チョイスコントロール上の各項目は、アクセラレータキーを定義することができます。アクセラレータキーを設定すると、そのキーを押下することで自動的にフォーカスをその選択項目に移動させることができます。上図の例では、*W* を押下すると *White* が選択されます。

アクセラレータキーは、表示文字の前にアンパサンド（&）を追加することで表示されます。

注： アンパサンドを表示する必要がある場合、その前にバックスラッシュ（¥）を付加してください。

コンボボックスで表示されるドロップダウンリストの長さを制限するには

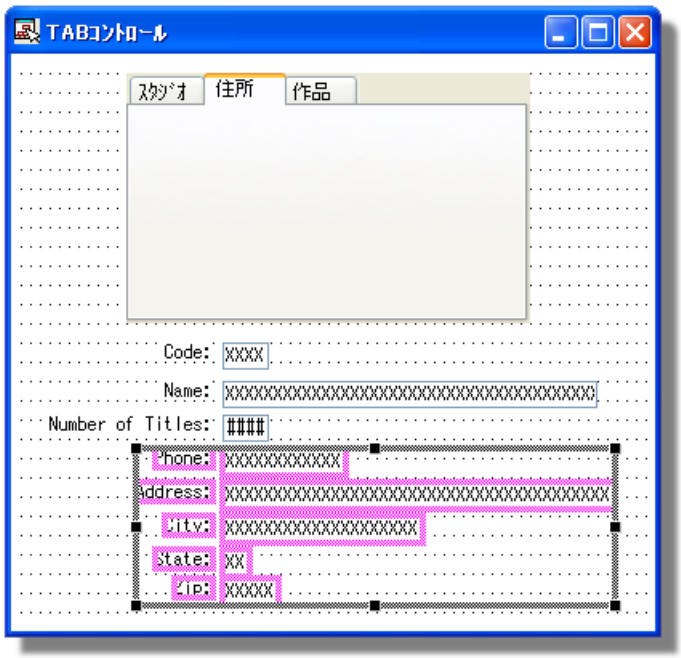


たくさんの選択肢がコンボボックスに定義されている場合があります。選択肢の表示行数を制限するには、**表示行数**特性に値を設定します。この例では、選択肢として最大 6 つの行が表示されます。

タブコントロールの異なるタブにコントロールを関連付けるには

タブコントロールを作成する場合、コントロールを 1 つのタブとリンクさせることができるます。リンクすると、そのタブが選択された場合のみ表示されます。すべてのタブにリンクさせることもできます。この場合、すべてのタブで表示されます。

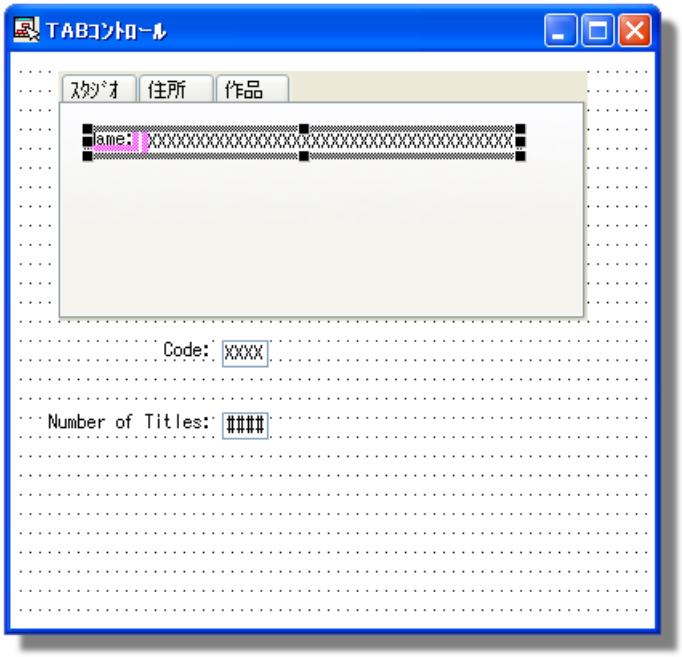
１つのタブでコントロールを表示させる



1. タブコントロールを選択します。
2. *Enter* を押して、リンクさせたいタブに切り替えます。
3. リンクさせたいコントロールを選択します。
4. リンクアイコン  を選択し、タブをクリックします。リンクされたコントロールはピンク色で囲まれた状態になります。
5. タブコントロール上に選択したコントロールを移動します。

これで、このコントロールは選択されたタブでのみ表示されます。この例では、アドレス項目はアドレスタブ上のみ表示されます。

すべてのタブでコントロールを表示させる

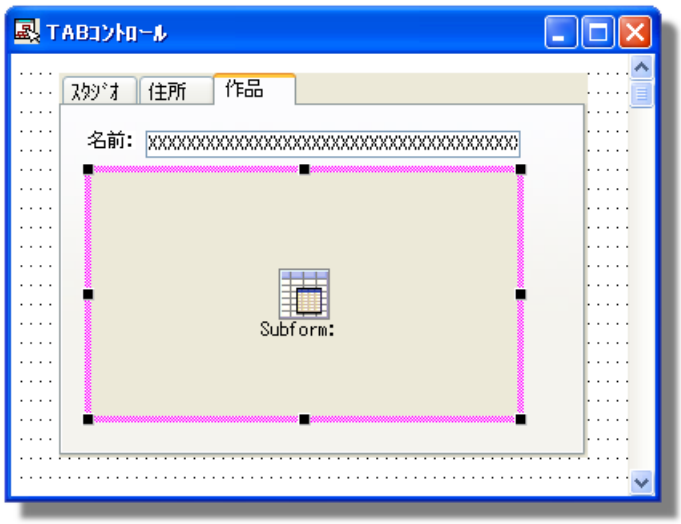


1. タブコントロールを選択します。
2. *Enter* を押して、どのタブも選択されていない状態に切り替えます。（上図を参照）
3. リンクさせたいコントロールを選択します。
4. リンクアイコン  を選択し、タブをクリックします。リンクされたコントロールはピンク色で囲まれた状態になります。
5. タブコントロール上に選択したコントロールを移動します。

これで、このコントロールはすべてのタブで表示されます。

タブコントロールの指定されたタブにタスクを関連付けるには

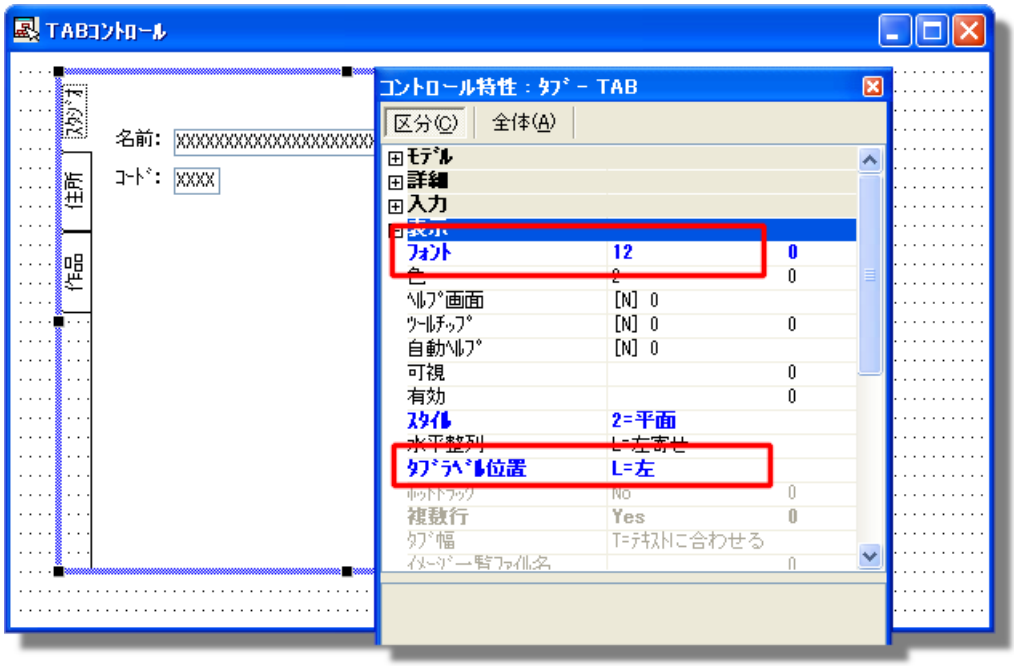
現在のタスクには定義されていないデータをタブ上に表示させる場合があります。表示されるデータがデータソースの内容であったり、別のレコードからの複合データの場合、特に必要になります。サブフォームコントロールをタブにリンクさせることでこのような処理を簡単に実現することができます。



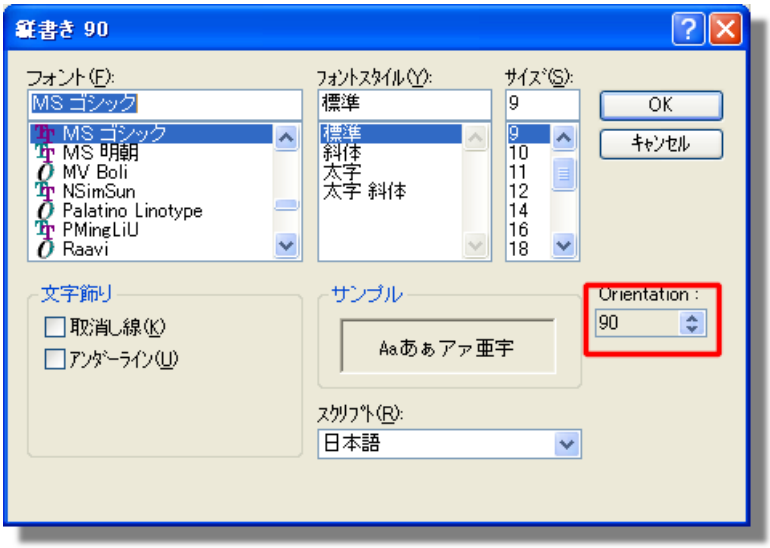
1. タブコントロールを選択します。
2. *Enter* を押して、リンクさせたいタブに切り替えます。
3. コントロールパレットからサブフォームアイコン  をクリックして選択します。
4. サブフォームをタブにドロップします。
5. タブフォームに実行したいタスク（プログラム）を定義します。

参照： 第 8 章：「タブコントロールに配置したサブフォームの表示を制御するには」（180 ページ）

垂直タブコントロールを定義するには



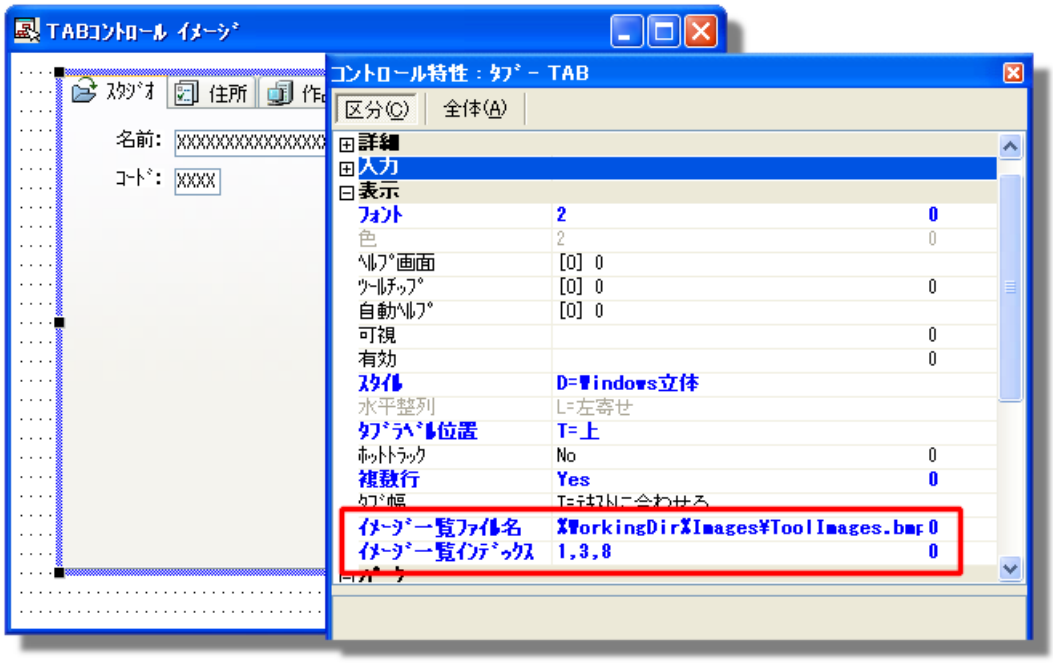
タブコントロールの**タブラベル位置**特性を ***L= 左***または ***R= 右***に変更することで、水平方向にタブが表示されるようになります。



垂直タブを使用する場合、文字が正しい方向に表示されるように、フォントを変更する必要があります。この例では、90 度に回転したフォントを使用しています。

注： **タブコントロール**の**スタイル**特性を ***D=Windows 立体***の場合は、***R= 右***および ***L= 左***の設定は機能しません。

タブコントロールの各タブにイメージを割り当てるには



テキストと一緒にイメージをタブと結び付けることができます。

タブコントロールの**イメージ一覧ファイル名**特性を指定することで実現できます。このイメージファイルは、基本的に**ツリー**コントロールや**プッシュボタン**で使用されるイメージを 1 つのファイルに連結したものです。このファイルは例えば、以下のような形式になります。

例の図のように、ファイルには、たくさんのイメージを定義することができます。各イメージのサイズは 16 X 16 ピクセルです。

イメージ一覧インデックス特性でどのイメージを表示させるかを指定することができます。この例では、３つのタブに対して 1、3、および 8 番目のイメージを指定しています。

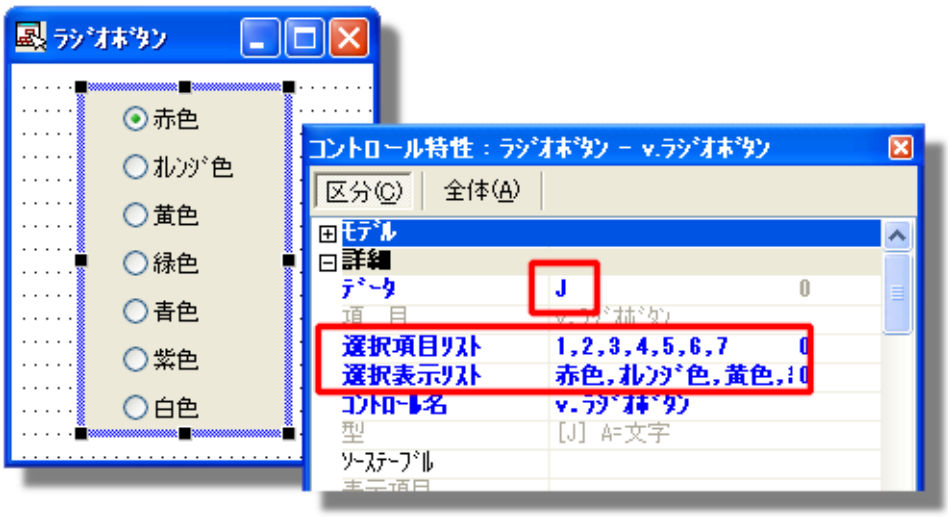
ラジオボタンを定義するには

２つの方法でラジオボタンを定義することができます。

1. 一般的な矩形表示コントロール内にボタンを表示する方法
2. 複数のコントロール内にボタンを含めるより柔軟な方法

２つのオプションの定義方法を以下に説明します。

1 つのコントロール内にボタンを含める



1 つのコントロール内にボタンを含めるラジオボタンを作成します。このタイプのラジオボタンは矩形のボックスで表示されます。作成するには以下のようにします。

1. 選択項目を格納する変数項目を作成します。この例では、項目 **L** になります。
2. コントロールパレットからラジオボタン  を選択し、フォームにドロップします。
3. **データ**特性に、変数項目を定義します。
4. **選択項目リスト**特性に、変数項目に設定したい値の選択肢を入力します。
5. **選択表示リスト**特性に、ラジオボタンに表示する選択肢を入力します。
6. **表示行数**や**フォント**などの表示特性を定義します。

これで実行時にラジオボタンから有効な値を選択することができます。

複数のコントロール内にボタンを含める

複数のコントロール内にボタンを含めるラジオボタンを作成します。このタイプはフォーム上の複数の部分にボタンを分けて表示させることができます。作成するには以下のようにします。

1. 選択項目を格納する変数項目を作成します。この例では、項目 **L** になります。
2. コントロールパレットからラジオボタン  を選択し、フォームにドロップします。
3. **データ**特性に、変数項目を定義します。
4. **選択項目リスト**特性に、変数項目に設定したい値の選択肢のうちの 1 つを入力します。この例では、*4* が設定されています。
5. **選択表示リスト**特性に、ラジオボタンに表示する選択肢のうちの 1 つを入力します。この例では、***緑色***が設定されています。
6. **表示行数**や**フォント**などの表示特性を定義します。

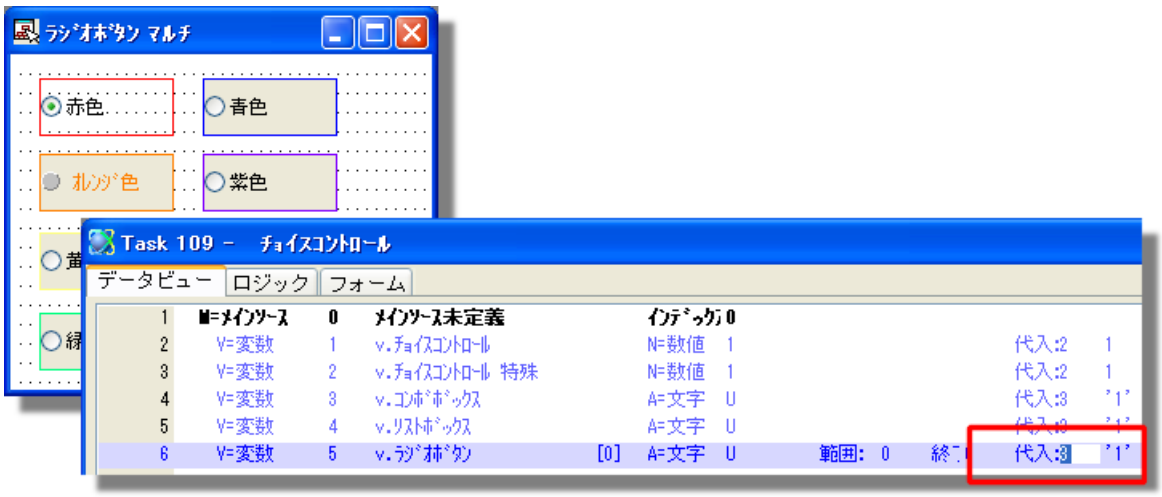
   これで実行時にラジオボタンから有効な値を選択することができます。

各々のラジオボタンに同じ変数項目を定義することで、これらをつなぎ止める状態になり、これらは 1 つのユニットとして動作します。すなわち、実行中にボタンの 1 つをクリックすると以前の選択肢が解除され、1 つのボタンのように動作します。しかし、これらは個別のコントロールであり、*Ctrl+* クリックで個別の修正したり移動したりすることができます。

チョイスコントロールにデフォルトオプションを設定するには

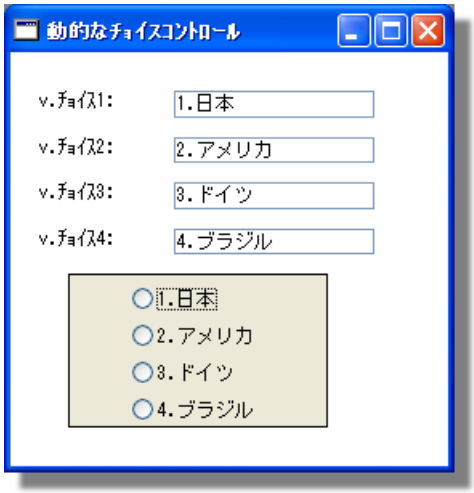
チョイスコントロールを作成する場合、デフォルトでは何も選択されていない状態で、空白や何も選択されていない状態で表示されます。これでは、よく選択する項目を表示させたいユーザには不親切なアプリケーションとなってしまいます。

デフォルトの選択肢を指定する



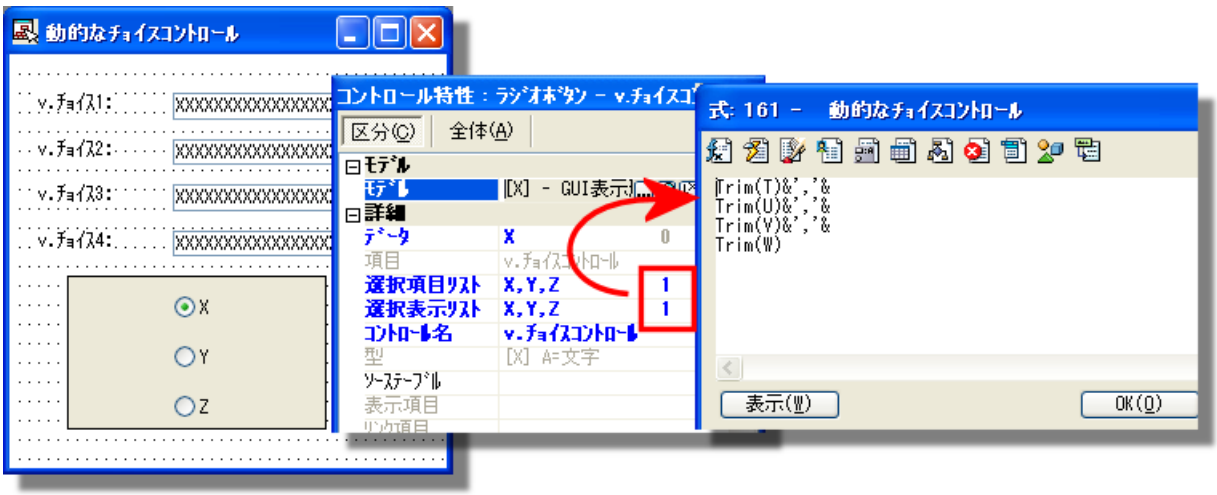
チョイスコントロールにデータ項目が定義され、その項目の初期設定ができれば、デフォルトオプションは簡単に設定できます。この例では、***赤色***の選択肢（**選択表示リスト**特性）は、*1*（**選択項目リスト**特性）とリンクしています。従って、**代入**特性に *1* を設定することで、***赤色***がデフォルトになります。

チョイスコントロールの選択肢を動的に設定するには



選択肢は固定して設定しますが、式を使用することで動的に指定することも可能です。この例では、動的な 4 つのユーザ入力をラジオボタンに設定しています。

動的な選択肢を設定する

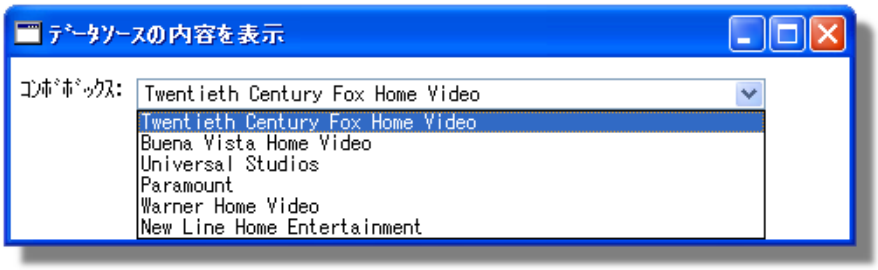


動的なチョイスコントロールを作成するには以下のようにします。

1. 選択項目を格納する変数項目を作成します。この例では、項目 **P** になります。
2. コントロールパレットからラジオボタン を選択し、フォームにドロップします。
3. データ特性に、変数項目 **P** を定義します。
4. 選択項目リスト特性を式で指定し、変数項目に設定したい値の選択肢を入力します。
5. 選択表示リスト特性を式で指定し、ラジオボタンに表示する選択肢のうちの 1 つを入力します。この例では選択項目リストと同じになっています。
   オプションで、選択表示リストに文字列を入力することができます。この値は開発環境でのみ表示され、開発者が表示内容を確認しやすくなります。上記の例では、**x**、**y**、**z** が設定されています。

   これで、実行時に式で評価された値がチョイスコントロールの選択肢となります。

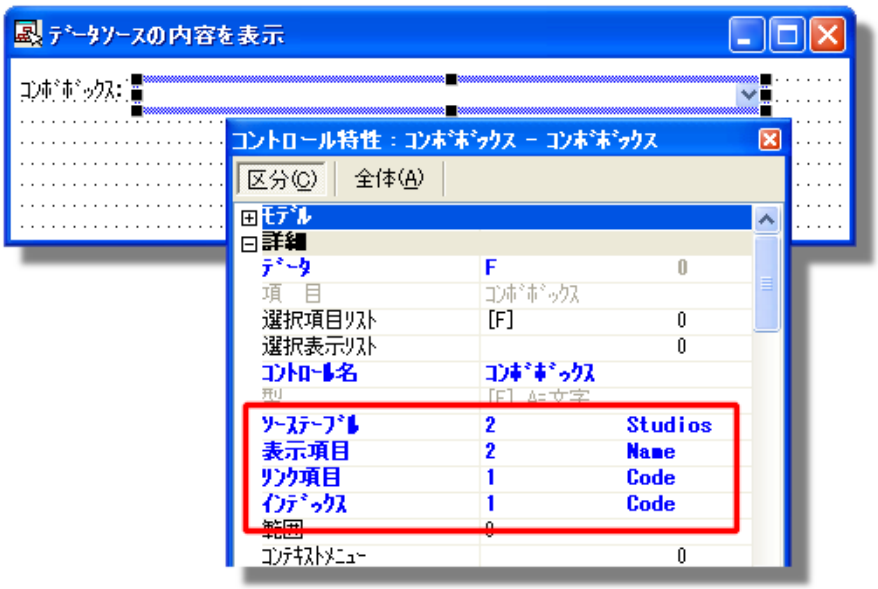
データソースの内容を選択肢にするには



データベーステーブルからデータを選択できるコントロールがあれば便利です。この方法を利用すると、テーブルが更新されると選択肢も自動的に変わります。

この例では、Studio テーブルからスタジオを選択肢しています。ユーザがスタジオ名を選択すると、タスク内ではスタジオコードが返るようになっています。これはどのようにして実現するかを説明します。

データベーステーブルにリンクしたチョイスコントロールを作成する



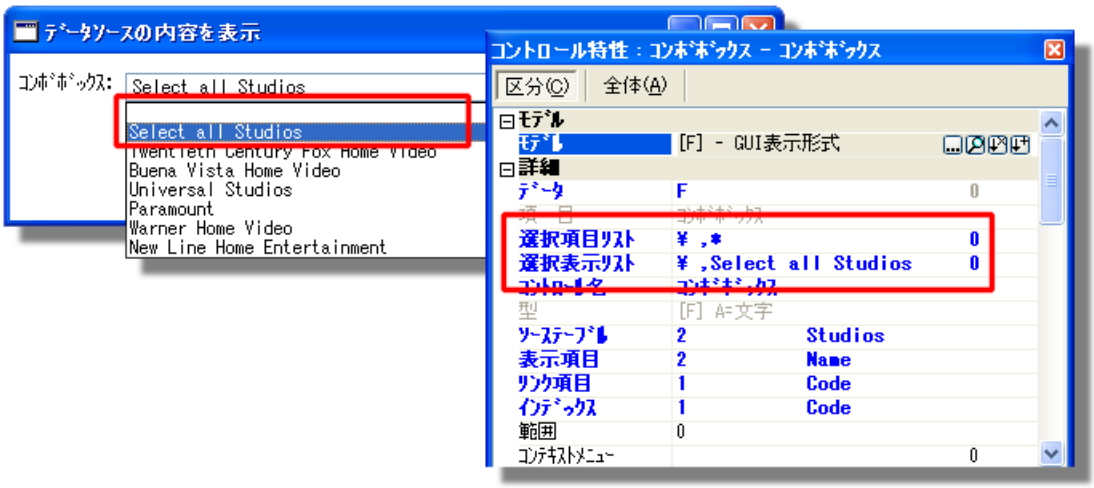
1. 最初に、データを格納する偏す項目を作成します。この例では、**v.Studio Code** を使用しています。
2. コントロールパレットからコンボボックス（または他のチョイスコントロール）を選択し、フォーム上にドロップします。
3. チョイスコントロールのデータ特性に変数を定義します。（例では、*L*）
4. ソーステーブル特性からズーム（*F5*）して、使用したいデータソースを選択します。この例では、*Studios* を使用しています。
5. 表示項目特性からズーム（*F5*）して、表示するカラムを選択します。この例では、*Studio Name* を使用しています。
6. リンク項目特性からズーム（*F5*）して、変数に値を返すカラムを選択します。この例では、*Studio Code* を使用しています。
7. インデックス特性からズーム（*F5*）して、選択肢の表示順を決めるインデックスを選択します。範囲指定が定義されている場合、範囲定義のインデックスと合うようにしてください。合わない場合、パフォーマンスに影響します。この例では、*Studio Code* をインデックスにしています。
8. 範囲特性からズーム（*F5*）して、選択肢の範囲を制限する条件を指定します。この例では範囲指定を行っていません。

   これで、タスクが実行されると、チョイスコントロールはテーブルから使用するデータ項目に移ります。

注： データソースが大きい場合、チョイスコントロールにリンクして使用すると性能が劣化する可能性があります。この方法は、比較的小さなデータソースまたは効率的なインデックスが定義されているデータソースで使用してください。また、チョイスコントロールにリンクされたデータソースは、宣言テーブルリストの中には含まれないことに注意してください。

データソースにリンクされたチョイスコントロールに追加オプションを結合するには

チョイスコントロールは、データをもとに選択肢を表示します。例えば、項目の範囲特性をもとに選択肢を表示したり、データソースに動的にリンクしている場合もあります。しかし、コンボボックスとリストボックスでは、範囲特性やデータソースの値に加え、より多くの値を選択肢として追加するオプションを持っています。



この例では、ソーステーブル **#2** にリンクされています（Studios テーブルです）。

1. **選択表示リスト**特性に２つの項目（***空白***と ***Select all studios*** という文字列です。）を追加します。空白は、バックスラッシュ（\）を付加します。
2. **選択項目**リストにも、２つの項目（***空白***と***アスタリスク***）を追加します。空白はリセットで、アスタリスクは全て選択することを表します。

参照： 「チョイスコントロールに特殊文字を表示するには」（265 ページ）

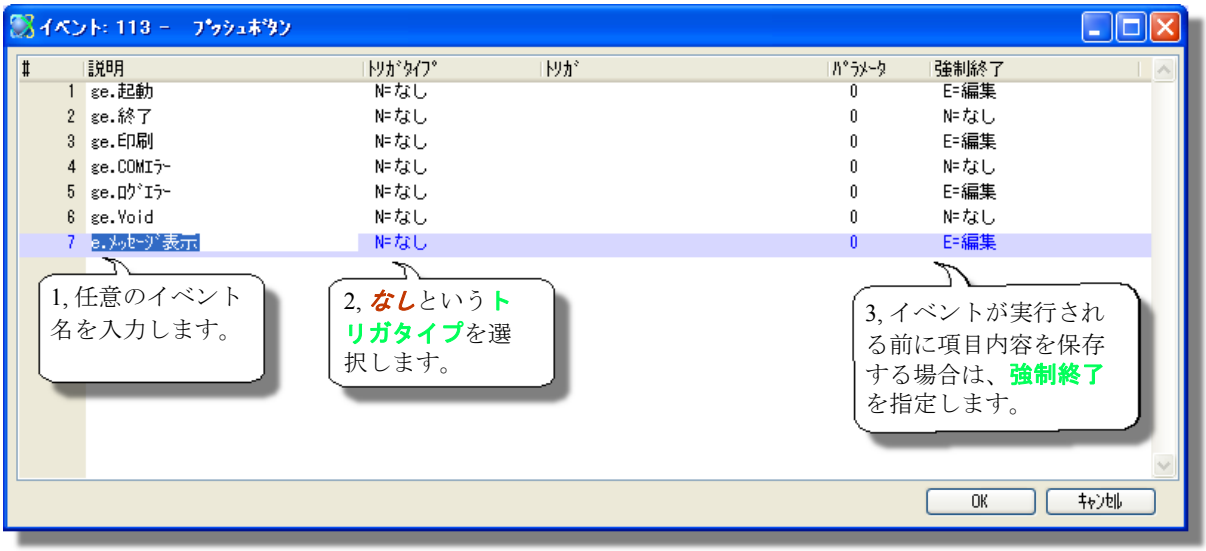
プッシュボタンとロジックを組み合わせるには

プッシュボタンをクリックするとロジックが実行されるようにするには以下の手順で行います。

1. ユーザイベントを作成します。
2. プッシュボタンコントロールをフォームに配置します。
3. イベントが実行されたときに処理するロジックユニットを作成します。

以下のセクションは、関連する手順を説明しています。

ユーザイベントを作成する

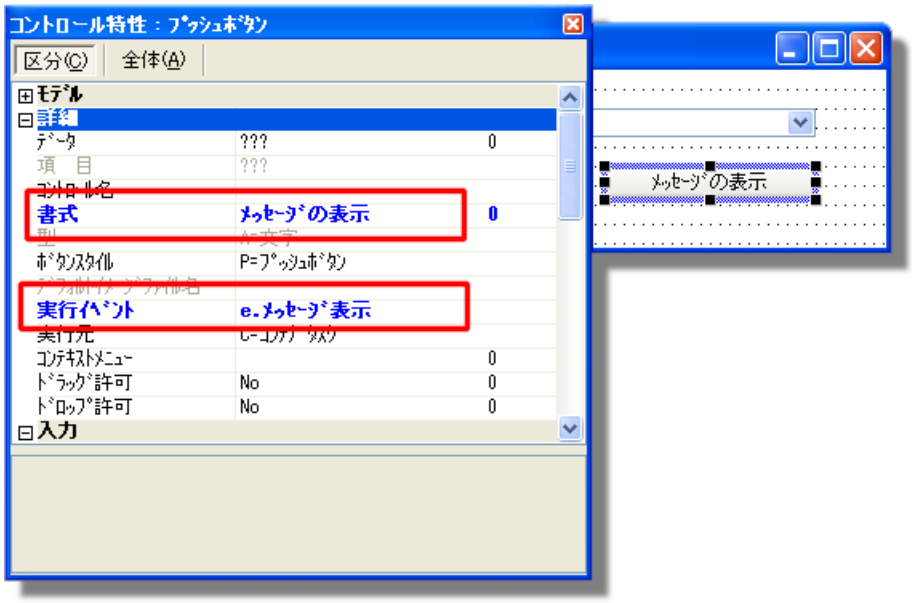


最初に、ユーザイベントを作成する必要があります。ユーザイベントテーブル（*Ctrl+U*）で作成します。1 行作成（*F4*）し、以下のようにパラメータを設定します。

1. 名前カラムに、*イベント名*を入力します。
2. トリガタイプでは、*N= なし*を選択します。このイベントがプッシュボタンで実行されるため、トリガは必要ありません。
3. 強制終了カラムでは、*E= 編集*または *N= なし*を選択します。*E= 編集*の場合は、プッシュボタンがクリックされてイベントが実行される前に、編集中のフィールドの値がデータ項目に格納されます。
4. プッシュボタンコントロールに割り当てるイベントを必要なだけ追加します。

注： グローバルなイベントを再利用することもできます。上の例において、グローバルなイベントは ge.. いう接頭辞が付加されています。ここには、ge. 印刷 や ge. 実行などの多くのプログラムで使用する標準的なグローバルイベントが定義されています。グローバルイベントを使用する場合、自動的に指定されたイベントを実行するボタンを作成することができるように、これらをモデルに割り当てることができます。

フォームにプッシュボタンを配置する

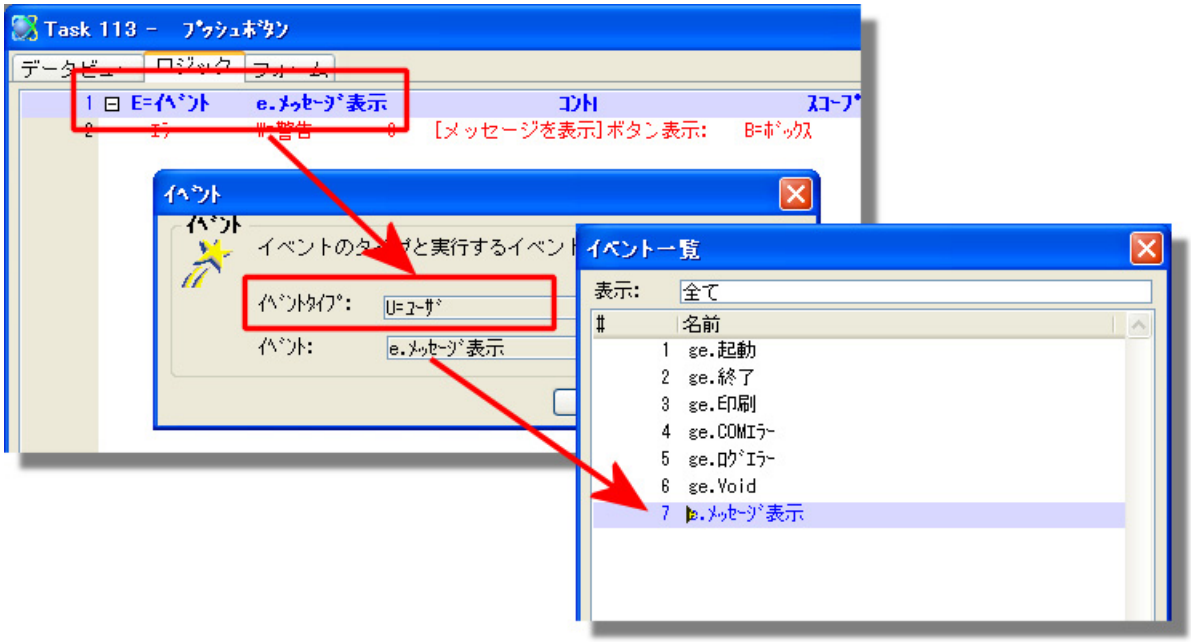


1. コントロールパレットからプッシュボタン 

を選択します。
2. フォームをクリックしてプッシュボタンコントロールをドロップします。必要に応じてサイズを変更します。
3. プッシュボタンコントロールの書式特性を入力します。ここには、プッシュボタンに表示されるテキストを入力します。
4. 実行イベント特性でズーム （*F5*）してイベントを選択します。

   これで、ユーザがプッシュボタンをクリックするとイベントは実行されます。この例では、ユーザがメッセージの表示ボタンをクリックすると、e. メッセージ表示イベントが実行されます。次に、イベントを処理するロジックユニットを作成する必要があります。

イベントが発行されると実行されるロジックユニットを作成する



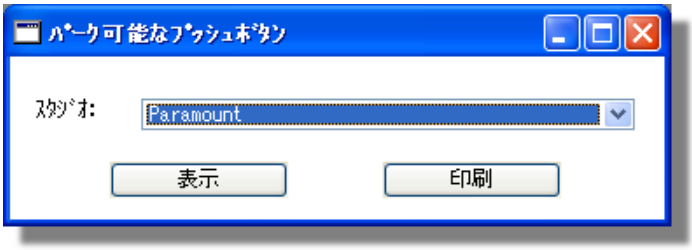
1. ロジックエディタを開きます。
2. ヘッダ行（*Ctrl+H*）を作成します。
3. コンボボックスから、*E= イベント*を選択します。イベントダイアログが表示されます。
4. イベントタイプとして、*U= ユーザ*を選択します。イベント一覧が表示されます。
5. 実行させたいイベントを選択します。
6. イベントが実行されたときに、実行される処理コマンドをイベントロジックユニットに追加します。

これで、実行可能なプッシュボタンが定義できました。

注 : この例では、*Tab* で移動できないプッシュボタンを使用しています。データ特性に項目を定義することで、*Tab* 移動を可能にすることができます。項目を設定すると、書式特性はデータ項目の書式が設定されるため、後で修正する必要があります。

参照 : 「プッシュボタンにキーボード操作を許可させるには」（283 ページ）

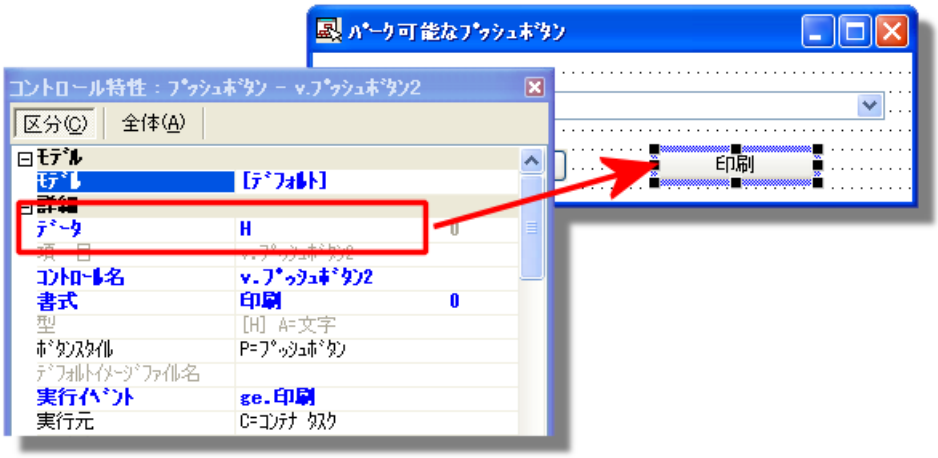
プッシュボタンにキーボード操作を許可させるには



プッシュボタンコントロールをフォームにドロップし、実行イベント特性を設定するだけでプッシュボタンを定義することができます。ユーザがマウスでクリックするか、アクセラレータキー（ホットキー）を押下することでプッシュボタンは動作します。しかし、ユーザは *Tab* で移動することはできません。

必要であれば、プッシュボタンに項目を設定することで *Tab* による移動が可能になります。プッシュボタンコントロールのデータ特性に項目を指定するには、以下の操作を行います。

パーク可能なプッシュボタンを作成する

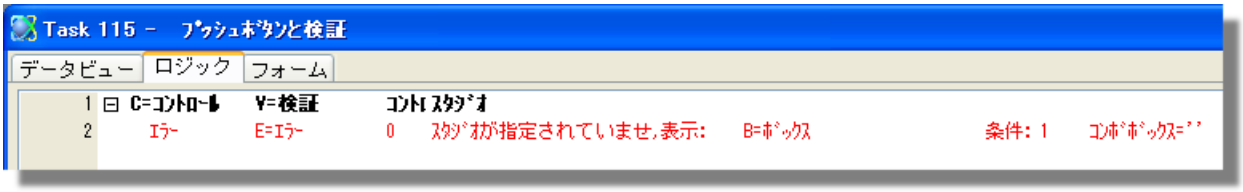


1. タスクのデータビューエディタを開き、プッシュボタンに設定する変数項目を作成します。
2. フォーム上にプッシュボタンをドロップします。
3. データ特性でズーム（*F5*）して、このプッシュボタンに設定する変数項目を選択します。
   これでユーザは、このプッシュボタンに *Tab* でパークさせることができます。コマンドパレットから TAB 順序の表示  を選択すると、*Tab* 移動が可能で TAB 順序の変更が可能であることを示す赤い数字が表示されます。

注： *Tab* 移動可能なプッシュボタンを作成すると、ボタン上のテキストは設定された項目に格納されているテキストが表示され、*Tab* 移動できないプッシュボタンとは異なって表示されるので注意してください。この対応には色々な方法があります。

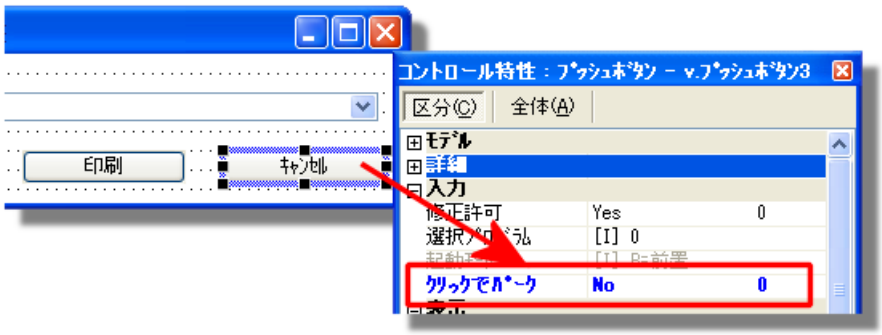
参照： 「パーク可能なプッシュボタン上にテキストを指定するには」（287 ページ）

プッシュボタンがクリックされたときに実行される検証ロジックをスキップするには



しばしば、フォーム上のコントロールは、それらに付属した検証ロジックを持つことがあります。この例では、スタジオを選択しないで Studio コントロールを通過することができません。通常、このような機能は必要とされているものです。例えば、ユーザは、印刷データを選択しないで印刷しようとはしません。

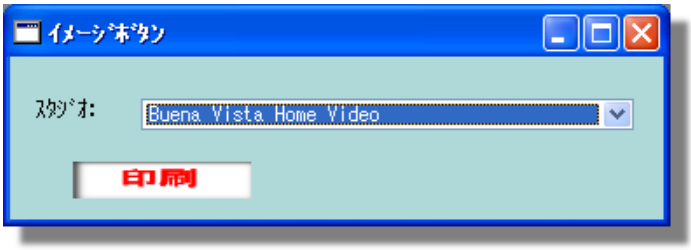
しかし、入力内容に問題があってもプッシュボタンを実行できるようにさせる必要があるかもしれません。例えば、終了やキャンセルのボタンを実行できるようにする場合が考えられます。



このような動作を実行させるには、クリックでパーク特性を *No* に設定します。プッシュボタンは設定前と同じように動作しますが、コントロールを通過してもコントロールロジックユニットが実行されません。

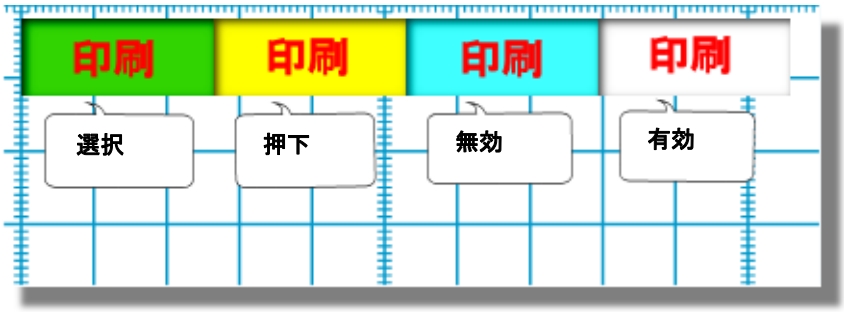
注： ユーザがキーボードを使用している場合は、効果がありません。従って、Studio フィールドを空白の状態のまま *Tab* 移動しようとした場合、エラーメッセージが表示されます。このような場合、プッシュボタンにアクセラレータキー（*Alt+C*）を定義することで回避できます。

イメージボタンを作成するには

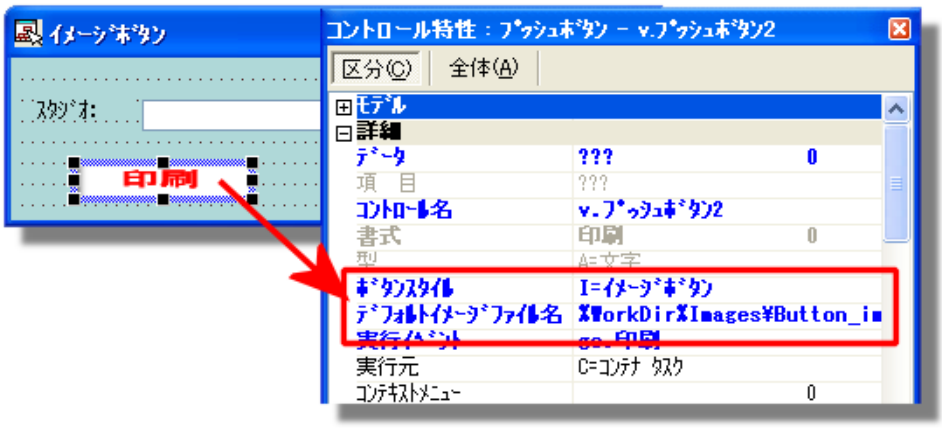


Magic でイメージボタンを使用することができます。***イメージプッシュボタン***には、同じサイズの 4 つのセクションを持つビットマップ（.bmp）ファイルを定義します。この例では、イメージエディタで 1 つのボタン用のイメージを作成し、同じイメージを 4 回繰り返しそれぞれ異なる色を設定しています。

実行時、ボタンの状態にもとづいて、Magic はイメージファイルの異なる部分を使用します。パークされておらず有効な状態の場合、4 番目のボタン（深緑）が表示されます。ボタンが使用不可の場合、3 番目のボタン（薄緑）が表示されます。ボタンがクリックされた状態の場合、2 番目のボタン（オレンジ）が表示されます。（項目が割り当てられており）*Tab* 移動でフォーカスがコントロール上にある場合に、最初のボタン（黄色）が表示されます。



イメージボタンを作成する

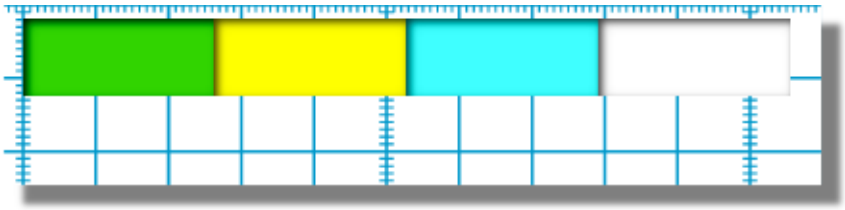


1. フォーム上にプッシュボタンをドロップします。
2. ボタンスタイル特性で、*I= イメージボタン*を選択します。
3. デフォルトイメージファイル特性でズーム（*F5*）して、イメージファイルを選択するか、直接ファイル名を入力します。ファイル名やパス名には論理名が使用できます。

これで、プッシュボタンを表示するために指定されたイメージファイルが使用されます。

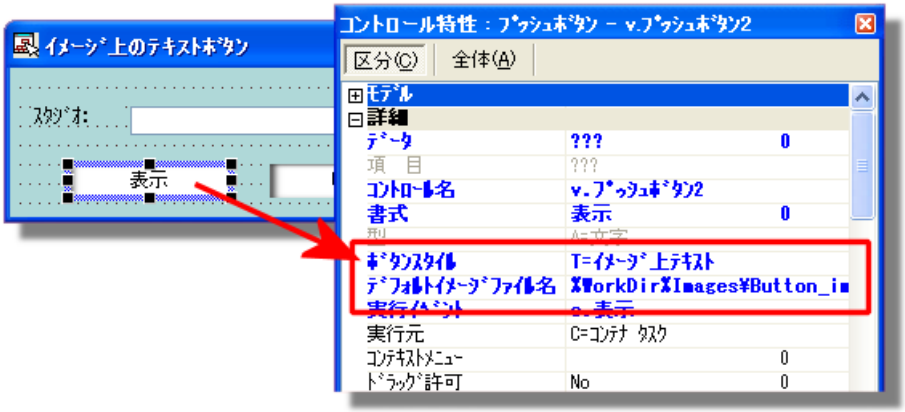
イメージとテキストを組み合わせてボタンを表示するには

Magic では、ボタンに表示されるテキストの背景にイメージを指定することができます。これにより、作成するイメージファイルが少なくなり、多言語機能を利用することで実行時に別の言語表記に変換することが可能になります。



イメージ上のテキストで使用するイメージファイルは、イメージボタンと同じ書式で作成しますが、テキストは指定しません。イメージファイルの作成方法については、「イメージボタンを作成するには」（285 ページ）を参照してください。

テキスト上のイメージボタンを定義する



1. フォーム上にプッシュボタンをドロップします。
2. ボタンスタイル特性で、*T= イメージ上のテキスト*を選択します。
3. デフォルトイメージファイル特性でズーム（*F5*）して、イメージファイルを選択するか、直接ファイル名を入力します。ファイル名やパス名には論理名が使用できます。
4. 書式特性でボタンに表示するテキストを入力します。この例では、*表示*が入力されています。

これで、背景にテキストが重なって表示されるプッシュボタンが表示されます。

パーク可能なプッシュボタン上にテキストを指定するには

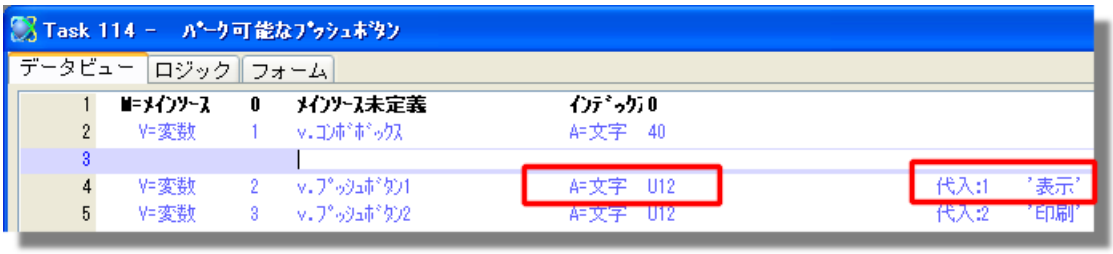
パーク不可のプッシュボタンを作成する場合、プッシュボタンコントロールの書式特性にテキストを入力することで、ボタン上に表示されます。しかし、パーク可能なプッシュボタンの場合、データ項目を定義するため、項目の内容がプッシュボタンに表示されます。

従って、この種類のボタンでは以下のようにしてテキストを指定します。

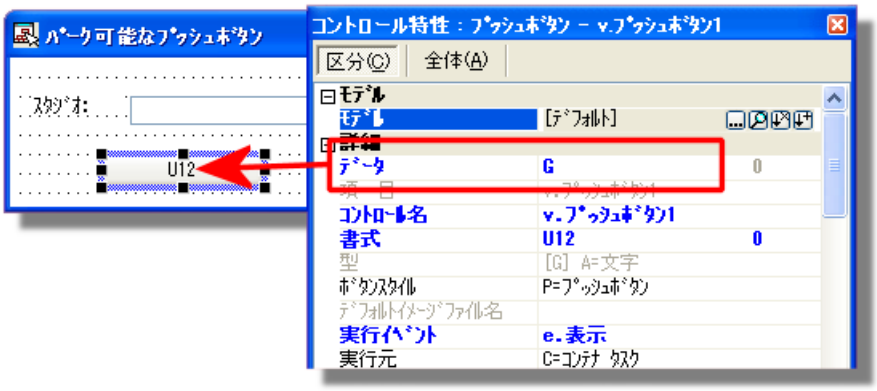
- 項目の代入特性で指定する。
- 項目のデフォルト値を指定する。
- プッシュボタンコントロールの書式特性にテキストを指定する。

詳細は、以下のセクションで説明します。

プッシュボタンのテキストを指定するために代入を使用する

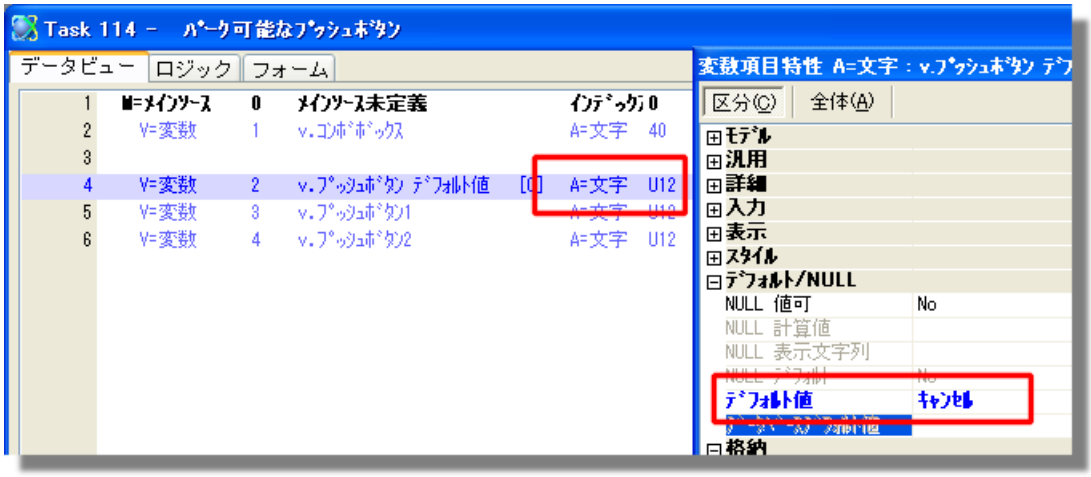


1. プッシュボタンのテキストが格納可能な長さの変数項目を作成します。
2. 代入特性を使用してテキストを指定します。この例では、*表示*が設定されています。



3. フォーム上にプッシュボタンをドロップします。
4. プッシュボタンのデータ特性に変数項目を定義します。
5. フォームエディタ上では、このボタンには指定されたテキストが表示されません。書式特性の内容が表示されます。しかし、プログラムを実行するとテキストが表示されます。

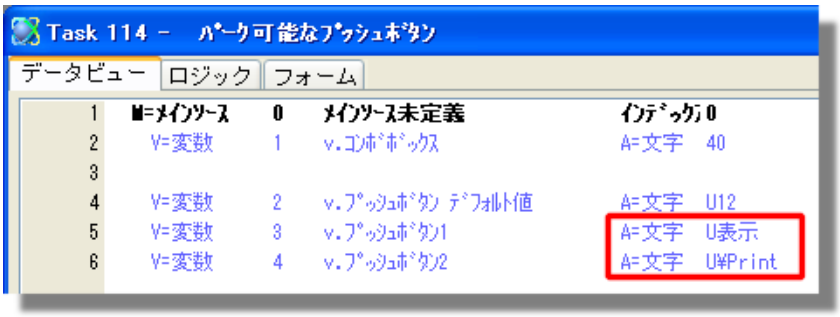
プッシュボタンのテキストを指定するためにデフォルト値を使用する



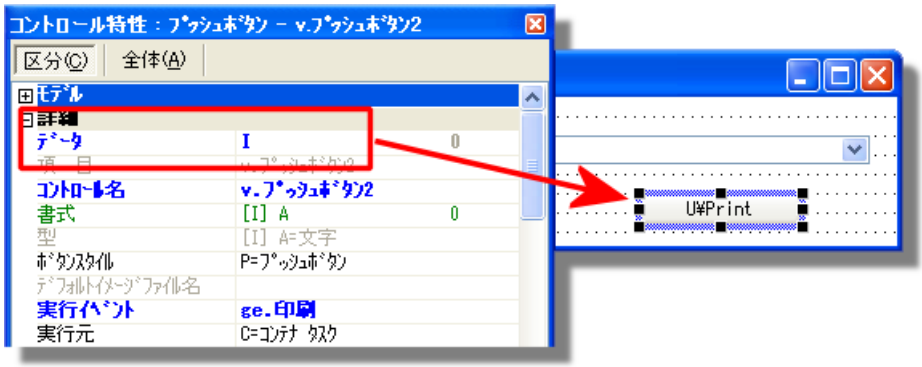
代入特性にテキストを指定することで、プッシュボタンにテキストが表示されます。しかし、代入特性の代わりに項目のデフォルト値特性にテキストを入力することでも同様に表示させることができます。

この方法は、モデルを使用する場合に便利です。しかし、データビューエディタ上でテキスト内容が参照できないため、ボタンに値が設定されているかどうかを確認できません。

プッシュボタンのテキストを指定するために書式特性を使用する

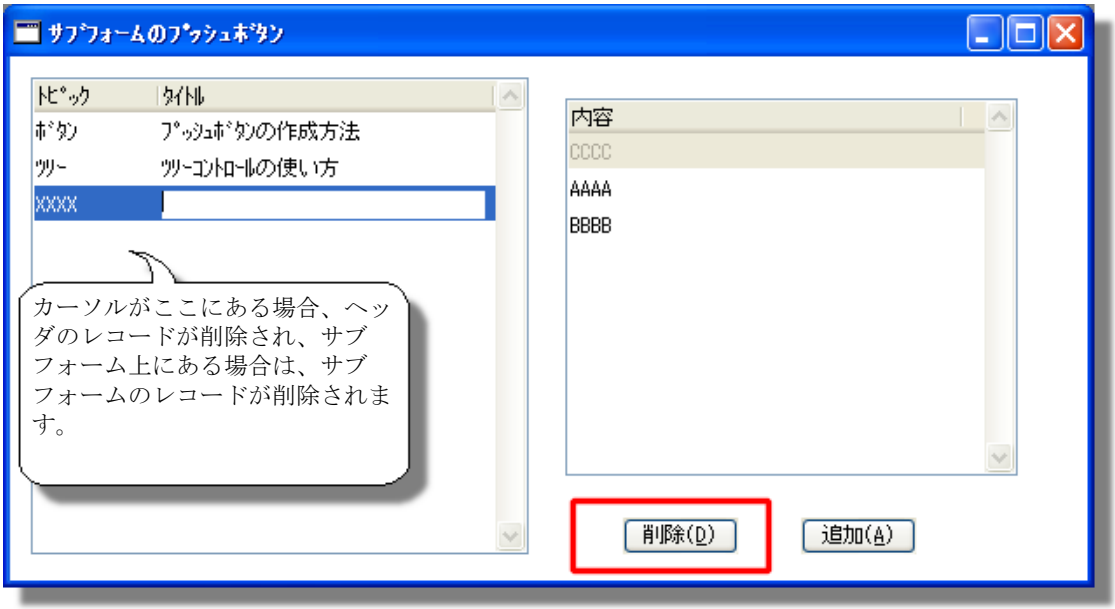


1. 文字型変数項目を作成します。
2. 少なくとも 1 つのプレイスフォルダ文字（この例では、U）を持つように、書式を指定します。残りの書式データをボタンに表示するテキストとして入力します。大文字は、前にバックスラッシュ（¥）を付加する必要があります。



3. フォーム上にプッシュボタンをドロップします。
4. プッシュボタンのデータ特性に変数項目を定義します。
5. これで、プッシュボタンコントロールの書式が変数項目から引き継がれます。この方法で、フォームエディタ上で、ボタンにテキストが表示されることを確認できます。

サブフォームやその親タスクに影響するプッシュボタンを定義するには



定義されたプッシュボタンに対して、現在のフォーカスがどこに位置しているかによって、異なる処理を実行させることができます。例えば、レコード一覧を表示させている親タスクがあり、各レコードが複数の子レコードを持っている場合、削除ボタンをクリックすると、カーソルがパークしている場所に応じて、親レコードや子レコードを削除するような処理が考えられます。

上の例において、削除ボタンがクリックされた場合、XXXX レコードを参照して処理すべきでしょうか？それとも CCC レコードを参照するべきでしょうか？ユーザは、カーソルが XXX の上にあるため、親レコードとその子レコードが全て削除されることを期待するはずです。しかしこのような場合は、カーソルがどこにパークしているかによって、異なる処理をさせる必要があります。

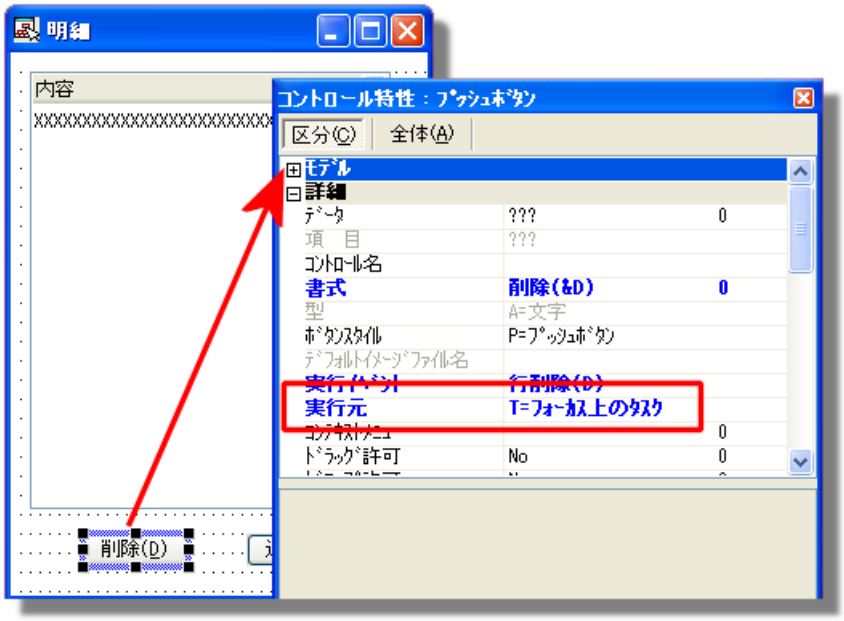
実行元特性は、この問題に対応するものです。この特性には、以下の２つのオプションがあります。

- *C= コンテナタスク*：イベントは、プッシュボタンが定義されているタスクで実行されます。
- *T= フォーカス上のタスク*：イベントは、フォーカスのある場所（タスク）で実行されます。

この例では、*T= フォーカス上のタスク*を設定する必要があります。

注： *フォーカス上のタスク*のオプションは、パーク不可（パーク可 =*No*）のボタンでのみ有効です。

実行元特性をフォーカス上のタスクに設定する



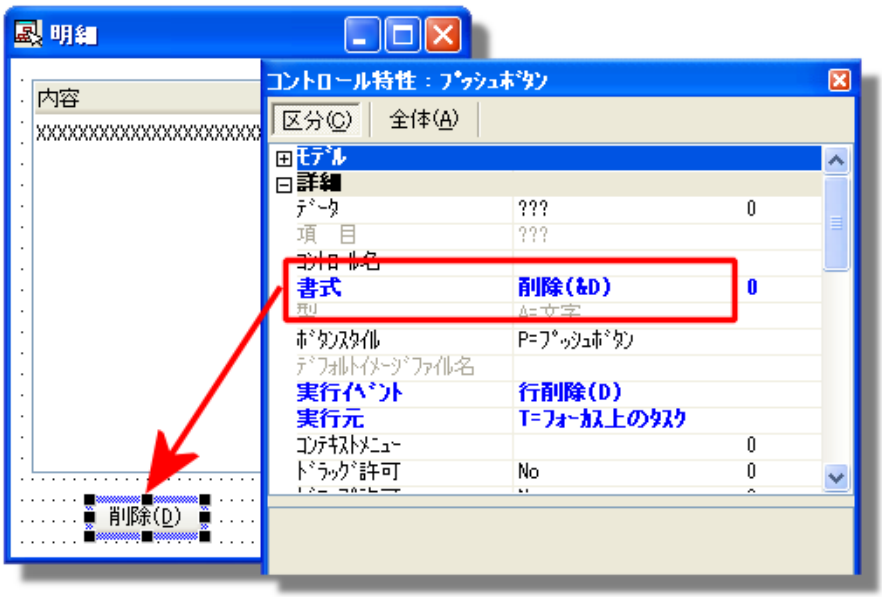
1. 変更が必要なプッシュボタンのコントロール特性を開きます。
2. 実行元特性でズームして、*T= フォーカス上のタスク*を選択します。

これで、現在フォーカスがどこにあるかによって、イベントが実行されるタスクが（親または子タスク）変わります。

アクセラレータをプッシュボタンに設定するには

大部分の人はマウスよりキーボードで操作するため、*アクセラレータ*（ホットキー）をプッシュボタンに設定すると便利です。パーク不可のプッシュボタンを使用している場合、特に必要です。アクセラレータを使用すると、ユーザはマウスからキーボードに手を移動する必要がなくなります。

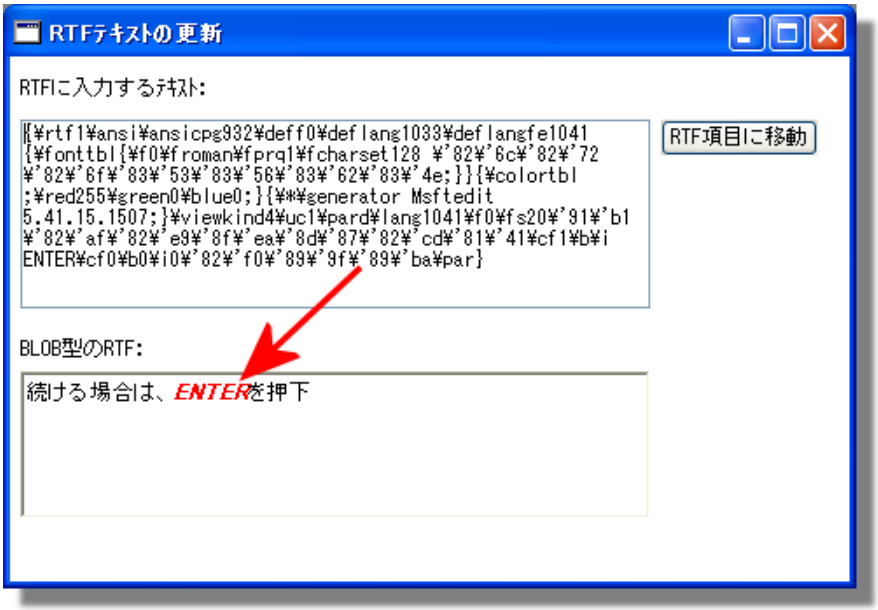
Magic ではアクセラレータを定義することができます。アクセラレータキーを設定するには、ボタンに表示するテキストの前にアンパサンド（&）を設定します。例えば、&C という文字が含まれるボタンは、C と表示され、*Alt+C* を押下することでそのボタンが押下されたことになります。



1. コントロールの書式特性に移動します。
2. アクセラレータキーとして使用したい文字の前に & を挿入します。この例では、文字 D を使用しています。
3. 指定された文字はにはアンダーラインが表示され、*Alt+ 文字*によってボタンが押下される動作になります。この例では、*Alt+D* がアクセラレータキーとなります。

注： 先頭文字は通常アクセラレータキーで、通常大文字で設定され、バックスラッシュを付加する必要があります。このためパーク可能なボタンを使用している場合、構文は多少異なります。上記の例では、U&¥ 削除となります。

動的なリッチテキストを作成するには

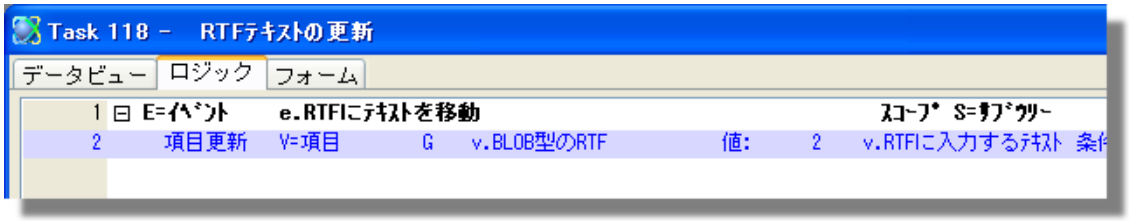


リッチテキスト（RFT）は基本的に HTML または XML のようなテキストマークアップ言語です。従って、フォーマットされたテキストを項目に格納することができ、指定されたフォントや色によって表示することができます。

ユーザは、コンテキストメニューを使用して実行時に RTF フィールド内のテキストを編集することができます。ほとんどのアプリケーションでは、フォーム上に RTF フィールドが定義されており、ユーザがデータのフォーマットを修正できるようになっています。もしフィールドの初期設定が必要であれば、他の文字型項目で行うように、**代入**特性や**デフォルト値**特性にテキストを設定することになります。

しかし、この例のように予めフォーマットされたテキストでフィールドを初期設定することができます。ここでは、テキストフィールド内に格納された RTF テキストを使用しています。

ユーザがボタンを**クリック**すると、**項目更新**処理コマンドによって BLOB 項目の値が更新されます。



テキストを BLOB 項目に格納するために、特別な作業は必要ありません。**項目更新**処理コマンドか**代入**特性を使用したり、**データ**特性に式を定義することで実現できます。

フォーム上に BLOB を表示させるには、**リッチエディット** を選択するか、GUI 表示形式の**スタイル**特性に ***I=リッチエディット*** と定義した BLOB 項目をフォームに配置します。

***ヒント:** リッチテキストを使用するには、このフォーマットのについて深く理解する必要はありません。Magic で RTF スタイルの BLOB 項目を編集し、**Blb2File()** 関数でテキストファイルに保存するだけです。この結果はカット＆ペーストで使用することができます。*

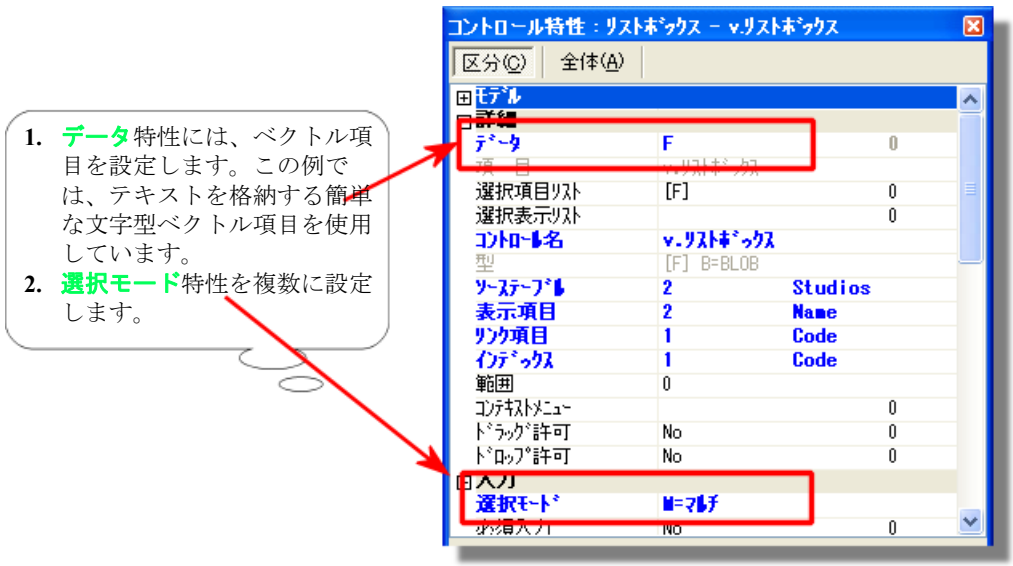
複数選択のリストボックスからデータを取得するには

リストボックスから一度に複数の項目を選択でさせることができます。複数選択のリストボックスは単一選択と同じように設定しますが、以下の特性の値が異なります。

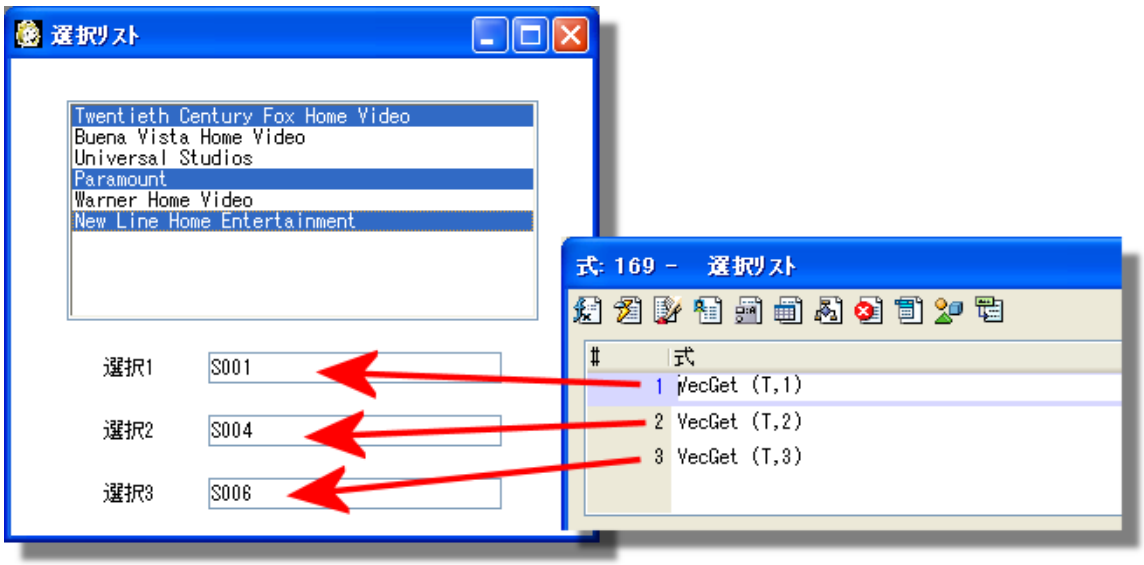
- **選択モード**：*M= マルチ*
- **データ**：ベクトル型の項目

以下のセクションでは、設定例を説明しています。

複数選択のリストボックスを使用する



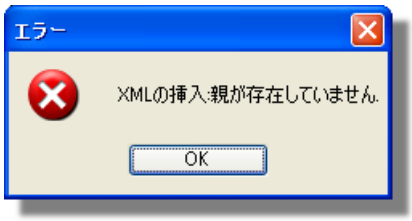
これで、**VecGet()** 関数を使用して更新されたベクトル項目の内容を取り出すことができます。



第 14 章 : XML

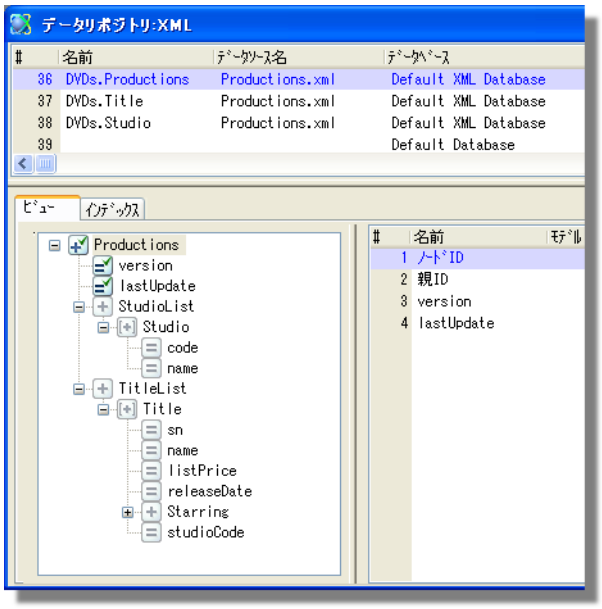
最初から XML ドキュメントを作成するには

XML ビューを定義することで、他のデータソースのように XML ドキュメントを処理することができます。XML ドキュメントは、平面的に見えますが全て階層構造になっています。また、すべての XML ファイルは、ルートディレクトリで初期設定する必要があります。初期設定しないと右に表示されているようなエラーメッセージが表示されます。



XML ドキュメントを初期設定する

例えば、この例のように、Productions.xml ファイルの中には３つのビューがあります。すべて同じ XML ドキュメントをアクセスしていますが、階層の異なる部分がそれぞれ異なるビューとして扱われます。現在のビューで使用されている部分は、対応する項目上で緑色のチェックマークが表示されます（右図を参照）。



さて、この XML ファイルにタイトルを書き込みたい場合は、最初にデータソース（Products）のルートディレクトリを作成する必要があります。現実的には、タイトルをデータソース **#37** に追加する前にデータソース **#36** に 1 つのレコードを書き込むことを意味しています。XML ファイルがまだ存在していない場合、ルートを作成するために書込リンクや簡単なバッチタスクを使用することができます。

データリポジトリで **APG** （***Ctrl+G***）を使用してデータ参照したり、レコードを作成することでこの処理を確認することができます。

ルートレベル（**#36**）でレコードを作成すると、より低いレベル（**#37** と **#38**）でレコードを作成することができ、レコードが正しく挿入されます。しかし、逆の場合は動作しません。

XML スキーマを見つけるには

XML ドキュメントを作成するには、最初に XML スキーマを準備する必要があります。スキーマはデータベース定義（データベース内の項目の書式を定義するもの）のようなものです。以前に XML を扱ったことがない場合は、事前に勉強しておくことをお勧めします。

XML ドキュメントを読み込むように依頼されているのであれば、スキーマはすでに存在するであろうし、それを使用することになります。他の人に渡すために XML ドキュメントを作成している場合、手動でスキーマを設計するか、*XML Spy* などの外部の製品を使用することになります。スキーマは、***.xsd*** という拡張子を持ったテキストファイルです。

どちらの場合も、スキーマが存在しているのであれば、Magic で簡単に XML 定義を作成することができます。

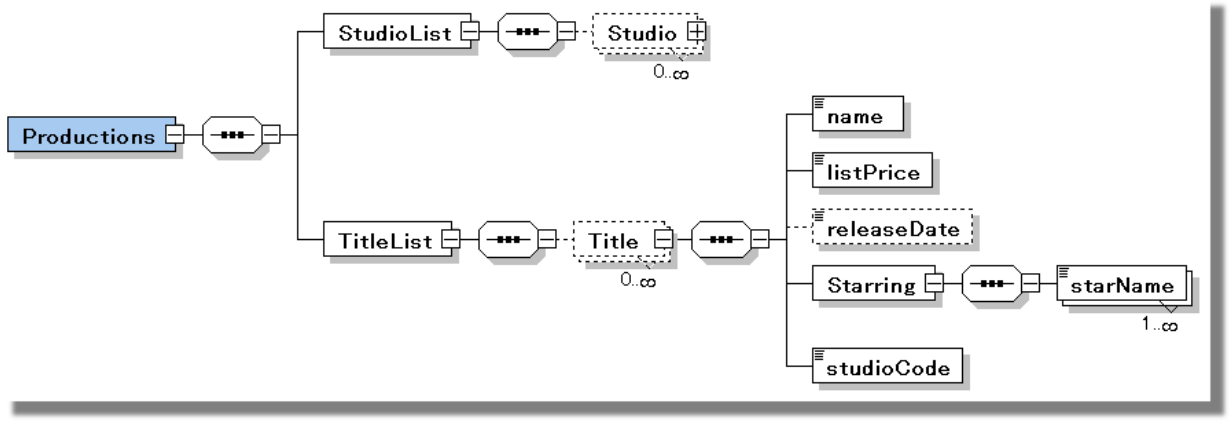
スキーマは複合的なデータの関係を記述しています。本質的に、スキーマはデータベース全体や複数に分離された構成内容、ネストされたテーブルについて記述することができます。データソースとしてこれらを使用するには、スキーマの一部を選択し、それらを **XML ビュー**と呼ばれる個別ののフラットなデータソースとして記述する必要があります。

XML スキーマ

ここでは、以下のスキーマファイルを使用しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="Productions">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="StudioList">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="Studio" minOccurs="0" maxOccurs="unbounded">
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element name="name" type="xs:string"/>
                           </xs:sequence>
                           <xs:attribute name="code" type="StudioCodeType" use="required"/>
                        </xs:complexType>
                     </xs:element>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
            <xs:element name="TitleList">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="Title" minOccurs="0" maxOccurs="unbounded">
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element name="name" type="xs:string"/>
                              <xs:element name="listPrice" type="xs:float"/>
                              <xs:element name="releaseDate" type="xs:date" minOccurs="0"/>
                              <xs:element name="Starring">
                              <xs:complexType>
                              <xs:sequence>
                              <xs:element name="starName" type="xs:string" maxOccurs="unbounded"/>
                              </xs:sequence>
                              </xs:complexType>
                              </xs:element>
                              <xs:element name="studioCode" type="StudioCodeType"/>
                           </xs:sequence>
                           <xs:attribute name="sn" type="xs:string" use="required"/>
                        </xs:complexType>
                     </xs:element>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
         <xs:attribute name="version" type="xs:int" use="required"/>
         <xs:attribute name="lastUpdate" type="xs:date" use="required"/>
      </xs:complexType>
   </xs:element>
   <xs:simpleType name="StudioCodeType">
      <xs:restriction base="xs:string"/>
   </xs:simpleType>
</xs:schema>
```

このスキーマの内容を図式化すると以下のようになります。



また、表にすると以下のようになります。

要素	属性	書式
Productions	**Complex type**	
StudioList	**Complex type**	
Studio	**Complex type**	
Name	Simple Type	xs:string
Code	Simple Type	StudioCodeType
TitleList	**Complex type**	
Title	**Complex type**	
name	Simple Type	xs:string
listPrice	Simple Type	xs:float
releaseDate	Simple Type	xs:date
Starring	**Complex type**	
starName	Simple Type	xs:string
studioCode	Simple Type	StudioCodeType
sn	Simple Type	xs:string
version	Simple Type	xs:int
lastUpdate	Simple Type	xs:date
StudioCodeType	Simple Type	

項目のいくつかは **Complex type** でそれ以外は **Simple type** です。Complex type は、テーブルの行（レコード）のようなものです。Simple type は、テーブルのカラムのようなものです。

従って、Complex type に対して個々のデータソース定義を作成します。この例では、Complex type の Studio に対してデータソースを作成します。

Simple type は、通常 type= という定義を含んでいます。これは、データが string、float、integer または他のスキーマ要素（StudioCodeType ）で記述されたものです。Magic が、データソース内で各項目に対してデフォルトデータ型を作成する場合に使用されます。

参照 : 「スキーマを使用しないで XML ドキュメントを処理するには」（319 ページ）

XML ビューを作成するには

XML ビューは、Magic で XML を使用して処理するためのキーとなるものです。いったん **XML ビュー**が作成されると、他のデータソースと同じように XML ドキュメントを使用することができます。

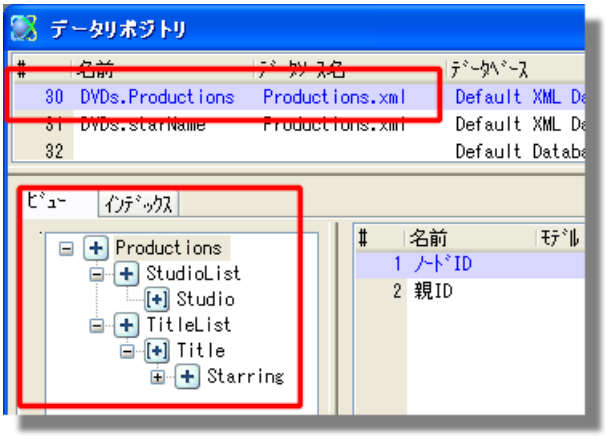
必要条件： 以下が必要になります。

- **データベース**テーブルに、***XML File*** というタイプが定義されたデータベース
- XML スキーマ（「XML スキーマを見つけるには」（296 ページ）を参照してください）。

XML ビューを作成する

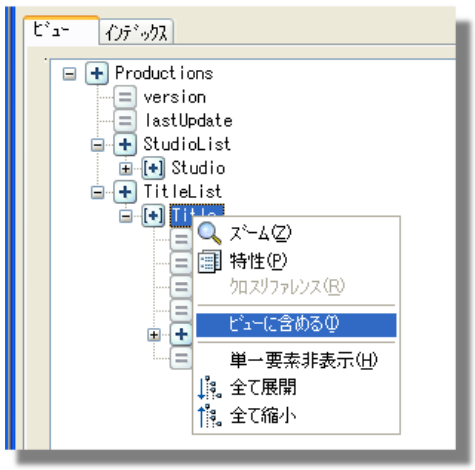
1. **データ**リポジトリに 1 行追加します（***F4*** または、**編集→行作成**）。
2. **データベース**カラムで、タイプが ***XML File*** に設定されているデータベースを選択します。

スキーマ情報を取得するために ***F9***（**オプション→定義取得**）を押下し、**ファイル選択**ダイアログが開きます。
スキーマファイル（***.xsd***）を選択すると、**名前**と**データソース名**のカラムにスキーマの最初の要素名がデフォルトとして設定されます。
この例では、Productions です。

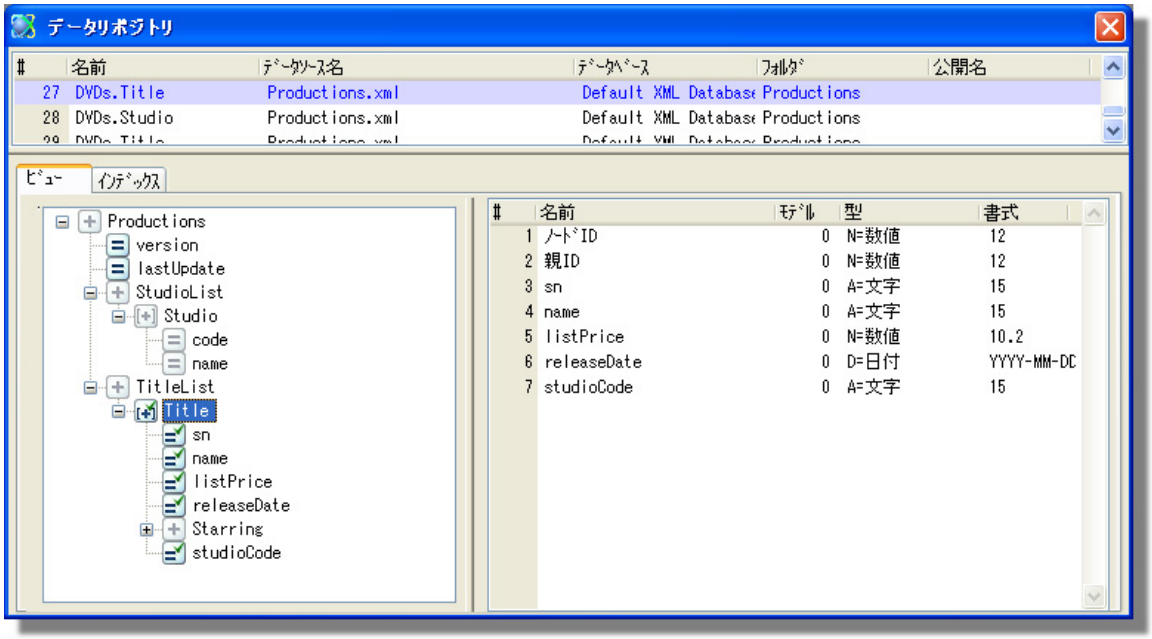


.

3. **ビュー**タブには、***.xsd*** 内の情報をもとに自動的に値が設定されます。
   複合要素（+ アイコンを持つもの）のみ参照できます。また単一要素を表示するには、コンテキストメニューから**単一要素表示**を選択してください。右の図では、単一要素が表示されます（メニューは**単一要素非表示**に変わります）。



4. タイトル要素内にビューを作成した場合、タイトル要素に移動し、コンテキストメニューの**ビューに含める**を選択します。これで、**ノード ID** と**親 ID** によってルート要素とリンクされるサブ要素を持つことになります。

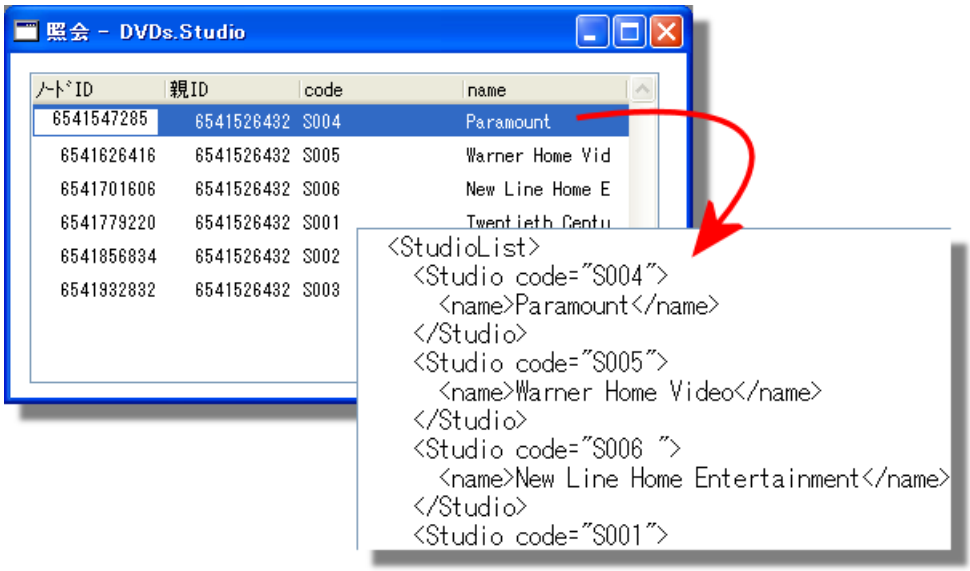


5. データが格納される実際の XML ドキュメントの名前を反映するために、**データソース名**を変更します。パス名を直接記述しないで論理名や相対パスを使用することを推奨します。
6. この **XML ビュー**を呼び出す際に必要な**名前**を（必要であれば）変更します。

これで他のデータソースと同じように、TitleList にアクセスすることができます。

注： XML ドキュメント内の他の要素についても同じように **XML ビュー**を作成する必要があります（「XML ファイル内の合成要素にアクセスするには」（301 ページ）を参照してください）。また、XML ドキュメントに書き込む前にルート要素を初期化する必要があります（「最初から XML ドキュメントを作成するには」（295 ページ）を参照してください）。

ノード ID と親 ID を更新するには



更新はできません。**ノード ID** と**親 ID** は Magic 内部で使用され、プログラムで更新しようとしても無視されます。

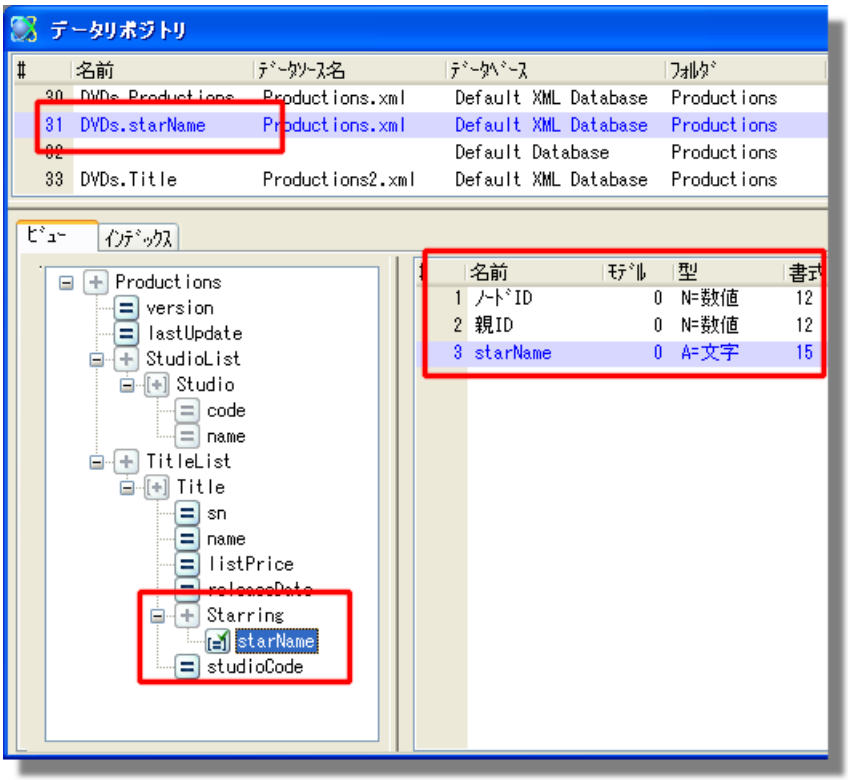
この例のように、実際のドキュメント内に明示的に保存されませんが、XML ドキュメントがオープンされると、繰り返し要素内の各レコードはユニークな**ノード ID** が設定されます。**親 ID** は XML ドキュメント内の親要素を参照します。この例では、ルートノードになります。

しかし、XML プログラムの作成にはこれらのことを理解する必要がありません。**ノード ID** と**親 ID** は無視できます。

XML ファイル内の合成要素にアクセスするには

XML ドキュメントを使用するということは、定義上、階層型データベースを使用することになります。一般的に、XML ドキュメントは複数のレベルを持っています。これらが一緒に保存されていても、SQL や ISAM テーブルの階層的なデータのように処理されます。

繰り返し要素



この例では、Productions という XML ドキュメントには複数の繰り返し要素が定義されています。

- ルートレベル：Productions は 1 レコードのみ持っています。
- Productions 以下
- StudioList- 複数の要素があります。
- TitleList- 複数の要素があります。
- TitleList 以下
- Starring list- 複数の要素があります。

各レベルに対して 1 つの **XML ビュー**が必要です。Magic は、このような設定にもとづいてレコードが書き込まれると、階層を正しく維持するための処理を実行します。

合成要素を選択する

1. 含めたい単一要素上の 1 つのレベルであるノードを選択します。
2. コンテキストメニューから、**ビューに含める**を選択します。
3. 子レコードのために、リンクフィールドも含めるようにします。「既存の XML ドキュメントを修正するには」（302 ページ）を参照してください。

これで、単一要素は **XML ビュー**にフィールドとして表示され、Magic プログラムからアクセスすることができます。詳細は、「XML ビューを作成するには」（298 ページ）を参照してください。

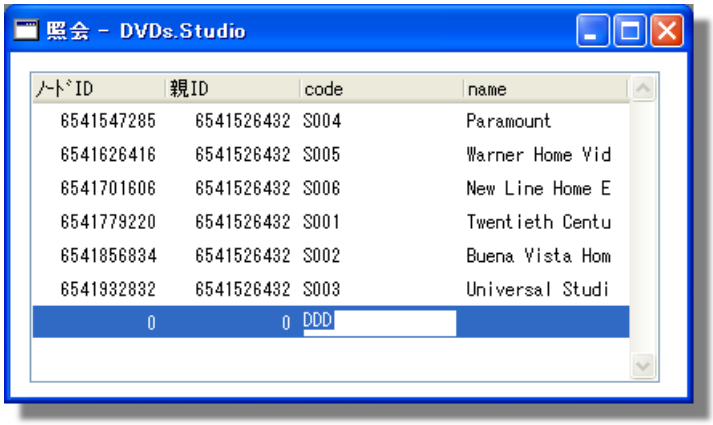
注： 単一要素は、要素名の前に = アイコンが表示され、複合要素は + アイコンが表示されます。このため単一要素と複合要素は区別することができます。単一要素がビュー上に表示されない場合、コンテキストメニューから単一要素表示を選択します。

参照： 「既存の XML ドキュメントを修正するには」（302 ページ）

既存の XML ドキュメントを修正するには

いったん XML ドキュメントが作成され、**XML ビュー**が設定されたら、ISAM や SQL テーブルのようにアクセスできます。基本的なレコードの処理は自動的に行われます。

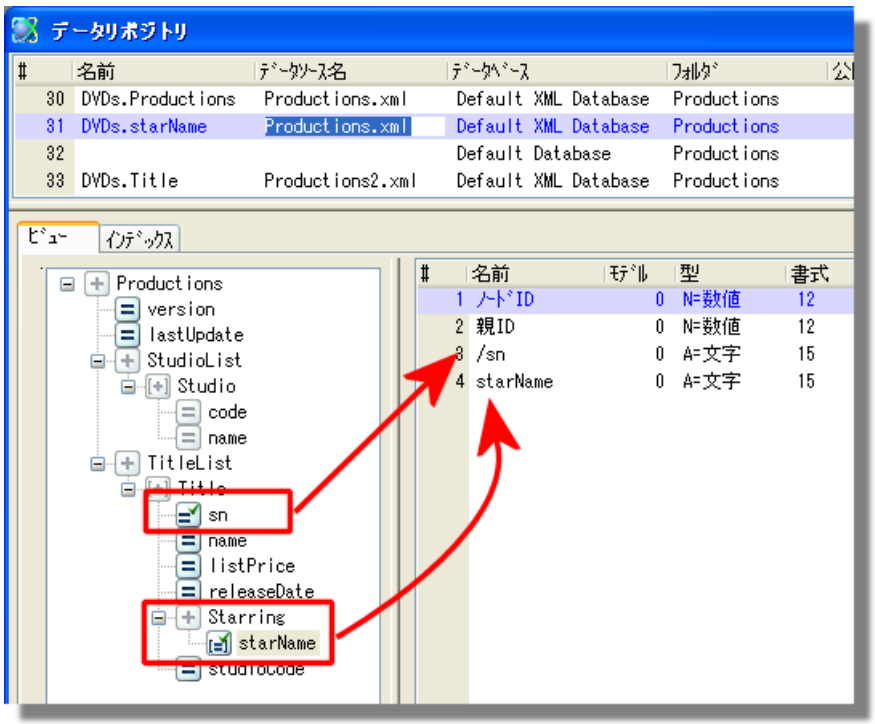
親レコードにアクセスする



他のテーブルと同じように親レコードを使用することができます。また、登録、修正、削除の処理を行うこともできます。この例では、ブラウザクライアントのプログラムを作成するために、**APG**（*Ctrl+G*）を実行し、テストレコードを追加しています。ノード ID や親 ID に対しては何も行いません（「ノード ID と親 ID を更新するには」（300 ページ）を参照してください）。

ただし、先頭のレコードは、XML ドキュメントのルートディレクトリではありません。この例では、Studio データ（これは ISAM または SQL の DBMS では子レコードとして考慮されません）にアクセスしていますが、XML ドキュメント内では、ProductctionsXML ドキュメントの 1 つの要素になります。レコードを追加する前に、ルートディレクトリ Productions を書き込まなければなりません。「最初から XML ドキュメントを作成するには」（295 ページ）を参照してください。

子レコードにアクセスする



さて、子レコードを作成する際、親とリンクさせるにはどのようにすればいいでしょうか？

この場合、２つの手順で実現することができます。

最初に、**XML ビュー**を作成している時に、子レコードに対してビューにリンク項目を含めます。

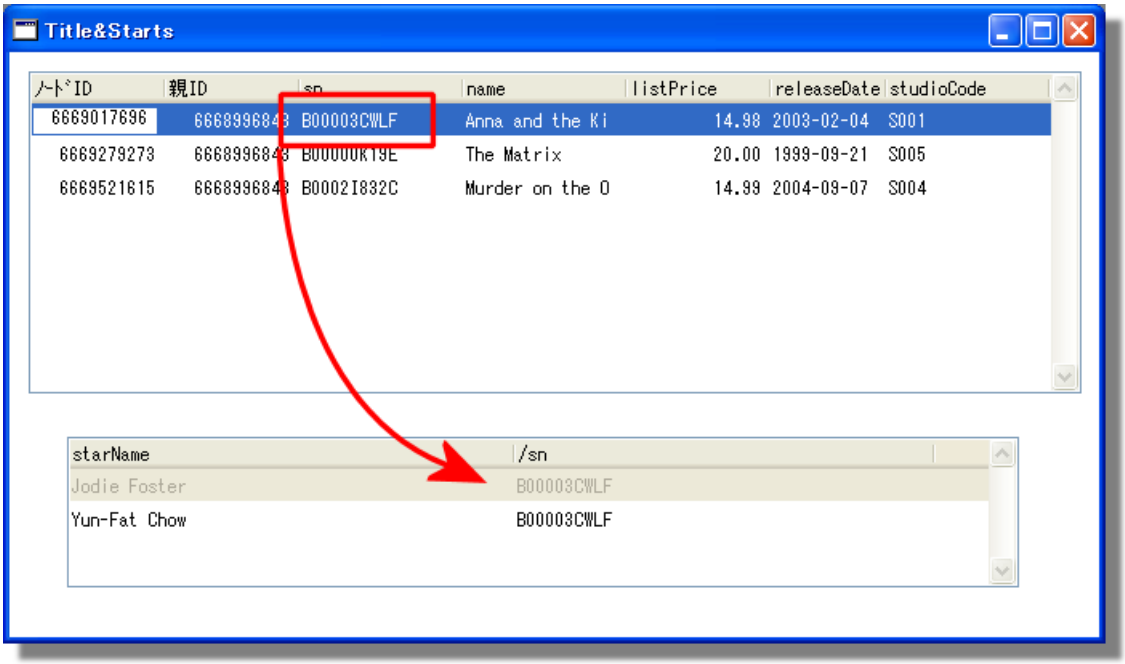
以下の手順で使用したいリンク項目を選択します。

1. リンクで使用する親レベルの要素上にカーソルを置きます。
2. コンテキストメニューから、**ビューに含める**を選択します。

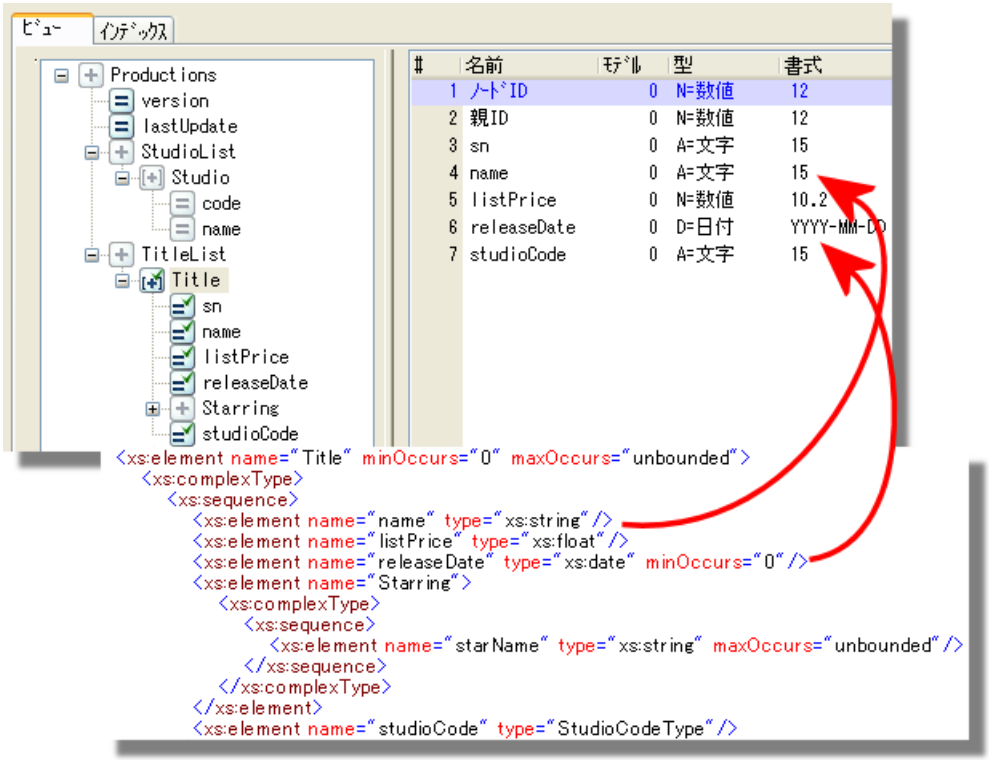
これで、リンク項目がビューに含められます。この例では、タイトルの sn（serial number）項目が子レコードで選択されます。それがリンク項目であることを示すために、名前の前にスラッシュ（/）が追加されます。

これで、レコードがサブタスク内で使用される場合、リンク項目は自動的に更新されます。

この例では、この映画と関連する俳優のみ表示するように範囲パラメータを使用しています。しかし、/sn 項目には**代入**特性が設定されていません。それにもかかわらず、プログラムを実行した場合、/sn 項目は範囲のために使用された親 sn によって初期設定されます。

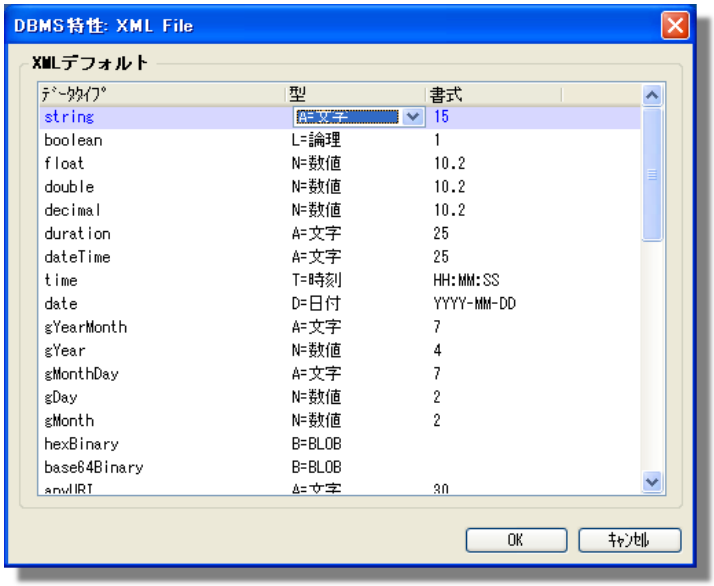


XML データ型に対応する Magic のデータ型を決定するには

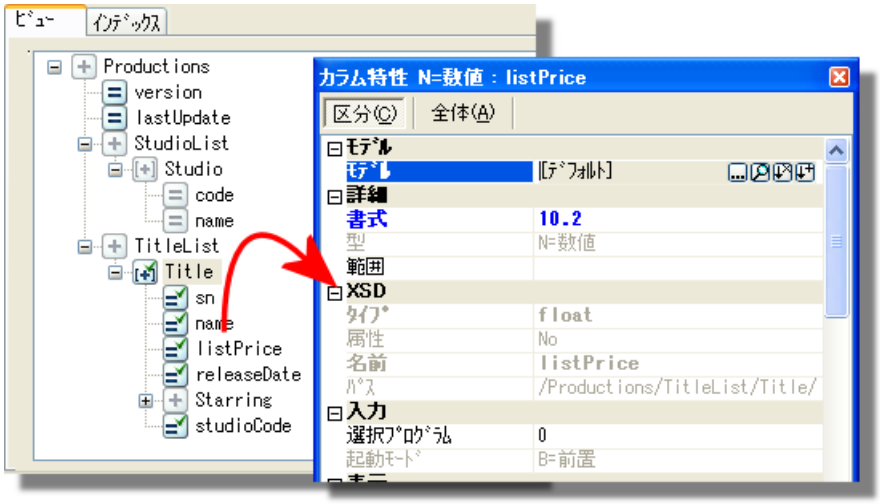


XML ビューを作成する場合、Magic は XML スキーマを参照して、**XML デフォルト特性**（**設定→ DBMS →特性**）の定義内容にもとづいてデータが変換されます。この例では、XML データ型 float は、Magic のデータ型の 10.2 に変換されます。

他のツールで作成された XML ドキュメントがある場合、**XML ビュー**の項目定義を変更することで、実際のデータを反映する **XML ビュー**を変更することができます。例えば、実際の XML ドキュメントではより長いタイトル名が必要なため、名前を 15 文字から 40 文字に変更しています。



スキーマの設定を表示する

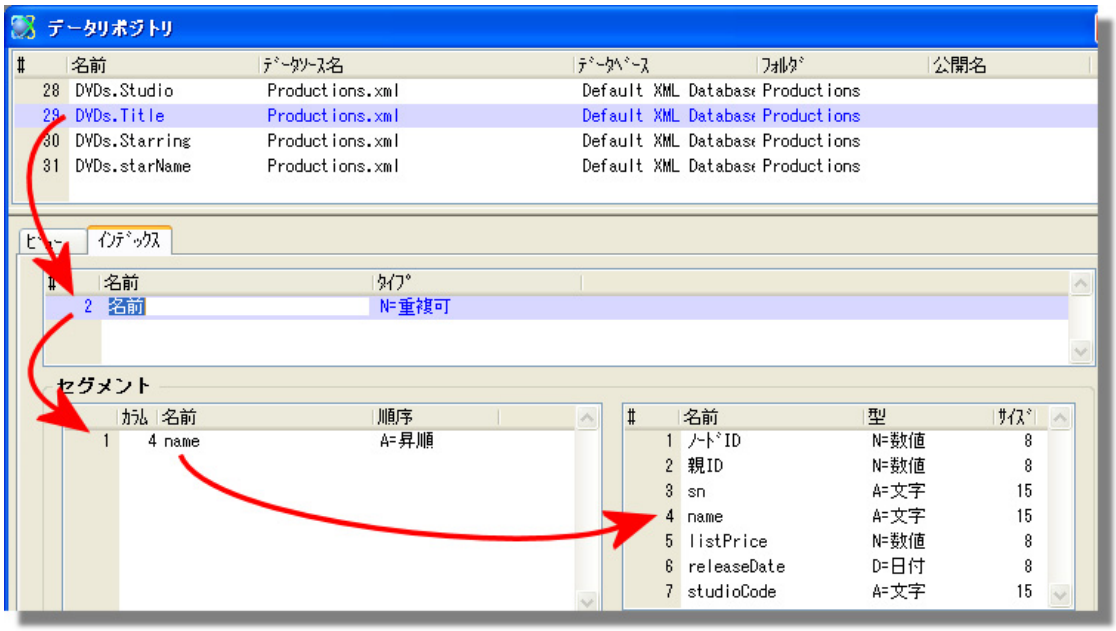


要素が **XML ビュー**に含まれている場合、スキーマ内にどのように定義されているか確認するためにスキーマを直接参照する必要はありません。**特性シート**の **XSD** セクションには、スキーマ設定が表示されます。

任意の順番で XML ドキュメントからデータを取得するには

XML ビューは、データを格納するために一時的なテーブルを作成していることを知っておいてください。一時的なテーブルのため、必要であればインデックスを追加して他のデータソースと同じように使用することができます。

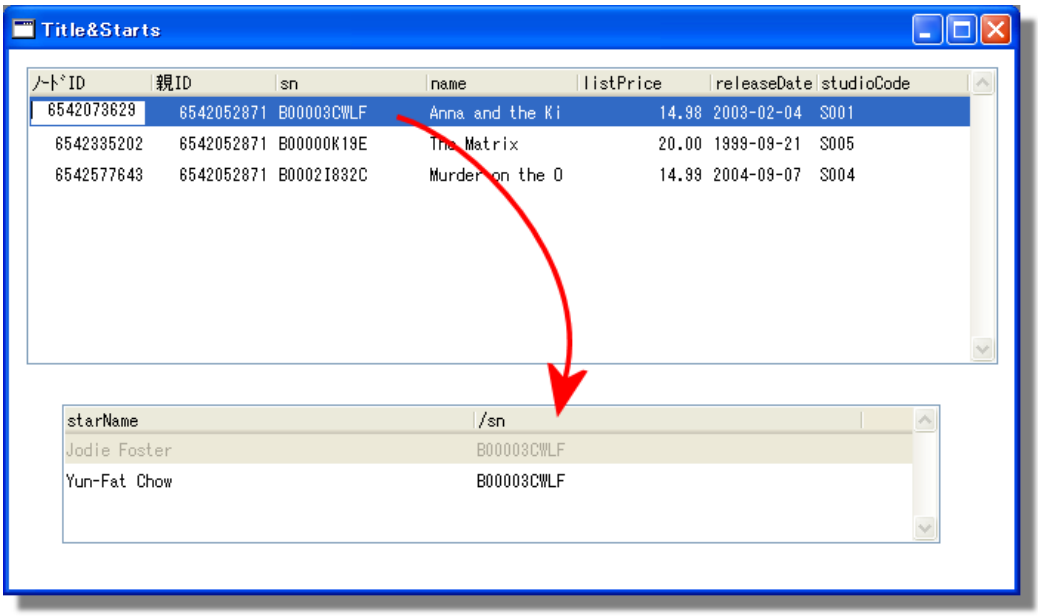
XML ビューのために代替インデックスを作成する



1. **データ**リポジトリで使用する **XML ビュー**を選択します。
2. **インデックス**タブを**クリック**します。
3. ***F4*** を押下して１行追加します。
4. インデックスの名前を入力します。右に ***Tab*** 移動します。
5. 重複レコードを許可する場合は、**タイプ**カラムで ***N= 重複可***の選択します。
6. **セグメント**エリア（画面の左下部分）を**クリック**します。***F4*** を押下して１行追加します。
7. **ズーム**（***F5***）して、**項目一覧**に移動し、インデックスセグメントに追加する項目を選択します。
8. 必要な項目に対して、ステップ **#6** と **#7** を繰り返して追加します。

これで、プログラムで作成されたインデックスを使用すると、データはそのインデックスにもとづく順番に表示されます。

XML ドキュメント内のマルチ発生要素を処理するには

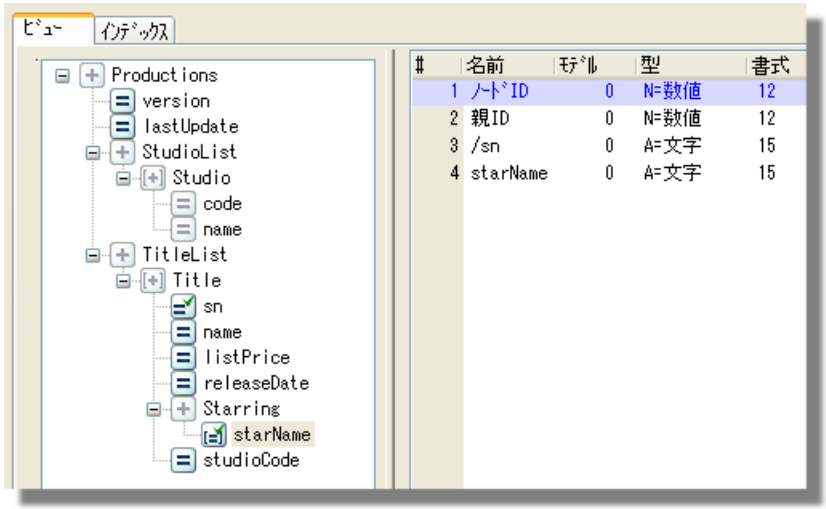


XML ドキュメント内のマルチ発生要素は、ISAM/SQL のデータソースと同じ方法で子レコードとして処理されます。子レコードは子タスク（子タスクのフォームやサブフォーム）で表示されます。範囲が指定されるため、親のレコードと関連するレコードのみ表示されます。

以下は、循環型の XML 要素を使用してプログラムを作成する方法について説明しています。

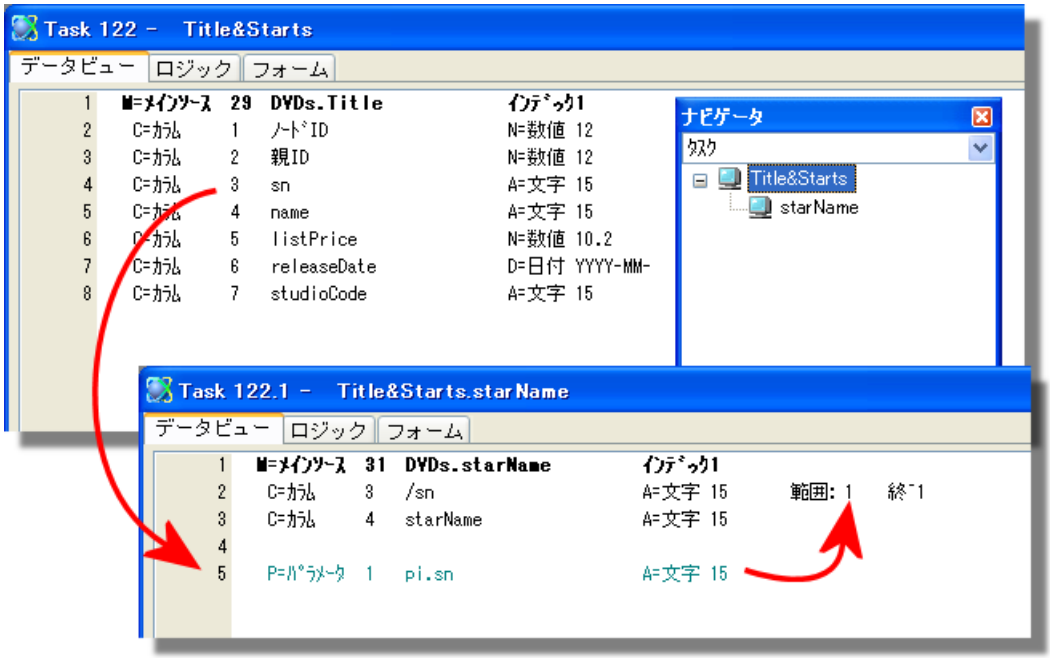
ヒント : *アイコンの表示内容で、スキーマ内の繰り返し要素を確認できます。繰り返し要素は、アイコンが角括弧で囲まれた状態で表示されます。これらは maxOccurs=unbounded または maxOccurs=>1 と定義されているため、スキーマ内で繰り返し要素が確認できます。*

繰り返し要素を表示する



1. 最初に、繰り返し要素のための **XML ビュー**を定義する必要があります。この例では、リンク項目である sn と繰り返し要素 starName が含まれています。

2. XML ビュー上で sn によって要素にアクセスすることができるように、インデックスを作成します。



3. 親要素 Title を表示するタスクを作成します。このタスクで、/sn 項目を定義します。/sn 項目をパラメータとしてサブタスクに渡します。
4. また、starName XML ビューにアクセスするサブタスクを作成します。ここで使用されるインデックスは、/sn 項目で、ビューの範囲指定のためにパラメータ同じ項目を使用します。
5. サブフォームは、サブタスクを表示するために親タスクのフォーム上で使用されます。（第 8 章：「サブフォーム」（175 ページ）を参照してください）。

これで、繰り返し要素がサブフォームに表示されます。

同じ XML ドキュメントを異なるユーザで共有させるには

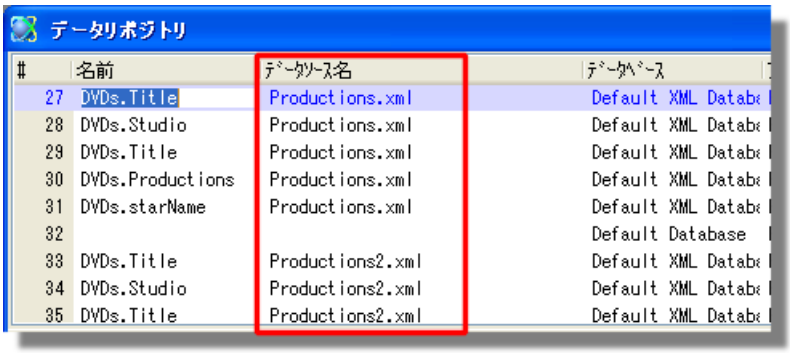
複数のユーザが、XML ドキュメントにアクセスする必要がある場合、読み込み専用モードでドキュメントがオープンされているかを確認する必要があります。

XML ドキュメントは、DBMS で管理されていないため、複数のユーザで同時に更新するはできません。同時に複数のユーザによって更新される可能性のある XML ドキュメントを作成する必要がある場合、SQL/ISAM データソースとしてデータを格納し、必要に応じて保存データから XML ドキュメントを作成するようにしなければなりません。

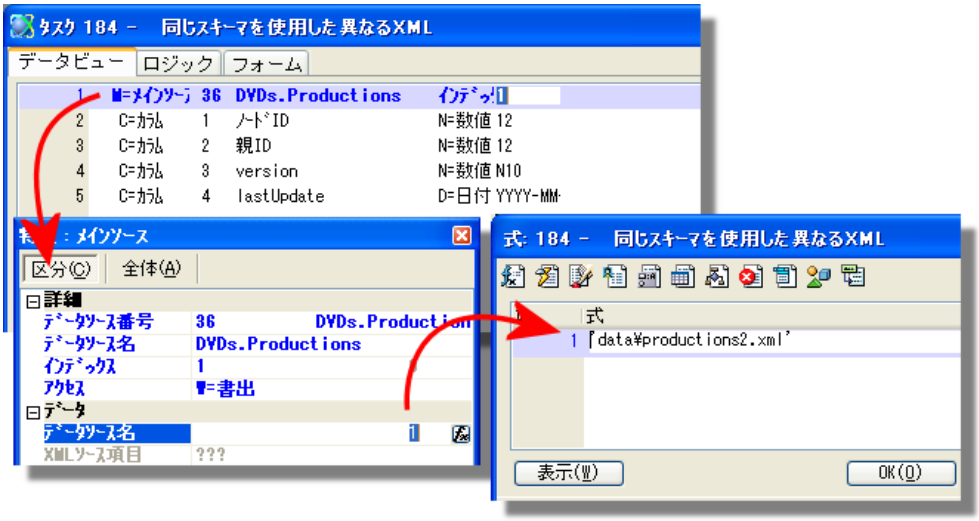
XML ドキュメントにアクセスモードを設定する

1. データソースの設定を変更したいメインソースまたはリンクのヘッダ行に移動します。
2. *Alt+Enter* を押下して、特性シートを開きます。
3. アクセス特性を、*R= 読込*に設定します。このタスクは XML ドキュメントを読み込み専用モードでアクセスします。これで、異なるユーザがが同時にこのタスクを使用することができます。
4. 共有特性を、*R= 読込*か *W= 書込*に設定します。このタスクが XML ドキュメントを読み込んでいる間、他のタスク（または外部の製品）が、XML ドキュメントを更新できるようにする必要があれば、書込に設定します。*N= なし*には設定しないでください。この設定にした場合、一度に 1 人のユーザだけがデータを参できるようになります。

同じスキーマを使用した異なる XML ドキュメントを作成するには



XML ビューは、**定義取得機能**（*F9*）を使用して設定されたスキーマにもとづいて作成されます。しかし、データは、**データ ソース名**カラムに入力された名前によって定義されたファイルに格納されます。

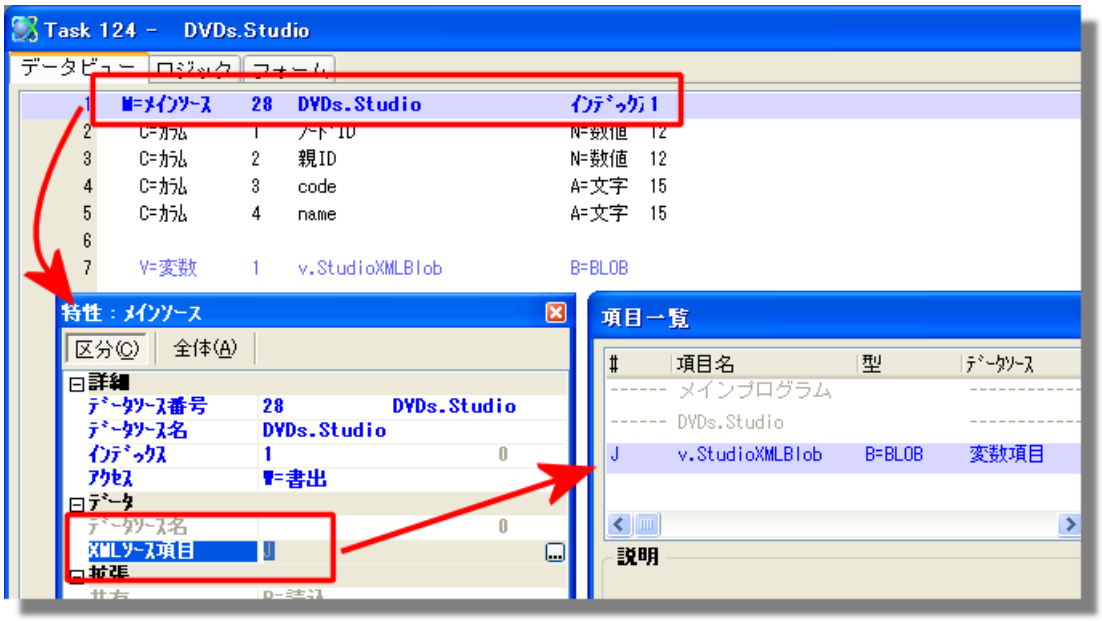


注： ドキュメントにアクセスするプログラムで XML ドキュメント名を上書きすることができます。これは、SQL/ISAM ファイルの場合と同じように**データ ソース名**特性に式を定義することで可能です。この例では 、ファイル名が直接設定されていますが論理名やデータ項目を使用することもできます。

Magic のデータ項目に格納された XML ドキュメントにアクセスするには

BLOB データをファイルに変換せずに、BLOB から直接 XML ドキュメントにアクセスすることで **XML ビュー**を使用することができます。

データソースとして BLOB を使用する



1. メインソースに **XML ビュー**を指定します。
2. **メインソース特性** （*Alt+Enter*）を開きます。
3. **XML ソース項目**特性に移動し、**ズーム** （*F5*）して項目一覧を開きます。
4. XML ドキュメントを含んでいる BLOB を選択します。

これで、ファイルと同じ方法で XML ドキュメントを使用することができます。

参照： 「同じスキーマを使用した異なる XML ドキュメントを作成するには」（310 ページ）

入出力ファイルとして BLOB 項目を使用する

XML データビューではなく XML 関数を使用する場合、**入出力ファイル**テーブルの**式 / 項目**カラムに BLOB 項目を定義しておく必要があります。

1. *Ctrl+I* を押下して**入出力ファイル**テーブルを開きます。
2. *F4* を押下して 1 行追加します。
3. **メディア**カラムで ***V= 項目***を選択します。
4. **式 / 項目**カラムで、アクセスしたい BLOB 項目を選択します。

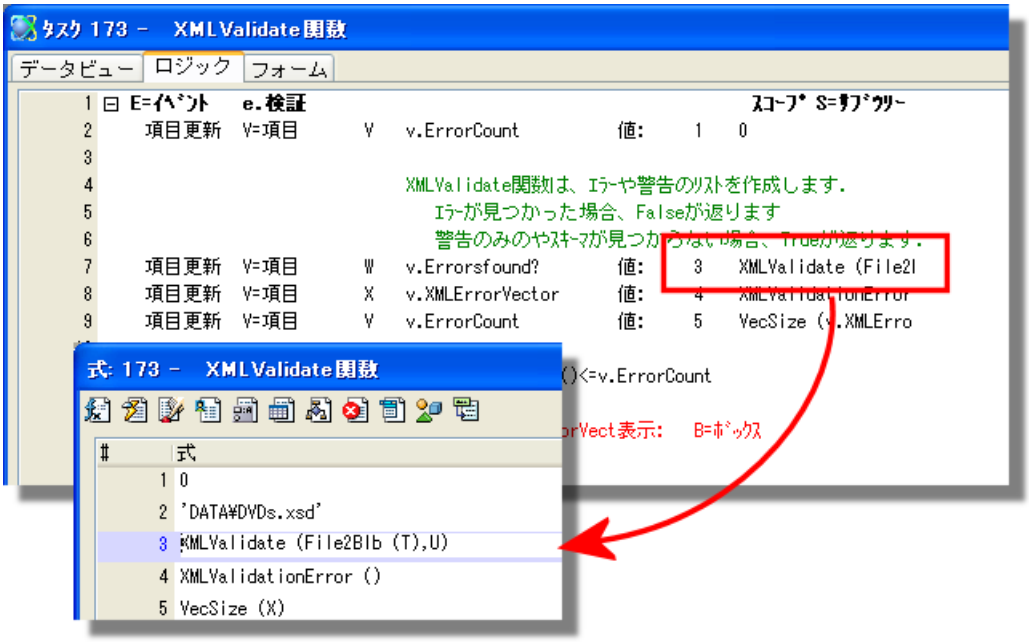
これで、XML 関数を使用して入出力ファイルにアクセスすることができます。タスクの世代番号とファイルの連番を指定することで入出力ファイルを参照できます。

参照 : 「スキーマを使用しないで XML ドキュメントを処理するには」（319 ページ）

XML ドキュメントを検証するには

XML ドキュメントを使用する前に、予め検証しておくことを推奨します。XML が正しい書式になっていなかったり、スキーマ定義と合っていない場合、正しい結果が得られません。

XML ドキュメントを検証する場合、**XMLValidate()** 関数を使用します。この関数は、XML ドキュメントとスキーマを比較し、エラーのリストを作成します。



XMLValidate() 関数を使用する

XMLValidate() 関数を使用する基本的な手順は以下の通りです。

1. **XMLValidate()** 関数を実行し、結果を論理データとして格納します。エラーがある場合は、False が返ります。警告だけであったり、スキーマが指定されなかった場合は、True が返り、メッセージも出力されます。
2. **XMLValidationError()** 関数を起動してエラーや警告内容をベクトル項目に格納します。
3. **VecSize()** 関数を使用して、どれだけのエラーや警告がベクトル項目内ににあるかを確認します。
4. ベクトルデータを読み込むために、**ブロック While** 処理コマンドを使用してループ処理を実行します。ここで、例にあるような警告ボックスを使用して 1 つずつメッセージを表示したり、テーブルに格納してまとめて参照できるようにします。

「XML ドキュメントの検証エラーを取得するには」（314 ページ）で説明されているような検証エラーを処理するグローバル関数を作成することもできます。

XMLValidate() 関数の構文は以下の通りです。

XMLValidate(*XMLBLOB*, *[XSDSchema]*)

パラメータ：

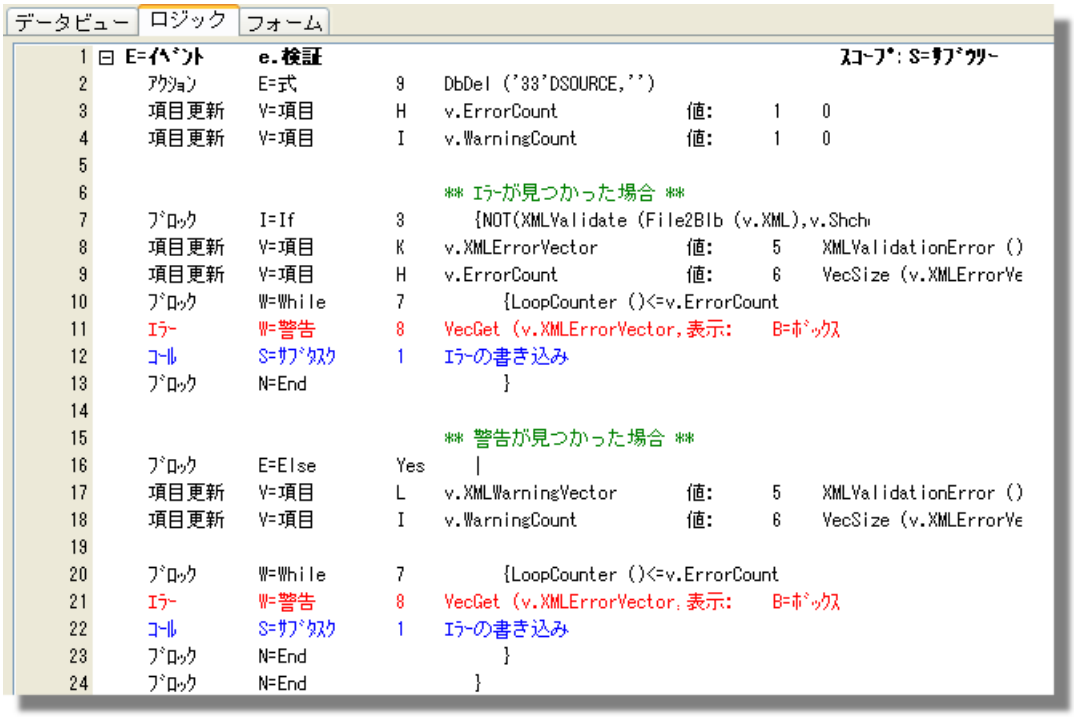
- ***XMLBLOB*** ：XML ドキュメントが含まれている BLOB 項目
- ***XSDSchema*** ：スキーマの位置を URL 形式で指定する

戻り値: XML ドキュメントにエラーが見つかった場合は、False が返ります。エラーがないか警告内容のみの場合は、True が返ります。

参照： 『リファレンスヘルプ』の XMLValidate 関数のトピック
「XML ドキュメントの検証エラーを取得するには」（314 ページ）

XML ドキュメントの検証エラーを取得するには

XMLValidate() 関数を実行した後、エラーがあれば **XMLValidationError()** 関数を使用してエラー情報を取得することができます。この関数はテキストメッセージ（Unicode）のベクトルデータが返ります。**VecSize()** 関数を使用することでエラーメッセージの数を取得することができ、テーブルへの格納や、印刷や表示を行うことができます。



参照： 「XML にアクセスしている間に発生したエラーを処理するには」（315 ページ）

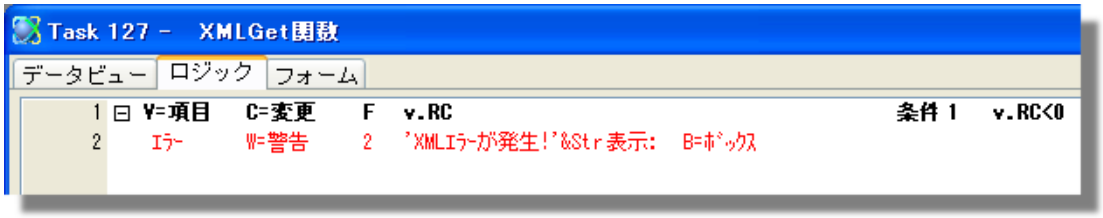
XML にアクセスしている間に発生したエラーを処理するには

XML 関数のいくつかは、エラーが発生したことを確認するための戻り値を返します。例えば、
XMLSetEncoding() 関数は適切に動作した場合は、0 が返り。そうでなければ負数が返ります。

注： 戻り値は負数になる場合があるため、格納する項目は、負値に対応した書式（、例えば N や 5N などを含む書式）にする必要があります。

グローバルなエラーハンドラを作成する

すべてのエラーコードは負の整数です。従って、ここで示されているように、戻り値が負数の場合に実行されるハンドラを作成することができます。



メインプログラム内で戻り値を使用することで、すべての XML 関数のエラーを処理することができます。同じ戻り値がすべての XML 関数で使用されるため、ユーザフレンドリなエラーメッセージを作成することもできます。エラーコードは、『リファレンスヘルプ』に記述されています。

ドキュメント対するエンコードを指定するには

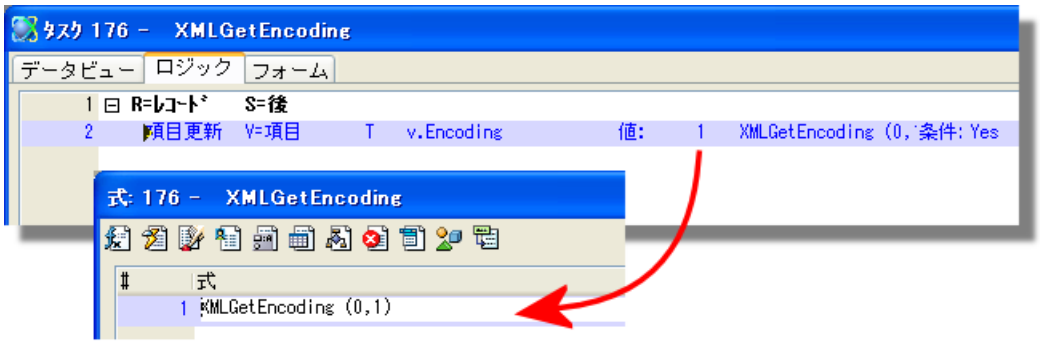
以下に示されるように、XML ドキュメントのエンコーディングは XML ドキュメントのヘッダ（の最初の行）に指定します。

```
<xml version="1.0" encoding="Shift-Jis"?>
```

エンコーディングが指定されない場合、utf-8 として処理されます。エンコーディング指定は、XML を構文解析する際にパーザによって使用されます。文字によっては、あるエンコーディングでは有効であっても、別の指定では無効になる場合があります。

Magic で XML を処理する場合、エンコーディングの指定内容を意識する必要はありませんが、
XMLGetEncoding() 関数を使用することで取得することができます。

XMLGetEncoding() 関数を使用する



XMLGetEncoding() 関数の構文は以下の通りです。

XMLGetEncoding(*generation*, *I/O entry*)

パラメータ：

- *generation*：タスクの世代番号です。0 が現在のタスク、1 は親タスクになります。
- *I/O entry*：XML ドキュメントが割り当てられている入出力ファイルの番号です。

戻り値は、XML ドキュメントに設定されている encoding 指定の値が文字列で返ります。

参照： 『リファレンスヘルプ』の XMLGetEncoding および XMLSetEncoding 関数のトピックを参照してください。

Base64 でエンコードされた XML データを処理するには

XML ビューを使用している場合、項目が Base64Binary としてスキーマで定義されていれば、Magic はテキスト⇔バイナリ値の自動変換を行います。

例えば、ここに、2 つの Base64 の要素が追加された Sample.xsd があります。

```
<xs:element name="Studio" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="RTF" type="xs:base64Binary"/>
        /xs:sequence>
        <xs:attribute name="code" type="StudioCodeType" use="required"/>
```

このスキーマを使用して **XML ビュー**を作成すると、以下のように表示されます。

この例では、RTF テキストを格納するために BLOB が使用されていますが、イメージやその他のバイナリデータを含めることもできます。

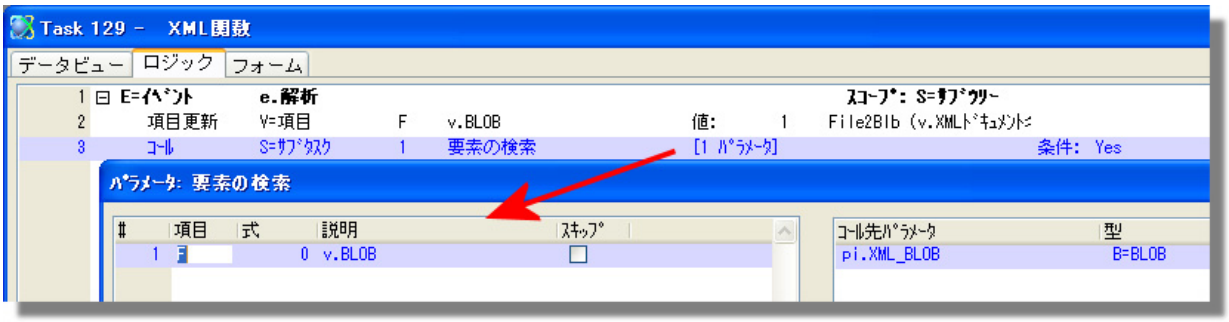
RTF 項目にデータを入力し、XML ファイルに格納された場合、以下のように保存されます。

```
<Studio code="S001">
    <name>Universal Studios</name>
    <RTF>c2QgIGFzZGYgYWRmIGQKDWFkZiBhc2RmCg1hZHNmYXNkZmFkc2YKDWFzZCBmNZg==</RTF>
  </Studio>
```

Magic は、上記の処理を全て行うことできます。

注：　XML ビューに対して 1 つの BLOB のみ定義できます。

XML ドキュメントをパラメータとして渡すには

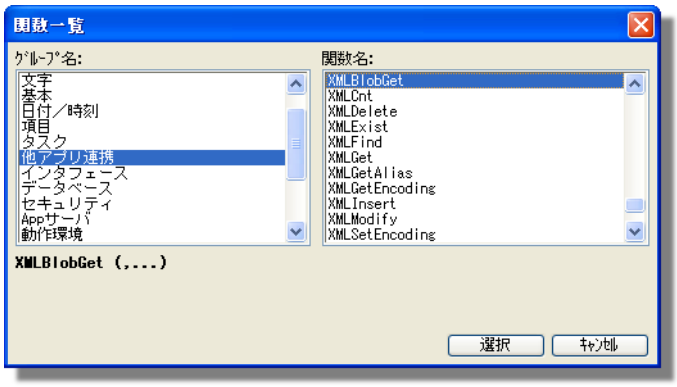


パラメータとして XML ドキュメントを渡す場合、最も簡単な方法は、ドキュメントを BLOB 項目に格納した上で、直接アクセスすることです。「Magic のデータ項目に格納された XML ドキュメントにアクセスするには」（311 ページ）を参照してください。

ファイルを BLOB に変換するには、**File2BLB()** 関数を使用します。使用方法は、『リファレンスヘルプ』を参照してください。

スキーマを使用しないで XML ドキュメントを処理するには

データリポジトリで **XML ビュー**を使用するには、スキーマを使用しなければなりません。スキーマのない XML ドキュメントがある場合、サードパーティの製品を使用するか、手動で作成する必要があります。



これ以外には、Magic の組み込み機能を利用してアクセスすることもできます。これらの機能は、スキーマを使用しない代わりに XML ファイルの内部フォーマットに関する情報を取得しなければなりません。これらの機能の使用方法は、以下で説明しています。

「XML ドキュメント内のデータの取得、更新、挿入を行うには」（320 ページ）

「XML ドキュメント内のデータ要素と階層構造を識別するには」（322 ページ）

XML ドキュメント内のデータの取得、更新、挿入を行うには

XML ファイルにアクセスするために **XML ビュー**を定義することで、他のデータソースと同じように XML データソースをタスクで使用することができます。**XML ビュー**の使用については、「最初から XML ドキュメントを作成するには」（295 ページ）を参照してください。

しかし、XML ファイルが定義された入出力ファイルを処理するために XML 関数を使用することもできます。これは **XML ビュー**を使用する場合より作業量が増えることになります。これらの関数は、主に Magic V9Plus との互換性を維持するために存在しています。

必要条件： データの更新や挿入を行う場合、ファイルのオープンモードを ***書込***（***追加***ではない）にする必要があります。また、データの内容が属性かどうかによって構文が多少異なります。

フォーマット要素パスと属性名の詳細については、「XML ドキュメント内のデータ要素と階層構造を識別するには」（322 ページ）を参照してください。 これらはかなり複雑な関数です。『リファレンスヘルプ』の関数のトピックを参照してください。

XML データを取得する

XMLGet() 関数は、XML ファイル内の要素を取得します。構文は以下のとおりです。

XMLGet(*generation*, *file*, *element path*, *attribute name***)**

パラメータ：

- *generation*：タスクの世代番号です。0 が現在のタスク、1 は親タスクになります。
- *file*：XML ドキュメントが割り当てられている入出力ファイルの番号です。
- *element path* ：要素パスを表す文字列です。
- *attribute name*：要素内の属性を表す文字列です。

戻り値：関数の処理が成功した場合、要素または属性を表す文字列が返ります。処理が失敗した場合は、空白が返ります。

XML データを更新する

XMLModify() 関数は、XML ファイル内の要素を更新します。構文は以下のとおりです。

XMLModify(*generation*, *I/O entry*, *element path*, *attribute*, *value [*, *auto convert]***)**

パラメータ：

- *generation*：タスクの世代番号です。0 が現在のタスク、1 は親タスクになります。
- *I/O entry*：XML ドキュメントが割り当てられている入出力ファイルの番号です。
- *element path*：要素パスを表す文字列です。
- *attribute*：要素内の属性を表す文字列です。
- *value*：修正される要素／属性の値を含む文字列です。
- *auto convert* ：（オプション）True が設定された場合、正規の XML フォーマットに変換します。

戻り値：関数の処理が成功した場合、0 が返ります。処理が失敗した場合は、負数のエラーコードが返ります。

XML データを挿入する

XMLInsert() 関数は、XML ファイル内に要素を追加します。構文は以下のとおりです。

XMLInsert (*generation*, *I/O entry*, *element path*, *attribute*, *value [*, *before/after flag*, *reference element*, *auto convert]***)**

パラメータ：

- *generation*：タスクの世代番号です。0 が現在のタスク、1 は親タスクになります。
- *I/O entry*：XML ドキュメントが割り当てられている入出力ファイルの番号です。
- *element path*：要素パスを表す文字列です。
- *attribute*：要素内の属性を表す文字列です。
- *value*：挿入される要素／属性の値を含む文字型、BLOB 型（RTF）のデータです。

- *before/after flag*：（オプション）追加する場所（*reference element* の前 / 後）を指定します。
- *reference element* ：（オプション）追加する際の参照用に指定する要素名
- *auto convert* ：（オプション）True が設定された場合、正規の XML フォーマットに変換します。

戻り値：関数の処理が成功した場合、0 が返ります。処理が失敗した場合は、負数のエラーコードが返ります。

XML データを削除する

XMLDelete() 関数は、XML ファイル内の要素を削除します。構文は以下のとおりです。

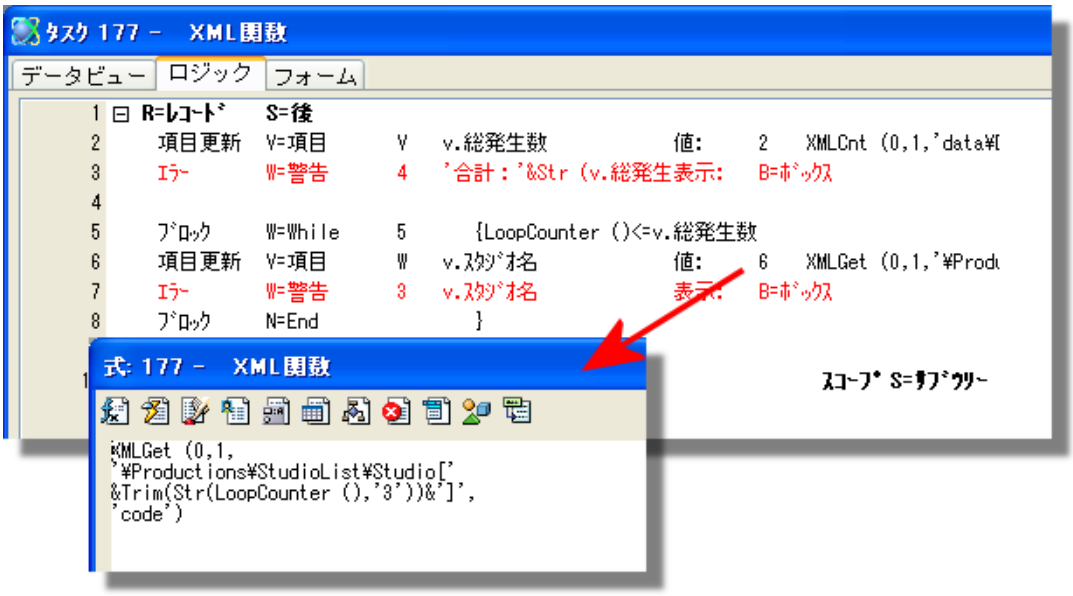
XMLInsert (*generation*, *I/O entry*, *element path*, *attribute***)**

パラメータ：

- *generation*：タスクの世代番号です。0 が現在のタスク、1 は親タスクになります。
- *I/O entry*：XML ドキュメントが割り当てられている入出力ファイルの番号です。
- *element path*：要素パスを表す文字列です。
- *attribute*：要素内の属性を表す文字列です。

戻り値：関数の処理が成功した場合、0 が返ります。処理が失敗した場合は、負数のエラーコードが返ります。

XML ドキュメント内のデータ要素と階層構造を識別するには



XML データソースにアクセスするために、**XML ビュー**を使用している場合、他のデータソースのカラムと同じように要素にアクセスすることができます。階層構造は、**データ**リポジトリで簡単に参照することができます。「最初から XML ドキュメントを作成するには」（295 ページ）を参照してください。

XML 関数を使用する場合は、XML ファイル内の要素パスを指定する必要があります。

これらの関数の詳細情報は、「XML ドキュメント内のデータの取得、更新、挿入を行うには」（320 ページ）を参照してください。例として **XMLGet()** 関数を使用してみます

XMLGet(*generation*, *file*, *element path*, *attribute name*)

- *generation*：タスクの世代番号です。0 が現在のタスク、1 は親タスクになります。
- *file*：XML ドキュメントが割り当てられている入出力ファイルの番号です。
- *element path* ：要素パスを表す文字列です。
- *attribute name*：要素内の属性を表す文字列です。

パス指定された各要素は、前のスラッシュによって分離されたその名前とインデックス（複数存在する場合）によって識別されます。最後のパラメータは（もしあれば）属性のために使用されます。

例として以下の XML から取得してみます。

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<Productions lastUpdate="2006-01-01" version="1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
 <StudioList>
  <Studio code="S004">
   <name>Paramount</name>
  </Studio>
  <Studio code="S005">
   <name>Warner Home Video</name>
  </Studio>
  <Studio code="S006 ">
   <name>New Line Home Entertainment</name>
```

例えば、要素名 Studio code の３番目の値を取得する場合以下のような式を指定します。

```
XMLGet (0,1, '/Productions/StudioList/Studio[3]','code')
```

この場合、S006 が返ります。

要素を取得する場合は、以下の式を指定します。

```
XMLGet (0,1, '/Productions/StudioList/Studio[3]/name', '')
```

この場合、New Line Home Entertainment が返ります。

通常は、インデックスはハードコーディングされず、項目や **LoopCounter()** 関数などを使用することになります。

参照： 「XML ドキュメント内のデータの取得、更新、挿入を行うには」（320 ページ）

XML ドキュメント内の混在内容を扱うには

混在内容の要素には、要素とテキストの両方が含まれています。例えば以下のような XML になります。

```
<?xml version="1.0" encoding="UTF-8"?>
<notice>
    To
    <name>Fred Flicker</name>
    Your DVDs are are overdue. They were due on
    <duedate>07/21/2003</duedate>
    Please bring them in ASAP.
    We really appreciate it. Thanks!
    <greeting>Sincerely,</greeting>
    <outlet>AAA Rentals</outlet>
</notice>
```

この例では、4 つのテキスト項目（通常表示）と 4 つの要素項目（**ボールド表示**）が定義されています。

最初に、複合ルートを持つ XML ドキュメントを受け取った場合。例えば、**XML ビュー**を使用する場合に、混在内容を処理するには、以下の 2 つの関数を使用します。

- **DbXmlMixedGet()**
- **DbXmlMixedSet()**

XML ビューの存在を確認するには

XML ビューを使用して XML ファイルの処理を行う場合、XML ビューにレコードが存在しているかどうかをあらかじめ確認することができれば、不要な処理を行う必要が無くなります。このような場合、**XMLExist()** 関数を使用することで簡単に確認することができます。

この例では、4 つのテキスト項目（通常表示）と 4 つの要素項目（**ボールド表示**）が定義されています。

最初に、複合ルートを持つ XML ドキュメントを受け取った場合。例えば、**XML ビュー**を使用する場合に、混在内容を処理するには、以下の 2 つの関数を使用します。

- **DbXmlMixedGet()**
- **DbXmlMixedSet()**

第 15 章 : COM

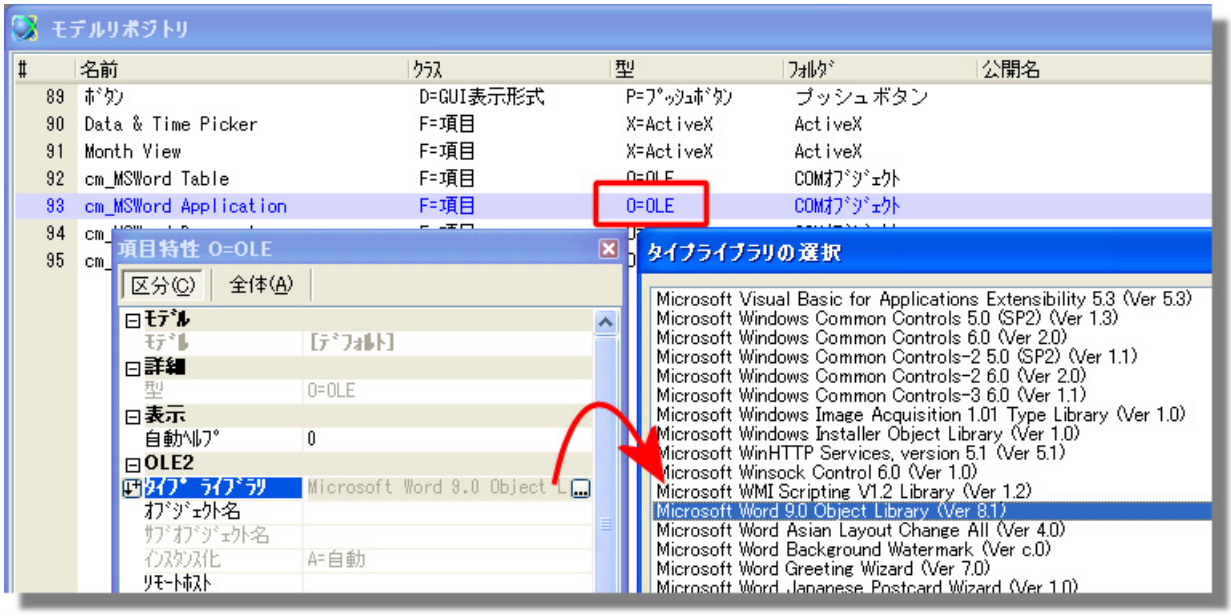
使用する COM オブジェクトを定義するには

COM オブジェクトを使用するには、あらかじめオブジェクトを定義しておく必要があります。COM オブジェクトは、他の項目と同じようにタスクのデータビューエディタで定義することができます。

COM オブジェクトの定義は、日付型や文字型の項目よりは多少設定する作業が多くなります。ライブラリをリストから選択する必要があり、また各ライブラリからオブジェクトを選択する必要があります。COM オブジェクトは、Word や Excel などの他の製品とリンクするために使用することが多いため、それらの製品がバージョンアップされた場合 COM 定義を修正する必要が発生する可能性があります。

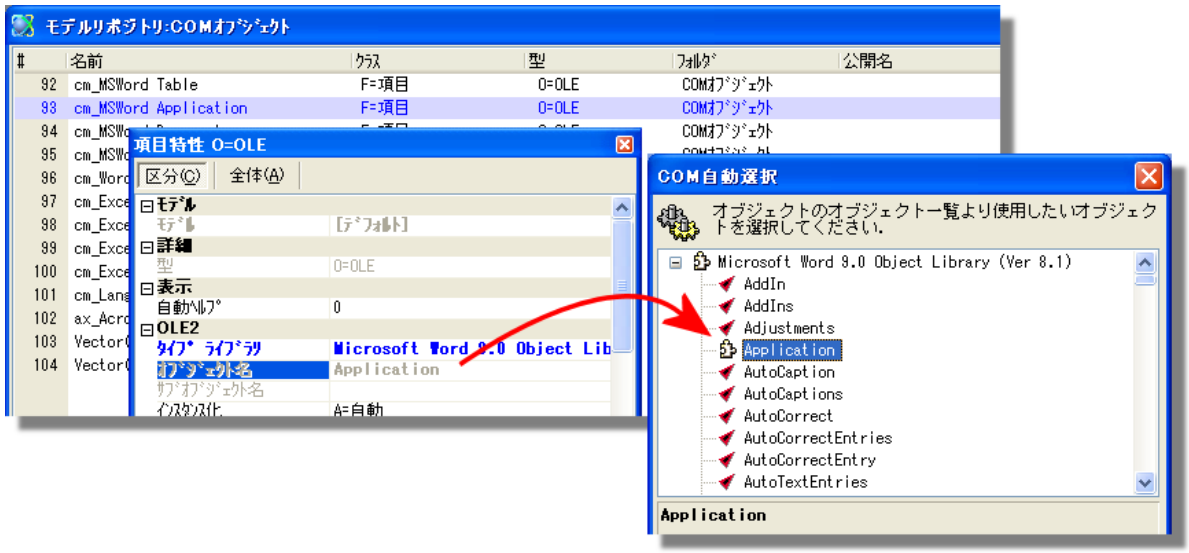
このような理由のために、モデルに COM を定義し、必要に応じてそれらを再利用することを推奨します。詳細は、「COM オブジェクト定義を再利用するには」（346 ページ）を参照してください。

COM オブジェクトを定義する



1. モデルリポジトリで一行追加（*F4* または、編集→行作成）します。
2. 名前カラムに任意の名前を入力してください。
3. クラスカラムはデフォルト（*F= 項目*）のままにします。
4. 型カラムは *O=OLE*（フォームに表示させる場合は、*X=ActiveX*）に設定します。
5. タイプライブラリ特性からズームして、使用したいライブラリを選択します。PC にインストールされているすべてのライブラリが表示されるため、非常に大きなリストになる場合があります。選択したいセクション名の先頭の文字を入力することで位置付けを行うことができます。*Enter* を押下して使用したいライブラリを選択します。

6. **オブジェク名**特性から**ズーム**して、このライブラリから使用したいオブジェクトを選択します。



7. 必要であれば、同じ方法で**サブオブジェクト名**を選択します。

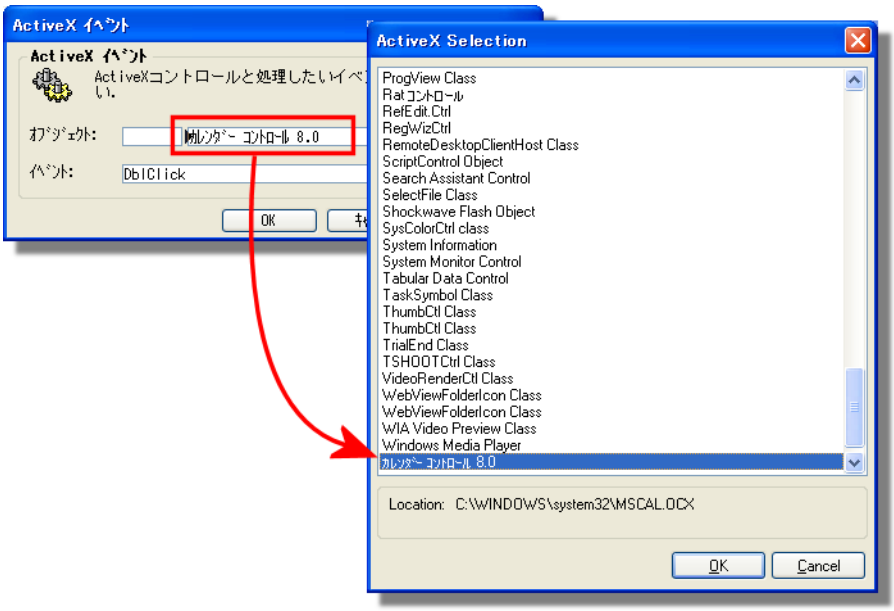
これで、COM オブジェクトを宣言するためにこのモデルを使用することができます。この場合、項目を定義する際にこのモデルを設定するだけで済みます。

注： COM ライブラリのリストを見て、何を選択していいのかが分からなくなる場合があります。COM ライブラリは、基本的には膨大な関数リストのようなものです。ライブラリに関するドキュメント（例：Word の開発者用ドキュメント）を参照しない限りどの関数が必要なのかは、実際分かりません。しかし Magic は、オブジェクトを簡単に試してみることで、オブジェクトの内容を確認することができます。

同じタイプの複数の COM オブジェクトに対して 1 つのイベントハンドラを設定するには

同じタイプの複数の COM オブジェクトを持っている場合、それらの COM オブジェクトのどれにでも対応する 1 つの ActiveX イベントのハンドラ（ロジックユニット）を設定することができます。この例では、カレンダーコントロール Ver8.0 でダブルクリックした場合に実行するロジックユニットを定義しています。

クラスによるハンドラを設定する

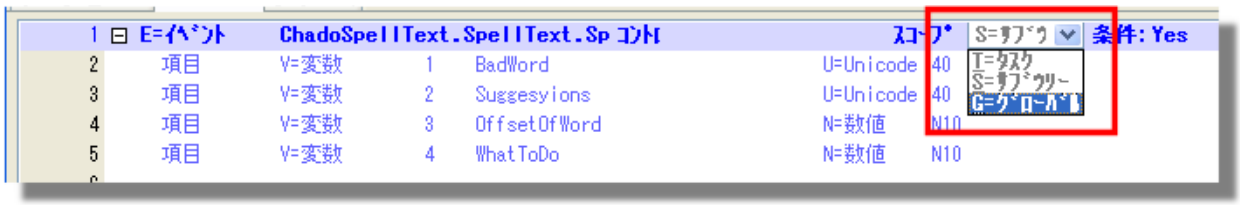


1. *Ctrl+H* を押下してヘッダ行を作成します。
2. E を入力し*イベント*を選択します。イベントダイアログが表示されます。
3. イベントタイプで *X=ActiveX* を選択します。*Tab* で次に移動します。ActiveX イベントダイアログが表示されます。
4. 通常は、ActiveX 項目を選択するために、オブジェクトでズームしますが。今回は、図のようにそのとなりの欄でズームします。これによって、ActiveX 項目を使用しないで直接 COM オブジェクトを選択することができます。
5. 次にイベントからズームします。処理させたい ActiveX イベントを選択します。この例では、DblClick（ダブルクリック）が選択されています。

下位タスクが実行中に、COM オブジェクトに対するイベントを処理できるようにするには

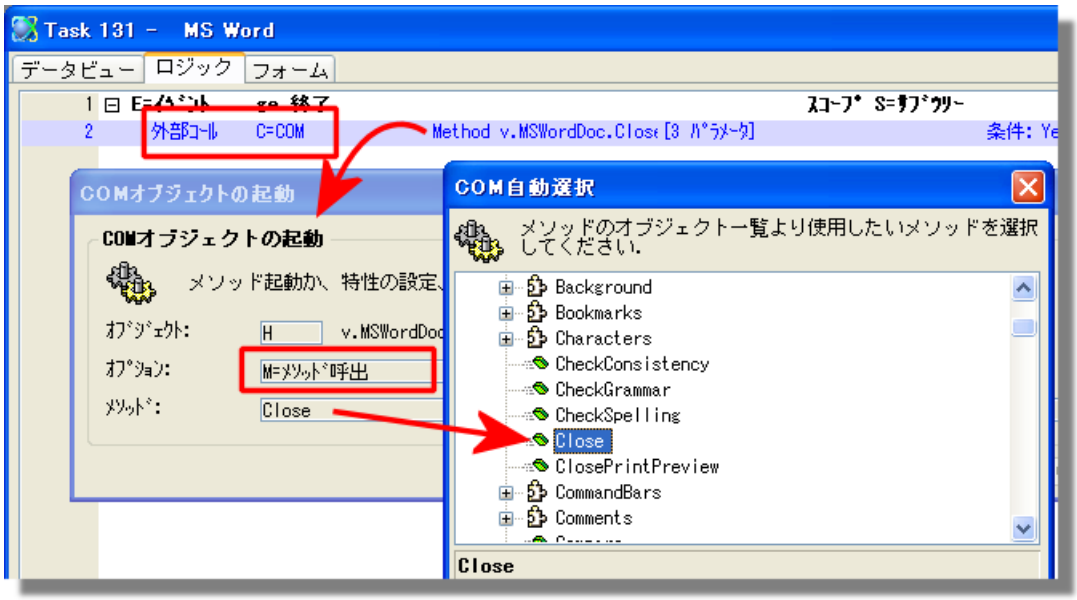
同じような処理を実行する複数の COM オブジェクトを持っている場合、親タスクで動作するイベントハンドラを作成することで便利に扱うことができます。例えば、1 つのタイプのすべての COM オブジェクトのエラーを処理するために、メインプログラム内でグローバルに動作するエラーハンドラを作成することでプロジェクト内のすべてのプログラムで利用することができます。

これには以下のようにします。



1. 特定のタイプのどのような COM オブジェクトに対しても反応するイベントを作成します。この例では、サードパーティーのスペルチックツールを使用してスペルエラーに対するグローバルなイベントロジックユニットを定義しています。このようなロジックユニットの定義方法については、第 15 章：「同じタイプの複数の COM オブジェクトに対して 1 つのイベントハンドラを設定するには」（327 ページ）を参照してください。
2. スコープ特性を適切に設定します。例えば、メインプログラムで定義されたイベントロジックユニットをどのプログラムからでも有効にするには、*G= グローバル*に設定します。親タスクに定義し、サブタスクでも反応させたい場合は、*S= サブツリー*に設定します。

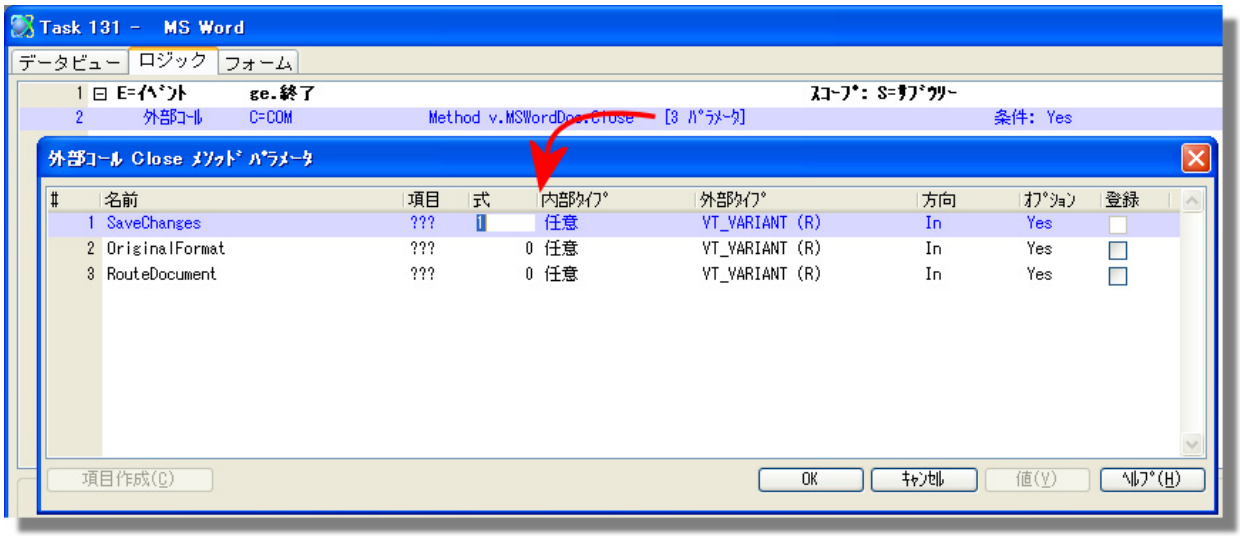
COM オブジェクトのメソッドを呼び出すには



COM オブジェクトが宣言されると、プログラムやサブタスクを起動するように、オブジェクトのメソッドを呼び出すことができます。ただし、**外部コール**という異なる処理コマンドが使用されます。この処理コマンドは Magic 以外のプログラムを呼び出す場合に使用されます。

COM メソッドを呼び出す

1. **ロジックユニット**内で 1 行追加します。
2. ***I*** を入力するかドロップダウンリストから選択して、**外部コール**処理コマンドを設定します。
3. ***C*** を入力するかドロップダウンリストから選択して、***COM*** を起動タイプとして設定します。***Tab*** 移動します。
4. **ズーム**して **COM オブジェクト**ダイアログを表示します。
5. **オブジェクト**から**ズーム**して COM オブジェクトを選択します。
6. **オプション**でドロップダウンリストから ***M= メソッドの呼び出し***を選択します。
7. **メソッド**から**ズーム**して、起動するメソッドを選択します。
8. ***Esc*** を押下して **COM オブジェクト**ダイアログを閉じます。**パラメータ**カラムに ***Tab*** 移動し、**ズーム**します。



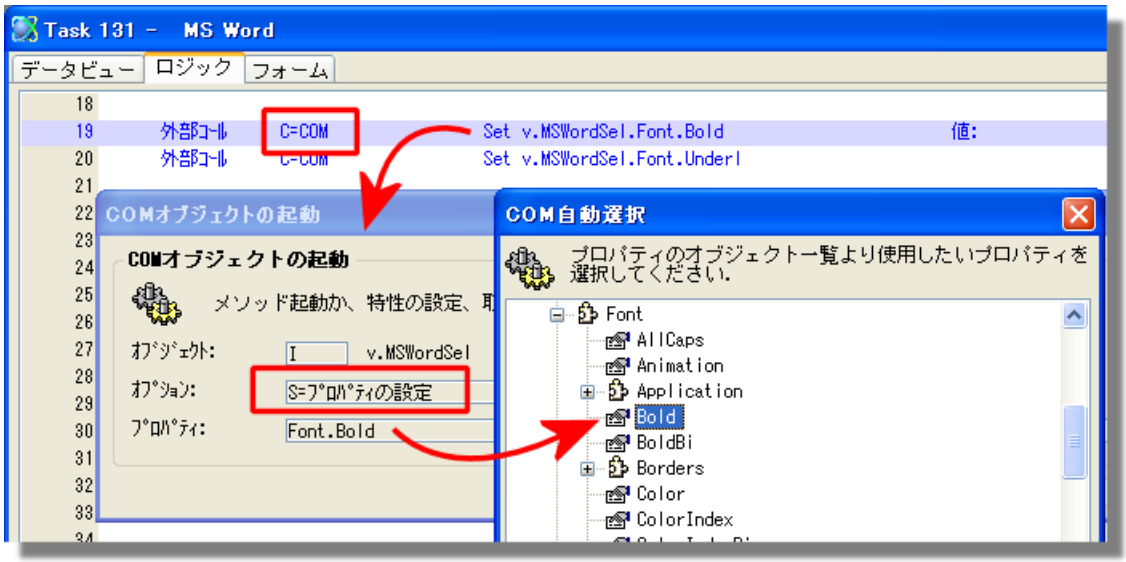
9. このメソッドに渡す**パラメーター一覧**が表示されます。一覧上に表示されるパラメータの情報（入力用か出力用か、オプションかどうか、データタイプは何なのか）を確認してください。
   登録カラム内のボックスを**クリック**すると、パラメータのデータ型に対応した項目が作成されます。Magic によってデータ変換が行われるため、他のプログラミングツールを使用する場合ほどはデータ型について正確に知る必要はありません。例えば、この例では VT_VARIANT タイプ（ポインタ内に渡す必要があることを意味しています）を使用します。値を式で指定（この場合、項目カラムは 0 になります）するだけで、あとは Magic が処理します。

数値を渡す場合も同様に、その数値が float 型か long 型か、あるいは packed integer かどうかということを意識する必要はありません。

これで、COM オブジェクト内のメソッドを呼び出す処理が作成されました。

ヒント :エ数を減らすために、コール COM 処理コマンドをコピーすることができます。同じメソッドにを繰り返し呼び出すような場合（例えば、COM オブジェクトを使用して Word 文書をフォーマットしていたり、段落を追加したりする必要がある場合）、このような操作は便利です。既存の処理コマンドをコピーするには、Ctrl+Shift+R か Ctrl+C を使用し、貼り付ける場合は Ctrl+V を使用します。

COM オブジェクトのプロパティを設定 / 取得するには



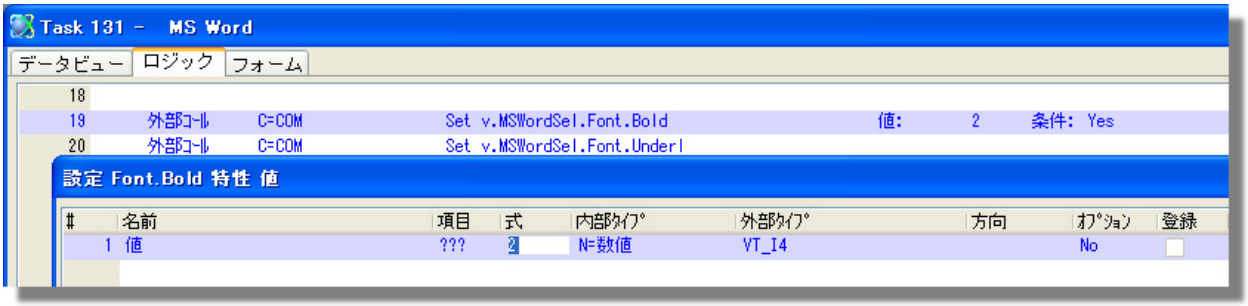
COM オブジェクトが宣言されたら、***プロパティの設定***オプションを使用して特性値を変更したり、***プロパティの取得***オプションを使用して特性値を取得することができます。

プロパティの設定オプションは、オブジェクトを変更するために使用されます。例えば、ActiveX オブジェクトに対して、オブジェクトの色やどのように表示されるかを変更するためにめに使用できます。この例（Word 文書の作成）では、現在のフォントスタイルをボールドに変更するために***プロパティの設定***オプションが使用され、これらの特性値を取得するために***プロパティの取得***オプションが使用されます。

プロパティの設定 / 取得オプションは、オブジェクトの値を変更したり取り出す場合にも使用されます。例えば、カレンダーオブジェクト内で、デフォルト日付を設定するために設定オプションが使用され、ユーザが選択した日付を取り出すために取得オプションが使用されます。

COM オブジェクトでプロパティの設定 / 取得オプションを使用する

1. ロジックユニット内で 1 行追加します。
2. *I* を入力するかドロップダウンリストから選択して、外部コール処理コマンドを設定します。
3. *C* を入力するかドロップダウンリストから選択して、*COM* を起動タイプとして設定します。*Tab* 移動します。
4. ズームして **COM オブジェクト**ダイアログを表示します。
5. オブジェクトからズームして COM オブジェクトを選択します。
6. オプションでドロップダウンリストから ***S= プロパティの設定***（***G= プロパティの取得***）を選択します。
7. プロパティからズームして、アクセスするプロパティを選択します。
8. *Esc* キーを押下して **COM オブジェクト**ダイアログを閉じます。パラメータカラムに *Tab* 移動し、ズームします。



9. このプロパティに設定 / 取得するパラメーター一覧が表示されます。一覧上に表示されるパラメータの情報（オプションかどうか、データタイプは何なのか）を確認してください。
   登録カラム内のボックスをクリックすると、パラメータのデータ型に対応した項目が作成されます。Magic によってデータ変換が行なわれるため、他のプログラミングツールを使用する場合ほどはデータ型について正確に知る必要はありません。例えば、この例では VT_I4 タイプ（signed long integer）を使用します。値を式で指定（この場合、項目カラムは 0 になります）するだけで、あとは Magic が処理します。

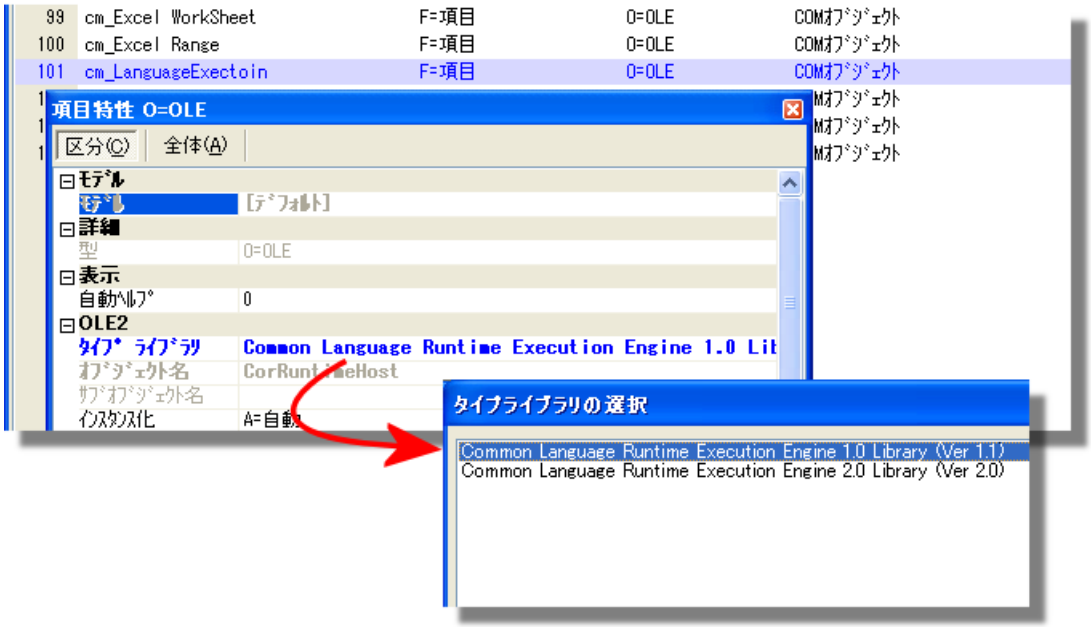
これで、COM オブジェクトのプロパティの設定 / 取得処理が作成されました。

ヒント :工数を減らすために、コール COM 処理コマンドをコピーすることができます。同じメソッドにを繰り返し呼び出すような場合（例えば、COM オブジェクトを使用して Word 文書をフォーマットしていたり、段落を追加したりする必要がある場合）、このような操作は便利です。既存の処理コマンドをコピーするには、Ctrl+Shift+R か Ctrl+C を使用し、貼り付ける場合は Ctrl+V を使用します。

COM オブジェクトの参照先を変更するには

COM 規格の非常に良いところは、それらが、前方互換で設計されていることです。すなわち、COM オブジェクトが規格に準拠している場合、COM オブジェクトの新バージョンをインストールしても、今まで定義していたメソッドがそのまま使用できるということです。規格はこれに関して様々な詳細をカバーし、独自の COM にオブジェクトを作成するときも、この規格に準拠する必要があります。

しかし、ライブラリ名は変更されます。例えば、毎年 PC 上の Microsoft Word をアップグレードした場合、Microsoft の COM ライブラリの全てのバージョンが登録されることになります。COM オブジェクトが古いバージョンを選択されている場合、新バージョンをインストールしてもこの選択内容は自動的に更新されません。この場合、古い COM オブジェクトがインストールされていない別の PC に新しいオブジェクトをインストールしても、オブジェクトへの呼び出しは失敗します。



例えば、ここに、Ver1.0 の COM オブジェクトがあります。Ver.2.0 もこの PC に存在しているとします。**タイプライブラリ**特性から**ズーム**すると、これらの 2 つのライブラリが同じオブジェクトと認識され、新しいライブラリが選択できる状態になっています。

このプロジェクトを古いライブラリが含まれていない PC 上で実行した場合、古いライブラリ名が示され、**ズーム**して新しいライブラリを選択できるようになります。

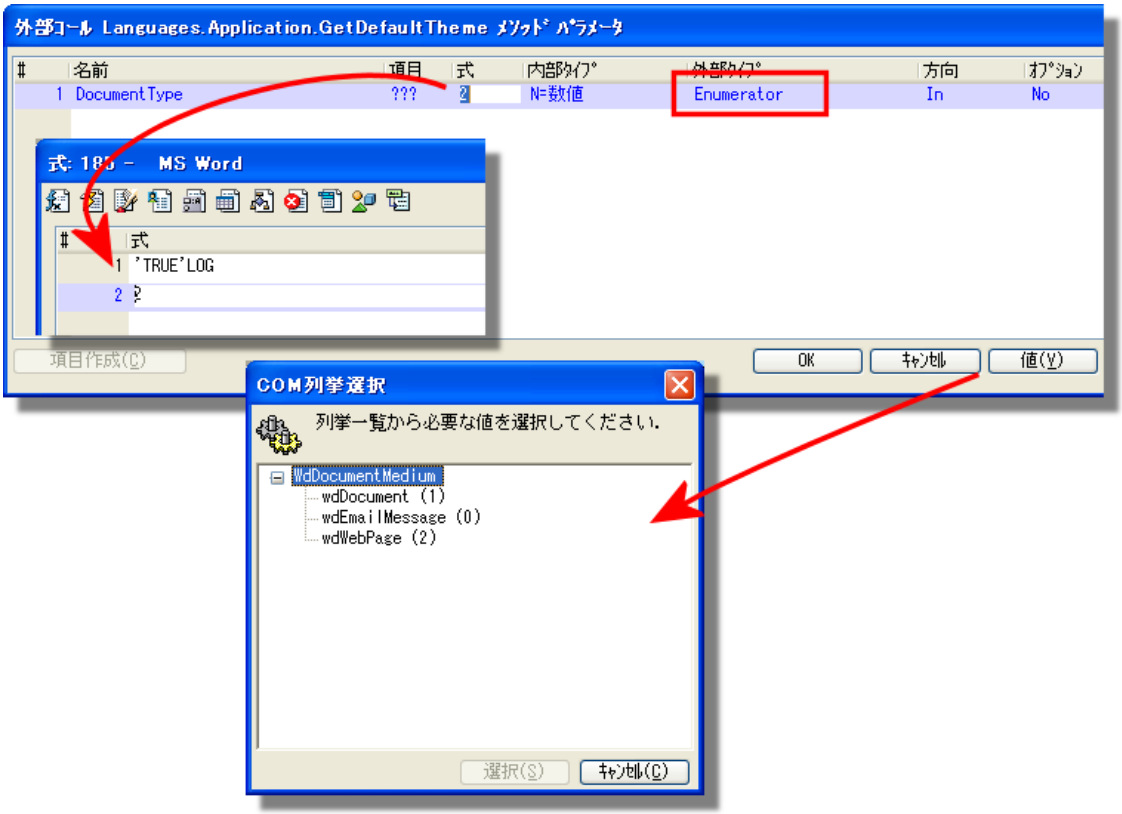
Magic では、オブジェクト名は変更できず、このオブジェクト内の（同じライブラリの別バージョン以外の）別のライブラリにも変更できません。これによって、不用意にオブジェクトが変更されることで、参照処理が失敗することを防止しています。

COM ライブラリの参照先を変更する

1. タスクの**データビューエディタ**や**モデル**リポジトリ内の COM オブジェクトが定義されている項目に移動します。
2. **特性シート**（*Alt+F1*）を開き、**タイプライブラリ**特性に移動します。
3. **ズーム**（*F5* または、**ダブルクリック**）して**タイプライブラリ選択**ボックスを開きます。このオブジェクトに対する複数のレビジョンが存在するかどうかによって、1 つだけ表示されたり複数表示されたりします。
4. *Enter* を押下して、アップグレードされたオブジェクトを選択します。

これで、このオブジェクトの呼び出し処理は、アップグレードされたライブラリを使用することになります。

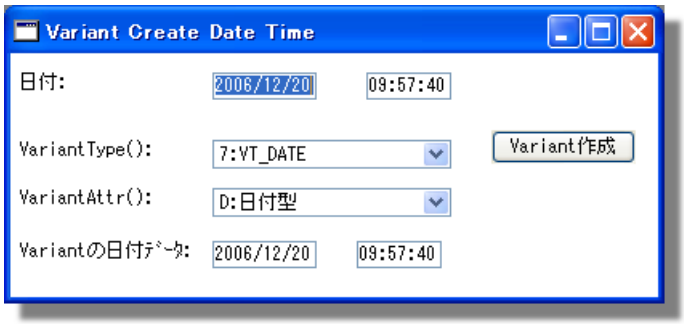
列挙型パラメータ値を設定するには



列挙型は、定数の固定されたセットから構成されるデータタイプです。例えば、週における日数や１年における月、または段落整列の形式のようなものです。しばしば、COM オブジェクト内で、複数のオブジェクト間で選択肢を持っている場合、選択をするために、整数を渡します。

どの整数がどの選択肢を表すかは、COM オブジェクトに対してドキュメントを調べるために必要な何かであり、実際に試してみることで何が起こるかを確認することができます。

Variant 型に対してデータの設定 / 取得を行うには



多くの COM オブジェクトは、VT_VARIANT データ型を使用しています。VT_VARIANT を定義することで、本質的には、どのようなデータ型でも保持できます。しかし、使用しているオブジェクトは特定の機能のために一定のフォーマットのデータを期待しているはずです。

通常、Magic は自動的に変換処理を実行します。例えば、特定の VT_VARIANT パラメータが配列を期待している予期していて、ベクトルを設定した場合、ただベクトルを渡し、Magic はそれを処理します（設定例は、「COM オブジェクトから配列値を取り出したり設定したりするには」（340 ページ）を参照してください）。

同様に、オブジェクトが VT_VARIANT で負の整数を返し、負数を有効にした数値型項目で受け取った場合、Magic は正しく変換処理を行います。

しかし、管理水準をより高めたい場合は、、受け渡しするデータとして BLOB 型を使用し、Variant 関数を使用することで、送信する Variant タイプを正確に設定することができます。

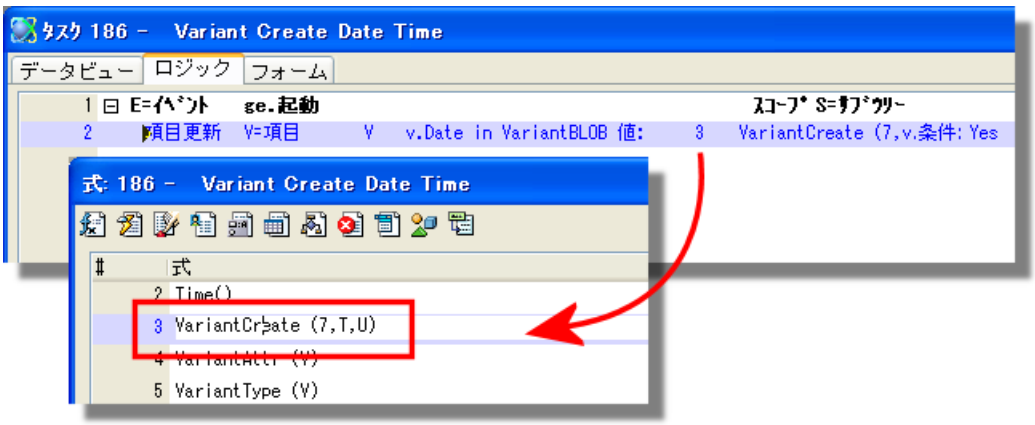
オブジェクトから返される Variant からデータを取得するために同じ関数を使用することもできます。

以下に挙げるのが、使用できる関数です。

- **VariantCreate()**：BLOB Variant にデータをコピーします。「Variant を作成する」（335 ページ）を参照してください。
- **VariantGet()**：BLOB Variant から項目にデータをコピーします。「Variant からデータを取得する」（336 ページ）を参照してください。
- **VariantGetVector()**：BLOB Variant からベクトル項目にデータをコピーします。『リファレンスヘルプ』を参照してください。
- **VariantAttr()**：Variant の Magic データ型を取得します。「VariantAttr() 関数を使用してデータ型を取得する」（338 ページ）を参照してください。
- **VariantType()**：Variant のデータ型（を表す数値）を取得します。「VariantType() 関数を使用してデータタイプを取得する」（339 ページ）を参照してください。

Variant を作成する

VariantCreate() 関数を使用することで Variant データを実装させることができます。この例では、日付と時間を VT_DATE 型の Variant に移動するために **VariantCreate()** 関数を使用しています。これは実際に日付 / 時刻を保持することができます。



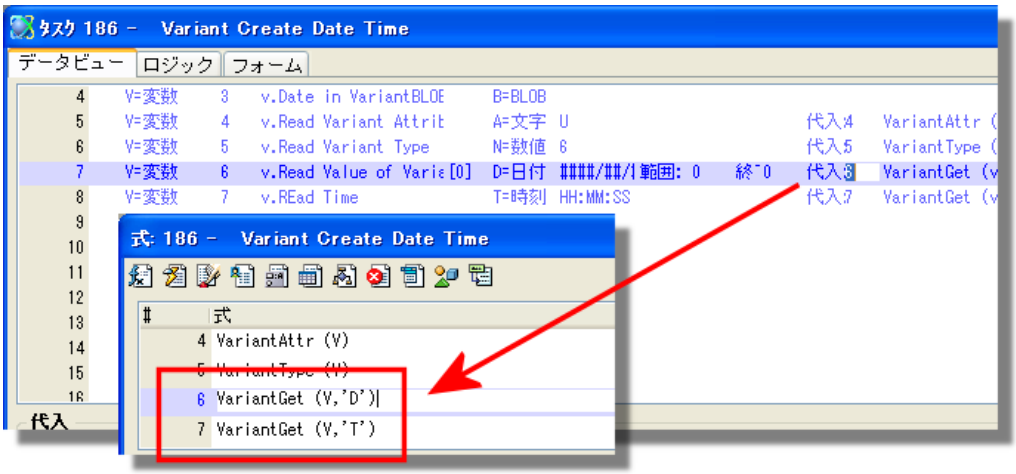
VariantCreate() の構文は以下の通りです。

VariantCreate(*VT Type*, *Value*, *Time value*)

パラメータ：

- *VT Type*：データタイプを表す数値です。例えば、この例では 7 が指定されています。データタイプの一覧は、「Variant のデータタイプ」（337 ページ）を参照してください。
- *Value*：Variant に設定する値です。どのようなデータも指定できます。
- *Time value*：（オプション）VT_Date 型に時刻を設定することが可能になります。この例では、３番目のパラメータとして時刻が設定されています。

Variant からデータを取得する



VariantGet() 関数の構文は以下の通りです。

VariantGet(*Variant Value*, *Attribute*)

パラメータ：

- *Variant Value*：Variant を表す BLOB 項目です。
- *Attribute*：取得するデータの型を表す文字です。コードの一覧は、「Variant の Magic データ型」（336 ページ）を参照してください。

この例では、Variant は日付と時刻を保存する VT_DATE 型で、**VariantGet()** 関数を使用して両方のデータを取得できます。

Variant の Magic データ型

型を表す文字	Magic のデータ型
A	文字型
N	数値型
L	論理型
D	日付型
T	時刻型
B	BLOB 型
U	Unicode 型

Variant のデータタイプ

Variant 関数で使用されるデータタイプには以下のものがあります。

タイプ値	列挙型のシンボル	内容	
0	VT_EMPTY	値が設定されていない	
1	VT_NULL	SQL スタイルの Null	
2	VT_I2	2 バイト整数 （符号付）	–32,768 to 32,767
3	VT_I4	4 バイト整数 （符号付）	–2,147,483,648 to 2,147,483,647
4	VT_R4	4 バイト不動小数点数	1.1E -38 to 3.4E +38 （7 桁）
5	VT_R8	8 バイト不動小数点数	2.2E -308 to 1.7 E +308（15 桁）
6	VT_CY	通貨型	
7	VT_DATE	日付型	
8	VT_BSTR	サイズ情報をもった文字列型	
9	VT_DISPATCH	IDispatch インターフェイス	
10	VT_ERROR	エラー型	
11	VT_BOOL	ブール型	
12	VT_VARIANT	バリアント型 （バリアント型配列にのみ使用）	
13	VT_UNKNOWN	IUnknown インターフェイス	
14	VT_DECIMAL	12 バイト数値（符号付）	
16	VT_I1	1 バイト整数（符号付）	–128 to 127
17	VT_UI1	1 バイト整数（符号なし）	0 to 255
18	VT_UI2	2 バイト整数（符号なし）	0 to 65,535
19	VT_UI4	4 バイト整数（符号なし）	0 to 4,294,967,295
22	VT_INT	int 型	
23	VT_UINT	unsigned int 型	
36	VT_RECORD	ユーザ定義型	
8192	VT_ARRAY		データ型の配列
16384	VT_BYREF		データ型への参照

COM オブジェクトの Variant 値のタイプと対応する Magic データ型を決定するには

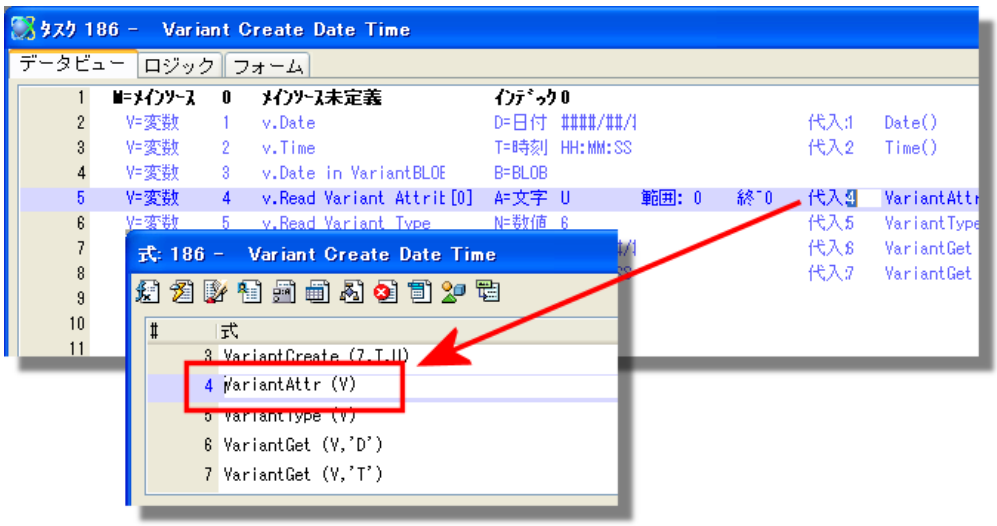
多くの COM オブジェクトは、VT_VARIANT タイプを使用しています。VT_VARIANT を定義することで、本質的には、どのようなデータタイプも保持できます。従って、このタイプのパラメータが設定されている場合、どのようなデータタイプが渡されるかを知る必要はありません。

実際は、試してみたり、オブジェクトに関するドキュメントを参照することで、どのようなオブジェクトが期待され送られるかを知ることができます。Magic はデータの変換処理を行うため、手動で変換処理を行う必要はありません。例えば、数値型項目内に VT_VARIANT タイプを受け取るように設定しオブジェクトが数値を返した場合、その通りに処理は実行されます。

しかし、1 つのオブジェクトがある呼び出しに対して Variant 内に数値を返し、別の呼び出し対しては文字型データを返す場合があるかもしれません。このような場合は、BLOB データとしてデータを受け取り、内容を確認する必要があります。

- **VariantAttr()** 関数を使用することでこのような処理を行うことができます。この関数はパラメータとして BLOB 型を指定でき、Variant のデータタイプを表す文字が返ります。そのコードを **VariantGet()** 関数で使用することによりデータを取得することができます。**VariantType()** 関数を使用することもできます。この場合、Variant のデータタイプ（VT_I8 など）が返ります。

VariantAttr() 関数を使用してデータ型を取得する



VariantAttr() 関数の構文は以下の通りです。

VariantAttr(*Variant*)

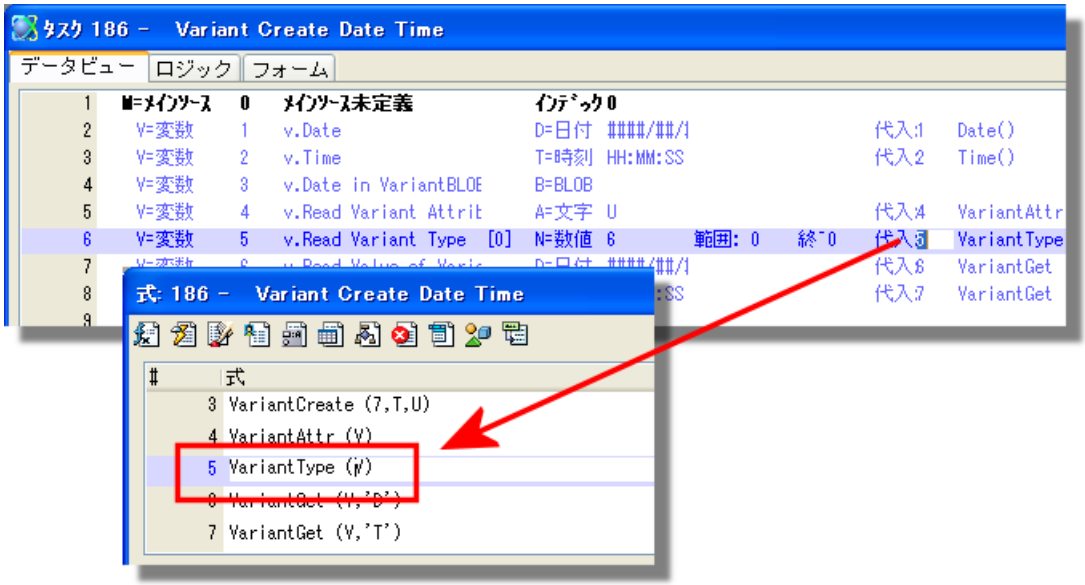
パラメータ：

- *Variant*：Variant が格納された BLOB 項目です。

この関数は、Magic のデータ型を表す文字を返します。これらのデータ型の一覧は、「Variant の Magic データ型」（336 ページ）を参照してください。

この例では、VT_DATE タイプが指定されているため、D が返ります。これは Magic の日付型を表しています。これでデータ型はわかりました。**VariantGet()** 関数を使用することで項目内の Variant からデータを取り出すことができます。

VariantType() 関数を使用してデータタイプを取得する



VariantType() 関数の構文は以下の通りです。

VariantType(*Variant*)

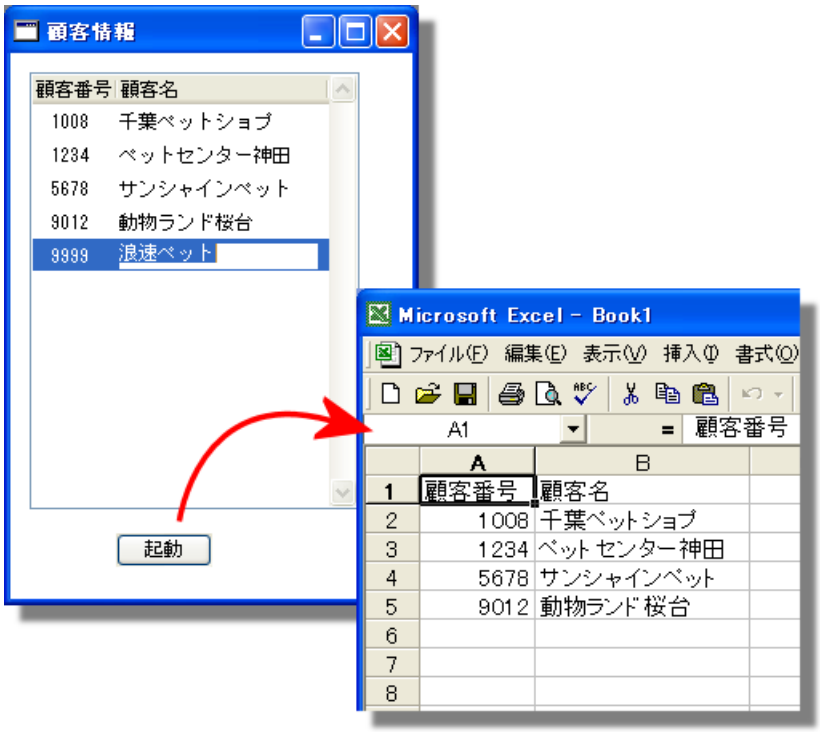
パラメータ：

- *Variant*：Variant が格納された BLOB 項目です。

この関数は、Variant のデータタイプを表す文字を返します。これらのタイプの一覧は、「Variant の Magic データ型」（336 ページ）を参照してください。

この例では、Variant は VT_DATE タイプのため、7 が返ります。

COM オブジェクトから配列値を取り出したり設定したりするには



配列と COM オブジェクトを使用した処理を実行することで、Magic と COM オブジェクトの本当の力を知ることができます。上記の例では、わずかな記述を行うだけでテーブル内のデータが Excle のシートにコピーする処理を追加することができます。

COM オブジェクトの間で他のパラメータと同じようにベクトルデータを受け渡しします。COM オブジェクトに渡すデータがが数値か文字列かを知る必要がありますが詳細内容（数値の場合、float か long かなど）を意識する必要はありません。

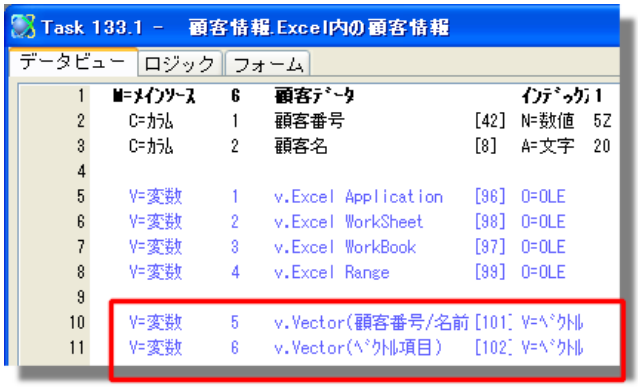
また、ベクトル内にいくつかの項目を定義する必要があります。この例では、何人のユーザがテーブルの中にいたかに見逃さないように、**Counter(*0*)** 関数を使用します。これによって、Excel シート内の正しい数を割り当てることができます。

COM オブジェクトに配列を渡す

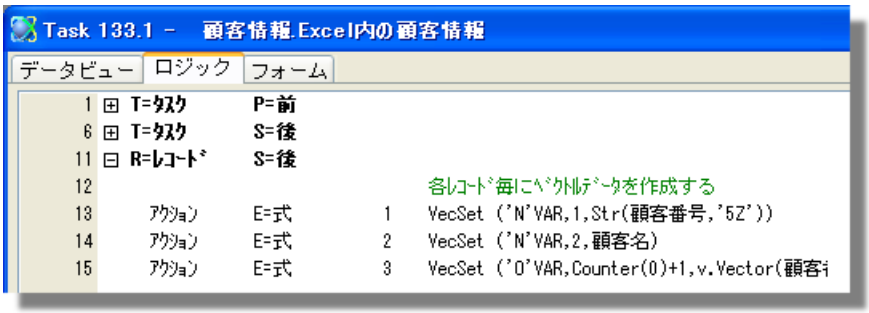
1. 最初に、ベクトル項目を設定します。この例では、２次元の配列を使用しています。各ラインにはユーザ番号とユーザ名を含んだ 1 次元配列を表しています。

   最初のベクトルは文字列の配列です。それがいっぱいの場合、2 つの要素（ユーザ番号とユーザ名）を含んでいます。これは、表計算の 1 行に相当します。

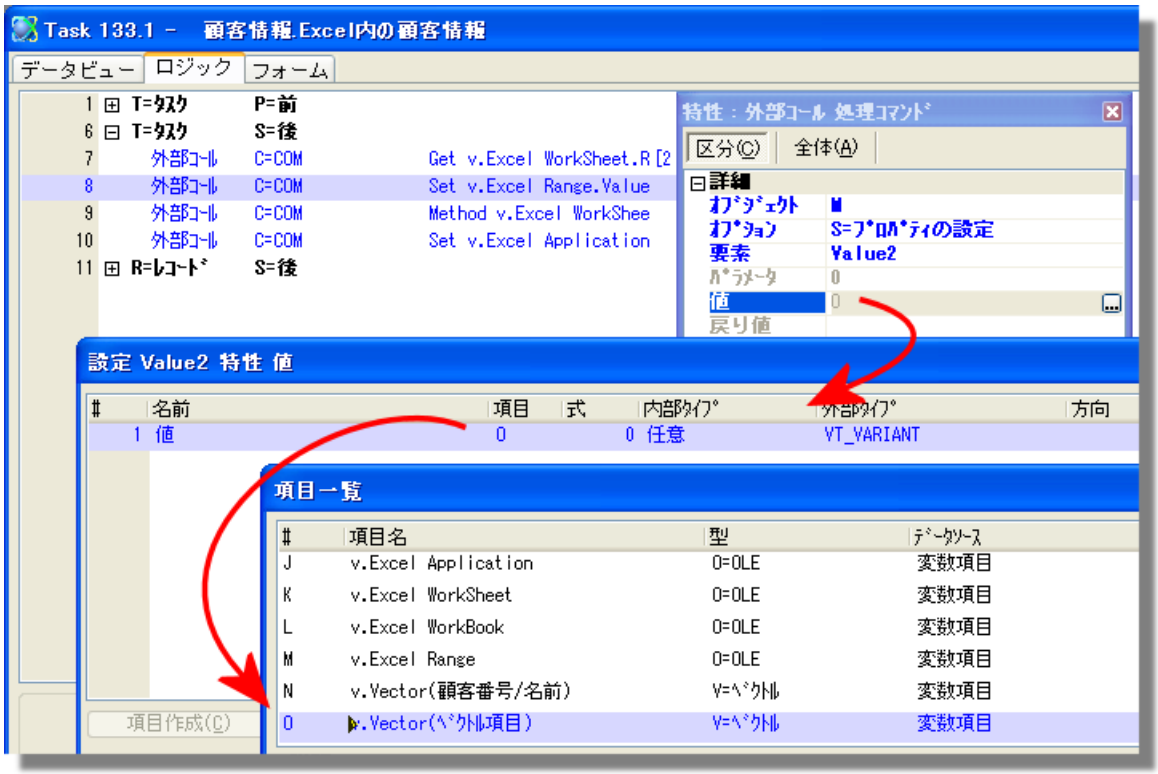
   2 番目のベクトルは、ベクトルで各行は、最初のベクトルのモデルを使用しています。これは表計算のシート全体に相当します。



.



2. 次に、COM オブジェクト内にベクトルを渡すために、**VecSet()** 関数を使用して、ベクトル内にデータを設定します。



3. ベクトルにデータが設定されると、簡単に渡すことができます。他のデータ項目と同じようにベクトルは単に渡されるだけです。

COM オブジェクト内のコレクションを処理するには

コレクションオブジェクトは、最近使用されたファイルや郵便番号または、この例のような「言語」といった項目のリストです。

リストにはユニークな識別子を含めなければなりません。識別子には、以下のものがあります。

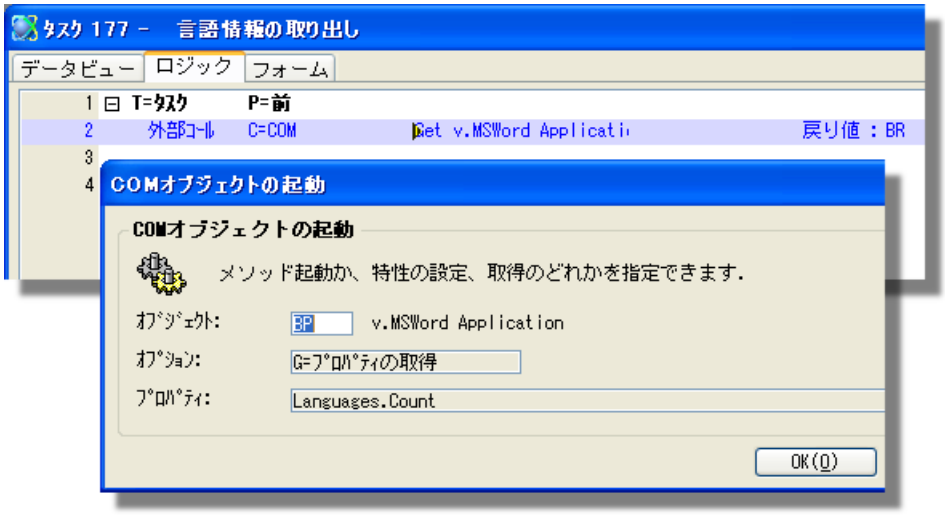
- インデックス……1 からリスト内の項目数までの連続番号
- キー……コレクション内の項目を識別するためのユニークな文字列（または数字）

各コレクションはインデックスかキーのどちらかを含みます。両方を含めることはありません。上記の例では、Microsoft Word からの言語リストには ID のキータイプが含まれています。ID が何であるかは分からないので、リスト全体を抽出することは難しくなります。

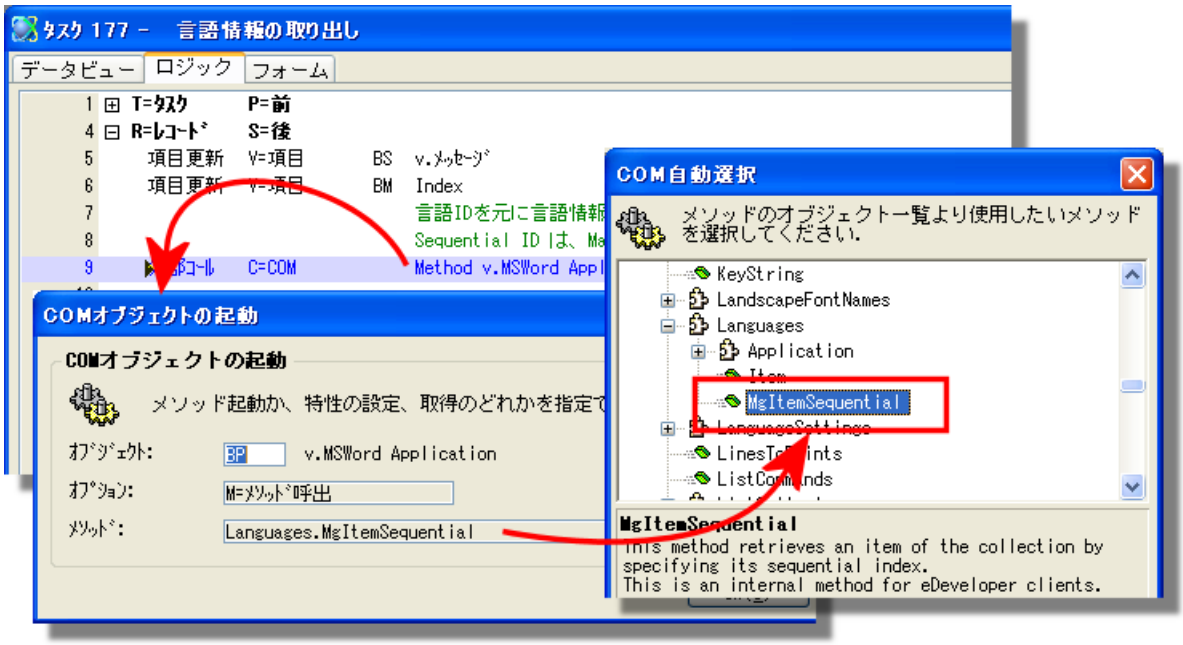
しかし、**MgItemSequential** と呼ばれるコレクションのために特別なメソッドを使用することで、この問題は解決されます。このメソッドはネイティブのオブジェクトの一部ではありませんが、他のすべてのメソッソと一緒にメソッド一覧に表示されます。それは、1 から項目数分のインデックスを使用して、リストから項目を取り出すことが」できます。項目がテーブル内にある場合、上記のように Magic での他のテーブルと同じように使用することができます。

以下の例は、Microsoft Word からサポートされた言語リストを取り出すものです。

コレクションを取り出す

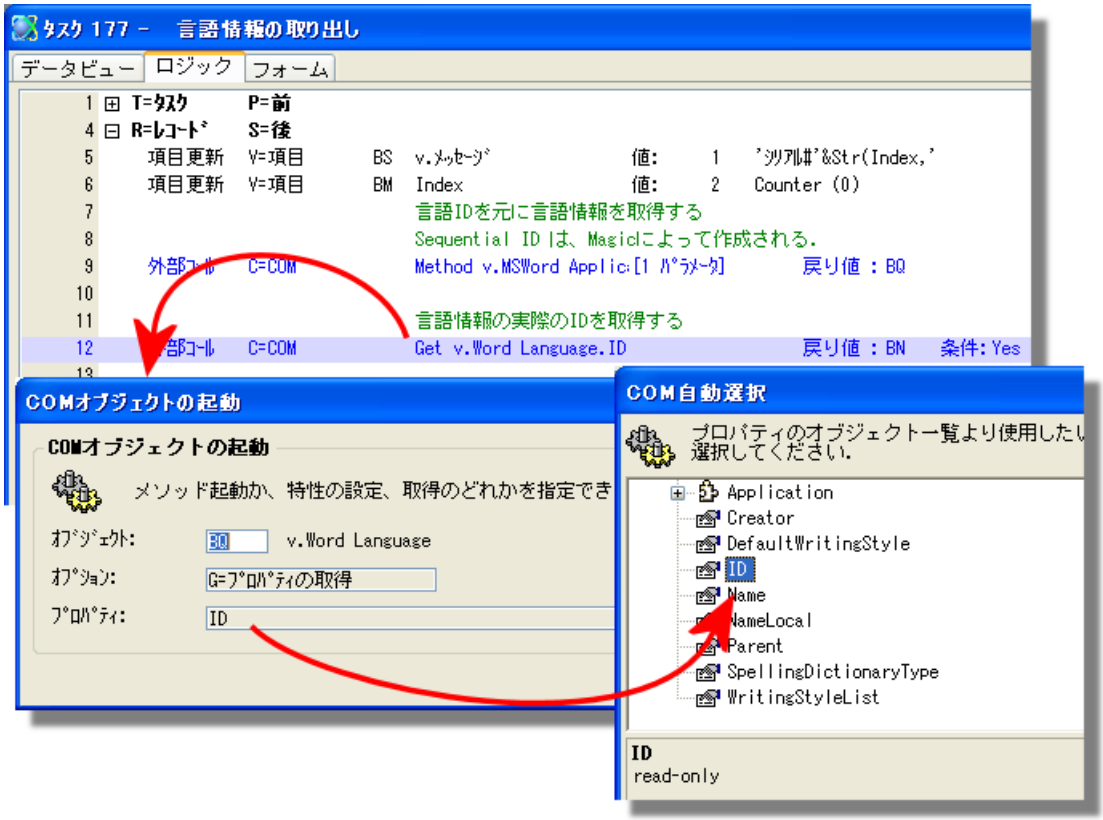


1. 最初に、コレクション全体のサイズを決定します。この例では、Word Application.Languages オブジェクト内に Count と呼ばれるメソッドがあります。このメソッドは言語の総数を返します。このメソッドは**タスク終了条件**特性で使用され、言語リスト全体を取り出すまで、タスクはループされます。



2. リスト内の各項目に対して、項目を取り出すために MgItemSequential メソッドを使用します。MgItemSequential は、ネイティブなオブジェクトには存在していませんが、COM の**自動選択一覧**に表示されます。

戻りのパラメータはオブジェクトを戻します。この場合は、Word Application.Language オブジェクトが返ります。



3. 次に、オブジェクトのプロパティを取得するために MgItemSequential によって返されたオブジェクトを使用します。この場合に、ID と名前を取得します。これらはテーブル内に格納されています。

タスクが実行されると、利用可能な Magic テーブル内にコレクションが格納されます。

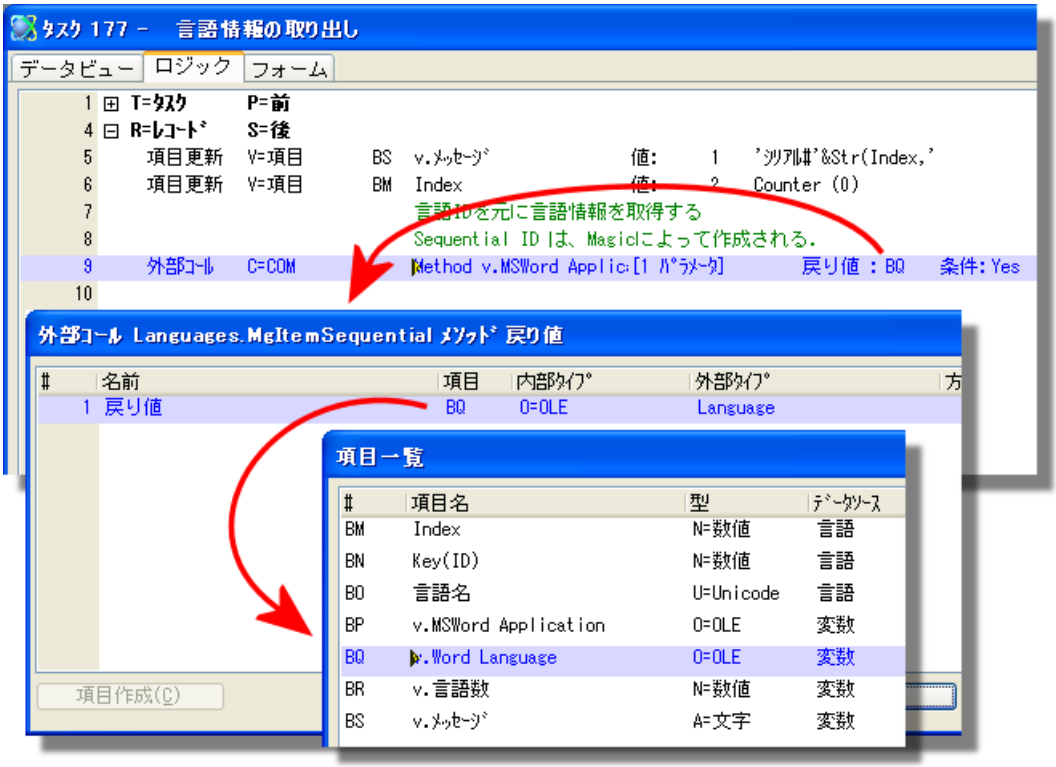
COM オブジェクト間でパラメータとして COM オブジェクトの受け渡しを行うには

COM オブジェクトはパラメータとして頻繁に他の COM オブジェクトを受け渡します。Magic ではこのような処理を簡単に行うことができます。COM オブジェクトはデータ項目のため、他のデータ項目と同じように渡すことができます。

COM オブジェクトを受け取る

この例は、「コレクションを取り出す」（343 ページ）で説明した MgItemSequential メソッドを使用します。

必要条件： プログラムがアクセスできる場所に COM オブジェクトを定義しておく必要があります。



1. パラメータの受け渡しを行う場所に移動します。この場合、戻り値として COM オブジェクトを受け取ります。
2. 他の項目と同じように**項目**カラムから**ズーム**します。COM オブジェクトが一覧表示され、***Enter*** を押すことで選択できます。

これでメソッドが起動されると、次の**外部コール**処理コマンドによって WordApp.Language という COM オブジェクトが利用できるようになります。

COM オブジェクト定義を再利用するには

品質がよく、信頼性が高く、保守が容易なアプリケーションを開発するには、なるべく項目を再利用することです。これはエラーを減らすだけでなく、アプリケーションの保守性を容易にし、またプログラムを早く作成することができます。Magic は、モデルリポジトリを使用することで、データ定義やコントロール定義を簡単に再利用することができます。

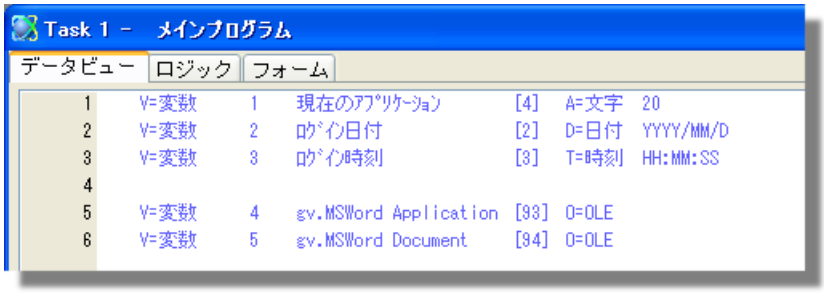
COM オブジェクト定義は、タスク内に直接定義するように、モデルリポジトリに定義します。唯一の違いは、一度定義することで、任意に再利用できるということです。また、モデルリポジトリでオブジェクトが定義された場合、クロスリファレンス（*Ctrl+F*）を使用してそのモデルが使用されているオブジェクトを検索することができます。

モデルリポジトリ内に COM オブジェクトを設定する方法については、「使用する COM オブジェクトを定義するには」（325 ページ）を参照してください。

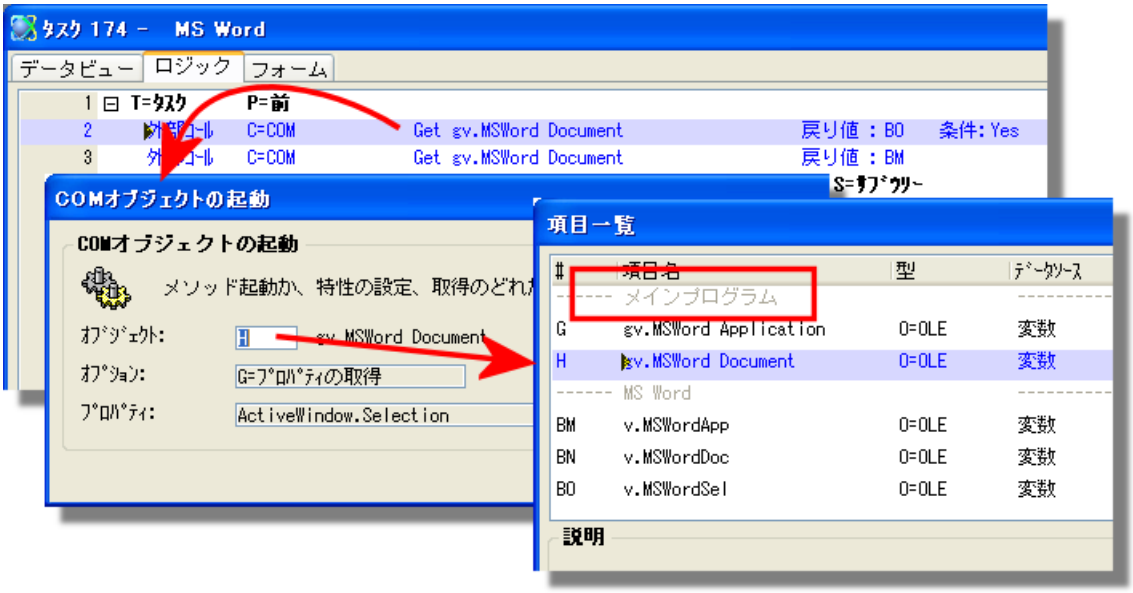
プログラム全体に渡って COM オブジェクトのインスタンスを保持するには

COM オブジェクトのインスタンスを生成する必要があり、異なるプログラムが実行中もそれをオープンしたままにする必要があるかもしれません。このためには、メインプログラム内で COM オブジェクトを定義します。これによって、プログラムリポジトリ内のどのプログラムでも利用可能になります。必要であれば、オブジェクトを初期設定するためにメインプログラムのタスク前を使用することで、プログラムが最初に実行するとインスタンスが使用できる状態になっています。

メインプログラム内で COM オブジェクトを定義する



1. タスク内に COM オブジェクトを定義するように、メインプログラムのデータビューエディタでオブジェクトを定義します。



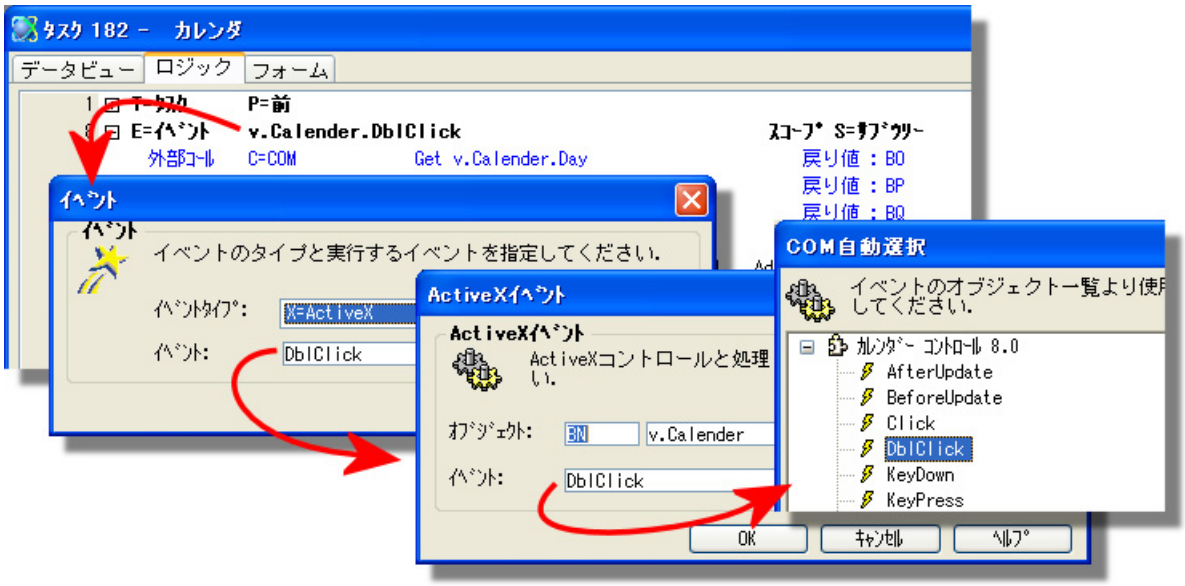
これで、COM オブジェクトを選択する際に、項目一覧の先頭のメインプログラムのヘッダの下に COM オブジェクトが一覧表示されます。

注： COM オブジェクトは、パラメータとしてプログラムの間で渡すことによって共有することもできます。

COM オブジェクトによって起動されたイベントを受け取るには

COM オブジェクトは、独自のイベントを発生します。これらのイベントは、Magic が自動的に検出され、そのイベントに対して実行させたい処理を定義することができます。

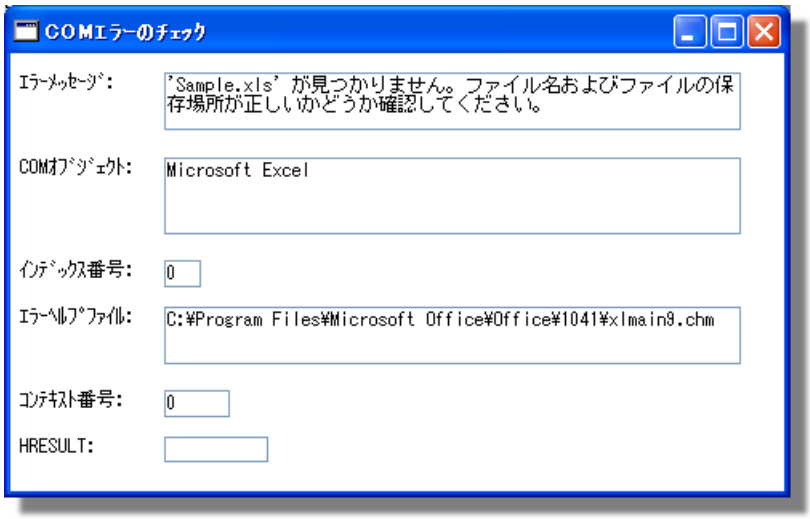
COM イベントに対するイベントハンドラを作成する



1. ロジックエディタ内にヘッダ行（*Ctrl+H*）を作成します。
2. *E* を入力して、**イベント**を選択します。
3. イベントダイアログボックスが開きます。**イベントタイプ**で、*X=ActiveX* を選択します。
4. **イベント**から**ズーム**します。**Active イベント**ダイアログボックスが開きます。
5. **オブジェクト**から**ズーム**します。アクセス可能な ActiveX オブジェクトの一覧が表示されます。イベントを受け取るオブジェクトを選択します。
6. **イベント**から**ズーム**します。**COM 自動選択**ダイアログボックスが表示されます。ここにはオブジェクトと起動されるイベントの一覧が表示されます。受け取るイベントを選択します。
7. OK を**クリック**して、すべてのダイアログボックスを閉じます。

これで、オブジェクトが選択されたイベントを実行すると、処理される**ロジックユニット**が作成されます。

COM オブジェクトによって発生したエラーを処理するには



COMError() 関数を使用して COM オブジェクトで発生したエラー情報を受け取ることができます。**COMError()** は最後に発生した COM エラーの内容を返します。構文は以下のとおりです。

COMError(*number*)

パラメータの ***number*** は、返される情報の種類を指定します。戻り値は文字列で返ります。

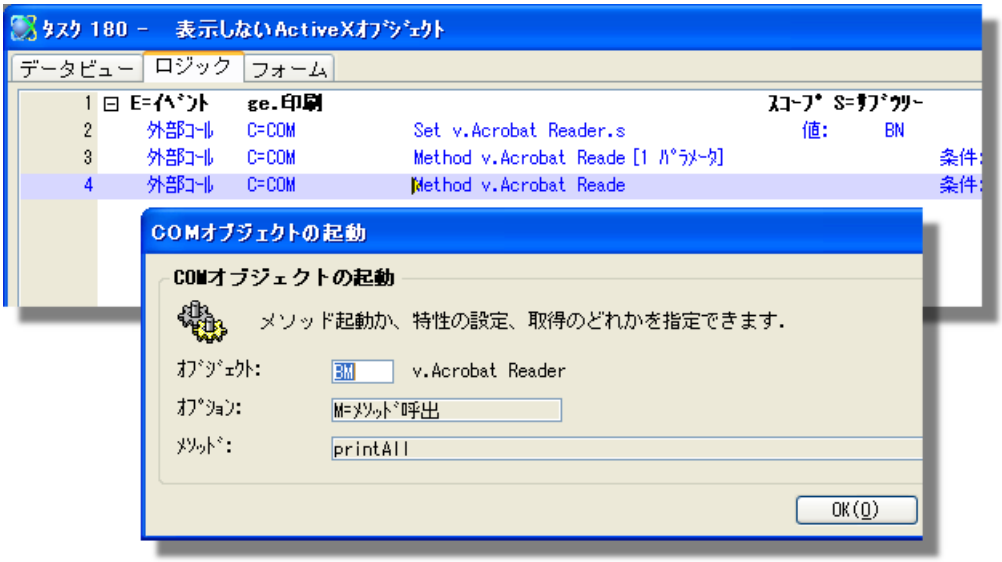
- 1 …… エラーの内容
- 2 …… COM オブジェクトの名前
- 3 …… インデックス番号
- 4 …… エラーに対するヘルプファイル
- 5 …… コンテキスト番号
- 0 …… HRESULT コード

ヒント : *プログラムやユーザ定義関数内にこの関数を定義することでカプセル化することができます。これによって、エラーをチェックしたり、エラーが見つかったときにメッセージを表示させたり、ログを出力したりすることができます。*
また、COM オブジェクトをデバッグしている間、この情報が Magic のアクティビティログに出力されます。

フォーム上に表示しないで ActiveX コントロールを処理するには

フォームに配置することで、ActiveX オブジェクトは、ウィンドウに表示されるように設計されています。しかし、表示しないで使用する方が便利な場合があります。この例では、PDF を印刷するために Acrobat Reader を使用していますが、ユーザに、どのように印刷させるかを指定させないようにしています。Acrobat Reader の ActiveX オブジェクトには、必要な機能が備えられています。以下はどのようにして行うかを説明しています。

不可視の ActiveX オブジェクトを使用する



1. 通常行うように COM オブジェクトを定義します（詳細は、「使用する COM オブジェクトを定義するには」（325 ページ）を参照してください）。
2. ActiveX オブジェクトを使用して実行するロジックを作成します。この例では、ファイル名を設定するために、プロパティの設定で src を指定し、印刷するために printAll というメソッドを呼び出しています。
3. ActiveX オブジェクトをフォームに配置します。ただし、コントロールのサイズ小さくします。
4. **ActiveX** コントロールの**コントロール特性**を開き、**可視**特性を ***No*** に設定します。

これでロジックは、表示されていた状態と同じ方法で **ActiveX** コントロールを使用して動作します。

COM メソッドと Magic のロジックを公開するには

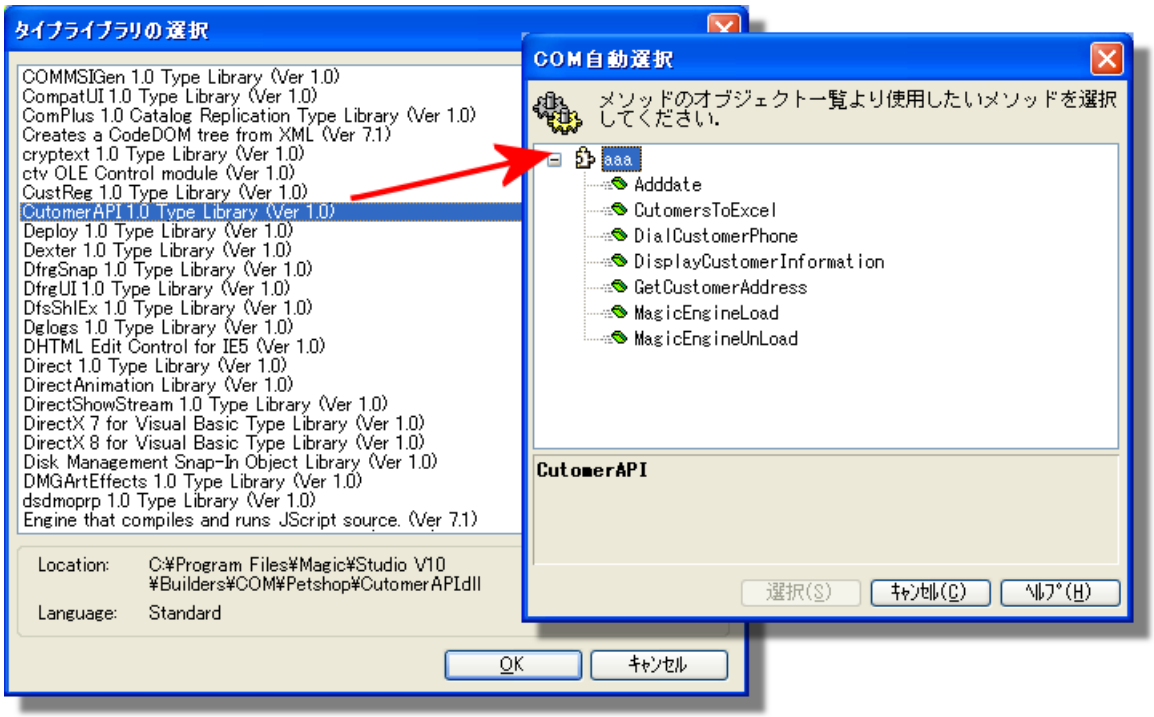
Magic では、アプリケーション内の API として作動する独自の COM にオブジェクトを作成することができます。この方法を使用することで、Magic 以外のアプリケーションツールから、Magic のプログラムにアクセスすることができます。これは、**COM インタフェースビルダ**を使用することで可能になります。

インタフェースビルダは、COM オブジェクトの動作方法に関するたくさんのコントロールを与えてくれます。以下は、関係するステップの要約です。

COM インタフェースを作成する

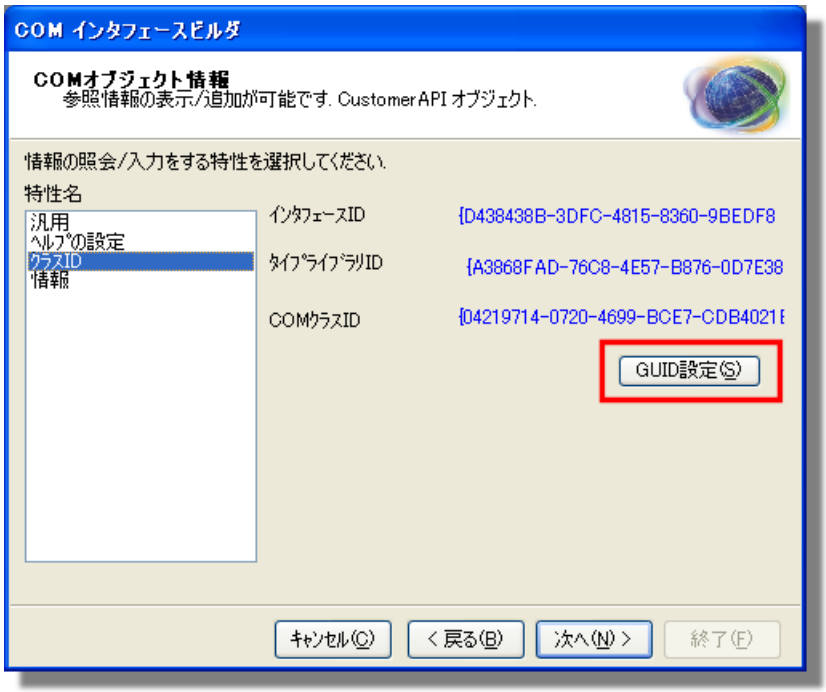
1. COM オブジェクトとして公開したいプログラムを決定します。これらのプログラムには公開名が定義されており、**外部**カラムがチェックされています。
2. **オプション→インタフェースビルダ→ COM** を選択します。**COM インタフェースビルダ**ウィザードが起動されます。次へをクリックします。
3. 次の画面で、既存のコンポーネントを修正したり、新規作成することができます。新規コンポーネントを作成するために、新規ボタンをクリックします。
4. 次に**インタフェース設定**ダイアログが表示されます。ここで COM オブジェクトの名前を入力します。***リモート***または***ローカルエンジン***を選択します。次へをクリックします。
5. 次に **COM オブジェクトプロパティ**ダイアログが表示されます。ここでは、COM オブジェクトに多くの特性（アプリケーション名、メッセージングサーバ、ユーザ名、パスワードなど）を設定することができます。また、ヘルプファイルに定義されているリンクキーや COM オブジェクトにアクセスしている際に表示されるヘルプテキストを指定することもできます。アプリケーションに対応する内容を設定した後、次へをクリックします。
6. 次に、**プログラム追加**ダイアログが表示されます。ここには、**外部**カラムでマークされたすべてのプログラムが表示されます。追加 >> または全て追加のボタンをクリックすることで選択されたプログラムに必要なプログラムが移動します。
7. 選択された各プログラム毎に、パラメータの設定や、ヘルプを定義するためのパラメータと詳細のボタンを使用することができます。またここでメソッドに新しく名前をつけることができるため、メソッド名は、Magic のプログラム名と同じである必要はありません。必要に応じて、メソッドを設定し、次へをクリックします。
8. 次に、**COM オブジェクト情報**ダイアログが表示されます。ここには、COM オブジェクトのバージョン番号やヘルプライブラリの位置、クラス ID を設定することができ、ドキュメントの一般的な情報の一部を入力することができます。必要に応じて設定し、次へをクリックします。
9. 次に、**COM オブジェクトパス**ダイアログが表示されます。ここには、DLL の生成先のパスを指定します。入力欄にはデフォルト値が表示されています。次へをクリックします。
10. 最後に、作成コンポーネントダイアログが表示されます。終了をクリックするとコンポーネントが作成されます。

最後に、**成功！**ダイアログが表示されます。これは DLL が正常に作成されたことを意味しています。この DLL を使用するには、PC に登録する必要があります。この場合、regsvr32.exe を使用します。このユーティリティを実行するには、Windows の**スタート→ファイル名を指定して実行**を選択し、regsvr32.exe ^DLL 名を入力するか、作成用のバッチファイルを作成します。しかし、テストのための簡単な方法は、regsvr32.exe のショートカットを作成し、そのショートカットに DLL をドラッグすることで自動的に登録されます。



他の COM オブジェクトのように選択することで、Magic から作成された COM オブジェクトの確認を行うことができます。図の左側は、オブジェクトが登録され、他のオブジェクトと一緒にリストに表示されていることを示しています。図の右側は、外部コール処理コマンドでオブジェクトを選択したときにメソッドの一覧が表示されていることを示しています。

公開するロジックのクラス ID を設定するには

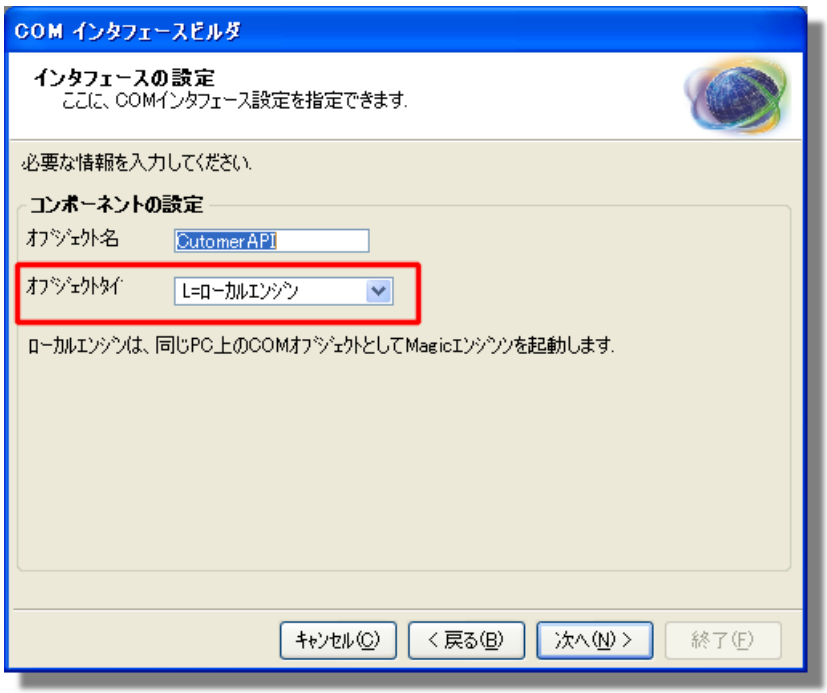


COM オブジェクトを作成すると、Magic が自動的にクラス ID を作成します。しかし、独自の ID を指定する必要がある場合、**COM インタフェースビルダ**の **COM オブジェクト情報**ダイアログで行うことができます。GUID 設定ボタンをクリックすると、任意の ID を入力することができます。

参照 : 「使用する COM オブジェクトを定義するには」（325 ページ）

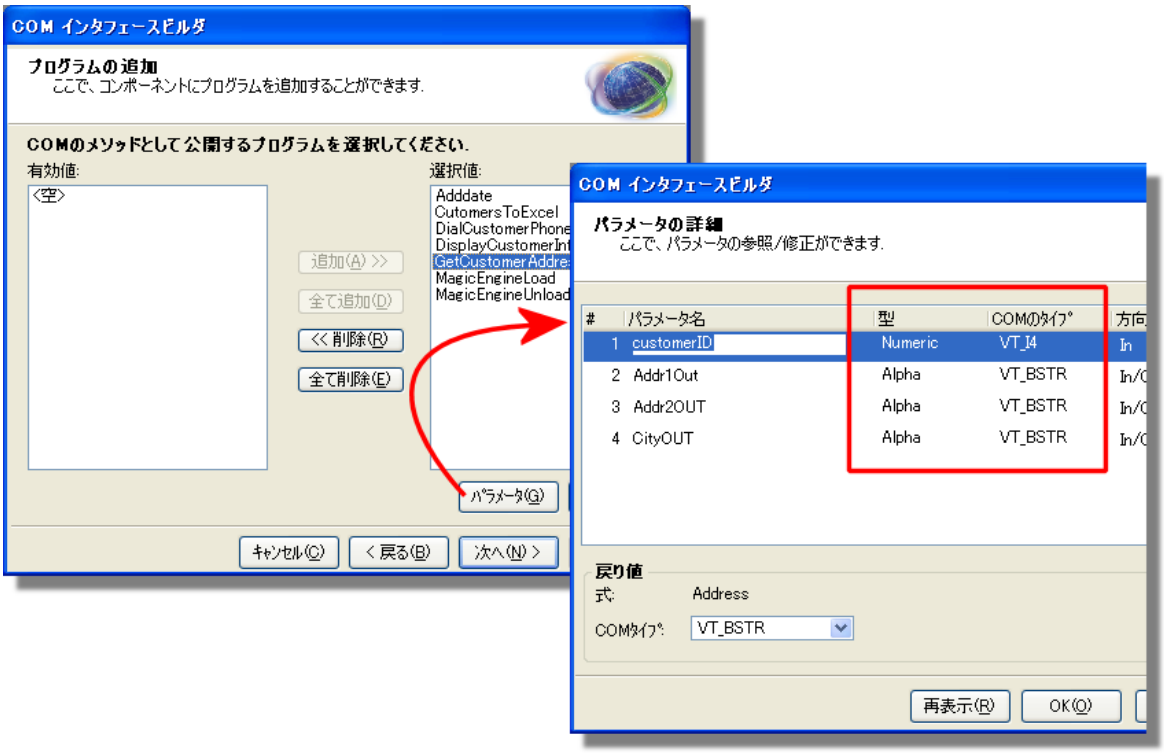
Magic にローカルにアクセスする COM クライアントを設定するには

COM オブジェクトは、ローカル PC 上やリモート PC 上で動作するように設定できます。



1. COM オブジェクトを作成する際に、オブジェクトタイプの指定が要求されます。COM オブジェクトが、ローカル PC 上で動作するように設定するには、**オブジェクトタイプ**を***ローカルエンジン***に設定するする必要があります。
2. また、DLL を実行する PC 上に登録する必要があります。RegSvr32 を使用して登録します。複数の PC 上にインストールする場合は、バッチファイルなどを使用して自動化処理します。
3. Magic の実行エンジンが有効でなければなりません。COM オブジェクトを作成する時に、エンジンの位置を指定することができますが、指定されない場合、システムはレジストリを参照し、そこに登録された Magic エンジンを使用します。

Magic ロジックを公開する際に COM のデータタイプを決定するには



COM インタフェースビルダを使用して、COM オブジェクトを作成する時に、パラメータを定義することができます。

メソッドパラメータの詳細を表示 / 修正する

1. **オプション→インタフェースビルダ→ COM** を選択します。**COM インタフェースビルダ**のウィザードが起動されます。**次へ**を**クリック**します。
2. 次の画面で、既存のコンポーネントを修正したり、新規作成することができます。表示したいメソッドを持つコンポーネントを選択します。
3. **プログラム追加**ダイアログに達するまで、**次へ**を**クリック**します。ここには、この COM オブジェクトにあるメソッドの一覧が表示されます。表示したいメソッドを選択し、**パラメータ**ボタンを**クリック**します。
4. このメソッドの**パラメーター一覧**が表示されます。COM データタイプは **COM のタイプ**カラムに表示されます。この内容は変更できません。Magic が自動的に Magic のデータ型に割り当てます。

参照 : COM のデータタイプについては、「Variant の Magic データ型」（336 ページ）を参照してください。

［このページは意図的に空白にしています。］

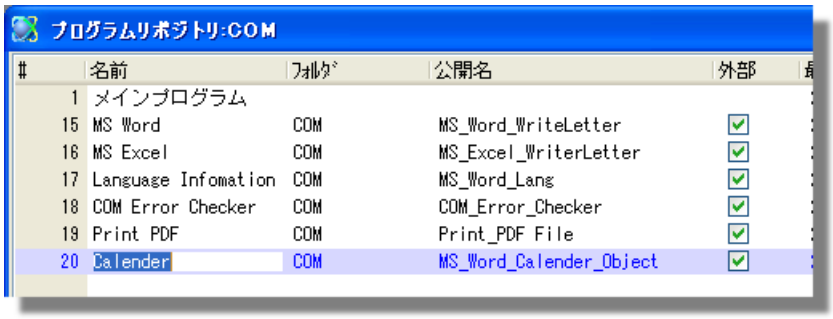
第 16 章 : コンポーネント

プロジェクト間で Magic のオブジェクトを再利用するには

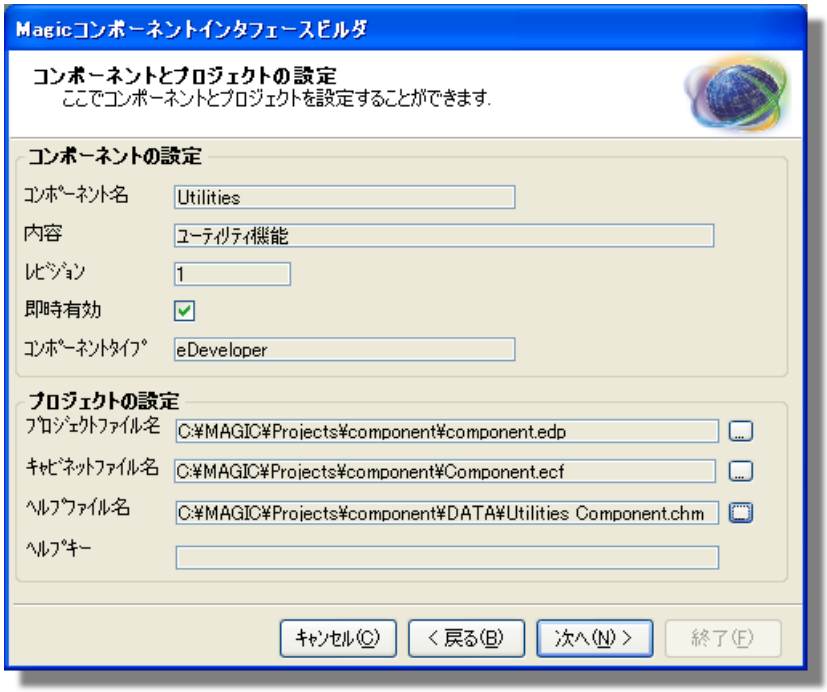
Magic Studio は、1 つのプロジェクト内でオブジェクトを再利用する機能があります。リポジトリ内にモデルやデータソースを定義することにより、プログラム作成する工数を減らすことができます。しかし、複数のプロジェクト間で再使用可能なモデルやデータソース、およびプログラムのライブラリを作成することにより同じように再利用が可能になります。この場合、使用するプロジェクトを変更せずにライブラリをアップグレードすることができます。

これらのライブラリは、Magic コンポーネントと呼ばれます。どの Magic プロジェクトからでもコンポーネントを作成し、***.edp*** または ***.ecf*** ファイルのどちらかでコンポーネントを利用することができます。アプリケーションのどの部分がどのコンポーネントを使用しているかということを管理することができるため、異なる機能を持った異なるコンポーネントをを作成することができます。いくつかの機能は、廉価バージョン用に利用することができるため、特に商業用のアプリケーションを開発する上で有益です。

Magic コンポーネントを作成する



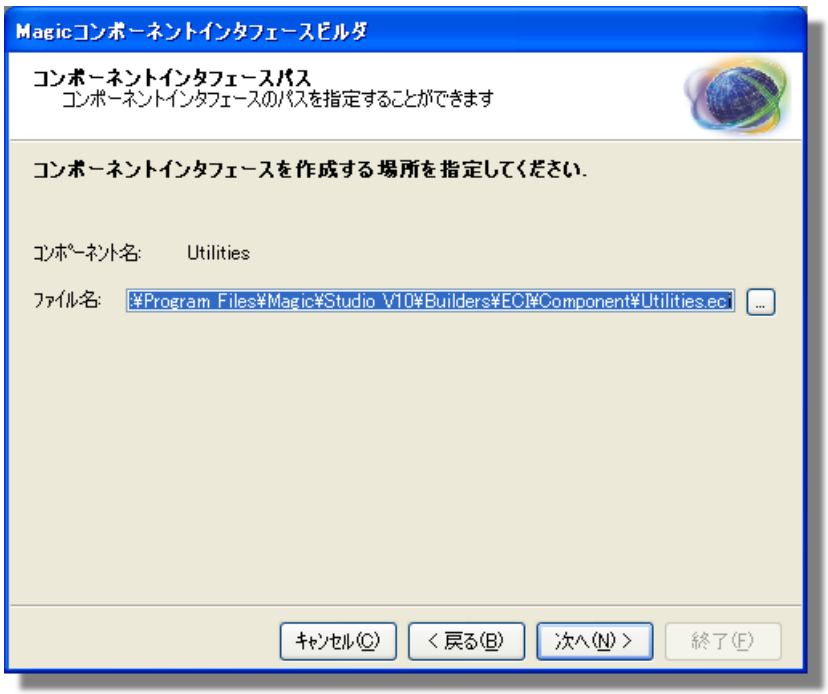
1. コンポーネントとして公開するオブジェクトを決定します。それらのオブジェクトに公開名を指定します。プログラムの場合は、更に**外部**カラムをチェックする必要があります。
2. **オプション→インタフェースビルダ→ Magic** を選択します。**Magic コンポーネントインタフェースビルダ**のウィザードが起動されます。次へをクリックします。
3. 次の画面で、既存のコンポーネントを修正したり、新規作成することができます。新規コンポーネントを作成するために、新規ボタンをクリックします。



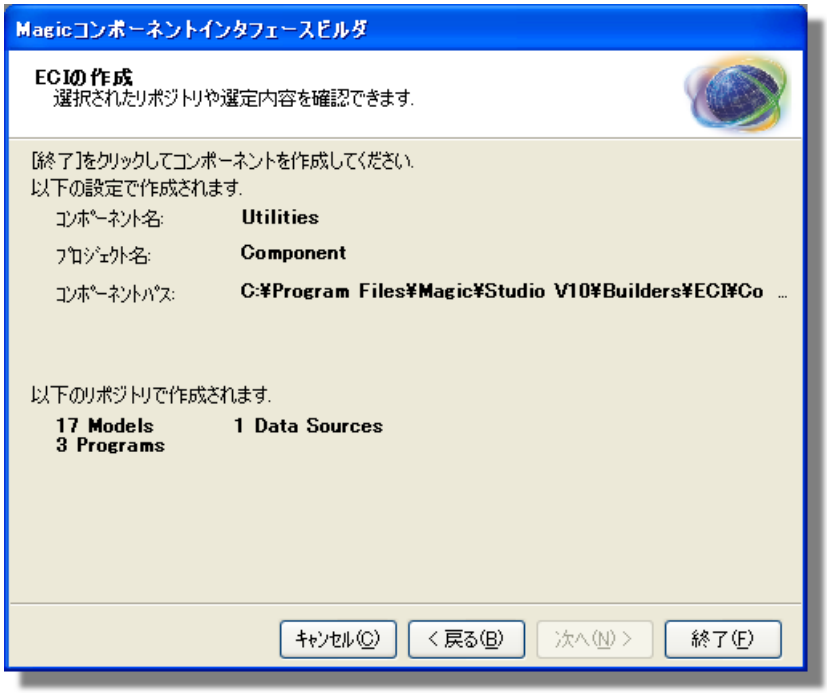
4. 次に、**コンポーネントとプロジェクトの設定**ダイアログが表示されます。ここには指定可能な多くの項目が表示されます。*F1* を**クリック**するとより詳細な説明が表示されます。ここでは**コンポーネント名**と**プロジェクトファイル名**を指定しなければなりません。次へを**クリック**します。

5. 次に、リポジトリの追加ダイアログが表示されます。ここでは、コンポーネントに表示されている中から使用したいリポジトリを選択できます。「有効」から「選択済み」へ項目を移動するために、追加 >> や全て追加のボタンをクリックします。次へをクリックします。

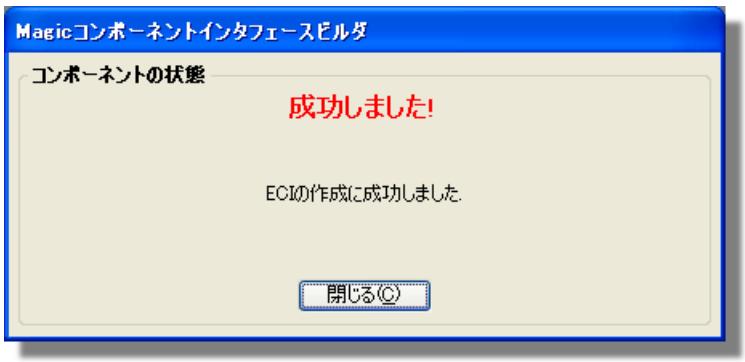
6. 次に、選択された各リポジトリ内のオブジェクトを選択することのできる一連のダイアログが表示されます。コンポーネントとして必要な項目のみを選択します。ここでは、モデルをコンポーネントに追加しています。



7. . 次に、作成する ***.eci***（eDeveloper Component Interface）ファイルのパスを入力するダイアログが表示されます。これは、コンポーネントを使用するプロジェクトがアクセスする場合に必要なファイルです。次へを**クリック**します。



8. 最後に、確認のために選択内容が表示されるダイアログが表示されます。コンポーネントを作成する場合は終了ボタンを**クリック**し、選択内容を変更するに場合は戻るボタンを**クリック**します。



9. ***.eci*** ファイルが生成されると成功しましたメッセージが表示され、コンポーネントが使用できるようになります。

注： ***.eci*** **ファイルはテキストファイルで、Magic.ini ファイルに似た構成になっています。必要に応じて、手動で編集することができます。**

これで、コンポーネントが利用できる準備が整いました。

参照： 「プロジェクトにコンポーネントを読み込むには」（362 ページ）

オブジェクトを公開するには

コンポーネントの作成中に、コンポーネントとして何を公開し、何を公開しないかを指定できます。コンポーネントとして公開されるには、オブジェクトは以下の基準を満たす必要があります。

1. オブジェクトには、**公開名**が定義されなくてはなりません。
2. オブジェクトがプログラムの場合、**外部**カラムのボックスがチェックされていなければなりません。
3. コンポーネントの作成中に項目を選択する必要があります（「Magic コンポーネントを作成する」（357 ページ）に説明があります）。

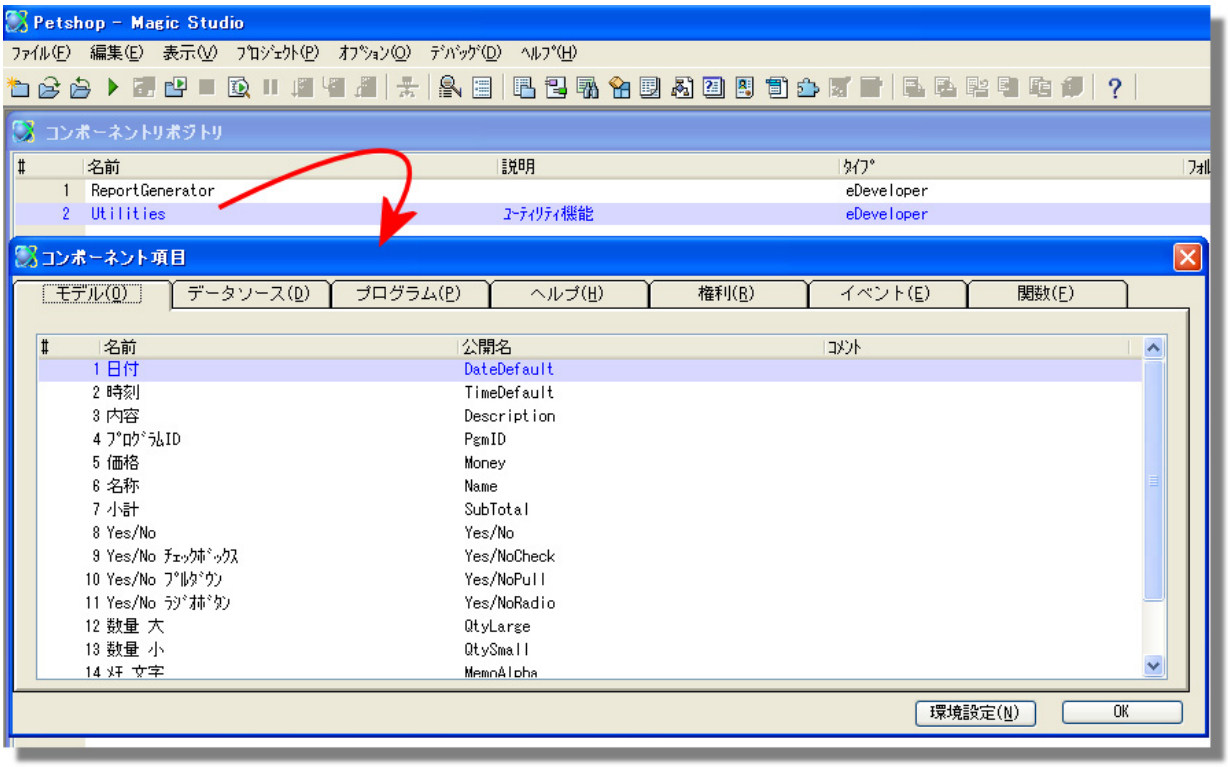
参照： 「プロジェクト間で Magic のオブジェクトを再利用するには」（357 ページ）

プロジェクトにコンポーネントを読み込むには

プロジェクトにコンポーネントを読み込む前に、*.eci* ファイルを作成する必要があります。これは、プロジェクト内にコンポーネントを読み込む際に使用されるテキストファイルです。*.eci* ファイルの作成方法は、「Magic コンポーネントを作成する」（357 ページ）を参照してください。

.eci ファイルが作成されたら、プロジェクトで利用可能にするために以下の手順で読み込み処理を行います。

Magic コンポーネントを使用する



1. **プロジェクト→コンポーネント**（***Shift+F7***）を選択します。**コンポーネント**リポジトリが開きます。
2. ***F4***（**編集→行作成**）を押下して 1 行追加します。使用するコンポーネントの名前を入力します。この名前は、コンポーネントの名前と同じにする必要はありません。これは表示用としてのみ使用されます。
3. **ズーム**（***F5*** または、***ダブルクリック***）してこのコンポーネントの内容が記述された *.eci* ファイルを選択します。
4. 再度**ズーム**すると、すべてのコンポーネント項目が一覧表示されます。コンポーネント用の動作環境を設定するボタンが下にあり、コンポーネント内の各アイテム毎のタブが表示されます。
5. これらの一覧内の各項目から**ズーム**することで**モデル特性**やプログラムに対するパラメータなどの情報が表示されます。しかし、プロジェクト内から特性値を変更することはできません。使用することだけ可能です。

これで、他のオブジェクトを使用するように、プロジェクト内でコンポーネント項目を使用することができます。例えばモデルを選択すると、そのプロジェクト内にあるモデルと同じ選択一覧にこのコンポーネントが表示されます。

注： 実行時は、*.eci* ファイルは不要です。コンポーネントが同じパス上に存在していることを確認する必要がありますが、一端コンポーネントを読み込むと、*.eci* ファイルは使用されません。

ホストアプリケーションで使用している既存のコンポーネントの変更を有効にするには

コンポーネントの修正内容を反映させるには、コンポーネントを作成し直し、ホストプロジェクトでこれを再読込する必要があります。変更内容を反映させるには 2 つの方法があります。 1 つ目は、実行環境のコンポーネントを差し替える場合で、ホストアプリケーションは変更されません。 2 つ目は、開発環境でコンポーネントを使用していて、コンポーネント内の新しい項目を追加する場合です。両方の場合に対して以下で説明します。

実行環境でコンポーネントを変更する

実行環境にインストールされているコンポーネントを変更する場合は、古いコンポーネントを新しいものに交換するだけです。例えば、コンポーネントのバグを修正したり、より速く動作するような処理を組み込んだり、その他の内部処理を修正した場合、このような変更作業が発生します。オブジェクトとそれらのパラメータの名前が変わらない限り、ホストアプリケーションは修正する必要がありません。

項目をコンポーネントに追加しても、ホストアプリケーションは新しいオブジェクトを認識しません。開発時に追加されたオブジェクトを使用していないため、これは認識されておらず、何も変わりません。しかし、既存のオブジェクトの公開名を変更したり、プログラムのパラメータを変更した場合、致命的なエラーが発生する場合があります。

開発環境でコンポーネントを変更する

開発エンジンでアクセスしているコンポーネントの内容が変更された場合、変更内容を反映させるため ***.eci*** ファイルを変更する必要があります。変更しないのであれば追加されたオブジェクトを参照しないようにします。このような場合、以下の手順で作業する必要があります。

1. コンポーネントが自分で作成したものであれば、新しい ***.eci*** ファイルを作成し直します。この操作は、最初から作成するのではなく既存のコンポーネントを修正する作業になります。詳細は、「Magic コンポーネントを作成する」（357 ページ）を参照してください。もし別のサードパーティから受け取ったコンポーネントであれば、新しいコンポーネントと一緒に新しい ***.eci*** ファイルを取得します。
2. **プロジェクト→コンポーネント**（***Shift+F7***）を選択し、更新したいコンポーネント上にカーソルを置きます。
3. **オプション→コンポーネントの読込 / 再読込**を選択します。使用する ***.eci*** ファイル名を選択するダイアログが表示されます。***.eci*** ファイルを選択し、開くを**クリック**します。

これでコンポーネント一覧は再読込され、新しいコンポーネントを使用して動作することができます。

ヒント : ***古いコンポーネントを削除してから再読込を行わないでください。このような操作を行うと、コンポーネントオブジェクトへのすべての参照情報が失われます。***
読込 / 再読込の操作によって参照内容が正しくリフレッシュされます。

コンポーネント内のオブジェクトの名前を変更する

Magic Studio 内では、オブジェクトの名前は固定ではありません。これは内部の参照システムを使用して項目を参照しているからです。しかし、オブジェクトを他のプログラムで利用可能になるように設定した場合、他のプログラミングツールと同じように、実際のテキスト名にもとづいて参照されます。

このため、コンポーネントオブジェクトの名前を変更する良い方法がありません。オブジェクトの名前を変更し、コンポーネントを再読込した場合、オブジェクトへのすべてのリンクが破損してしまい、オブジェクト名カラムに「項目は有効ではありません」が表示されます。

従って、コンポーネントを設計する際に、COM ライブラリの開発と同じような考え方で開発することを推奨します。オブジェクトの公開名を変更したり、プログラムに渡すパラメータを変更したり、オブジェクトを削除しないようにしてください。例えば、新しい改善されたカレンダオブジェクトを組み込みたい場合、新しく作成したカレンダ 2 を呼ぶようにしてから、オリジナルのカレンダを削除するようにします。

コンポーネント用のヘルプファイルを提供するには

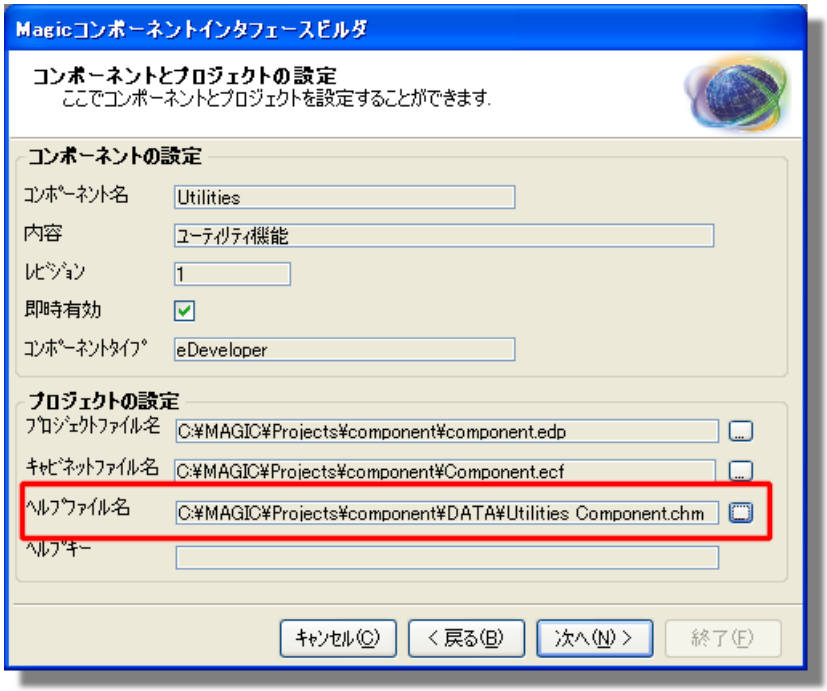
コンポーネント用のヘルプファイルを提供することで、コンポーネントの内容が理解されやすく、使用しやすいものになります。

Windows のヘルプファイルを作成するには、サードパーティ製のオーサリングツールが必要となります。すべてのヘルプ情報を入力し、コンパイルして、***.chm***（ヘルプ）ファイルを作成します。ヘルプファイルの内部には、各ヘルプトピックに対するユニークな数値インデックス（マップ ID）や複数のキーワードを定義することがができます。

Magic のコンポーネントにおいて、このインデックスやキーワードを設定することで、コンポーネントリポジトリ上で ***F1*** を押下すると関連するヘルプトピックを表示させることができます。

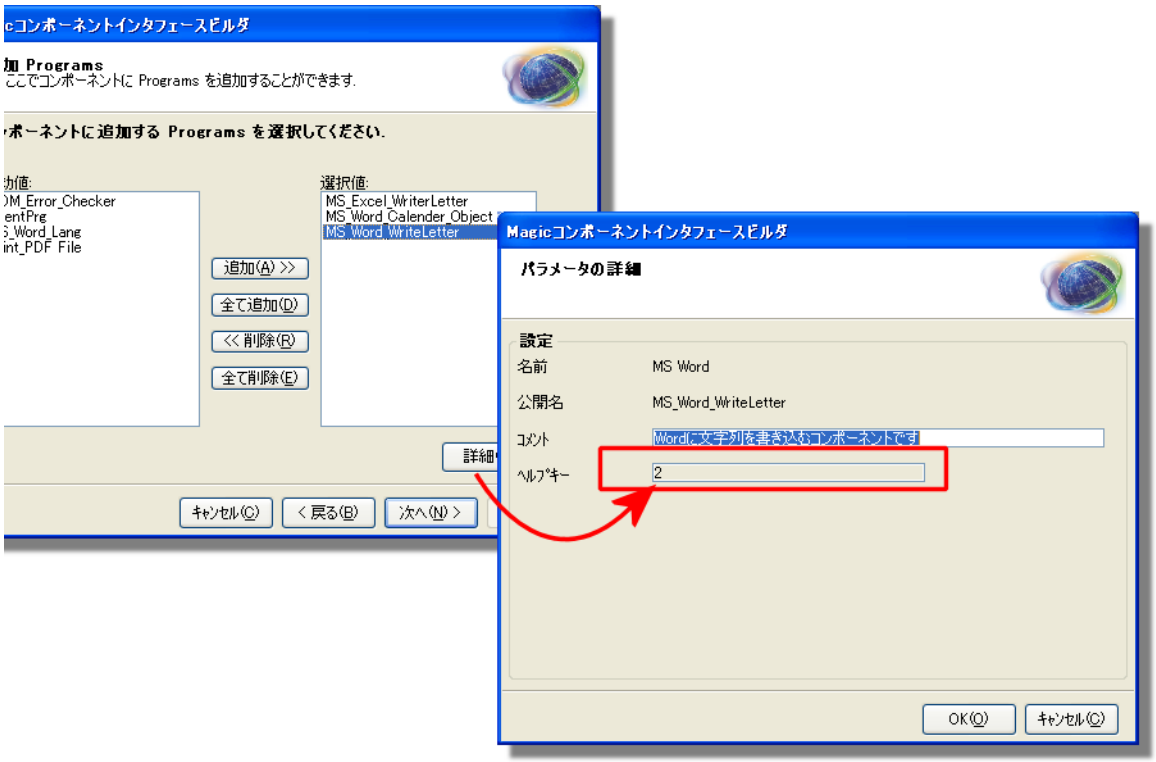
コンポーネントを作成する際に、ヘルプファイルが存在する場所の指定と、どのような状況でどのインデックスやキーワードを割り当てるかをあらかじめ検討しておく必要があります。

ヘルプファイルを組み込む



1. コンポーネントとプロジェクト設定画面で、使用したいヘルプファイルの名前とパスを指定します。パス名を直接入力するよりも、%WorkingDir% のような論理名を使用することを推奨します。
2. ヘルプキーには、表示させたいヘルプトピックのインデックス（マップ ID）となる数値かキーワードとなる文字列を指定します。

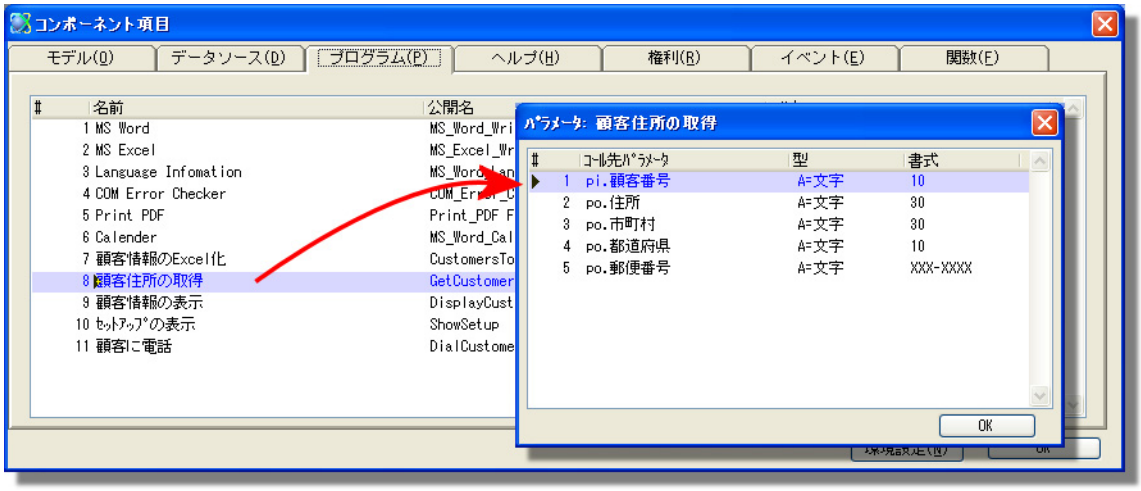
注： ヘルプファイルとヘルプキーの両方が設定されていない場合、Magic のリファレンスヘルプが表示されます。



3. ヘルプを追加したいオブジェクトに到達するまでウィザードを進めます。
4. オブジェクトに対する詳細ボタンを**クリック**します。パラメータや**ヘルプキー**を追加するダイアログが表示されます。このオブジェクトに対応するヘルプトピックのマップ ID やキーワードを入力します。

これで、このコンポーネントが組み込まれると、**コンポーネント**リポジトリの**コンポーネント項目**テーブルにおいて該当する項目にカーソルが位置付けられている状態で ***F1*** を押下すると、指定されたヘルプトピックが表示されます。また**コメント**に入力された内容も**コンポーネント項目**テーブルに表示されます。

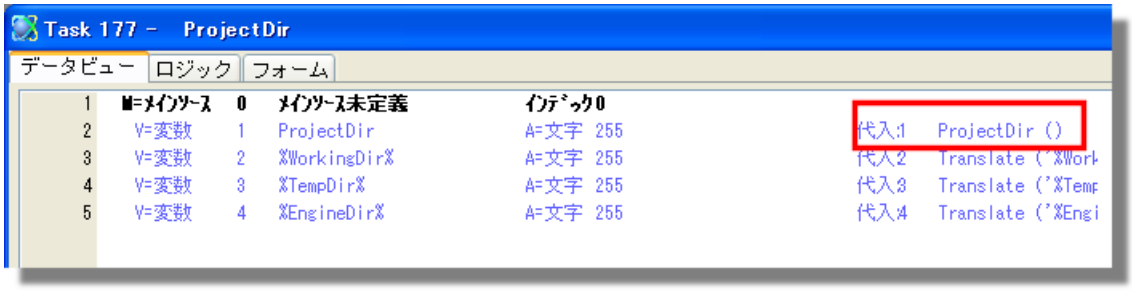
コンポーネントのオブジェクトに関する詳細情報を参照するには



プロジェクト内でコンポーネントを使用している場合に、各オブジェクトでズームする（*F5* または、ダブルクリック）ことでオブジェクトに関する詳細情報を参照することができます。例えば、図の例では顧客住所の取得からズームするとそのオブジェクト（プログラム）に対するパラメーター一覧が表示されます。

これ以外の情報は表示されません。コンポーネントは「ブラックボックス」として扱われ、実行する場合に必要な情報以外は表示されません。

コンポーネントが存在するディレクトリにアクセスするには



コンポーネントが実行している間は、**ProjectDir()** 関数を使用することでコンポーネントの存在するディレクトリにアクセスすることができます。

この関数は実際のパス名を返します（論理名ではありません）。%WorkingDir% や %TempDir% などのディレクトリ情報を表す論理名は、ロードされたすべてのコンポーネントに対して同じ値で評価されますが、**ProjectDir()** 関数は、実行中のコンポーネントに対応したパス名が返ります。

現在実行中のアプリケーションがコンポーネントかどうかを確認するには

Magic のプロジェクトは、どのように組み込むかによって、ホストまたはコンポーネントとして実行させることができます。従って、1 つのプロジェクトファイルに 2 つの役目を持たせることができます。その際、どのモードで実行しているかを **IsComponent()** 関数を使用して確認することができます。関数の構文は以下の通りです。

IsComponent()

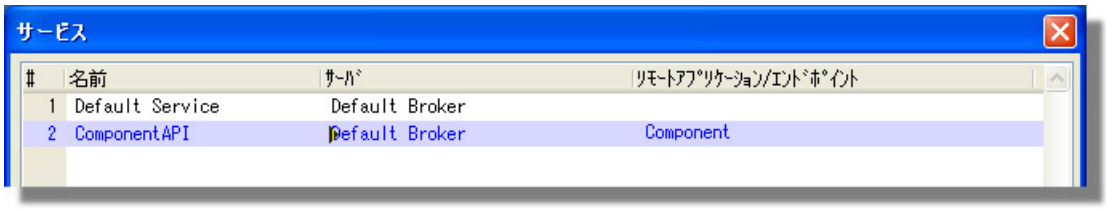
戻り値：タスクがコンポーネントとして実行している場合は、True が返ります。

コンポーネントとして定義されない別のアプリケーション内のプログラムを呼び出すには

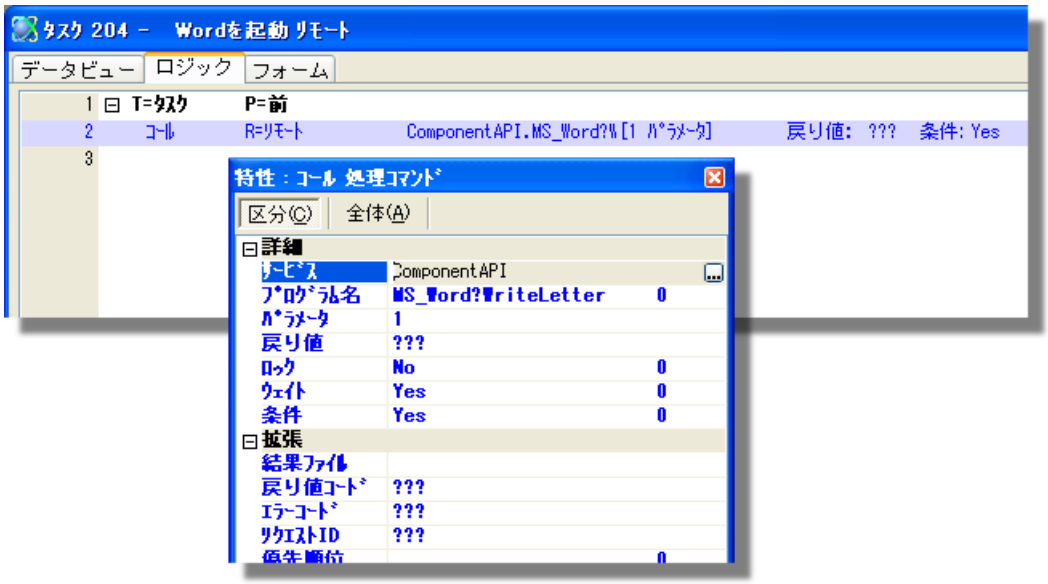
コンポーネントを使用しないで別のアプリケーション内のプログラムを呼び出すこともできます。COM オブジェクトや SOAP サービスとして呼び出されるプログラムを組み込む場合を含めて、いくつかの方法があります。しかし、最も直接的な方法として、コールリモートとコール公開名の２つの処理コマンドを使用する方法があります。

これらのタイプの呼び出しを使用する場合、実行モードで起動されるまで起動するオブジェクトの情報を持っていないことに注意してください。従って、名前（プロジェクトファイル名や公開名）とパラメータが正しく設定されているかどうかは、開発者の責任になります（Magic ではチェックされません）。

コールリモートを使用する



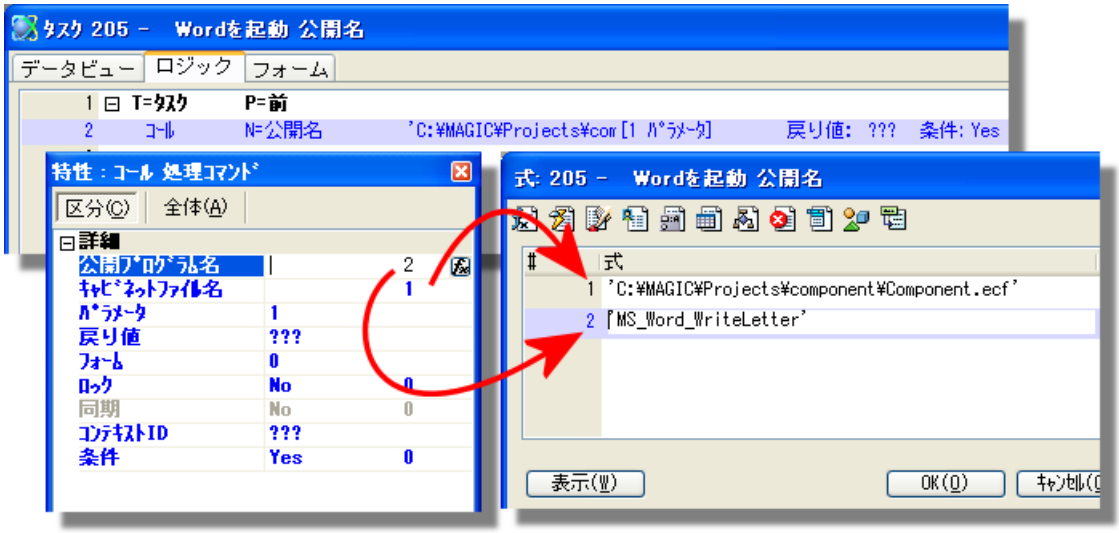
1. 最初に、サービステーブル（オプション→設定→サービス）に他のアプリケーションを表すサービスを設定します。リモートアプリケーション / エンドポイントカラムからズームして、使用したいアプリケーションを選択します。
2. 次に、タスクにコールリモート処理コマンドを定義します。
   - 任意のヘッダ行内で *F4* を押下して 1 行追加します。
   - *C* を入力してコール処理コマンドを選択します。
   - *R* を入力して *R= リモート*を選択します。



3. 処理コマンド特性（*Alt+Enter* または特性シートをクリック）を開きます。
4. サービス特性からズームして、実行させたいアプリケーションを表すサービスを選択します。
5. プログラム名特性を入力します。ここには、指定されたアプリケーションに定義されているプログラムの公開名を入力します。開発者の責任で正しく入力する必要があります。
6. パラメータ特性からズームしてプログラムに必要なパラメータを定義します。この設定も開発者の責任で正しく行ってください。自動的に比較しません。

これで、プログラムが実行されると、他のアプリケーションのプログラムを呼び出すことができます。

コール公開名を使用する



1. タスクにコール公開名処理コマンドを定義します。
   - 任意のヘッダ行内で *F4* を押下して 1 行追加します。
   - *C* を入力してコール処理コマンドを選択します。
   - *N* を入力して *N= 公開名*を選択します。
2. 処理コマンド特性（*Alt+Enter* または、特性シートをクリック）を開きます。
3. 公開プログラム名特性から式エディタにズームします。実行させたいプログラムの公開名を定義します。
4. キャビネットファイル名特性でズームしてプログラムが定義されているキャビネットファイルを指定します。
5. パラメータ特性からズームしてプログラムに必要なパラメータを定義します。

これで、プログラムが実行されると、指定されたキャビネットファイルのプログラムを呼び出すことができます。

ヒント :パス名を直接入力する方法は、開発時には内容が確認できますが、メインプログラム で定義された変数にパス名を格納したり、論理名を使用することを推奨します。

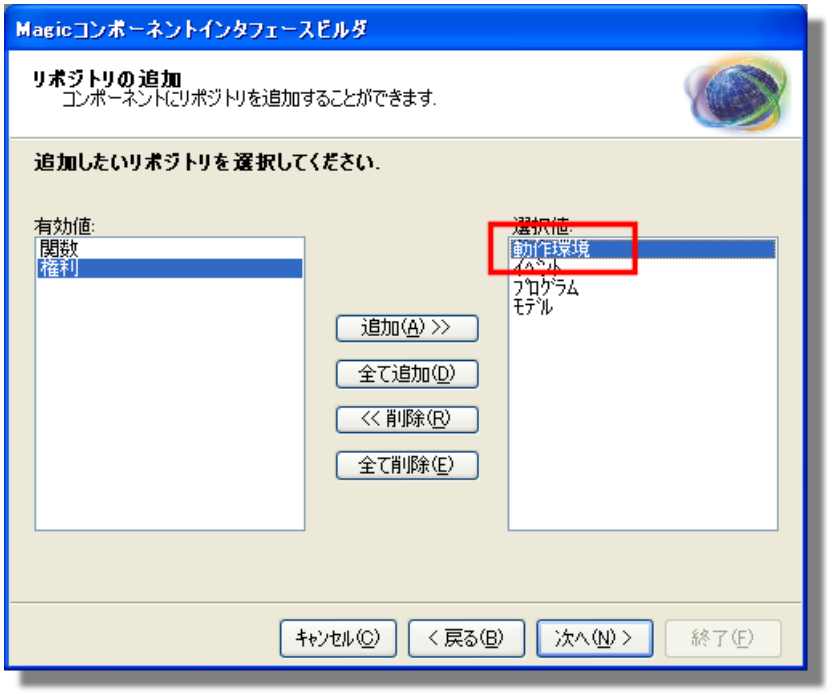
アプリケーション間で再帰呼び出しを処理するには

コンポーネント間の再帰呼び出しはできません。現在のアプリケーション内のオブジェクトを参照するコンポーネントオブジェクトを読み込もうとした場合、上記のようなメッセージがステータス行に表示されます。

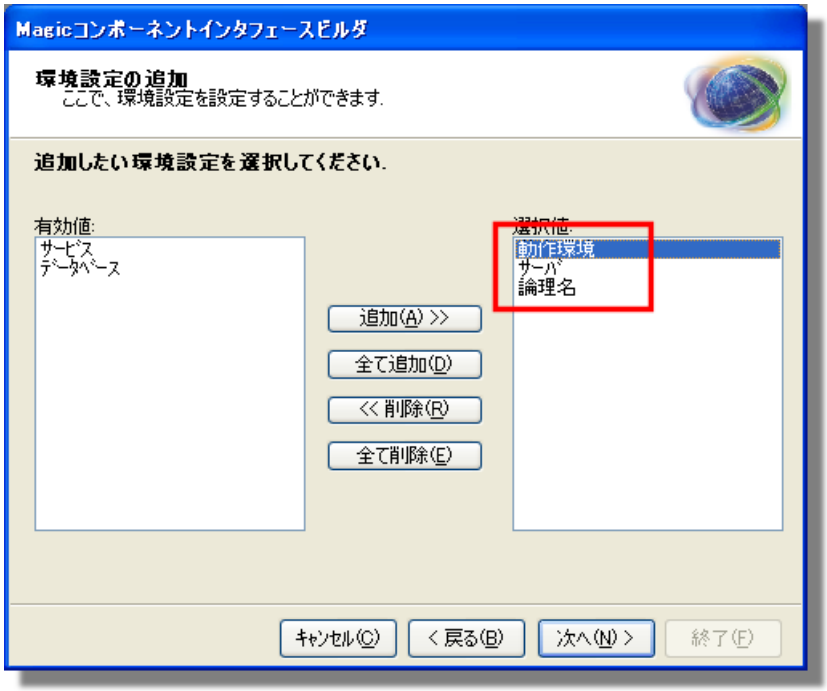
一般的に、アプリケーションを構造化することはよい考えです。このためこのようなアプリケーションは再帰的な呼出しを必要としません。例えば、多くのアプリケーション間で使用されるユーティリティのライブラリを持つことは可能ですが、ユーティリティはそれらのアプリケーション内のプログラムを呼び出すことはしません。

しかし、もし再帰的な呼出しが必要な場合は、**コール公開名**や**コールリモート**の各処理コマンドを使用して組み込むことができます。定義方法は、「コンポーネントとして定義されない別のアプリケーション内のプログラムを呼び出すには」（369 ページ）を参照してください。

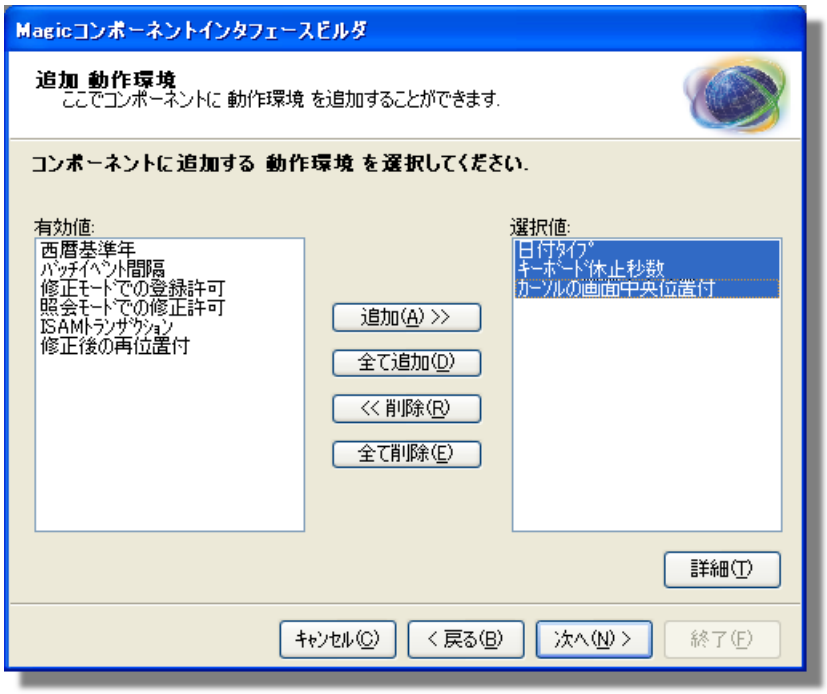
コンポーネントの動作環境を組み込むには

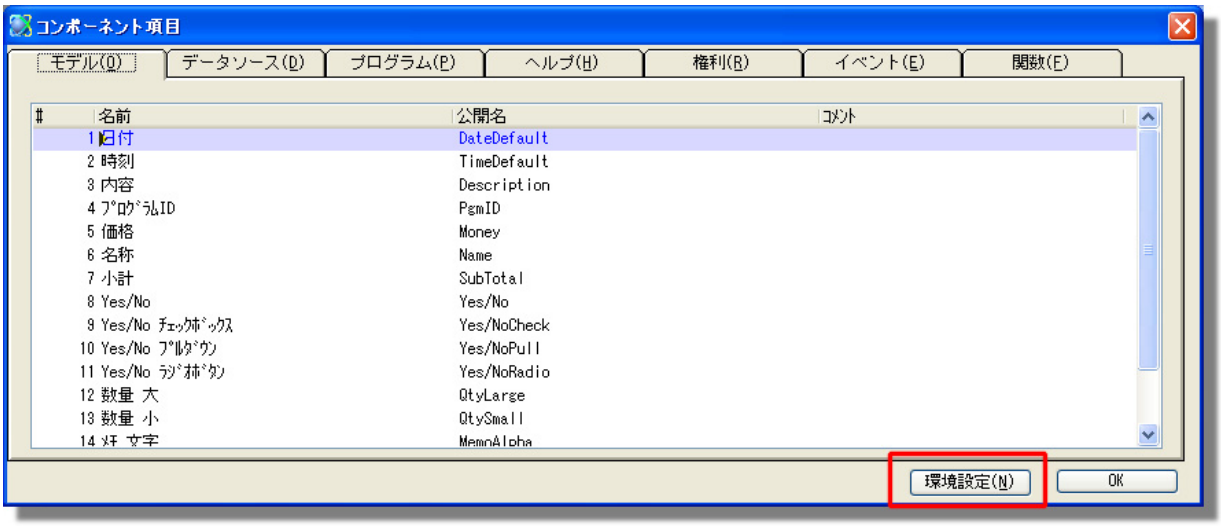


コンポーネントを作成する際に、そのコンポーネントのために必要な動作環境を設定することができます。ウィザードのリポジトリ選択で***動作環境***（*Environment*）を選択すると、各環境情報を個別に指定する一連のダイアログが表示されます。



この例では、コンポーネントの動作環境、サーバ、および論理名の各情報を組み込まれるコンポーネントの一部でになるように定義しています。これらの各項目に対して、詳細な設定ダイアログが開き、必要な情報を設定することができます。





コンポーネントが使用されると、環境設定ボタンをクリックすることで設定内容が表示されます。

コンポーネントへのアクセスを最適化するには

コンポーネントは、最初に実行される前にメモリにロードされる必要があります。コンポーネントが最初に実行される場合に、遅延の無いようにするには、プロジェクトがロードされる際にコンポーネントもロードされるように指定することです。

この設定は、2 つの場所で行うことができます。コンポーネントを作成する場合に、**即時起動**のフラグを ***.eci*** に指定することができます。この設定がデフォルト設定となります。

また、**コンポーネント**リポジトリ内で、**コンポーネント特性**（*Alt+Enter*）を利用することでこのデフォルト設定を変更することができます。

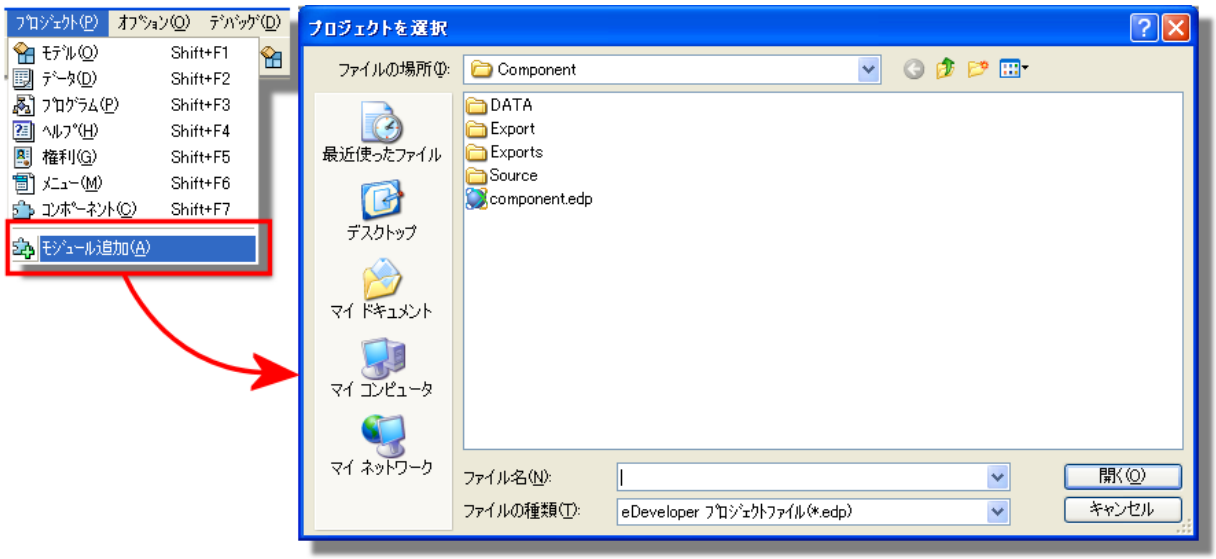
コンポーネント用プロジェクトを一括管理するには

１つのアプリケーションを開発する場合、ホストアプリケーション用のプロジェクトと複数のコンポーネント用プロジェクトを使用する場合があります。その際、関連するプロジェクトファイルを一括で管理できたほうが効率的に開発できます。

Magic Studio では、コンポーネント用のプロジェクトをホスト側のプロジェクトの***モジュール***として登録することで、プロジェクトを簡単に切り替えることができるようになります。

モジュールとしてプロジェクトを登録する

1. ホストアプリケーション用のプロジェクトを開いた状態で、**オブジェクトを選択**ダイアログ（**プロジェクト→モジュール追加**）を開きます。
2. コンポーネント用のプロジェクトファイルを選択します。
3. **ナビゲータペイン**にモジュール用ツリーが表示され、選択されたプロジェクトがツリーに追加されます。

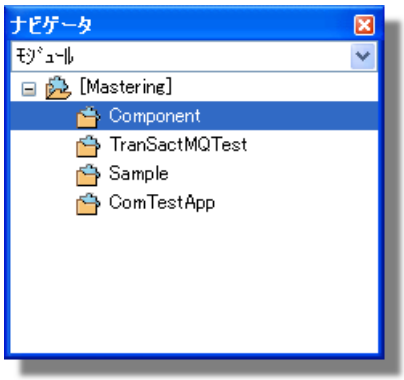


注：　モジュールのプロジェクト情報がホスト側のプロジェクト（edp）ファイルに登録されるだけで、実際にプロジェクト関係のファイルがホスト側に追加されるわけではありません。

登録されるプロジェクトのパスは、ホストアプリケーションを基準に設定されます。他の PC にコピーする場合は、同じ位置関係になるようにする必要があります。

登録したモジュールを開く

1. **ナビゲータペイン**で***モジュール***を選択し、**モジュールツリー**を開きます。
2. オープンしたいモジュール名を**ダブルクリック**します。
3. 指定されたプロジェクトが開きます。



登録したモジュールを削除する

1. **ナビゲータペイン**で***モジュール***を選択し、**モジュールツリー**を開きます。
2. オープンしたいモジュール名を**クリック**して選択します。
3. ***F3***（**コンテキストメニュー→行削除**）を押下すると削除確認のダイアログが表示されます。**はい**を**クリック**すると削除されます。

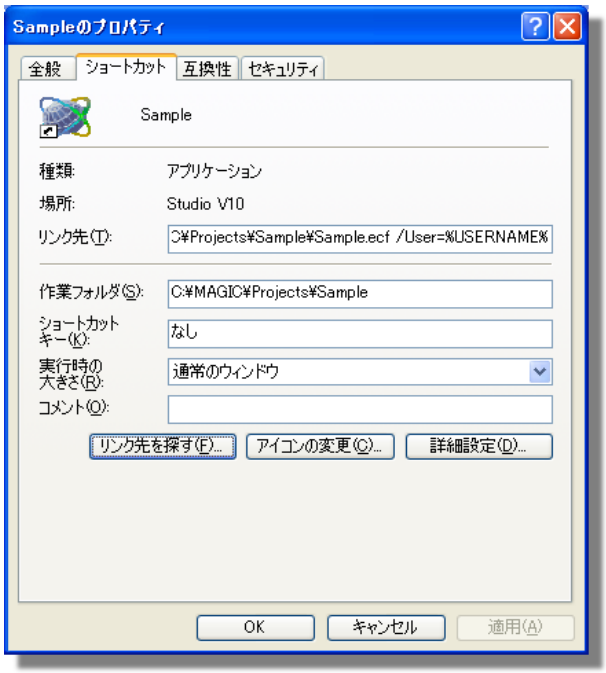
注： ホスト側のプロジェクト（edp）ファイル内のモジュール情報が削除されるだけで、実際にプロジェクト関係のファイルが削除されるわけではありません。コンポーネントとしている場合にも影響しません。

第 17 章 : 動作環境

Windows のログイン ID を使用して Magic にログオンするには

Magic には独自のログインダイアログがありますが、これを使用しないで、Windows のログイン ID を使用して自動的に Magic にログオンすることができます。これは簡単で、ユーザの操作時間を短縮することができます。

ユーザが Windows にログインすると、システム変数 %USERNAME% にはユーザ ID が含まれます。この内容は、Magic 起動時のショートカットに指定することで渡すことができます。



Magic にログインするために Windows のユーザ ID を使用する

1. 通常の方法と同じように、Windows 上で Magic を起動するショートカットを作成します。
   - **リンク先 :** *.ecf* ファイルの名前とパス
   - **作業フォルダ** : 作業フォルダとなるパスを入力します（通常は、*.ecf* ファイルの存在する場所と同じパスを指定します）。
2. リンク先を指定した後、**/USER=%USERNAME%** を追加します。この例では、結果として以下のように指定されます。

   ```
   C:¥eDeveloper10_Projects¥Examples¥Examples.edp /User=%USERNAME%
   ```

3. Windows のログイン ID に合わせるために、Magic に同じユーザ ID を登録します。パスワードは空白にします。
4. **動作環境**ダイアログ（**オプション→設定→動作環境**）の**パスワードの入力**（**システム**タブ）選択し、*No* に設定します。
5. **動作環境**ダイアログの**ログオン許可**（**システム**タブ）選択し、*No* に設定します。

これで、ユーザは、ユーザ ID やパスワードを入力せずに自動的に Magic にログオンすることができます。

注 : **ログオン**ダイアログが表示できる状態の場合、他のユーザによって簡単にログインできてしまうため、この機能を使用する場合は慎重に行う必要があります。これによって、例えば管理者用のデータを参照できてしまうような問題が発生するかもしれません。

このような問題を回避するには以下のような方法があります。

- （上記に説明されるような）アイコンを使用する場合を除いて、Magic にログオンすることができないようにしてください。ユーザが管理者権限で PC にアクセスできないようにすることでこの方法が可能になります。
- ユーザ ID を指定する場合は、**パスワード入力**を *Yes* に設定してパスワードを入力しなければいけないように設定します。
- 重要なデータにアクセスするユーザに対してはパスワード入力を強制してください。この方法を使用した場合、管理者は通常のユーザとは異なるショートカットを作成し、パスワードを指定するようにします。

```
C:¥eDeveloper10_Projects¥Examples¥Examples.ECF /InputPassword=Y
```

Magic Studio 起動時に自動的にプロジェクトをオープンするようにするには

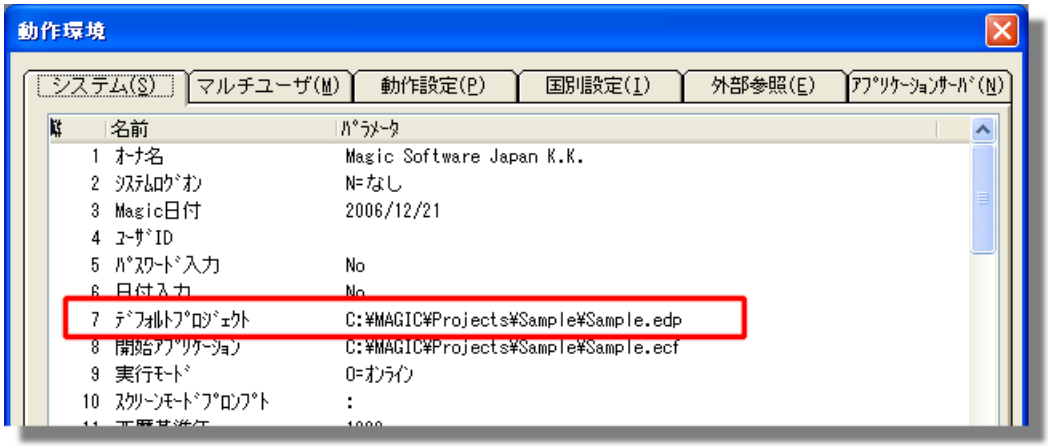
Magic Studio を起動するには２つの基本的な方法があります。

- プロジェクト（***.edp***）ファイルやプロジェクトのショートカットを**クリック**して起動する。
- Windows のスタートメニューやデスクトップ上のショートカットを**クリック**して Magic Studio を直接起動する。

直接 Magic Studio を起動した場合、デフォルトではプロジェクトはオープンされません。しかし、**動作環境**ダイアログで**デフォルトプロジェクト**を設定することによって指定したプロジェクトをオープンさせることができます。ここには、プロジェクトファイル名とパス名を指定します。

注：　この設定は、**開始アプリケーション**と同じようなものですが、実行アプリケーション（*.ecf* ファイル）の読込は行われません。

デフォルトプロジェクトを設定する



1. 上記で示されるように、**動作環境**ダイアログ（**オプション→設定→動作環境**）の**デフォルトプロジェクト**（**システム**タブ）でプロジェクトのパスとファイル名を入力することによってデフォルトプロジェクトを設定することができます。
2. **動作環境**ダイアログで指定する代わりに、Magic のインストールディレクトリ内にある Magic.ini を編集することでも指定できます。この例では、以下のように指定します。
   ```
   DefaultProject = C:¥eDeveloper10_Projects¥Examples¥Examples.edp
   ```

これで Magic を再起動すると、指定されたプロジェクトがオープンされます。

Web 用のアプリケーションを開発する場合は、アプリケーションをテストするためのウェブサイトや特別なソフトウェアは必要ありません。Magic エンジンをサーバエンジンとして動作させることができます。以下に説明するような、作業が多少必要となります。Web 上でアプリケーションを実行させる方法についての基本的な知識も必要です。

サーバアプリケーションをテストする

必要条件：　Web アプリケーションを実行させる前に、以下の準備が必要です。

1. アプリケーションを実行させる PC に IIS サービスがインストールされていることを確認します。
2. Magic の**インターネットリクエスタ**がインストールされることを確認します（IIS がインストールされていれば、Magic のインストール時にデフォルトでインストールされます）。
3. **動作環境**ダイアログ（**オプション→設定→動作環境**）の**アプリケーションサーバを有効にする**（**アプリケーションサーバ**タブ）を ***Yes*** に設定します。設定内容を変更したら、プロジェクトを閉じて Magic を再起動します。
4. MRB を起動します（**スタート→プログラム→ Magic Studio V10 →ブローカとリクエスタ→ MRB の起動**）。MRB をサービスとしてインストールした場合、Windows 起動時に自動的に MRB も起動されます。

これで、特定プログラムをテストしたい場合以下のような操作を実行します。

1. テストしたいプログラム上にカーソルを置きます。
2. **デバッグ→ブラウザから実行**（***Ctrl+Shift+F7***）を選択します。

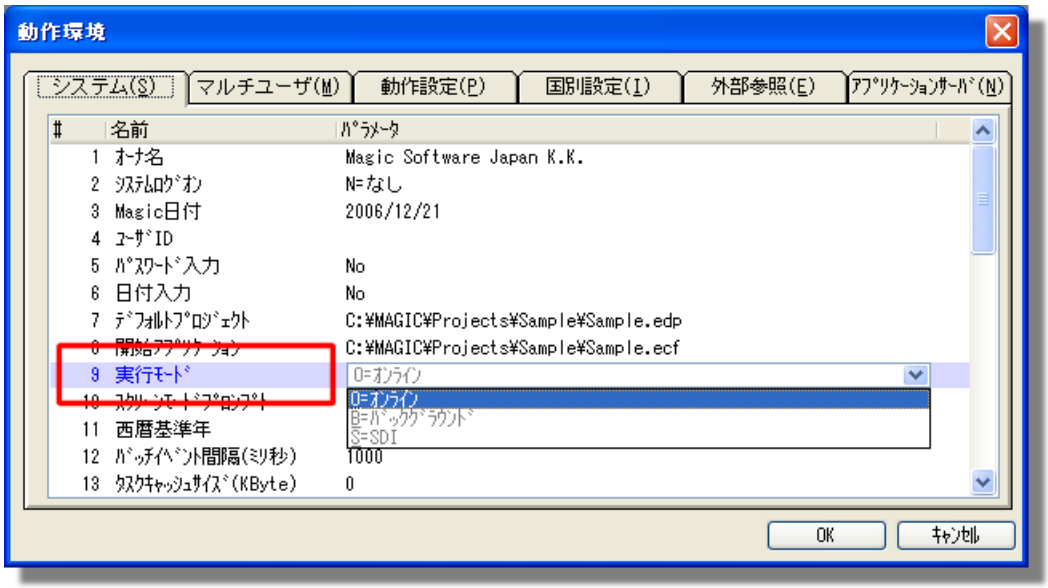
Magic は自動的にブラウザウィンドウを開き、そこでプログラムを実行します。特別なリンクを設定する必要はありません。それは自動的に作成されます。デバッグモードで実行していれば、**アクティビティモニタ**や**項目モニタ**などを使用することでアプリケーションの動作内容を確認することができます。

SDI アプリケーションとして実行するように指定するには

SDI（Single Document Interface ）アプリケーションは、個々のウィンドウが独自のメニューやタスクバー、ステータスバーを持っているものです。メインの MDI ウィンドウが背景に存在せず、SDI ウィンドウを閉じると、アプリケーションも終了します。

メインのコンテキストやメインプログラムを持っており、背景も存在していますがユーザには表示されません。

SDI アプリケーションを作成する

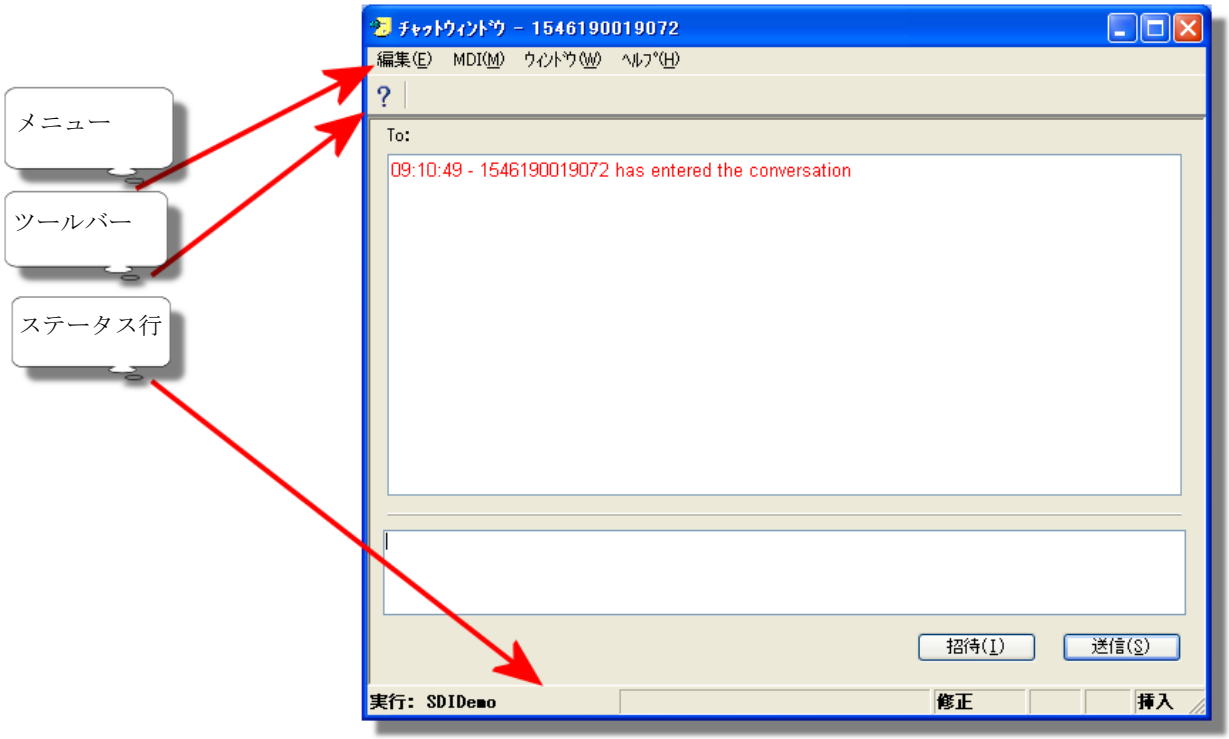


1. 動作環境ダイアログ（オプション→設定→動作環境）の実行モード（システムタブ）を *S=SDI* に設定します。
2. コンテキストがメインかどうかにもとづいて、タスク前でプログラムを実行します。このようにしないと、タスクが何回も起動されます。CtxGetName()='Main' を実行条件にすることで判断することができます。
3. メインプログラムのタスク特性でウィンドウを開く特性（インタフェースタブ）が *No* になっていることを確認します。このようにしないとエラーが発生します。

参照： 「SDI プログラムとして実行させるには」（382 ページ）

SDI プログラムとして実行させるには

SDI プログラムは、独自の表示環境を備えたプログラムです。



このプログラムには、親（MDI）はありません。親はデスクトップになります。これは独自のコンテキストで実行します。

Magic では簡単に SDI ウィンドウを作成することができます。以下の手順を実行します。

SDI コンテキストを定義する

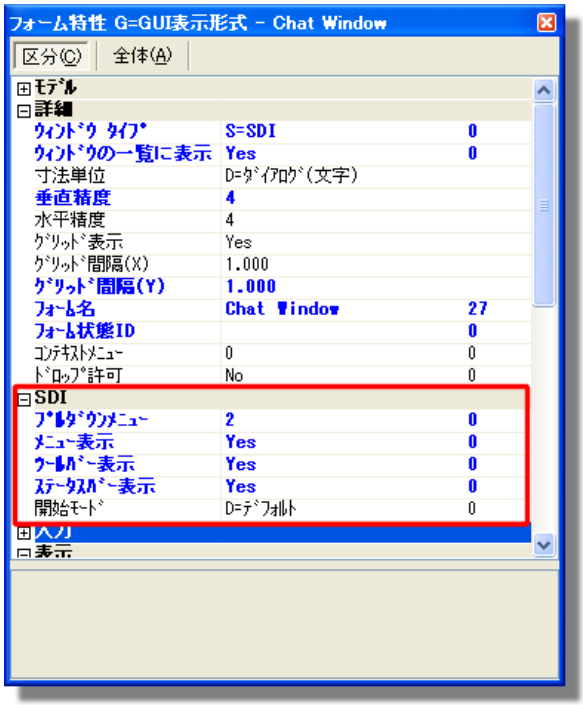
1. **フォーム特性**を開いて**ウィンドウタイプ**特性を ***S=SDI*** に設定します。
2. **タスク特性**を開き、**並行実行**特性（**拡張**タブ）をチェックします。
3. **メインプログラム**の**タスク特性**で**ウィンドウを開く**（**インタフェース**タブ）を ***No*** に設定されていることを確認します。このようにしないとエラーが発生します。

これで、SDI コンテキストの定義に必要な作業が終わりました。

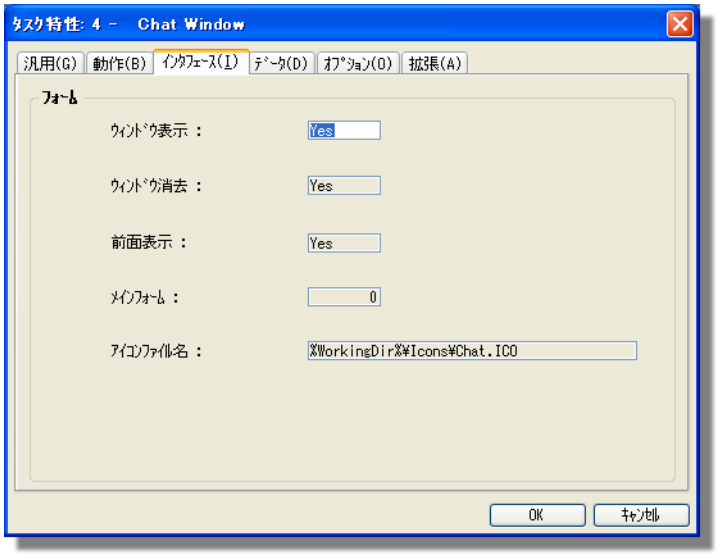
SDI コンテキストを修正する

1. フォーム特性：タスクを SDI と定義した場合、フォーム特性に **SDI** と呼ばれるセクションが表示されます。ここには以下の特性があります。
   - **プルダウンメニュー** と **メニュー表示** ：メニューリポジトリのメニューから選択
   - **ツールバー表示**：表示するか否か
   - **ステータスバー表示**：表示するか否か
   - **開始モード**：デフォルト、最大、最小。最大が指定された場合、Magic のフレームに制限されないためディスクトップ全体に表示されます。

   フォーム特性の残りの大半は有効で、他のフォームの場合と同じように動作します。**開始時の位置**特性の ***M=MDI の中央***の設定が例外です。MDI が存在しないため、この設定は意味がなく無視されます。

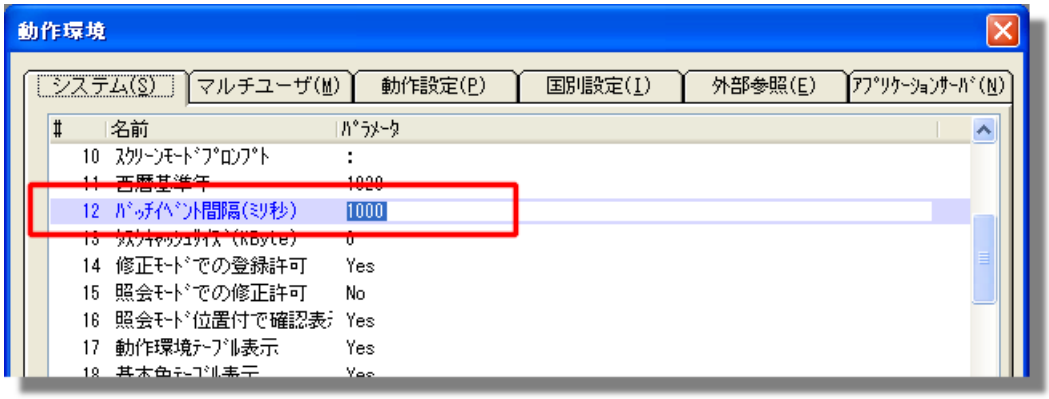


2. タスク特性で、フォームのアイコンを設定することができます。これはフォームの左上の隅で表示され、Windows で ***Alt+Tab*** を押下してウィンドウを切り替える際にも表示されます。ここに何も設定されない場合、アプリケーション特性で指定されたアイコンが表示されます。



参照： 「SDI アプリケーションとして実行するように指定するには」（381 ページ）

バッチタスクでの画面の再表示間隔を指定するには



バッチタスクを作成する場合、ユーザに対して処理経過を示すプログレス画面を表示させることが一般的に行われています。これによって処理の状態を確認することができます。この画面は再表示させる必要があり、これによってレコードサイクルやプログレスバーの移動を表示させることができます。しかし、画面が頻繁に更新されると、バッチタスクのスピードに影響を与えることになります。理想的な再表示間隔は、実行している PC のパフォーマンスに依存します。新しくより速い PC は、古い PC より短い再表示間隔にしても処理することができます。

従って、**動作環境**ダイアログの**バッチイベント間隔**を使用することで、この機能を実行環境に合わせて設定することができます。バッチイベント間隔は 1 ミリ秒の単位で設定します。これは、1000 と設定した場合、画面が 1 秒間に 1 回の割合で再表示することを意味しています。

バッチイベント間隔を指定する

1. **動作環境**ダイアログ（**オプション→設定→動作環境**）の**バッチイベント間隔**（**システム**タブ）に移動します。使用したいミリ秒の値を入力します。
2. Magic.ini ファイルの **BatchPaintTime** のパラメータに値を設定することでも可能です。
3. または、**INIPut()** 関数を使用して一時的に **BatchPaintTime** 設定を変更することもできます。特定のバッチプログラムのみ間隔を変更し、それ以外は元に戻したい場合は、この方法を使用します。

エンジンが非同期イベントのチェックを行う間隔を指定するには

バッチタスクが実行している場合、Magic エンジンはユーザからの入力処理を待つことなく、システムが許す限り高速でレコードを処理します。

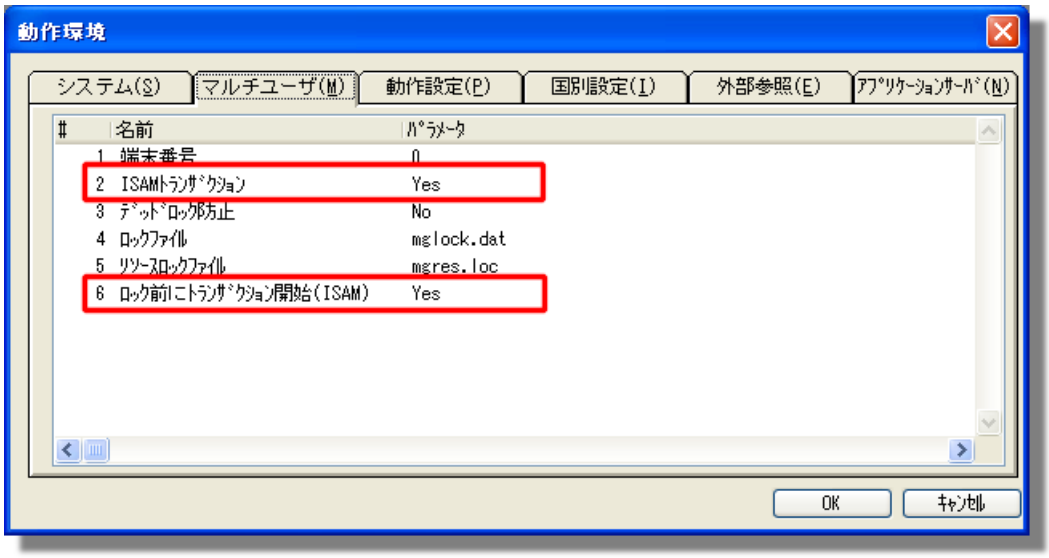
エンジンは未処理のイベントを受け取ることができ、それに対応した処理を実行させることができます。イベントのチェック間隔は、以下の３つの設定内容にもとづいて決定されます。

1. **イベント可**特性（**タスク特性→動作タブ**）：この設定が ***Yes*** または、**True** と評価される式が設定されている場合、エンジンはイベントをチェックします。これ以外はチェックされません。**イベント**には、キー操作も含まれます。このため、***No*** が設定された場合、、ユーザが **Esc** を押下してもバッチタスクをキャンセルすることができなくなります。
2. **バッチイベント間隔**（**オプション→動作環境→システムタブ**）：これはアプリケーション内のすべてのバッチタスクに影響しますが、必要に応じて、**INIPut()** 関数を使用することで実行中に変更することができます。
   - ***0***：イベントのチェックは行われません。
   - ***N***：N ミリ秒ごとにチェックされます。
3. **レコードイベント間隔**特性（**タスク特性→動作タブ**）：この設定は、現在のタスクでのみ有効で、エンジンに対し処理レコード数にもとづいたイベントのチェックを指示します。
   - ***0***：レコード件数にもとづいたチェックは行われません。
   - ***N***：N レコードごとにチェックされます。

これらの設定は、組み合わせて使用することができます。例えば以下のような設定が可能です。

- **イベント可**が ***No*** に設定した場合、他の設定内容に関わらずイベントのチェックは行われません。
- **バッチイベント間隔**を ***0*** に設定し、**レコードイベント間隔**を ***0*** に設定した場合、**イベント可**の指定内容に関わらずイベントはチェックされません。
- **バッチイベント間隔**を ***300*** に設定し、**レコードイベント間隔**を ***20*** に設定した場合、エンジンは 300 ミリ秒ごとおよび 20 レコードとにイベントをチェックします。

ISAM ファイルのトランザクションを組み込むには



ISAM 系の DBMS を使用している場合、Magic は、SQL 系の DBMS と同じようにトランザクションをサポートします。**タスク特性**の**トランザクションモード**特性や**トランザクション開始**特性（**データ**タブ）を設定することで、ISAM 系 DBMS に適切なトランザクション要求が送られます。

しかし、アプリケーション全体に対して ISAM トランザクションの有効／無効を切り替えることもできます。つまり、いくつかのタスクでトランザクションを有効にすることができますが、アプリケーション全体で無効にすることで、タスクのトランザクション設定は無視されます。

ISAM トランザクションを設定する

1. グローバルに ISAM トランザクションを設定するには、**動作環境ダイアログ**（**オプション→設定→動作環境**）の **ISAM トランザクション**（**マルチユーザ**タブ）に移動して ***Yes*** を設定します。
2. Magic.ini ファイルに設定する場合は、以下のようにします。

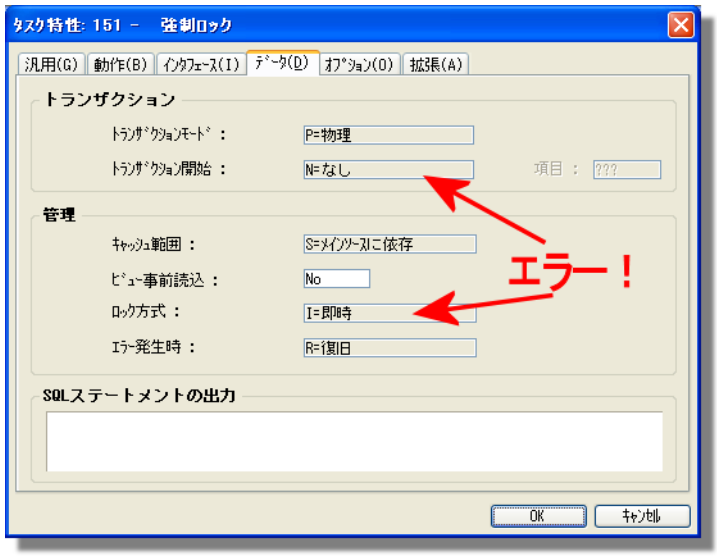
```
ISAMTransaction = Y
```

トランザクション内の ISAM の強制ロック

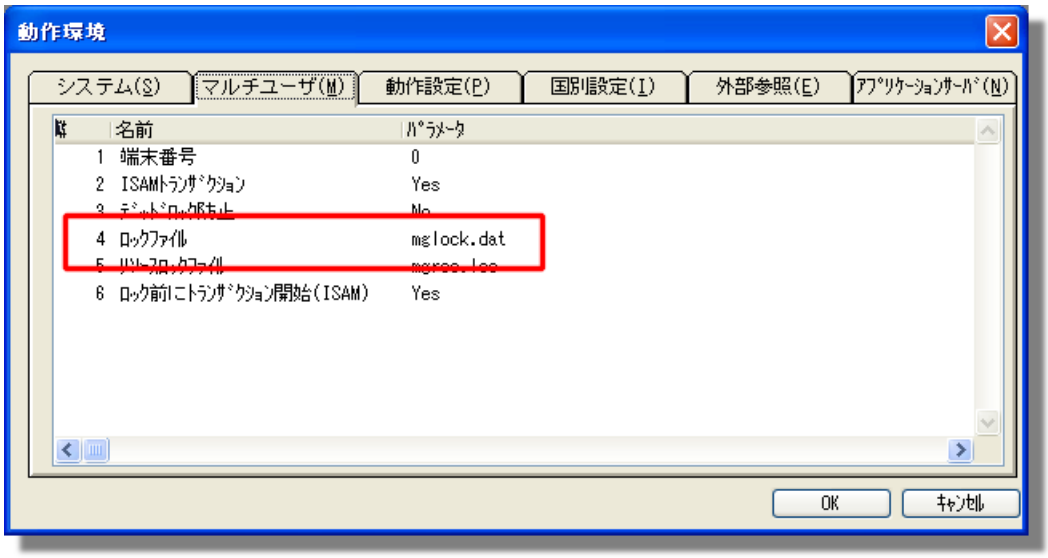
ISAM トランザクションが有効な場合、**トランザクション内の強制ロック**の設定を利用することができます。

もし**トランザクション内の強制ロック**が ***Yes*** に設定された場合、**ロック方式**特性が ***N= なし*** と設定されていなければ、**トランザクション開始**特性も ***N= なし*** に設定しなければなりません。すなわち、アクティブなトランザクションがない限り、レコードをロックすることができません。

ロック方式特性を設定した場合、トランザクションは開始されず、プログラムの**構文チェック**を実行するエラーが表示されます。



Magic のロックファイルの位置を指定するには



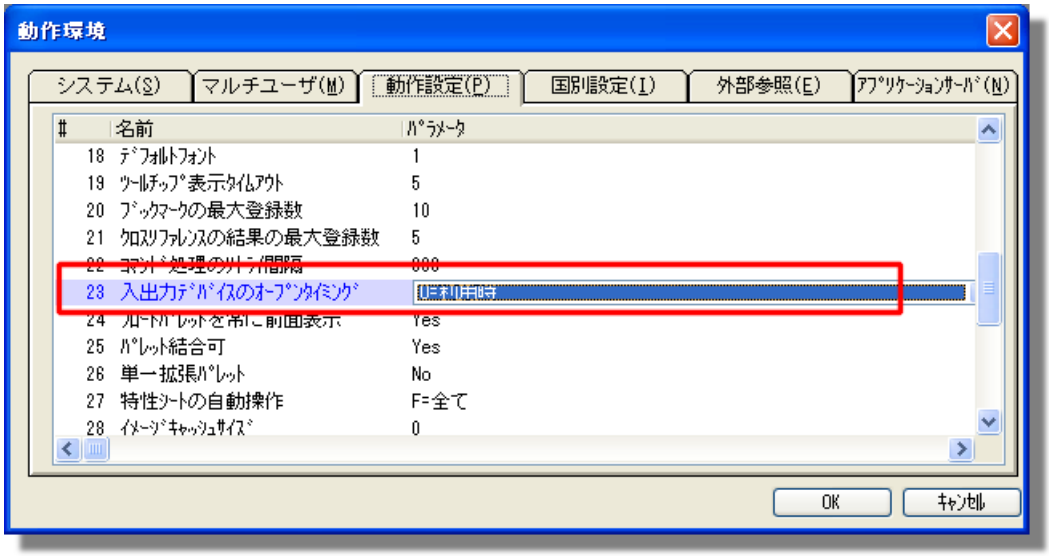
Magic がレコードをロックする場合、ロックファイルを使用してロック状況を常に監視しています。

ロックファイルの名前は、Magic.ini で指定されます。ファイルを直接アクセスするか、**動作環境**ダイアログ（**オプション→設定→動作環境**）の**ロックファイル**（**マルチユーザ**タブ）で変更することができます。

ここにはロックファイルのパスは指定しません。ロックファイルは、ロックされているファイルのディレクトリに作成されます。その位置は、データベーステーブルの位置カラムや**データ**リポジトリ内のデータソース名カラムで指定されます。

参照： 第 18 章：「Magic でデータベーステーブルを作成するには」（395 ページ）

入出力ファイルが実際に使用されるまで作成されないようにするには

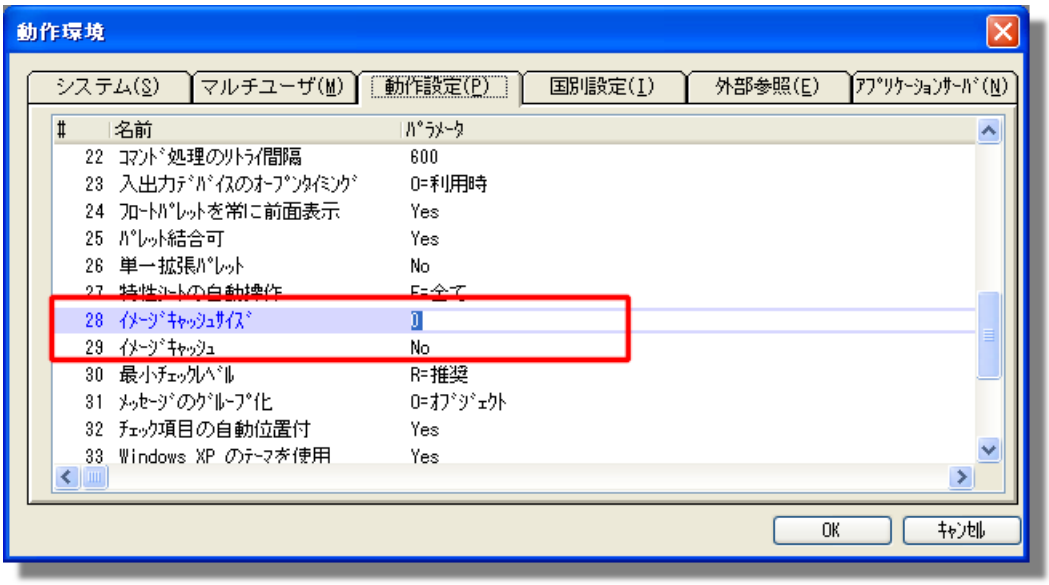


入出力ファイルがオープンされるとすぐに、新しいファイルまたはプリントジョブが作成されます。しかし、タスクが実際に処理を開始する前にファイルがオープンされるため、何も出力していないタスクが、空の入出力ファイルを作成してしまうことを意味しています。これによって白紙が印刷されてしまうため、指定した帳票を出力したい場合に不都合が生じてしまいます。

このような問題を防止するには、**動作環境**ダイアログ（**オプション→設定→動作環境**）の**入出力デバイスのオープンタイミング**（**動作設定**タブ）を使用します。このパラメータを ***U= 利用時***に設定した場合、出力処理が実行されるまで、入出力ファイルはオープンされません。すなわち、**フォーム出力**処理コマンドが実行された時点でオープンされます。***I= 即時***に設定された場合、入出力ファイルは、定義されたタスクが起動された時点で作成されます。

ヒント : *帳票ヘッダの出力処理の定義場所によって帳票の出力処理を最適化することができます。これらの処理コマンドが**タスク前**にある場合、レコードが処理される前に、フォームが出力され、白紙の帳票を出力させる可能性があります。しかし、この処理を**グループ前**に定義したり、ページヘッダとして自動的に出力させるようにした場合、実際に出力するレコードが存在するまで出力されなくなります。*

実行時のイメージ処理を最適化するには



たくさんのイメージファイルを処理すると処理速度を低下させることになります。このような問題を回避する 1 つの方法は、キャッシュを利用することです。同じイメージが何度もアクセスされる場合、メモリキャッシュから使用させることができます。

イメージキャッシュの動作は、２つの動作環境設定（**イメージキャッシュサイズ**と**イメージキャッシュ**）によって制御します。

イメージキャッシュサイズ

オプション→動作環境→動作設定→イメージキャッシュサイズ

この設定は、表示するイメージをキャッシュするために使用する最大メモリサイズ（KByte）を指定します。***0*** が指定された場合、キャッシュサイズの制限がないことを意味しています。キャッシュされたイメージサイズの合計が指定されたサイズを超えた場合、最も使用頻度の少ないイメージがキャッシュから削除されます。

イメージキャッシュ

オプション→動作環境→動作設定→イメージキャッシュ

この設定を ***Yes*** にした場合、イメージが最後に使用されてから変更されたかどうかをチェックし、変更された内容を反映させることができます。この機能は、注意して使用する必要があります。

この設定を ***No*** にするとタイムスタンプをディスクから取得する処理を行わないため処理が早くなります。しかし、処理中にイメージの内容が更新されるのであれば、***Yes*** に設定する必要があります。これによって古いイメージが表示され続けることはなくなります。

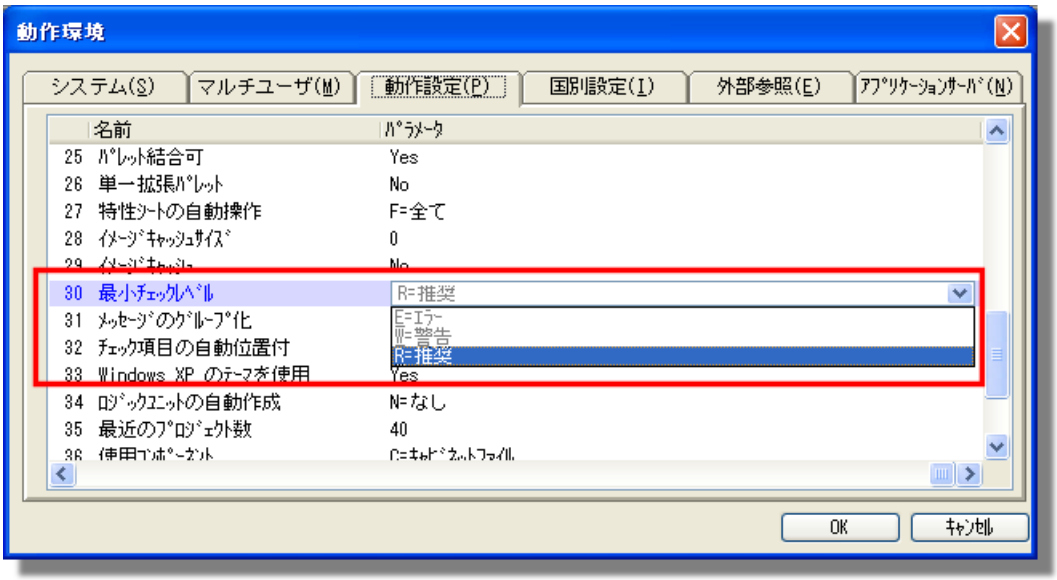
表示されたチェックメッセージを制御するには

Magic Studio には**構文チェック**ユーティリティが備えられています。もしオブジェクトに問題がある場合は、このユーティリティを実行することで確認することができます。オブジェクト上にカーソルを置き、***F8***（**オプション→構文チェック**）を押下することでこのオブジェクトに対して**構文チェック**を行うことができます。***Alt+F8***（**オプション→カーソル以降をチェック**）を押下することで現在のカーソル位置移行の全てのオブジェクトをチェックすることもできます。

エラーが存在していれば、**チェック結果**ペインと呼ばれる個別のウィンドウが表示されます。表示されるメッセージの内容にもとづいて対応することができます。

最小チェックレベル

オプション→動作環境→動作設定→最小チェックレベル



この設定は、チェックメッセージ内で表示させたい内容のレベルを指定します。

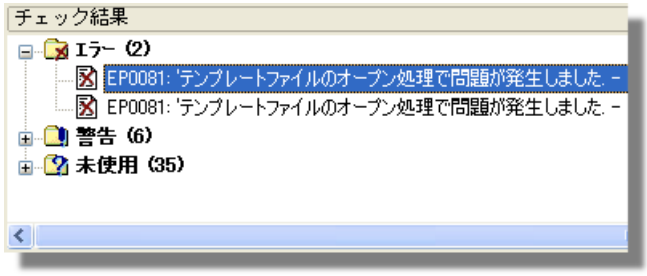
- ***R= 推奨*** : 推奨、警告、およびエラーを表示します。
- ***W= 警告*** : 警告とエラーを表示します。
- ***E= エラー*** : エラーだけを表示します。

メッセージのグループ化

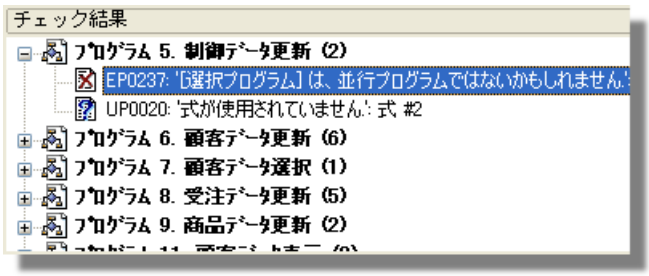
オプション→動作環境→動作設定→メッセージのグループ化

この設定は、チェックメッセージをどのようにグループ化させるかを指定します。

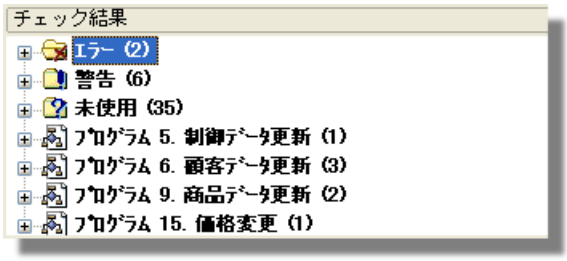
- ***T= タイプ***: リストの最上位に最も深刻な問 題を表示し、内容の深刻さの順にメッ セージを表示します。



- ***O= オブジェクト***：チェックされるオブジェクトの順にメッセージを表示します。



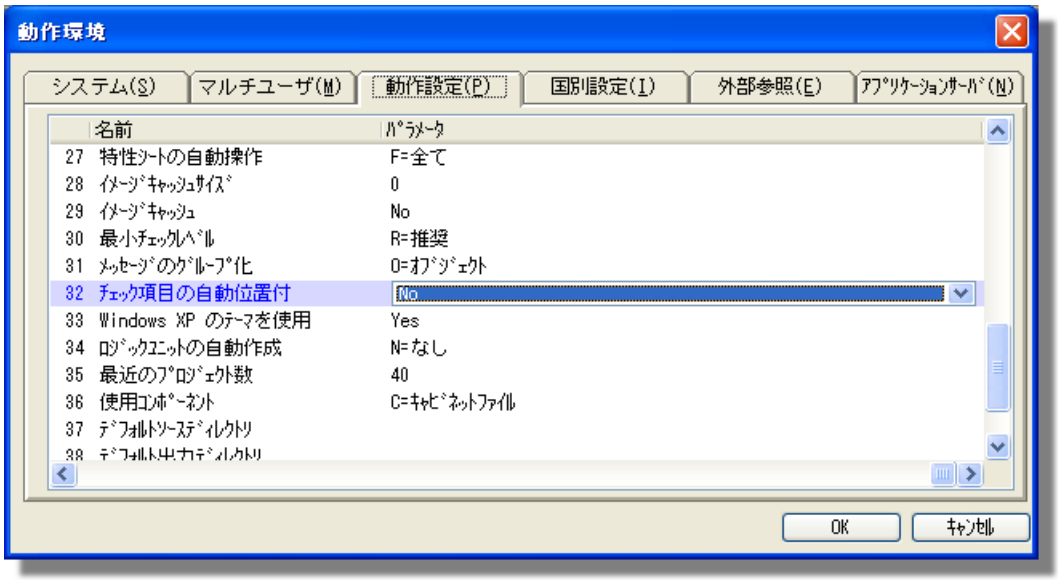
- ***B= オブジェクトとタイプ***：タイプとオブジェクトの両方の内容をもとに順番に表示します。



これらのリストに表示されるメッセージ上で**ズーム**（***F5*** または、**ダブルクリック**）することで、該当するオブジェクトにカーソルが移動します。これによってエラー原因を簡単に見つけることができます。

チェック項目の自動位置付

オプション→動作環境→動作設定→チェック項目の自動位置付



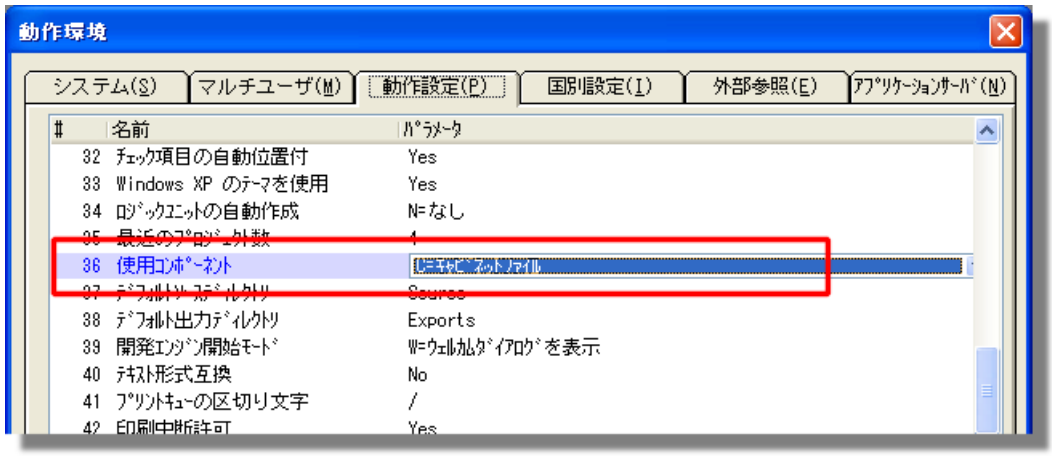
この設定を ***Yes*** にすると、**構文チェック**ユーティリティは、自動的にチェックリスト内の最初の項目に移動し、オブジェクトを開き、エラー箇所にカーソルを移動します。

チェックメッセージの表示

オプション→動作環境→システム→チェックメッセージの表示

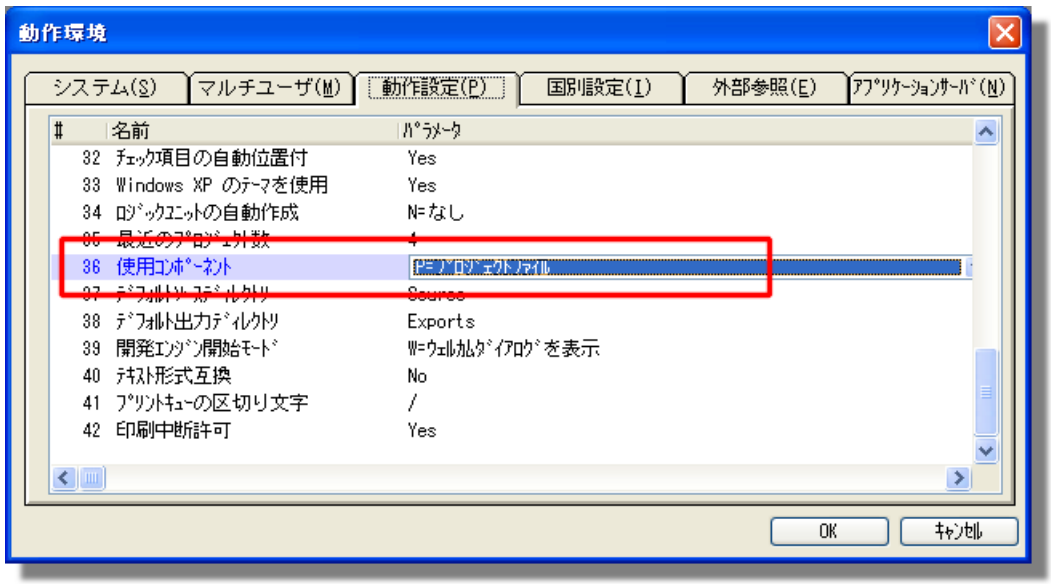
この設定を ***Yes*** にすると**チェックメッセージ**テーブルにアクセスできます。

プロジェクトファイルをコンポーネントとしてアプリケーションを開発するには

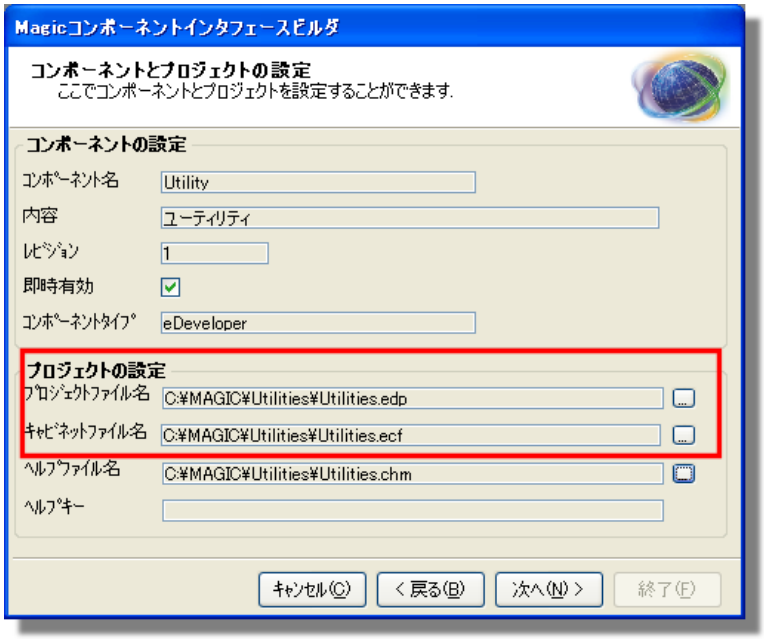


独自のコンポーネントを使用してプロジェクトを開発する場合、プロジェクト（***.edp***）ファイルまたはキャビネット（***.edf***）ファイルのどちらかをコンポーネントとして使用するように指定できます。コンポーネントを簡単に変更することができるため、プロジェクトファイルを使用した方が便利な場合があります。

プロジェクトファイルをコンポーネントとして使用する

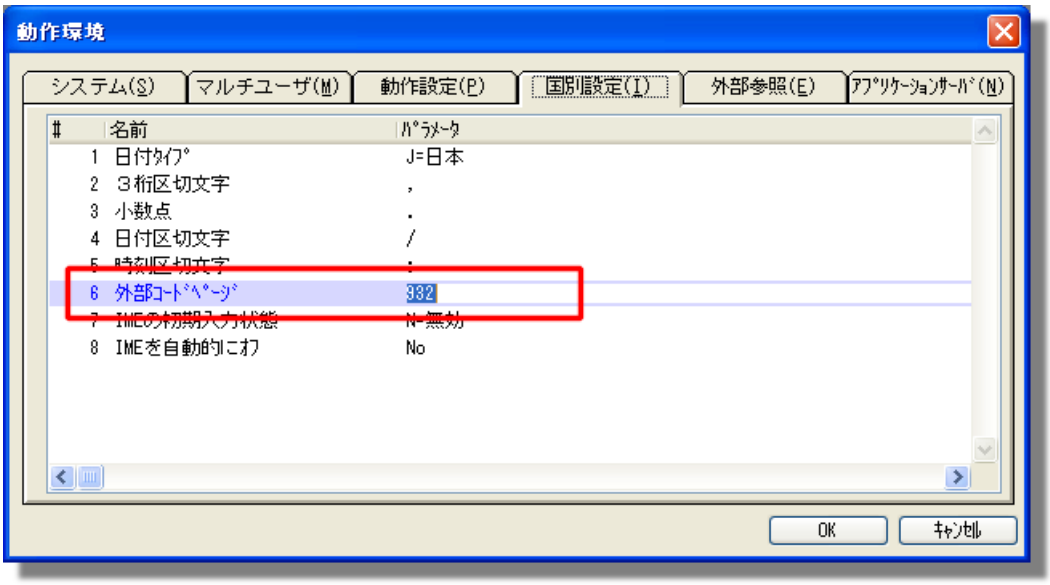


1. **動作環境**ダイアログ（**オプション→設定→動作環境**）の**使用コンポーネント**（**動作設定**タブ）を ***P= プロジェクト*** に設定します。



2. この設定より、コンポーネントの作成時に指定されたプロジェクトファイルが開きます。

ANSI を Unicode に変換するためのコードページを指定するには



Unicode から ANSI（またはその逆も）へのコード変換は、コードページを使用して行われます。デフォルトでは、OS のコードページを使用します。使用するコードページは、**動作環境**ダイアログ（**オプション→設定→動作環境**）の**外部コードページ**（**国別設定**タブ）で設定することができます。

CodePage() 関数を使用することで、現在使用されているコードページを変更することができます。

第 18 章 : データソースの定義

Magic でデータベーステーブルを作成するには

Magic は、簡単にデータベーステーブルを作成することができます。必要な作業はテーブルのカラムを指定するだけで、Magic は DBMS 内にテーブルを作成する処理を実行します。さらに、ISAM ファイルか SQL テーブル、またはメモリテーブルであるかどうかに関わらず、どのような種類のデータソースを作成する場合でも、操作内容は基本的に同じです。XML ファイルを定義する場合でも、基本的に同じ形式で行うことができます。

Magic で SQL データソースを使用して作業する場合、基本的に以下の 2 つのケースが考えられます。

1. テーブルは存在せず、Magic で新規作成する。
2. テーブルは既に存在しており、Magic でアクセスできるようにする。

2 番目のケースは、「既存のデータベーステーブルにアクセスするには」（402 ページ）で説明しています。ここでは、1 番目のケースについて説明します。

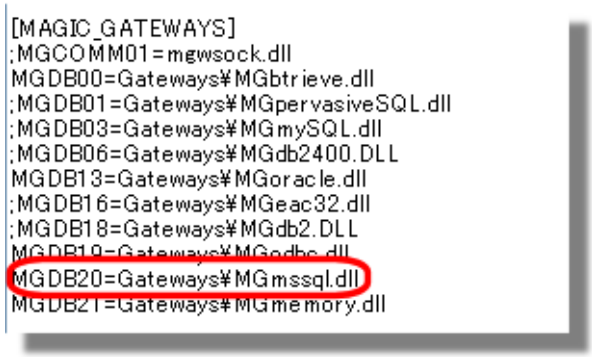
ここには、基本的な操作の概要が記述されています。以降に各詳細の説明があります。

- ゲートウェイと DBMS がロードされることを確認します。
- **データベース定義の設定**：DBMS に対応したゲートウェイをロードし、データベース定義を設定します。
- **テーブルの作成**：データソースリポジトリでテーブルを定義します。
- **カラムの作成**：テーブルに必要なカラムを定義します。
- **インデックスの作成**：テーブルに必要なインデックスを定義します。
- **テーブルの構文チェック**：テーブルの定義内容に問題がないかどうかをチェックするために、**構文チェック**ユーティリティを実行します。
- **レコードを作成してテーブルを確認する**：作成されたテーブルが動作することを確認するために、簡単な照会プログラム（***Ctrl+G***）を作成します。

データベース定義が正しく設定された場合、Magic は、自動的に SQL テーブル定義を作成する処理を実行します。

注： SQL テーブルを使用したこれらの処理は、メモリテーブルや ISAM ファイルでテーブルを作成する場合とほとんど同じになります。違いは、データベース定義が必要なことと、ISAM ファイルやメモリテーブルには命名時の制約がないということです。

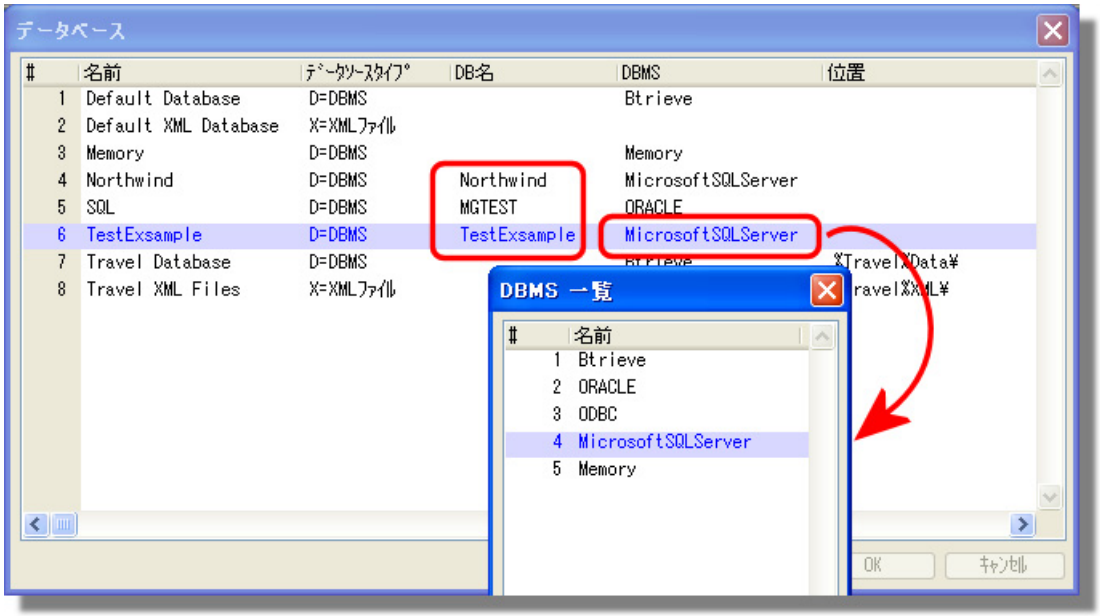
1. ゲートウェイと DBMS がロードされることを確認する



1. データベースを設定する前に、ゲートウェイがロードされていることを確認する必要があります。この設定は、Magic.ini で行います。Magic のインストール処理の際に、選択されたゲートウェイの内容にもとづいて Magic.ini の MAGIC_GATEWAY セクションが自動的に設定されます。
2. 新しい DBMS タイプにアクセスする場合は、対応するゲートウェイがインストールされていることを確認するために Magic.ini とインストールディレクトリをチェックしてください。
3. 当然ですが、使用する DBMS が PC にインストールされていなければなりません。この例では、Pervasive SQL の ISAM データベースと MS-SQL Server データベースにアクセスしているため、これらの製品が両方とも PC にインストールされていなければなりません。
4. ゲートウェイを追加する必要がある場合、次の手順に移る前に追加して、Magic を再起動する必要があります。

2. データベース定義を設定する

データベース定義は、データソース指定がどのように使用されるか決める上での鍵となります。実際、１つのデータソース設定はいくつかのデータ定義を繰り返し使用することができ、これにより同じデータソースが、割り当てるデータ定義によってメモリテーブルとなったり、SQL テーブルや ISAM テーブルとなったりすることができます。



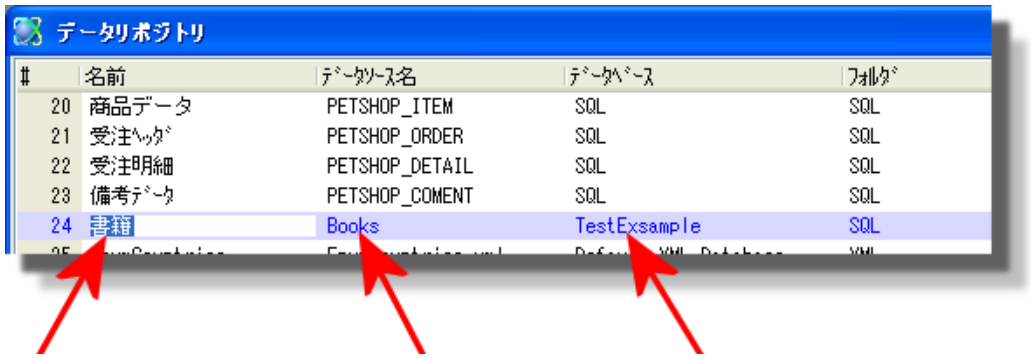
1. プロジェクトが開いていない状態で、**データベース**テーブル（**オプション→設定→データベース**）を開きます。利用可能な DBMS のみが **DBMS 一覧**に表示されます。ここから使用する DBMS を選択します。この例では、Microsoft SQL-Server と Oracle を使用しています。

2. 次に、データベース名を入力する必要があります。ここには DBMS 内のデータベース名を入力します。この例では、２つのデータベース（Northwind と TestExsample）を使用しています。
3. Magic がデータベース内にテーブルを作成する前に、DBMS マネジャー（または、それに類するもの）を使用してデータベースを作成しておく必要があります。
4. 必要に応じて ***Alt+Enter***（**編集→特性**）を押下してユーザ ID とパスワードを設定します。また、Magic でデータベース内にテーブルを作成する場合は、**データベース特性**の**開発でテーブル定義を変更**（**オプション**タブ）を ***Yes*** に設定する必要があります。

3. テーブルを作成する

次の手順は、実際のテーブル定義を作成することです。この作業を行う前に、使用する DBMS の制約条件を確認しておく必要があります。Magic 自体はあまり制約条件がありません、このためメモリテーブルを使用する際に命名規約などの制約条件を考慮する必要がありません。しかし、SQL 系の DBMS では使用可能な文字に関する制約があります。

1. データリポジトリ（*Shift ＋ F2*）に移動し、一行追加（*F4*）します。以下のようにテーブルを作成します。



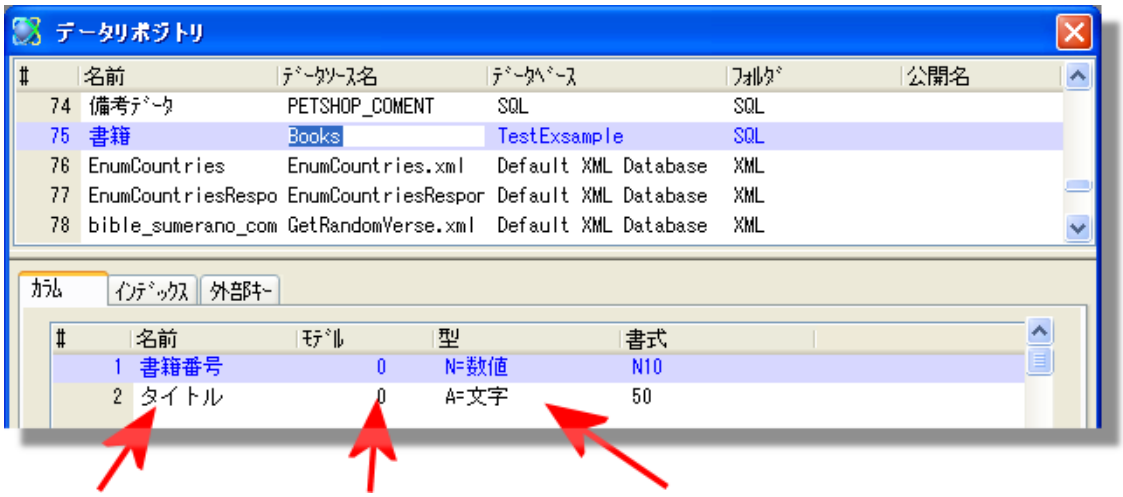
任意の**名前**を入力します。

データソース名のデフォルトは、名前カラムに入力した名前になります。この内容は変更できます。これは DBMS データベースの実際のテーブル名になります。

ここから**ズーム**して**データベース一覧**から**データベース**を選択します。

4. カラムを作成する

カラムを作成するには、下に表示されているタブ領域を**クリック**します。各カラム毎に、***F4*** を押下して行を追加し、以下のようにデータを入力します。各行には、そのカラムの基本的な特性が表示されています。より多くの特性値を設定する場合は、**特性シート**（***Alt+Enter***）を表示します。カラム定義の設定は、プログラム内で項目を定義する場合と同じような操作を行います。



各カラムの**名前**を入力します。入力された名前は、SQL テーブル内ではそのまま使用されませんが、なるべく同じ名前を指定する習慣にしてください。これは、SQL の規則に従うことを意味しています。

基本的な項目定義を行うようにしたい場合は、**モデル**を選択します。モデルの使用は、必須ではありませんが、保守作業が容易になります。

モデルを使用していない場合、カラムの**型**と**書式**を設定します。

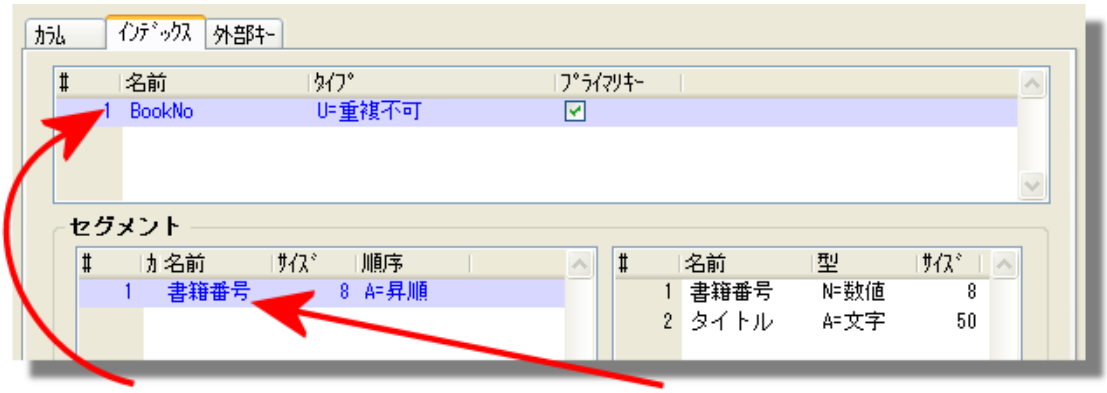
次に、**特性シート**（***Alt+Enter***）でこのカラムのより詳細な特性を設定する必要があります。ほとんどのカラムは、基本的な設定だけで使用できます。**項目モデル**に特性を設定することですべて終了する場合もあります。しかし、ここには、データを SQL データベースにどのように格納するかを正確に指定することのできるセクションがあります。

ここで重要な点は、データの各ビットに対する２つの内容があることです。Magic のデータ型（文字型、数値型、日付型、時刻型など）は単純で一般的なものです。しかし、DBMS のデータ定義（ZStirng、LStiring、Integer、Float など）や実際の SQL 定義（Char、Integer）に対する設定もあります。Magic は、デフォルトで使用頻度の高い定義内容を選択しますが、必要に応じて変更することができます。この設定内容の詳細は、「Magic の項目とデータベースカラム間の割付を定義するには」（408 ページ）を参照してください。

ヒント :主要なデータ定義をカプセル化するためにモデルを使用することを推奨します。これによって SQL でのデフォルトを１カ所で定義できるようになります。

5. インデックスを作成する

次に、インデックスを作成する必要があります。画面下の領域のインデックスタブをクリックして作成します。



1. 各インデックスに対して、*F4* を押下して行を追加します。インデックスの名前を入力し、ユニークかどうかや、プライマリキーかどうかを指定します。特性シート（*Alt+Enter*）を開くことで詳細の特性を設定することができます。

2. 次に、各インデックスに対して、*F4* を押下してセグメントを作成します。ズームして右側に表示されているカラム一覧からセグメントの一部になるカラムを選択します。また、セグメントの並び順（昇順か降順）を指定します。ここから特性シートを表示して詳細な特性を設定することができます。

6. テーブルの構文チェックを実行する

カラムとインデックスの入力後、エラーがないかどうかを確認するために構文チェックユーティリティ（*F8* または、オプション→構文チェック）を使用します。

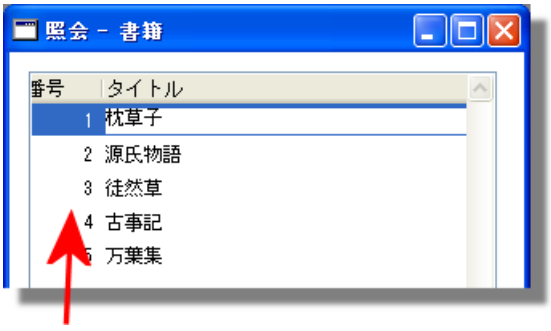
1. チェックしたいテーブル上にカーソルを置きます。
2. *F8* を押下します。

エラーがあればチェックペインに表示されます。

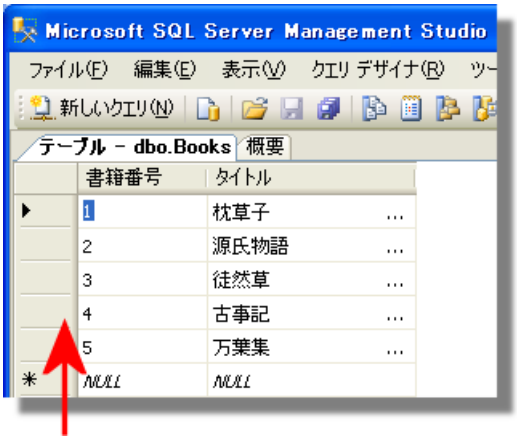
7. レコードを作成してテーブルを確認する

最後に、レコードが作成できることを確認します。簡単に確認する方法は、APG（Auto Program Generator）を実行することです。詳しいやり方は、「テーブル内のデータを暗号化するには」（406 ページ）を参照してください。

照会プログラムが正常に実行され、2、3 のレコードの追加ができれば、テーブルが正常に作成されたことをになります。DBMS ツールを使用することで、SQL DBMS に作成されたテーブルを参照することができます。



照会プログラムで作成されたテーブルが表示されます。



そして、MS-SQL Server には同じテーブルが表示されます。

DBMS に作成されたテーブルが既にある場合、DBMS 内の定義内容と Magic の定義内容を同期させる必要があります。一カ所（またはそれ以外）にある定義内容を常に維持することがとても良い方法です。Magic で定義を変更した場合、**データベース特性**の**開発モードでテーブルを変更する**（オプションタブ）特性を *Yes* に設定することで、DBMS 内のテーブルの再構築が自動的に行われます。

参照： 「データベーステーブルを参照するには」（409 ページ）

既存のデータベーステーブルにアクセスするには

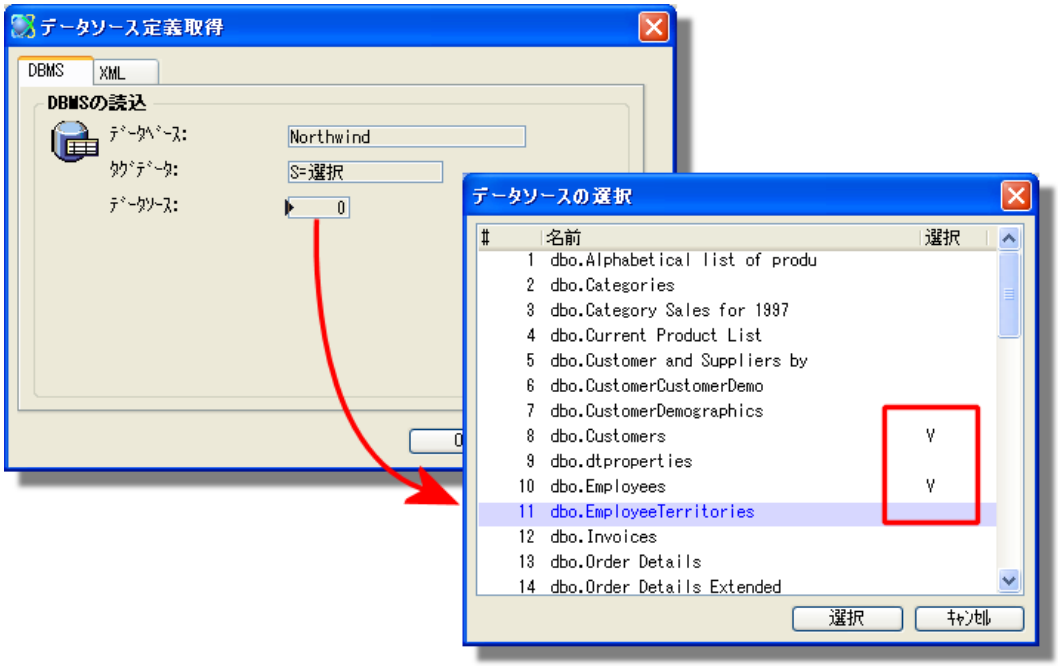
データベーステーブルがすでに DBMS 内に存在している場合、簡単に Magic アプリケーションでアクセスすることができます。

必要条件： 以下の環境設定が必要です。

- DBMS がインストールとされ、動作していること。
- インストールされたその DBMS に対応した Magic のゲートウェイがインストールされていること。
- DBMS に対応したデータベース定義が行われていること。

詳細は、「1. ゲートウェイと DBMS がロードされることを確認する」（396 ページ）と「2. データベース定義を設定する」（396 ページ）を参照してください。

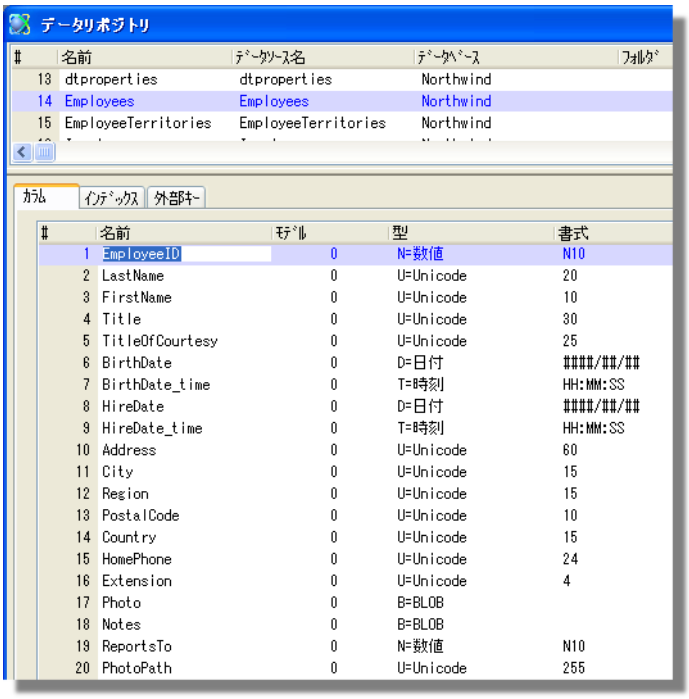
既存のデータベーステーブルにアクセスする



1. オプション→定義取得（*F9*）を選択します。定義取得ダイアログボックスが開きます。
2. データベースからズームして、データベースを選択します。
3. タグテーブルから、*S= 選択*または *A= 全部*のどちらかを選択します。
4. *選択*を選択すると、データソースからズームしてアクセスしたいデータソースを選択します。選択したいデータソースにカーソルを移動し、*Space* を押下すると選択状態になります。
5. データソースを選択したら、選択ボタンをクリックします。定義取得ダイアログに戻り、選択されたデータソースの数が表示されます。

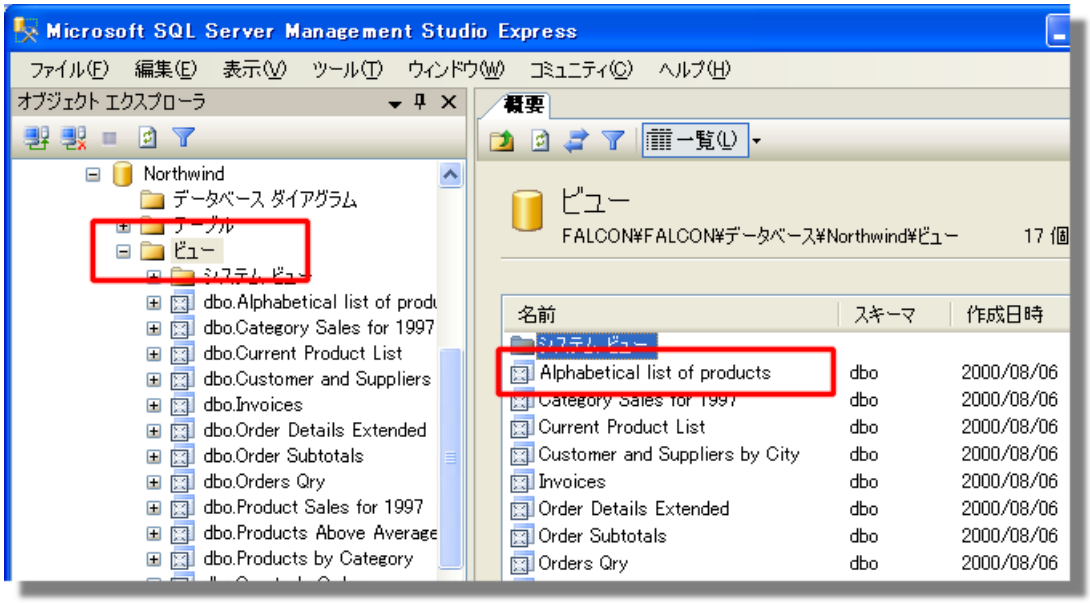
6. OK をクリックします。ダイアログが閉じ、データリポジトリに選択されたデータソースが追加されます。

これで、Magic アプリケーションにデータソース定義が追加されました。

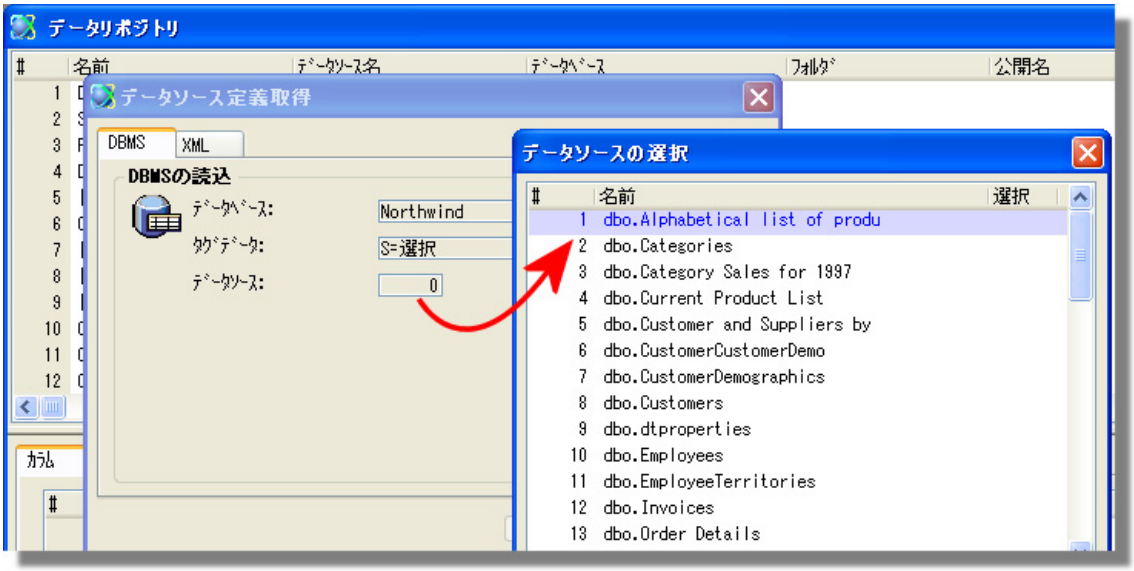


注： テーブルに日付と時刻型のカラムが存在する場合、「SQL の Date-Time のカラムを Magic の日付と時刻の項目の組み合わせに変換しますか？」というメッセージが表示されます。Magic での日付と時刻の組み合わせの処理に関する詳細説明は、第 12 章：「日付と時刻を組み合わせたデータを加算するには」（238 ページ）や第 12 章：「２つの日付 / 時刻データの差分を計算するには」（239 ページ）を参照してください。

データベースのビューからデータを取得するには



データベースの中には、データベーステーブル（表）とデータベースビューがあります。ビューは一種の仮想テーブルです。ビューにはデータは格納されません。むしろ、いくつかの標準にもとづいて 1 つ以上のテーブルからデータを抽出する SELECT ステートメントが格納されています。



しかし、データベーステーブルを扱うように、Magic はデータベースビューを扱います。定義取得を実行すると、データベースビューがテーブルと一緒に一覧表示されます。データベースビューを選択すると、他の SQL テーブルのように動作する Magic のテーブルを作成することができます。データベースビューの名前はブラケットで囲まれて表示されるため、それがデータベースビューかどうかを確認することができます。

定義取得の使用に関する詳細については、「既存のデータベーステーブルにアクセスするには」（402 ページ）を参照してください。

データベーステーブル定義を変更するには

Magic で定義されたデータベーステーブルがある場合、必要に応じてこれを変更することができます。どのように変更するかは、テーブルが別のアプリケーションによって使用されているかどうかや、開発環境で利用できるデータであるかどうかに依存します。ここでは、以下のケースについて説明します。

- **Magic でのみ使用**：データは Magic でのみ定義されます。
- **外部アプリケーションと共有**：データは DBMS 内やサードパーティ製品によって定義されます。例えば、経理用のアプリケーションや購入したデータベースからデータにアクセスする場合などが考えられます。

Magic 内でのみ定義されているデータベーステーブルを変更する

データベーステーブルが Magic 内でのみ定義されている場合は、簡単に変更することができます。プログラムからの定義は、同期され可能な限り自動的に維持されます。

- カラムを追加すると、リポジトリ内のプログラムは自動的に更新されます。
- カラムを削除する場合は、どのプログラムもそのカラムにアクセスしていないことを確認する必要があります。使用されるかどうかを確認するには、カラム上で**クロスリファレンス**ユーティリティ（***Ctrl+F***）を使用します。使用していなければ、それを削除します。
- カラムを変更すると、参照情報も可能な限り変更されます。フォーム上に配置されている場合は、表示上の特性は変更されません。また、項目の型を変更した場合（例：数値型から文字型など）、データを更新するための処理コマンドの定義内容も変更する必要があります。

さて、既存のデータはどうなるでしょうか？**データベース特性**の**開発モードでのテーブル変換**特性を ***Yes*** に設定した場合、Magic は自動的にデータの構成を変更し、万一に備えてバックアップコピーの作成を実行することもできます。

しかしこのような処理は、データを作成する運用環境で行おうとするものではありません。作成されたアプリケーションを提供する場合、既存データの書式を再定義するためのユーティリティを提供するべきです。簡単に行う方法として、**データ**リポジトリ内に新旧のテーブルを両方定義し、データを旧テーブルから新テーブルに移動するための Magic プログラムを作成することがあります。

外部テーブルと共有している Magic のデータソースを変更する

さて、別の場所で定義されるデータソースを Magic で利用する場合は、異なる問題が発生します。この場合、別のソースから定義を取得する必要があります。**開発モードでのテーブル変換**を ***No*** に設定しておく必要があります。この場合、Magic は既存のデータの変更を行いません。

既存のデータソース定義上で変更内容を入力することで、手動で内容を変更することができます。

または、**定義取得**ユーティリティ（***F9***）を使用して、自動的に定義内容を取得することで変更内容を反映できます。ここには、定義取得を使用した変更方法についての説明があります。

1. テーブル定義を取得するために、**定義取得**ユーティリティを使用します。詳細は、「既存のデータベーステーブルにアクセスする」（402 ページ）を参照してください。
2. 定義内容を印刷するか、そのスクリーンショットを取ります。これによって次のステップで、内容を確認することができます。
3. 既存のテーブル定義を編集します。カラムが追加された場合、空白のカラムを既存の定義に追加します。カラムが削除された場合、定義から削除します。この操作は、カラムの位置や大まかな内容を外部テーブルに合わせるためです。例えば、顧客名が新しい定義では 5 番目の行にあり、20 文字の長さで、6 行目に 30 文字のカラムが追加された場合、Dummy というカラムを追加します。文字の長さは考慮する必要はありません。
4. インデックスに対しても同じようにします。
5. ２つの定義内容が位置的に整合性が取れていれば、古テーブル定義を新しい定義で上書きします。

Magic は内部 ID を使用して位置でデータベースカラムを参照しているため、このような処理によって変更ができます。通常は、既存のテーブルが変更されることはあまりありません、また、項目を追加する処理には時間もかかりません。

テーブル内のデータを暗号化するには

パスワードを使用してアクセスを制限した場合でも、低レベルのツールを使用することで、ディスク内容を参照することができます。このため、非常に重要な内容が含まれているテーブルを暗号化する必要があります。

Magic では簡単に暗号化することができます。以下の手順で行います。

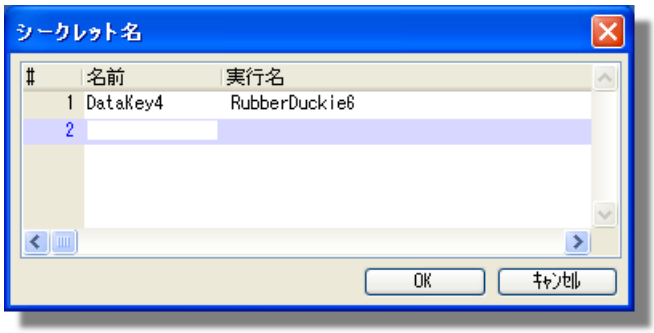
データベーステーブルを暗号化する

1. **データ**リポジトリ内の暗号化したいテーブルに移動します。
2. **データソース**特性（*Alt+Enter* または、**編集→特性**）を開きます。
3. **アクセスキー**特性に、任意の文字列を入力します。シークレット名を使用することを推奨します（「シークレット名を使用するには」（407 ページ）を参照してください）。
4. **テーブルの暗号化**特性を *Yes* に設定します。

これで、テーブル内のデータは暗号化されます。同じキーを持っているユーザのみデータを参照することができます。

注： すべての DBMS がデータの暗号化をサポートしている訳ではありません。使用している DBMS がサポートしていない場合、**アクセスキー**と**テーブルの暗号化**の各特性は灰色で表示されます。

シークレット名を使用するには

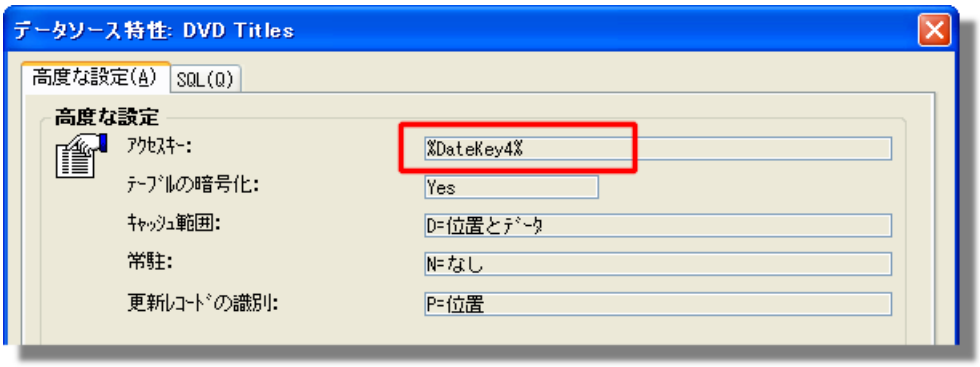


シークレット名を設定する

1. SUPERVISOR として Magic にログオンします。
2. シークレット名テーブル（オプション→設定→シークレット名）を開きます。
3. シークレット名を入力します。名前カラムに入力された文字列は、スーパーバイザ以外ではアクセスできないことを除けば、論理名のように動作します。

シークレット名は、ユーザ ID やパスワードを暗号化して格納することができます。

シークレット名を使用する



シークレット名を使用したい場合、論理名と同じ指定方法で入力します。この内容は実行時に変換されます。

このように設定されている場合、アプリケーションのインストール時にシークレット名の情報（セキュリティファイル）も一緒に提供する必要があります。シークレット名を使用せず実行名を指定することもできますが、実行環境と合っていない可能性が発生します。

Magic の項目とデータベースカラム間の割付を定義するには

Magic は、組み込まれた詳細内容へのアクセスを開発者に行わせないようにすることで、膨大な種類のデータタイプを容易に処理できるようにしています。Magic を使用している場合、数種類のデータ型（文字型、日付型、時刻型、論理型など）を扱いさえすればよく、これらの項目が実際にメモリやデータベースにどのように格納されるかを意識する必要はありません。

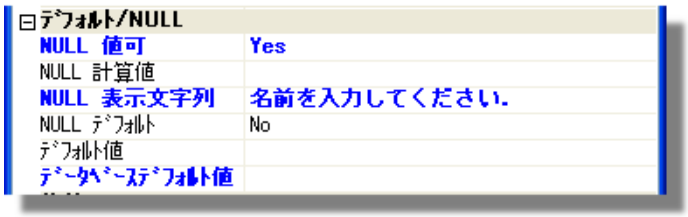
しかし、実際のデータベース記憶型式を制御することができます。これらの詳細は、データリポジトリの各カラムのカラム特性（*Alt+Enter*）で指定できます。

Magic の項目のデータ型に関する詳細は、**詳細**セクションで指定されています。この内容は、10 桁の数値型項目を定義するとよくわかります。この場合、*2,147,483,648* から *-2,147,483,648* までの値を格納することができます。

格納セクションでは、この項目が格納される実際のデータ形式を指定することができます。この例では、4 バイトの *Siged Integer* で格納されます。

デフォルト /NULL と **SQL** のセクションは、項目に組み込まれた内容の詳細な情報を設定することができます。この内容については、以下に概要がありますが、この上にカーソルを置き、*F1* を押下することで各特性の情報が確認できます。

デフォルト/NULL セクション



- **NULL 値可** : この特性が *No* と設定された場合、以下の 4 つの特性は灰色で表示され、この項目には反映されません。NULL は使用されなくなります。*Yes* に設定された場合、項目が最初に作成されると NULL（空白や 0 とは概念的に異なります）が設定されます。
- **NULL 計算値** : NULL が設定された場合に、実際に項目に含まれている値を指定します。この特性が指定されない場合、NULL の実際の値は Magic.ini ファイルの DBMS セクションで指定された値が設定されます。NULL 値は、通常は入力することのできない値です。しかし、プログラム内で **NULL()** や **ISNULL()** 関数を使用することができるので、実際の値を意識する必要はありません。
- **NULL 表示文字列** : NULL 値の場合にユーザに表示される値を指定します。例えば、NULL 値が 1901/01/01 などのような日付の場合に、「誕生日を入力してください。」というメッセージを表示させることができます。
- **NULL デフォルト** : この特性が *Yes* と設定された場合、項目のデフォルト値は NULL になります。*No* の場合、**デフォルト値**特性で指定された値がデフォルト値となります。
- **デフォルト値** : 項目が最初に作成された場合に、Magic が自動的に指定された値で項目を初期化します。手動で値を設定する必要はありません。ほとんどの項目は、*0* または*空白*がデフォルトとなります。しかし、特定の値で初期化したい場合は、ここに値を指定します。
- **データベースデフォルト値** :Magic はここで指定された文字列を CREATE TABLE ステートメントを使用して DBMS に送ります。この設定によって、DBMS でデフォルト値を設定することができます。例えば、MS-SQL Server のテーブルに defvalue というデータベースデフォルト値を作成する場合、Magic は以下のようにして作成します。

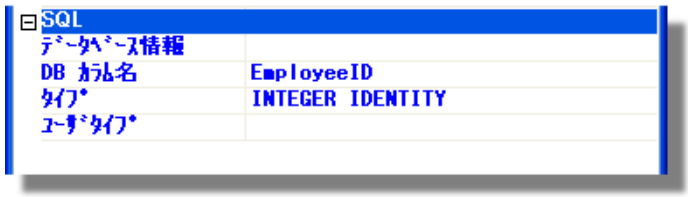
  ```
  CREATE TABLE owner1.table1 (Col1 CHAR(10) NOT NULL DEFAULT 'defvalue')
  ```

  Magic は何も追加することなく、この文字列を CREATE TABLE ステートメントに追加します。

格納セクション

- **文字セット**： 格納時の文字セットを *A=ANSI*、*O=OEM*、および *U=Unicode* から選択できます。
- **デフォルト記憶形式** : この特性を *Yes* に設定すると、DBMS によってデータ形式が決まり、**記憶形式**特性は無視されます。複数の DBMS に対して同じデータ定義を行いたい場合は、この特性を使用する方が便利です。
- **記憶形式**： ここには、実際のデータ型式が指定できます。ここから**ズーム**すると**記憶型式一覧**が表示され、選択することができます。SQL カラムに対しては、SQL セクションの**タイプ**特性でも指定することができます。
- **修正許可**： ユーザが実行時にこの項目にアクセスできるかどうかを指定します。
- **サイズ** : カラムに割り当てるバイト数を指定できます。
- **データベース定義** : *N= 標準*または、*S= 文字列*が指定できます。
- **更新形式** : この特性は、遅延トランザクションを使用する場合の SQL カラムにのみ有効です。差分と設定された場合、Magic はメモリ内の値との差分にもとづいて値を更新します。複数のユーザが同時に同じ項目の操作をしているような場合（例えば、現在の株の総数を更新するような場合）、この特性が重要になります。

SQL セクション



- **データベース情報** : 使用する DBMS に渡すデータベース固有の情報を指定することができます。
- **DB カラム名** : DBMS 内で定義されるカラム名を指定します。
- **タイプ** : 使用するデータタイプを指定します。通常は指定する必要がありません。Magic は、**記憶形式**特性で指定された値にもとづいたタイプを使用します。しかし、ここに直接指定することもできます。また、**定義取得**を使用して DBMS から定義をした場合、ここにデータのタイプが表示されます。
- **ユーザタイプ** : DBMS によっては、DBMS 内に独自の「ユーザタイプ」を定義することができます。ユーザタイプを定義した場合、ここでそのタイプを指定することができ、ユーザタイプは、テーブル作成時に Magic によって使用されます。

参照 : 「データベーステーブル定義を変更するには」（405 ページ）

複数の DBMS で動作するように設定するには

Magic でのハイレベルのデータ定義能力の一つは、複数のタイプのテーブルに対して同じデータソース定義を使用することができることです。例えば、MS-SQL Server テーブルによって動作し、さらに Oracle や DB2 UDB によって使用することのできる定義を持つことができます。

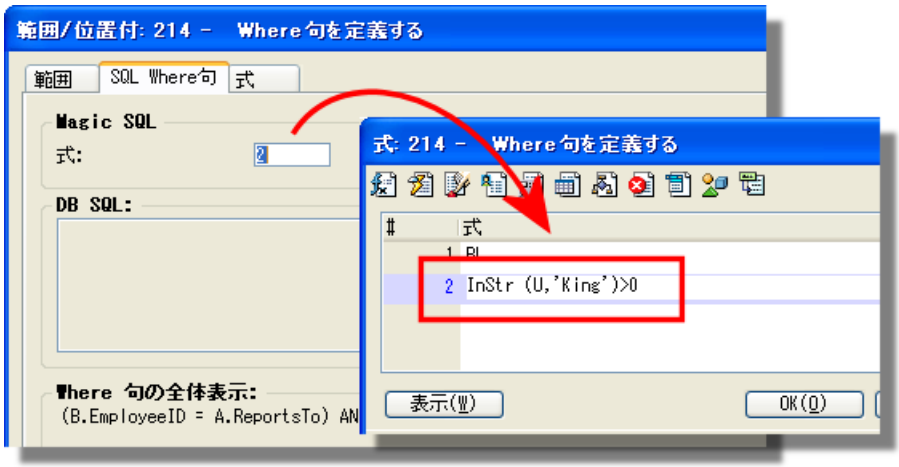
すべての DBMS は独自の処理を行うため、いくつかの制限があります。また、入力されるユーザ定義は各データベースに対して同じであることを知っておく必要があります。

しかし Magic は、より高いレベルでデータを定義したり、実行時に DBMS 固有の機能を設定することを可能にするためのオプションを提供しています。

移行可能な項目を定義する

カラム特性の**デフォルト記憶形式**を ***Yes*** に設定すると、記憶形式特性は無視され、使用する DBMS 固有のデータ形式が使用されます。例えば、数値型項目がある場合、ある DBMS は ***Integer*** と定義し、別の DBMS では ***NUMBER*** や ***PACKED DECIMAL*** と定義されるかもしれません。

移行可能な WHERE 句を定義する



WHERE 句の構文は、DBMS によって異なる場合があります。例えば、特定の文字列から文字列を取得する場合、DB2 UDB と Oracle は **Substr** を使用し、MS-SQL Server は **Substring** を使用します。このため、両方で動作する WHERE 句を作成することができません。

この問題は、**範囲／位置付**ダイアログ（**タスク環境→範囲 / 位置付**）の **Magic SQL**（**SQL Where 句**タブ）に式を定義することで解決します。ここに定義する構文は Magic の構文ですが、実行時に使用する DBMS 用の SQL WHERE コマンドが作成されるようになります。

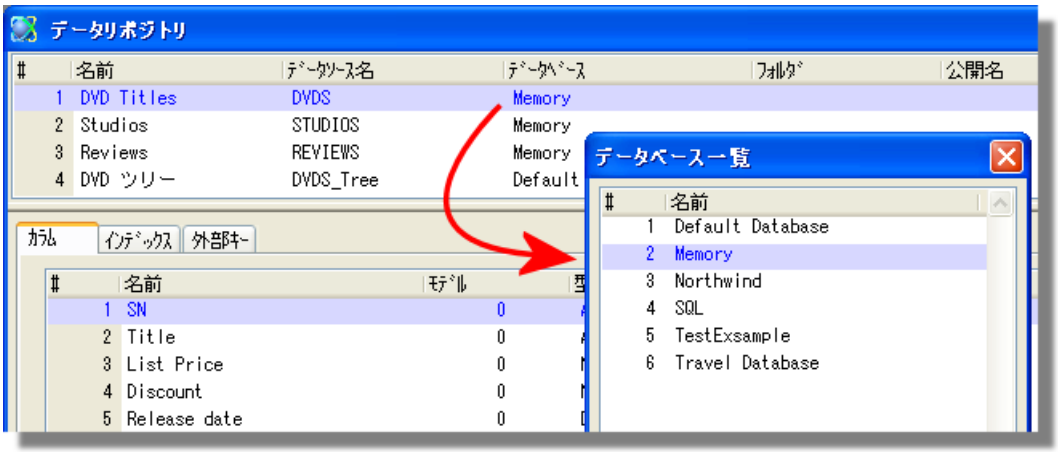
参照 : 第 20 章 :「タスクのデータビューに範囲を定義するには」（442 ページ）

永続性のないデータ用のテーブルを定義するには

Magic には配列機能がありますが、一時的なデータを保持するための小さなテーブルを作成し、他のテーブルのように扱う方が一般的により簡単になります。この場合、項目モデルやコントロールモデルを使用することが可能で、**Blb2File()** と **File2Blb()** のような関数を使用することで素早くテーブル化したり、保存したりすることができます。テーブルを使用することは、インデックスを考慮する必要がなく、データの検索に範囲や位置付け機能が利用できることを意味しています。

Magic で使用する永続性のないテーブルを***メモリテーブル***と呼んでいます。これは、他のテーブルと同じように作成することができます。

メモリテーブルを作成する



1. 他のデータベーステーブルと同じようにテーブルを作成します（詳細は、「Magic でデータベーステーブルを作成するには」（395 ページ）を参照してください）。
2. データベースカラムから、***Memory*** を選択します。

注： もしメモリテーブルがデータベース一覧に表示されない場合は、Magic.ini またはデータベーステーブル（オプション→設定→データベース）に定義されていないためです。通常は、Magic のインストール時に定義されます。メモリゲートウェイは、Magic.ini の [MAGIC_GATEWAYS] セクションの MGDB21 で定義されます。

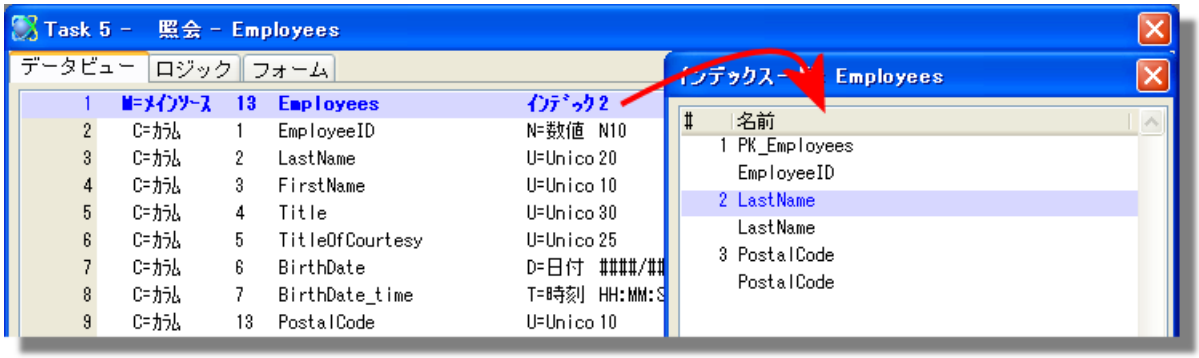
決められた順番でデータベーステーブルからレコードを取得するには

Magic でデータベーステーブルを使用している場合、これらのレコードを表示する順番を指定することができます。これには３つの方法があります。

- **インデックスを指定する**：データソースを選択した場合、使用するインデックスも選択します。
- **インデックスの並び順を指定する**：昇順か降順のどちらかを指定します。
- **動的ソートを実行する**：実行時にソートされるレコードを指定することもできます。この場合、データソースにインデックスを定義する必要はありません。このオプションはデータソースに定義されたインデックスを使用する場合より処理が少し遅くなりますが、レコード数が非常に多いテーブルでない限り利用できます。

詳細の定義方法を以下で説明します。

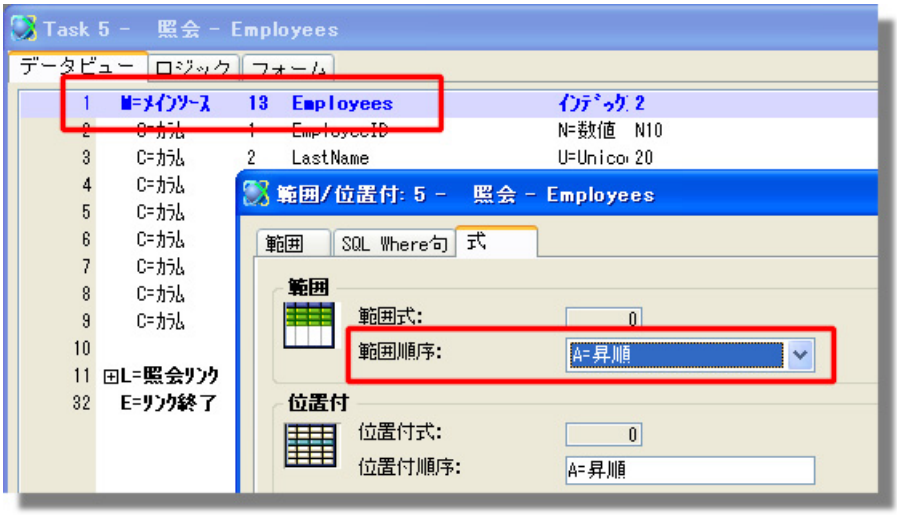
インデックスを指定する



プログラム内でデータソースを選択する場合、**インデックス**も設定することができます。ここから**ズーム**して使用するインデックス選択できます。この例では、#2 のインデックスを選択しています。この場合、従業員の姓をもとにレコードがソートとされます。

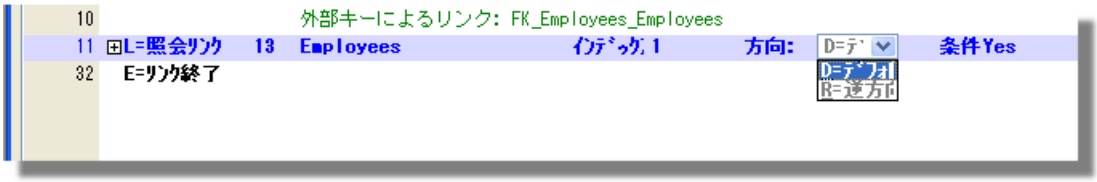
この設定は、メインソースとリンクテーブルの両方で有効です。インデックスを指定しない場合、レコードは作成順もしくは DBMS が決めた順番で表示されます。

メインソースに対する並び順を指定する



範囲／位置付ダイアログ（**タスク環境→範囲 / 位置付**）の**式**タブ（*Ctrl+R*）で**範囲順序**や、**位置付順序**を定義することでメインソースに対する並び順を指定することができます。**範囲順序**は、レコードが表示される順番を指定します。**位置付順序**は、位置付け指定されていたり、レコードの位置付けで位置付けカラムを使用している場合に使用されます。

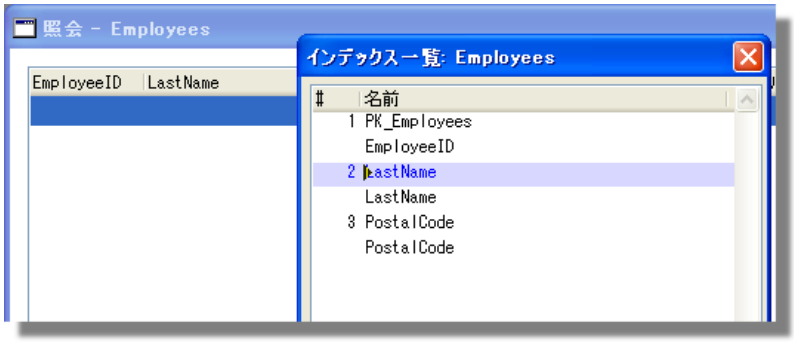
リンクテーブルの並び順を指定する



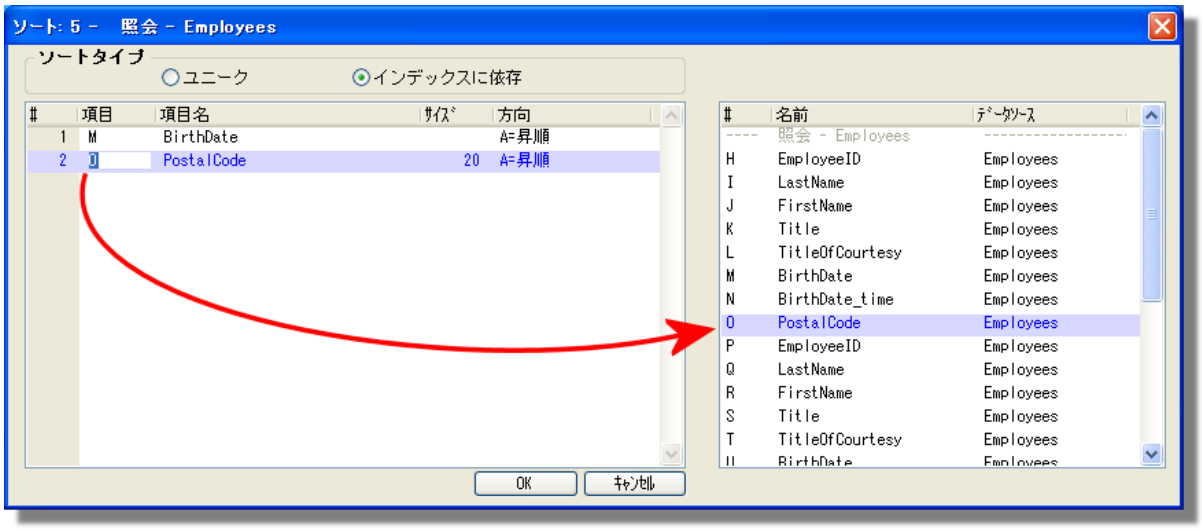
テーブルをリンクしている場合、1 つのレコードに位置付けているだけなので、方向の指定はそれ程重要ではありません。しかし、テーブル内の最初や最後のレコードを取得するような場合、指定内容はとても影響します。指定されたインデックス（例えば、郵便番号）の並び順が昇順に定義されている場合、**方向**特性を ***D= デフォルト***に設定した場合はテーブルの最小番号が返り、***R= 逆方向***を指定した場合は最大値が返ります。

動的なインデックスを作成する

データベーステーブル上で定義したインデックスの他に、実行時のみ有効な仮想的なインデックスを作成するように指定することができます。**オプション→インデックス**（*Ctrl+K*）を選択すると、**インデックス一覧**にこのインデックスが表示されます。



1. **ソート**テーブル（**タスク→ソート**または、*Ctrl+T*）を開きます。



2. ソートで使用する項目を以下の手順で定義します。
   - ***F4*** を押下して 1 行追加します。
   - **項目**カラムで**ズーム**して追加したい項目を選択します。
   - **方向**カラムで各項目に対して並び順（***A= 昇順***か ***D= 降順***）を設定します。
3. OK を**クリック**して終了します。

データベーステーブルを参照するには

実際に格納されているデータの内容を参照することは、アプリケーションを開発する上で便利です。Magic では、**APG**（**オプション→ APG** または、***Ctrl+G***）を使用することで簡単に行うことができます。

参照プログラムを作成する

1. **データ**リポジトリで、参照したいテーブル上にカーソルを置きます。
2. ***Ctrl+G***（**オプション→ APG**）を押下します。
3. **APG** ダイアログから以下のパラメータを選択します。
   - **モード**：一時的にデータ内容を参照する場合は、***E= 実行***。**プログラム**リポジトリにプログラムを作成する場合は、***G= 作成***
   - **オプション**：***Q= 照会***
   - **カラム**から**ズーム**することで、表示するカラムを選択したり、表示する順番を変更することができます。
4. **Enter** を押下するか、OK ボタンを**クリック**します。

モードで***実行***を選択した場合、**データベース**テーブル内のデータビューが表示されます。デフォルトモードは照会です。データ内容を変更する場合は、***Ctrl+M***（**オプション→修正**）を押下します。

モードで***実行***を選択した場合、プログラムが**プログラム**リポジトリ内に追加されます。***F7*** を押下することでこのプログラムを実行することができます。テスト中に重要なデータを参照することができるように、プログラムを変更することもできます。

ヒント :APG を使用すると、簡単にプログラミング作業を開始することができます。APG でプログラムを作成しこれを修正することで、意図するプログラムを作成することができます。

データベーステーブルからテキストにデータを出力するには

データを表計算などのアプリケーション渡したり、データを変換したりすために、データベーステーブルのデータをテキストファイルに出力することがあります。Magic は、このような処理を自動的に行うためのユーティリティとして **APG**（**オプション→ APG** または、***Ctrl+G***）を提供しています。

テキスト出力プログラムを作成する

1. **データ**リポジトリで、出力したいテーブル上にカーソルを置きます。
2. ***Ctrl+G***（**オプション→ APG**）を押下します。
3. **APG** ダイアログから以下のパラメータを選択します。
   - **モード**：一時的にデータ内容を参照する場合は、***E= 実行***。**プログラム**リポジトリにプログラムを作成する場合は、***G= 作成***
   - **オプション**：***E= 出力***
   - **カラム**から**ズーム**することで、出力するカラムを選択したり、出力する順番を変更することができます。
4. **テキストファイル**には、出力するテキストファイル名を入力します。
5. ***Enter*** を押下するか、OK ボタンを**クリック**します。

モードで***実行***を選択した場合、指定されたデータがテキストファイルとして出力されます。

モードで***作成***を選択した場合、プログラムが**プログラム**リポジトリ内に追加されます。***F7*** を押下することでこのプログラムを実行することができます。

注： テキスト出力機能は、非常に便利ではありますが、いくつかの制限があります。特殊な問題として、CR/LF を含んだ文字型項目や BLOB 型項目が存在した場合があります。このようなデータは、XML 形式で出力するなどの別の方法が必要になります。

テキストファイルのデータをデータベーステーブルに入力するには

テキストファイルからデータを入力することは、便利な方法である場合があります。表計算アプリケーションや既に出力されたファイルからデータを入力する場合、この機能は便利です（「データベーステーブルからテキストにデータを出力するには」（415 ページ）を参照してください）。Magic は、このような処理を自動的に行うためのユーティリティとして **APG**（**オプション→ APG** または、***Ctrl+G***）を提供しています。

テキスト入力プログラムを作成する

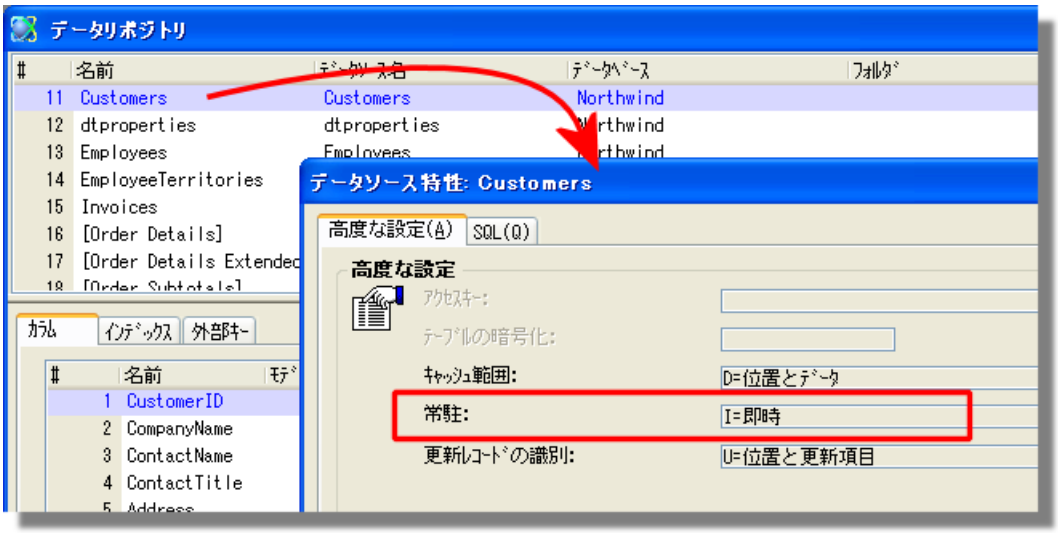
1. **データ**リポジトリで、入力したいテーブル上にカーソルを置きます。
2. ***Ctrl+G***（**オプション→ APG**）を押下します。
3. **APG** ダイアログから以下のパラメータを選択します。
   - **モード**：一時的にデータ内容を参照する場合は、***E= 実行***。**プログラム**リポジトリにプログラムを作成する場合は、***G= 作成***
   - **オプション**：***I= 入力***
   - **カラム**から**ズーム**することで、入力するカラムを選択したり、入力する順番を変更することができます。
4. **テキストファイル**には、入力するテキストファイル名を入力します。
5. ***Enter*** を押下するか、OK ボタンを**クリック**します。

モードで***実行***を選択した場合、指定されたテキストファイルからデータが入力されます。

モードで***作成***を選択した場合、プログラムが**プログラム**リポジトリ内に追加されます。***F7*** を押下することでこのプログラムを実行することができます。

注： テキスト入力機能は、非常に便利ではありますが、いくつかの制限があります。特殊な問題として、CR/LF を含んだ文字型項目や BLOB 型項目が存在した場合があります。このようなデータは、XML 形式として入力するなどの別の方法が必要になります。

データベーステーブルから一度にデータを取り込むには



プログラムで最も時間のかかる動作の１つには、データの読込処理があります。テーブルへのアクセスが頻繁に行われる場合、テーブルのデータを一度に読み込み、その内容を保持しておくことで、アプリケーションの処理が高速化されます。頻繁に更新されない読み込み専用のテーブルの場合、メモリ内にテーブル全体を読み込み、必要に応じてそれをアクセスすることは有効な処理になります。

しかし、手動でこのような処理を行う必要はありません。各データソースの**データソース特性**（*Alt+Enter*）には、**常駐**特性があります。この特性が *No* 以外の設定の場合、テーブルのデータはメモリに読み込まれアプリケーションが終了するまで常駐されます。このようなテーブルは、***常駐テーブル***と呼ばれています。

常駐テーブルは読み込み専用です。このテーブルは、一度データが読み込まれると頻繁に変わらないことを想定しています。しかし、アプリケーションの実行中に、内容を更新したい場合は、**DbReload()** 関数を使用することができます。

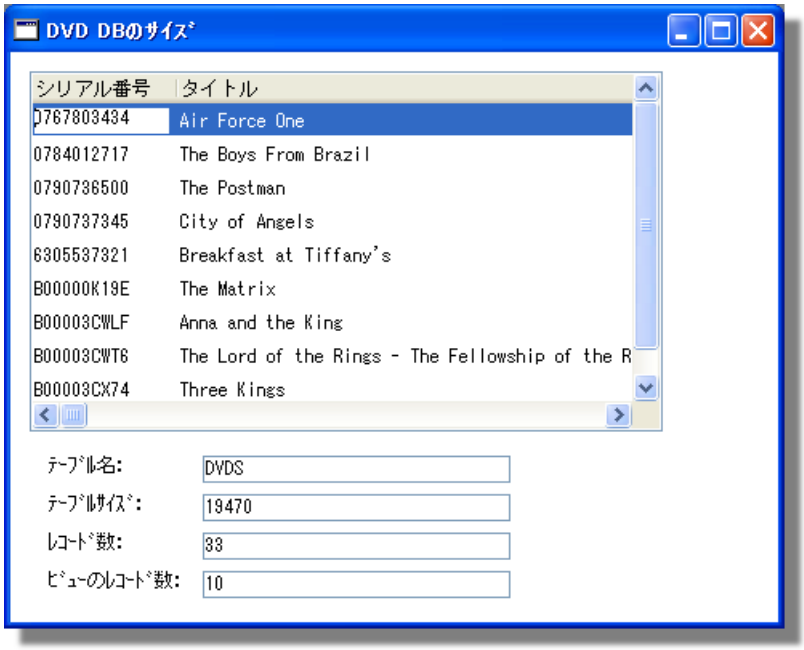
注： この機能を利用するには、**動作環境**ダイアログ（**設定→動作環境**）の**常駐テーブル読込**（**動作設定**タブ）を *Yes* に選定する必要があります。

データがメモリ内に読み込まれる場合、**常駐**特性の設定オプションによって以下のような動作になります。

常駐特性の設定オプション

- ***N= なし***：デフォルトの設定です。テーブルは常駐テーブルにはなりません。使用しているタスクが終了するとテーブルもクローズされます。
- ***I= 即時***：アプリケーションが起動されるとこのテーブルは直ちに読み込まれ、アプリケーションが終了するまでオープンされます。
- ***O= 利用時***：プログラムが最初にこのテーブルをオープンした時点で読み込まれ、アプリケーションが終了するまでオープンされます。
- ***B= ブラウザ上で常駐***：テーブルはブラウザから読み込まれます。この設定により、テーブルのリンク定義による再計算がローカル上で行われます。

データベーステーブルからレコードを取得する際のデータサイズを指定するには



データベーステーブルのサイズを知る必要があるかもしれません。この場合、以下の３つの関数を使用することができます。

- **DbSize()**：テーブルのサイズをバイト数で返します。
- **DbRecs()**：テーブルのレコード数を返します。
- **DbViewSize()**：現在のデータビューのレコード数を返します。

以下に説明するようにに、これらの関数は多少異なった動作をします。

DbSize() 関数を使用する

バイト数でテーブルのサイズを求めたい場合、**DbSize()** 関数を使用します。構文は以下の通りです。

DbSize(*dsource#*, *tablespec***)**

パラメータ

- *dsource#*：**データ**リポジトリでのテーブル番号です。
- *tablespec*：**データ**リポジトリに定義されているテーブル名以外のテーブルにアクセスする場合は、ここにテーブル名を指定します。

戻り値：バイト数を表す数値が返ります。

例：DbSize('1'DSOURCE,")

DbRecs() 関数を使用する

テーブルのレコード数を求めたい場合、**DbRecs()** 関数を使用します。構文は以下の通りです。

DbRecs(*dsource#*, *tablespec***)**

パラメータ

- *dsource#* ：**データ**リポジトリでのテーブル番号です。
- *tablespec* ：**データ**リポジトリに定義されているテーブル名以外のテーブルにアクセスする場合は、ここにテーブル名を指定します。

戻り値：レコード数を表す数値が返ります。

例：DbRecs('1'DSOURCE,")

DbViewSize() 関数を使用する

現在のデータビュー内のレコード数を求める場合、**DbViewSize()** 関数を使用します。構文は以下の通りです。

DbViewSize (*generation*)

パラメータ

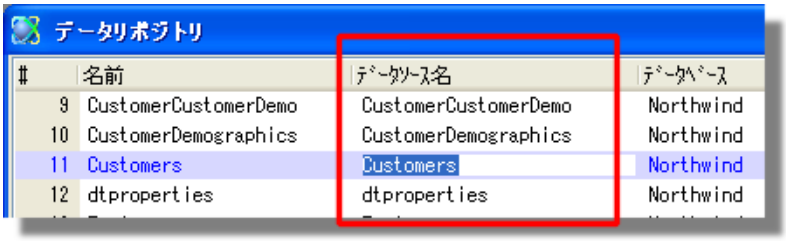
- *generation*：タスクの世代番号です。(0 は現在のタスク、1 は親タスクになります)

戻り値：レコード数を表す数値が返ります。テーブル内のレコード数ではなく、タスクで範囲指定をした場合にビューとして展開されたレコード数のみ返されます。

例： **DbViewSize(*0*)**

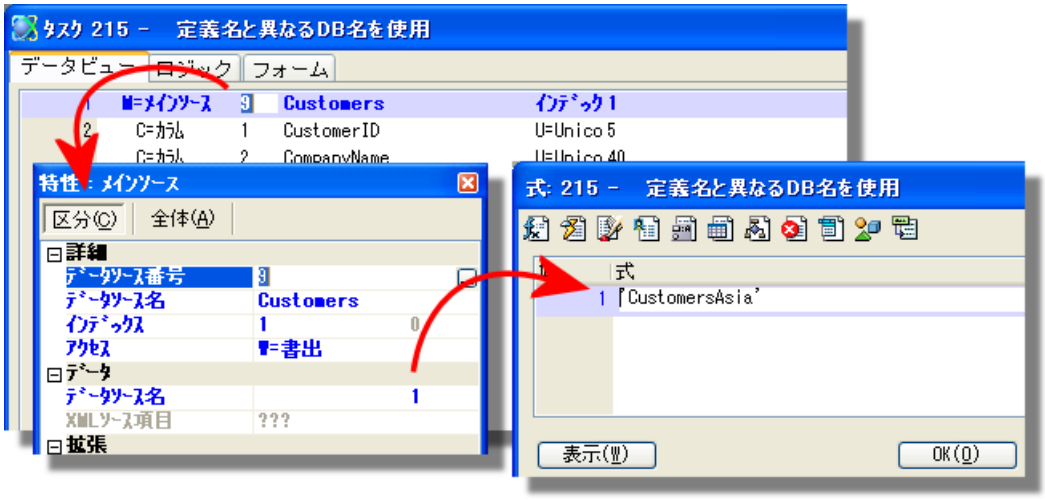
注： デフォルトでは、データビュー内のレコードは必要に応じて取り込まれたものだけのため、**DbViewSize()** は、範囲条件に合う実際のレコード数のみ返ります。この関数で正確なレコード数を取得したい場合は、タスク特性の**ビューの事前読込**特性を *Yes* に設定する必要があります。タスクが起動される前に、ビュー内のすべてのレコードが読み込まれます。これは **DbViewSize()** で正確な値を返すだけでなく、期待値どおりにスクロールバーを表示させるようにできます。

動的にデータソース名を設定するには



通常、データソース名はデータリポジトリのデータソース名カラムに設定された名前が使用されます。しかし、データソースの定義の際や、そのデータソースにアクセスする関数にて指定することで、プログラムが実際にアクセスするデータソースを変更することができます。さらに論理名を使用することで、データソース名を実行環境にもとづいた設定ができます。以下で、これらのオプションについいて説明します。

定義された名前と異なるデータソース名を使用する



メインソースやリンクテーブルを定義する際に、データソース名を変更することができます。この設定を行うには、項目特性のデータソース名特性で式を使用してデータソース名となる文字列を指定します。この例では、Cutomers というデータソースを CustomersAsia という名前ので使用するように設定しています。

関数でデータソース名を指定する

例えば、CustomersAsia テーブルのレコード数を取得する場合、以下のようなパラメータでは指定できません。

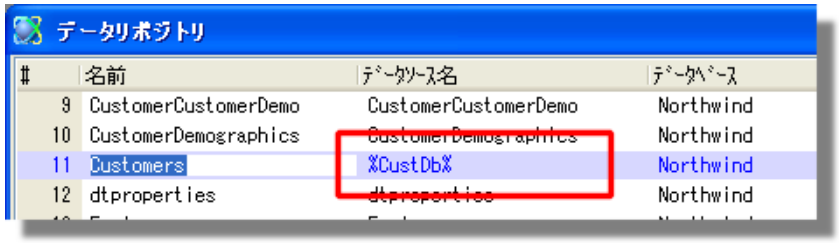
```
DbRecs('4'DSOURCE,'')
```

以下のように指定する必要があります。

```
DbRecs('4'DSOURCE,'CustomersAsia')
```

２番目のパラメータを使用することで、実際にアクセスするデータソース名を変更しています。DB 関数のいくつかは、このように２番目のパラメータでデータソース名を指定することができます。

データソース名に論理名を使用する



データソース名をプログラムで指定する場合、直接名前を指定することは推奨できません。**論理名**を使用するようにしてください。この方が、保守がしやすくなります。例えば、CustomersAsia の代わりに、CustomersEurope や CustomersAustralia などの別のテーブルにアクセスすることも簡単にできるようになります。また、異なる保存データ用に名前を変更したテーブルを使用することもできます。

顧客データベース用に論理名 %CustDB% を定義し、データソース名を %CustDB% と定義した場合、**DbRecs()** 関数では以下のように指定します。

```
DbRecs('4'DSOURCE、%CustDB%)
```

1 つのテーブルを定義することで、複数の実テーブルにアクセスするため、**データ**リポジトリで論理名を使用することもできます。

［このページは意図的に空白にしています。］

第 19 章 : メインプログラム

アプリケーションを初期設定するには

アプリケーションを設定する際に、アプリケーション内の全てのプログラムが実行される前にグローバルに設定する必要のある項目があるかもしれません。Magic では、これらの項目を**メインプログラム**で定義します。これは**プログラム**リポジトリの最上位にある特別なプログラムです。

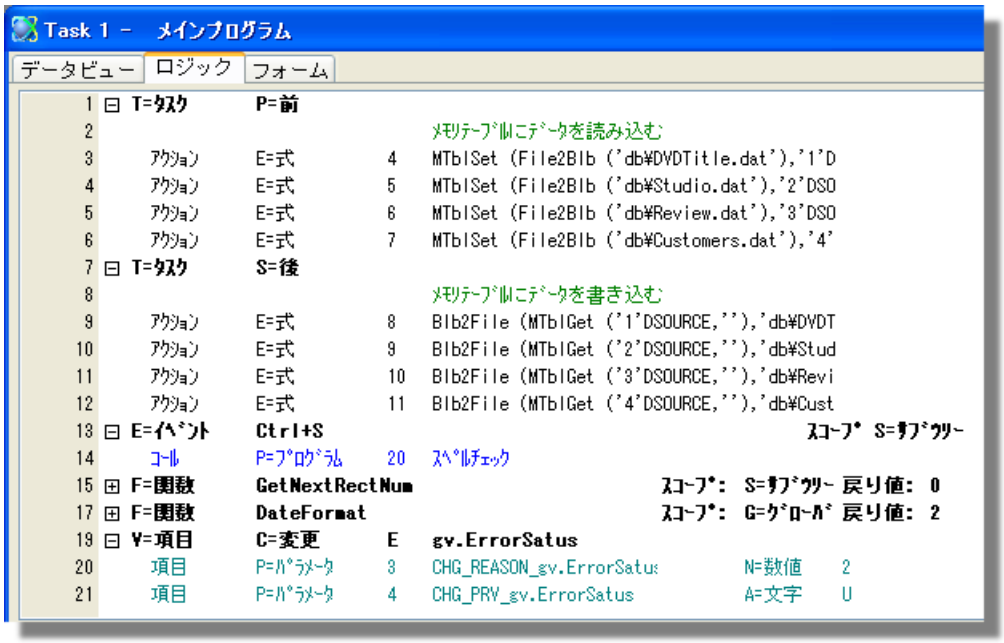
メインプログラムは、プロジェクト内の全てのプログラムの「親」タスクと考えることができます。(デバッグモードでテストする場合も) プログラムが起動される前に、**メインプログラム**の**タスク前**が実行されます。実行中のプログラムから、**メインプログラム**内の**ユーザ定義関数**や変数項目にアクセスすることができます。そして、プロジェクトがクローズされると (デバッグモードの場合は、プログラムが終了すると)、**メインプログラム**の**タスク後**が実行されます。

メインプログラムのデータビューエディタ

メインプログラムの**データビューエディタ**で、アプリケーションで使用する (OLE オブジェクトを含めて、) 変数項目を定義することができます。**タスク前**でこれらの項目を初期設定することができます。

SOAP サービスで使用される様々なタイプの変数項目 (CRLF データや BLOB など) が必要な場合、**データビューエディタ**は、これらを格納しておく便利な場所となります。

メインプログラムのロジックエディタ



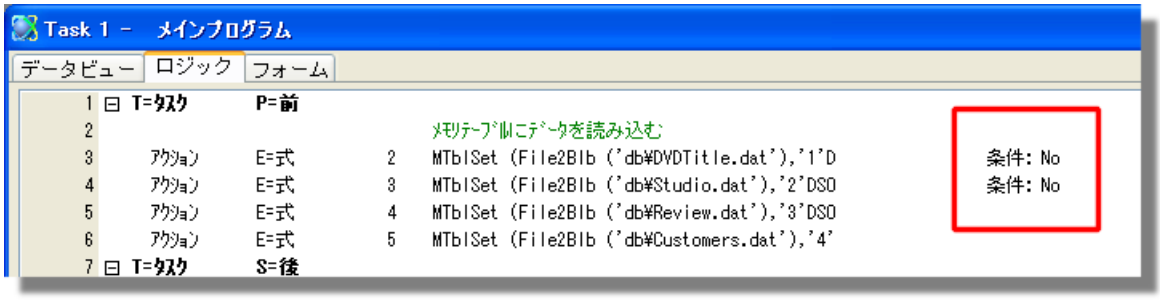
1. **タスク前**：プログラムが起動される前に実行させたい処理コマンドを定義します。この例では、ファイルを元に BLOB 化された Magic テーブルを（メモリ）テーブル化する処理が定義されています。ログオンしたユーザの記録をとるためのプログラムをここで呼び出すこともできます。
2. **タスク後**：ユーザがプロジェクトを終了する際に、実行させたい処理コマンドを定義します。この例では、メモリテーブルの内容を BLOB 化およびファイル化する処理が定義されています。ここにも、様々な処理を実行するためのプログラムを呼び出すことができます。
3. **関数：関数**ロジックユニットを定義することもできます。ここに定義された**ユーザ定義関数**は、グローバルにアクセスすることができ、Magic の**関数一覧**に表示されます。
4. **イベント**：ここに定義された**イベント**ロジックユニットはグローバルにアクセスできます。この例では、***Ctrl+S*** を押下すると、どのプログラム上であってもスペルチェックプログラムが実行されるように定義されています。
   メインプログラム内でこの種のロジックを定義することで、個々のプログラム内で同じ処理を何回も定義する必要がなくなります。
   イベントロジックユニットを定義することで、**エラー**タイプのイベントを使用してグローバルにエラーを処理したり、**ActiveX** イベントを処理したりすることもできます。
5. **項目**：**項目変更**ロジックユニットは、メッセージを表示させたり、ログを出力する目的で使用することができます。

メインプログラムのフォームエディタ

フォームエディタでは、アプリケーション全体で利用可能なフォームを定義することができます。例えば、ここに一般的な帳票ヘッダを定義することで、多くの異なる帳票用に使用することができます。

アプリケーションの背景を定義するために使用することもできます（「アプリケーションの背景 / 壁紙を設定するには」（426 ページ）を参照してください）。ここに定義されるフォームは、アプリケーションのグローバルなコンテキストメニューを設定する場所でもあります。

初期設定のプログラムを省略するには



メインプログラム内に定義されているあらゆる初期化処理は、条件カラムに条件式を設定することで簡単に処理を制御することができます。条件カラムを *No* に設定すると、初期化処理は実行されなくなります。

しかし、一定の条件（テスト環境など）でのみ初期設定をスキップさせたい場合があるかもしれません。プログラムをテストしている場合や **APG** を使用してテーブルを参照する場合も、メインプログラムは自動的に実行されます。この場合、初期設定処理よって起動が多少遅くなります。タスク前でユーザ操作が必要なプログラム（時刻の入力など）があった場合、これによって起動時間はさらに遅くなります。

従って、これらの状況で条件付きで処理を実行させないようにするために **RunMode()** 関数を使用することができます。この関数を実行すると以下の値が返ります。

- -1：アプリケーションがアプリケーションサーバとして動作している
- 0：アプリケーションがクライアント / サーバの実行モードで動作している
- 2：プログラムが開発モードで実行している（デバッグ→実行（*F7*）または オプション→ APG（*Ctrl+G*））
- 3：デバッグモードでプロジェクトが実行している（デバッグ→プロジェクトを実行 （*Ctrl+F7*））

例えば、開発モードの場合のみ初期設定処理を無効にするには、以下の条件式を設定します。

Runmode()<2

これで実運用の場合のみ処理が実行されるようになります。

アプリケーションの背景 / 壁紙を設定するには

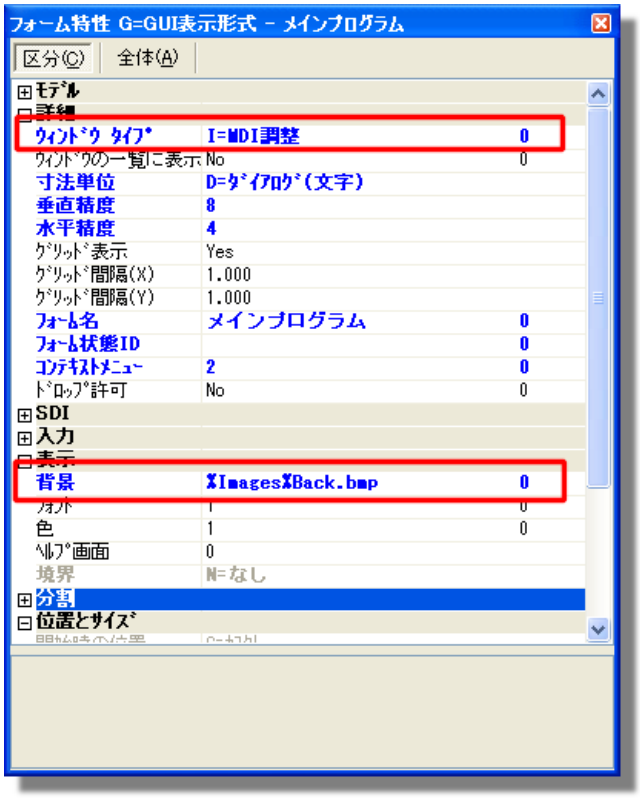
メインプログラムで背景や壁紙を設定することができます。プロジェクトが実行中に、他のタスクフォームのようにメインプログラムのフォームが表示されます。メインプログラムのフォーム上には、データ入力機能を持たせることはできませんが、プッシュボタンなどの操作用のコントロールを配置させることはできます。

この例では、背景として壁紙を使用し、アプリケーションのタイトルとバージョン番号が表示されたテキストボックスが表示され、起動ボタンも配置されています。

メインプログラムに背景を定義する

1. タスク特性のフォーム表示特性を *Yes* に設定します。
2. フォーム特性のウィンドウタイプ特性を *M=MDI に合わせる* に設定します。
3. フォーム特性の背景特性に表示させたい壁紙のファイルを定義します。

他のフォームと同じようにテキストやプッシュボタンなどのコントロールも配置することができます。



ヒント :背景特性に式を指定したり、タスク特性のメインフォーム特性で式を使用することで、背景表示を動的に指定することができます。

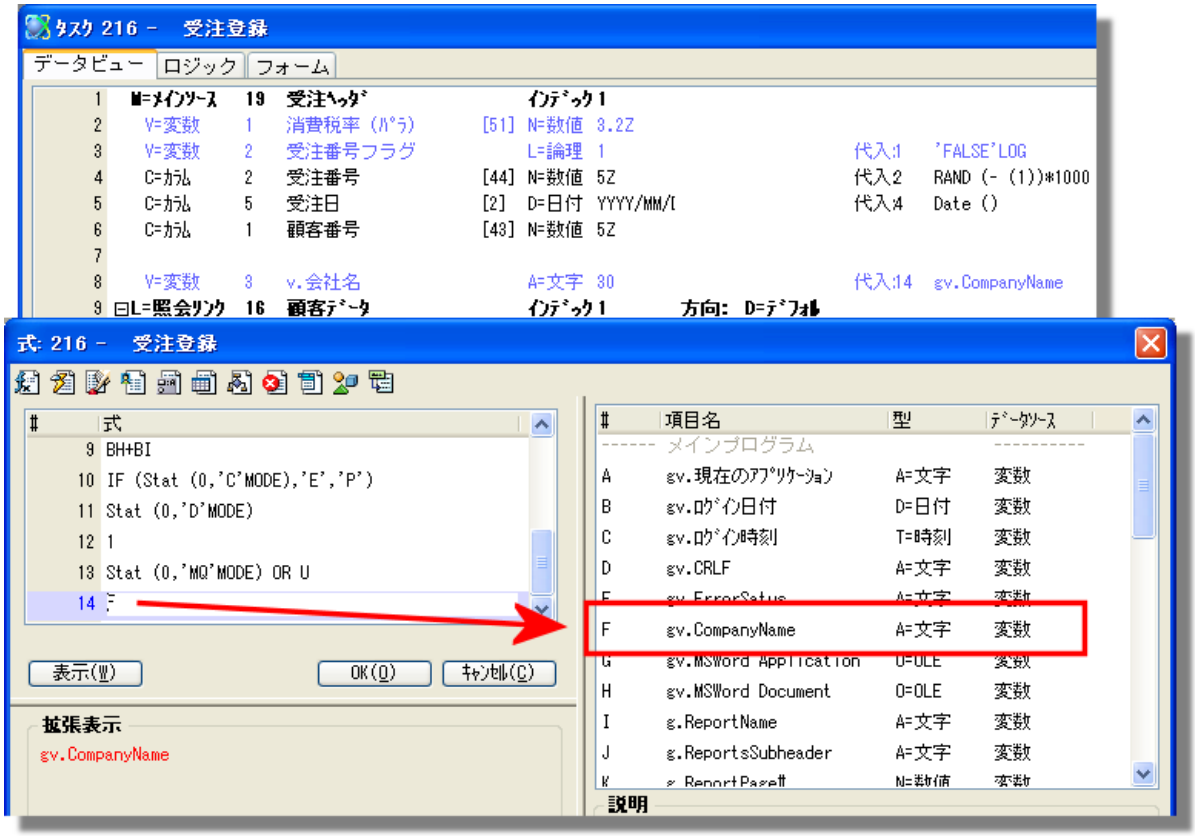
グローバル変数の設定と使用を行うには（コンテキスト単位）

メインプログラムで定義された変数項目は、プロジェクト内のどこからでも使用することができます。この例では、Company Name という変数項目を定義しているため、この変数を様々な処理で使用することができます。

代入特性を使用して変数項目の初期設定を行うことができます。**Translate(*'%Company%'*)** は、Magic.ini に定義されている論理名をもとに会社名に変換します。別の方法として、データベーステーブルから会社名を取り出すためにタスク前でプログラムを呼び出したり、テーブルをリンクしてデータを取り出すこともできます。

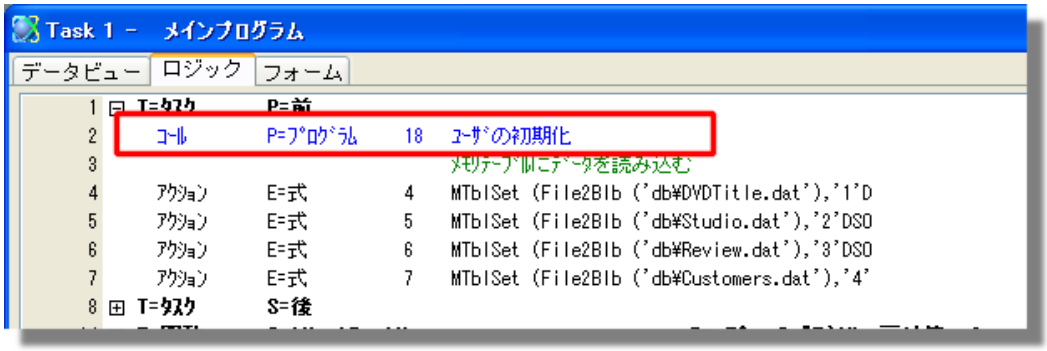
注： メインプログラムでテーブルにリンクする場合、データは読み込み専用となります。

グローバル変数を使用する



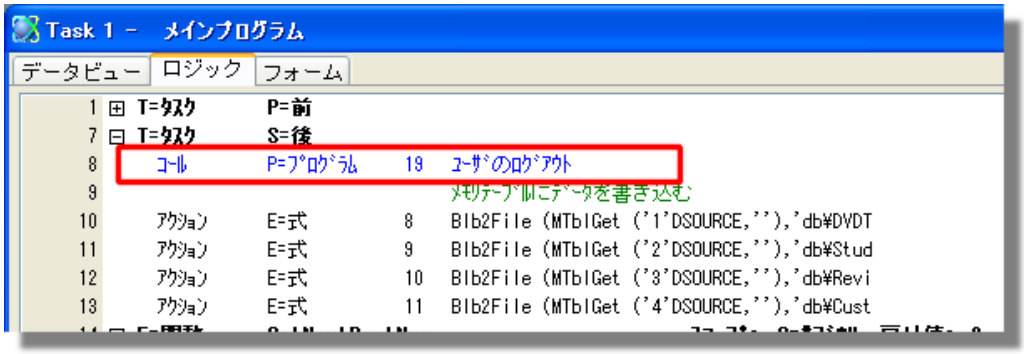
メインプログラムに変数項目を定義すると、他の項目と同じようにアクセスすることができます。また、どのタスクからでも項目一覧の最上位に表示されます。式エディタでこの変数項目から値を取り出したり、項目更新処理コマンドを使用して、内容を更新することができます。

アプリケーション起動時の手続き処理を実行するには



アプリケーションの起動時に実行させたいプログラムは、**メインプログラム**の**タスク前**で起動しなければなりません。コンポーネントプログラムを含む Magic 内のあらゆるプログラムをここで呼び出すことができます。呼び出すプログラムが同じプロジェクト内に存在する場合、呼び出されたプログラムは**メインプログラム**の変数項目にアクセスできるため、パラメータを渡す必要がありません。

アプリケーション終了時の手続き処理を実行するには



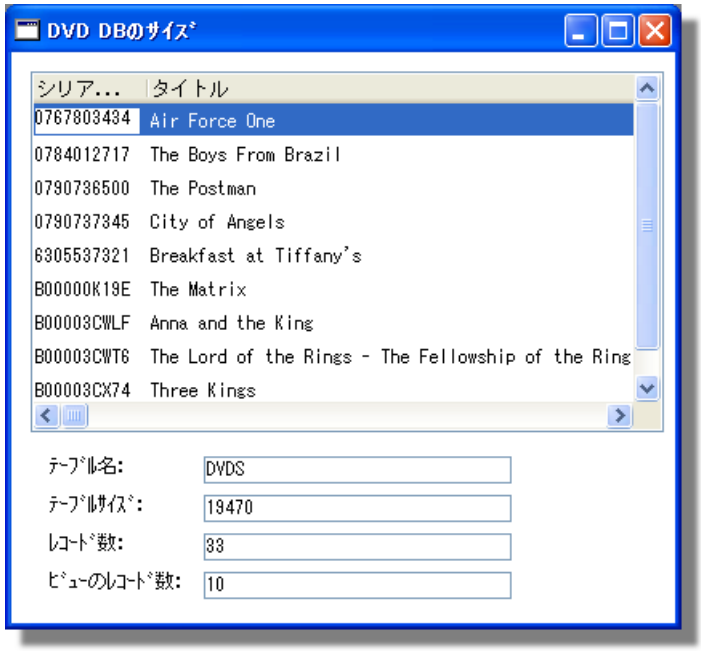
アプリケーションの終了時に実行させたいプログラムは、メインプログラムのタスク後で起動しなければなりません。コンポーネントプログラムを含む Magic 内のあらゆるプログラムをここで呼び出すことができます。呼び出すプログラムが同じプロジェクト内に存在する場合、呼び出されたプログラムはメインプログラムの変数項目にアクセスできるため、パラメータを渡す必要がありません。

［このページは意図的に空白にしています。］

第 20 章 : データビュー

タスクのデータビュー内のレコード数を取得するには

データベーステーブルのサイズを知る必要があるかもしれません。この場合、以下の３つの関数を使用することができます。



- **DbSize()** : テーブルのサイズをバイト数で返します。
- **DbRecs()** : テーブルのレコード数を返します。
- **DbViewSize()** : 現在のデータビューのレコード数を返します。

DbViewSize() 関数のついては以下で説明します。

DbViewSize() 関数を使用する

現在のデータビュー内のレコード数を求める場合、**DbViewSize()** 関数を使用します。構文は以下の通りです。

DbViewSize (*generation*)

パラメータ

- *generation* : タスクの世代番号です。(0 は現在のタスク、1 は親タスクになります)

戻り値 : レコード数を表す数値が返ります。テーブル内のレコード数ではなく、タスクで範囲指定をした場合にビューとして展開されたレコード数のみ返されます。

例 : **DbViewSize(*0*)**

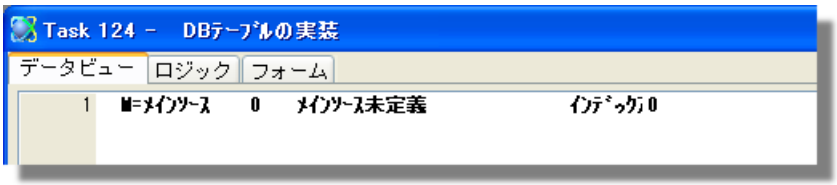
注： デフォルトでは、データビュー内のレコードは必要に応じて取り込まれたものだけのため、**DbViewSize()** は、範囲条件に合う実際のレコード数のみ返ります。この関数で正確なレコード数を取得したい場合は、タスク特性の**ビューの事前読込**特性を *Yes* に設定する必要があります。タスクが起動される前に、ビュー内のすべてのレコードが読み込まれます。これは **DbViewSize()** で正確な値を返すだけでなく、期待値どおりにスクロールバーを表示させるようにできます。

タスクのメインソースを定義するには

タスクの**メインソース**は、**データビューエディタ**の最初のヘッダ行として常に表示されています。**メインソース**の番号と名前およびレコード表示で使用されるインデックスが同じ行に表示されています。これ以外の特性は、**特性シート**（*Alt+Enter*）で指定できます。

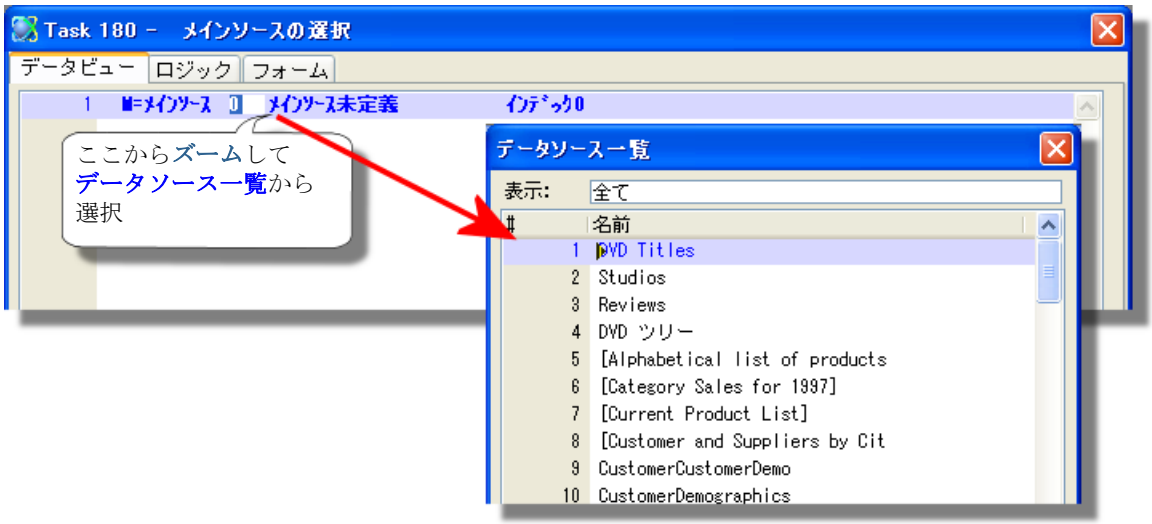
タスクが複数のレコードを検索することができる唯一の方法は、**メインソース**を定義することです。**メインソース**が定義されたら、これをもとにタスクは実行されます。表示される（バッチタスクの場合は、繰り返し処理が実行される）レコード数は、**メインソース**の範囲条件を満たしているレコードによって決定されます。

メインソースは必須ではありません。タスクに**メインソース**が定義されていない場合、最初のヘッダ行は以下のように表示されます。



メインソースのないオンラインタスクはレコードを表示しません。この場合、1 度に 1 つの（リンク）レコードを表示することができるだけになります。**メインソース**のないバッチタスクは、デフォルトで永久にループします。このため、**タスク特性**で**タスク終了条件**特性を指定する必要があります。

メインソースを入力する



1. **メインソース**の次のカラムにカーソルを移動します。
2. ズームして、**データソース一覧**を表示します。
3. 使用したいデータソース上にカーソルを置き、*Enter* を押下します。

タスクで処理されるレコードの順番を動的に設定するには

メインソースが指定されると、タスクはデフォルトで、データベースの全てのレコードを検索します。レコードが処理される順番は、開発者によって設定されます。処理の順番を指定するには３つの方法があります。

- インデックス番号の指定
- インデックスを式で指定
- タスクのソートテーブルで指定

これらの方法について以下で説明します。

インデックス番号を指定する

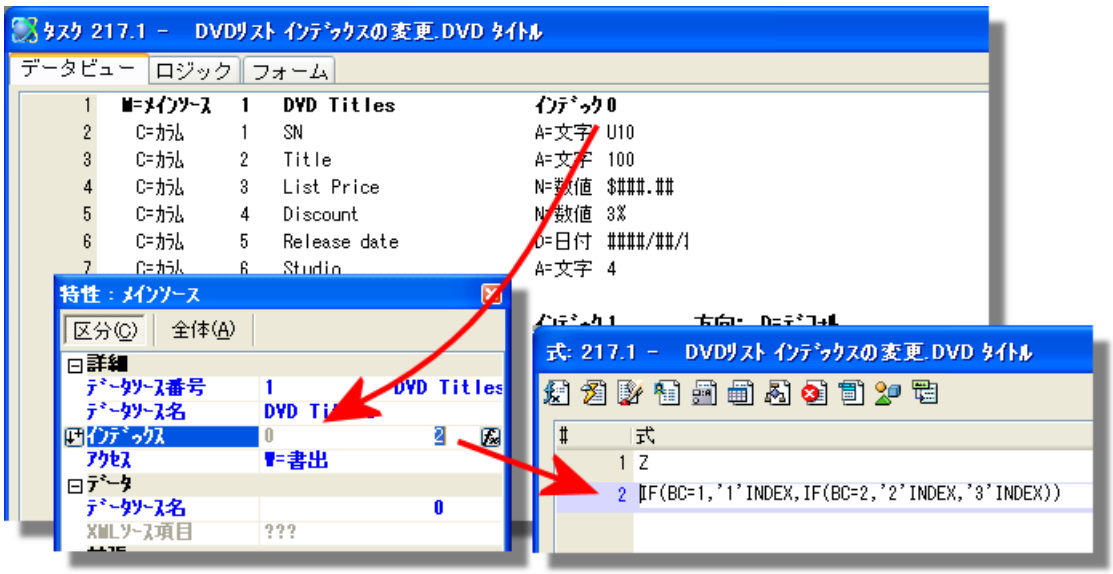
インデックスを最も簡単に指定する方法は、メインソースのインデックスカラムにインデックス番号を指定することです。この例では、タイトル毎に DVD をソートしています。インデックスは、**#2** が指定されています。

1. データビューエディタのメインソースに移動します。
2. インデックスカラムかズーム（*F5* またはダブルクリック）してインデックス一覧を開きます。
3. *Enter* を押下して使用するインデックスを選択します。

インデックスが追加／削除された場合、ここに表示されているインデックス番号も変更されます。

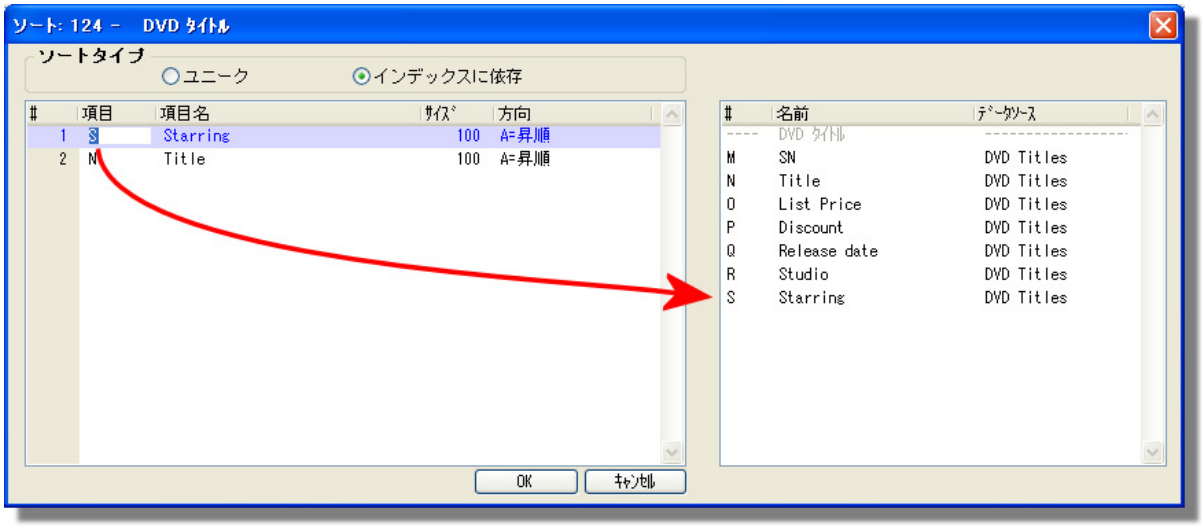
インデックス式を使用する

実行時に動的にインデックスを選択する場合は、式を使用してインデックスを指定します。この場合、ユーザがソート順序を選択することができます。また、ユーザによって選択された選択条件にもとづいた適切なソート順序が使用されるようにロジックを組み込むこともできます。



1. **メインソース**から *Alt+Enter* を押下して**メインソース特性**を開きます。
2. **詳細**セクションの**インデックス**特性に移動します。右側の**式**特性に移動します。ここから**ズーム**して、実行時にインデックス番号として評価される式を指定します。
   この例では、ユーザが入力した選択条件をもとにインデックスを選択しています。ユーザが選択した値（1、2、または 3）を使用するのではなく、対応する **INDEX** リテラルを指定するようにしていることに注意してください。インデックスが今後変更されても、**INDEX** リテラルは自動的に変更されるためこの指定方法は重要です。

タスクのソートテーブルを使用する



使用したいインデックスが定義されておらず、表示する順番を変更することだけが問題の場合は、Magic に内蔵されているソート機能を使用します。このソート機能ではどのような項目でも指定することができ、Magic は実行時にテーブルに一時的なインデックスを作成します。

1. ソート処理を実行させたいタスクを開きます。
2. ソートテーブル（タスク環境→ソートまたは、*Ctrl+T*）を開きます。
3. ウィンドウの左側は、何も設定されたいないか（まだソート定義されていない場合）、項目が一覧表示（既にソートが定義されている場合）されています。この一覧によってテーブルのソート順序が決まります。この例では、Starring と Title の２つの項目ソートされるように定義されています。
4. 一覧に項目を加えるには以下のようにします。
   - *F4*（編集→行作成）を押下して１行追加します。
   - 項目カラムで、項目のシンボル名を入力するか、ズームして一覧から選択します。
   - 項目の長さが長すぎる場合、ソート処理が効率的に実行されない場合があります。このような場合、サイズカラムで文字数を入力することによって、ソートで使用される文字数を短くすることができます。この例では、Starring と Title の２つの項目の長さは 100 桁ですが、最初の 20 桁のみ使用するように指定しています。
   - 必要に応じて方向カラムを降順に変更することもできます。日付項目に対してこの指定を行うことで、最新の情報から表示されるようになります。
5. 項目をリストから削除する場合は、*F3*（編集→行削除）を押下します。
6. 設定が終了したら、OK をクリックします。

ヒント：ソート処理は範囲処理の後に実行されます。レコードの範囲を指定してレコード数を制限した後にソートを行うことで、テーブルのソート処理が早く実行されることになります。

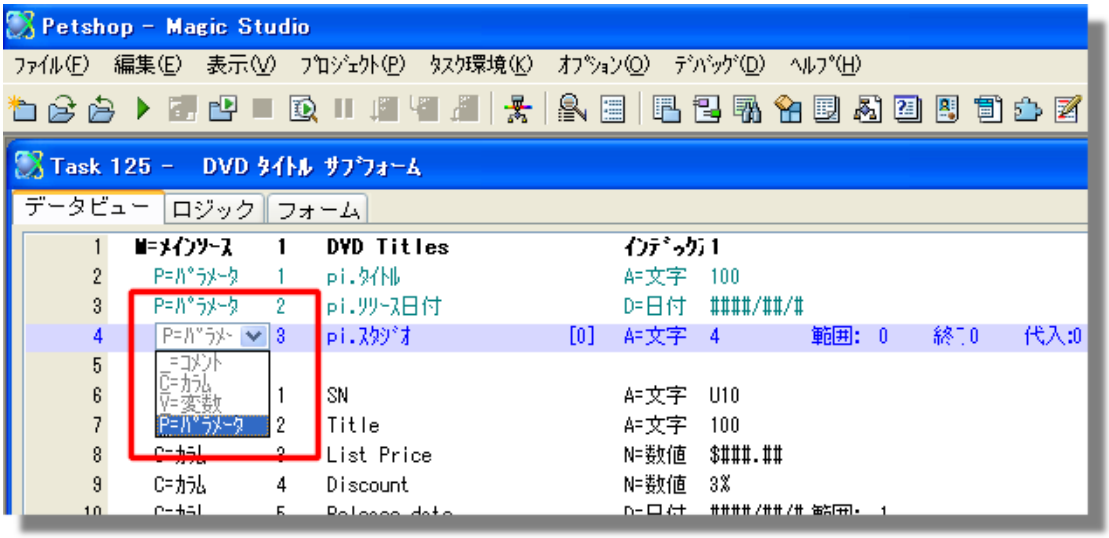
プログラムに受け渡しするパラメータを定義するには

様々なプログラム言語では、パラメータとしてデータの受け渡しができるようになっています。パラメータは、データを渡したり、受け取ったりすることができます。

相手から受け取るデータをパラメータとして定義するだけでなく、戻り値として定義することもできます。戻り値は、一般的に関数として動作するプログラムのために使用されます。戻り値を使用することで、**CallProg()** 関数を使用して式でプログラムを呼び出すことができます。戻り値は、コールプログラム処理コマンドの戻り値カラムで受け取ることもできます。

以下で、パラメータと戻り値の定義方法について説明します。

パラメータ項目を定義する

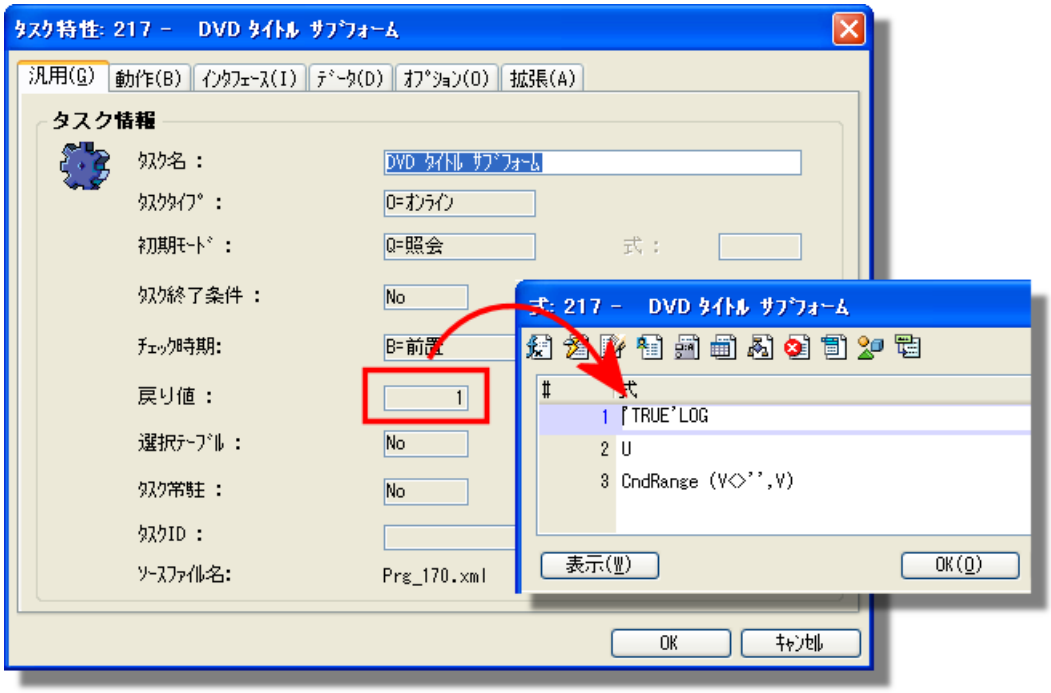


1. パラメータ項目は、他の変数項目と同じ方法で定義します。唯一の違いは、***カラム***または***変数***ではなく***パラメータ***を選択することだけです。
2. パラメータの**名前**を入力します。Magic から見て、入力された名前はあまり重要ではありませんが、このプログラムを利用する際、定義されたパラメータが入力用か出力用または共用のものかどうかが確認できるようにするために、何らかの命名規約を決めておくことを推奨します。例えば、入力パラメータは先頭に pi を付加し、出力用は po、共用は pio を付加するようにします。
3. 他の項目と同じように**型**や**書式**およびその他の**項目特性**を設定します。**項目モデル**を指定することで、パラメータの項目長を合わせ易くなります。
4. パラメータ項目の定義場所はあまり重要ではありません。**データビューエディタ**の先頭や下部に定義されていたり、他の項目の間に分散されて定義されているかもしれません。しかし、定義場所を（先頭に定義するとか）あらかじめ決めておいた方がメンテナンスが容易になります。

定義されたパラメータ項目は、**コール**処理コマンドの**パラメータ**テーブルに表示されます。また、コンポーネントや COM、または SOAP オブジェクトを作成する際にも使用されます。このプログラムを呼び出す処理を定義する際、**パラメータ**テーブルでパラメータ名やデータ型を確認することができ、**構文チェック**を実行するとパラメータの数とデータ型の整合性をチェックします。

注： パラメータを式で指定した場合、値を更新することができません。従って、項目 G が定義されており、G を項目カラムに入力した場合、更新することができます。項目 G を式で指定した場合、G の値はプログラムに渡りますが、項目の値は更新されません。値が変更される必要がない場合のみ、式でパラメータを指定することができます。

戻り値を定義する



1. タスク特性の戻り値（汎用タブ）に移動します。
2. ズームして式エディタを開きます。
3. 返したい値として評価される式を指定します。

タスクのデータ項目を定義するには

Magic には 3 種類の項目があります。

- **パラメータ**は、プログラムへのデータの受け渡しを行う場合に使用されます。変数項目と同じ方法で定義します。この例では、スタジオ番号（SN）に値を渡します。
- **カラム**は、データソースから選択される項目です。Magic では、データソースは SQL テーブルやメモリテーブル、または XML ファイルが使用されますが、全て同じ方法で扱われます。この例では、メインソース（DVD テーブル）とリンクテーブル（スタジオファイル）からカラムを選択しています。
- **変数**は、現在のタスクやその子タスクでのみ利用できる一時的な項目です。この例では、パーク可能なプッシュボタンを定義するために使用されています。

これらの項目が定義されると、各種類に対応した動作を実行します。主な違いは、タスクの終了時のデータの扱いです。

パラメータ項目に関する詳細は、「プログラムに受け渡しするパラメータを定義するには」（437 ページ）を参照してください。以下でカラムと変数項目の定義方法について説明しています。

カラムを定義する

1. リンクテーブルのカラムを選択する場合、リンクとリンク終了の間にカーソルを置きます。これ以外の場所にカーソルがある場合、**メインソース**のカラムとして扱われます。
2. ***F4***（**編集→行作成**）を押下して１行追加します。
3. ***C*** を入力するかプルダウンメニューから ***C= カラム***を選択します。***Tab*** 移動します。
4. **ズーム**（***F5*** または、**ダブルクリック**）してカラムを選択するか、カラム番号を入力します。***Tab*** 移動します。
5. 名前カラムに入力することで、必要に応じて変更することができます。同じデータソースをリンクしているなどで、同じ名前のカラムが複数定義されている場合、名前を変更する方がわかりやすくなります。データビューで名前を変更しても、**データ**リポジトリには影響しません。

これで定義できました。カラムの特性は**データ**リポジトリで設定されるため、ここで設定する必要はありません。

変数項目を定義する

1. 変数項目を作成したい行にカーソルを置きます。変数はどこにでも作成できます。
2. ***F4***（**編集→行作成**）を押下して、１行追加します。
3. ***V*** を入力するかプルダウンリストから ***V= 変数***を選択します。***Tab*** 移動します。
4. 変数名を入力し、***Tab*** 移動します。
5. **モデル**カラムには、必要に応じて変数に定義する**項目モデル**番号を設定します。この例では、変数 b.WatchClip は モデル **#40** を使用していますが、変数 v.VideoReturnCode にはモデルが使用されていません。
   モデルを使用することで、開発工数が削減され、プログラムの保守が楽になります。
   モデルを選択するには、ここから**ズーム**（***F5*** または、**ダブルクリック**）します。

モデルを使用している場合、変数項目の特性はすでに設定された状態になっています。必要に応じて、デフォルトの特性値を変更することができますが、モデルの使用方法についてよく理解した上で行う必要があります。モデルを使用していない場合、*Tab* 移動して次のステップを行ってください。

6. モデルの次のカラムは**型**カラムです。ここでは、項目の型（文字型、数値型日付型、時刻型、OLE 型など）をプルダウンメニューから選択することができます。*Tab* 移動します。
7. 次に**書式**カラムに移ります。変数の書式を入力します。入力する上での補助が必要であれば、ここから**ズーム**して、**書式**ダイアログを開きます。

OLE オブジェクトのように更に特性を設定する必要なものもありますが、ほとんどの変数はこの時点で、利用することができます。**特性シート**（*Alt+Enter*）を開いて、必要な特性値を設定することができます。

タスクのデータビューに範囲を定義するには

デフォルトでは、Magic はメインソースのすべてのレコードを表示します。例えば、APG を使用してプログラムを作成した場合、テーブルのすべてのレコードが表示されます。

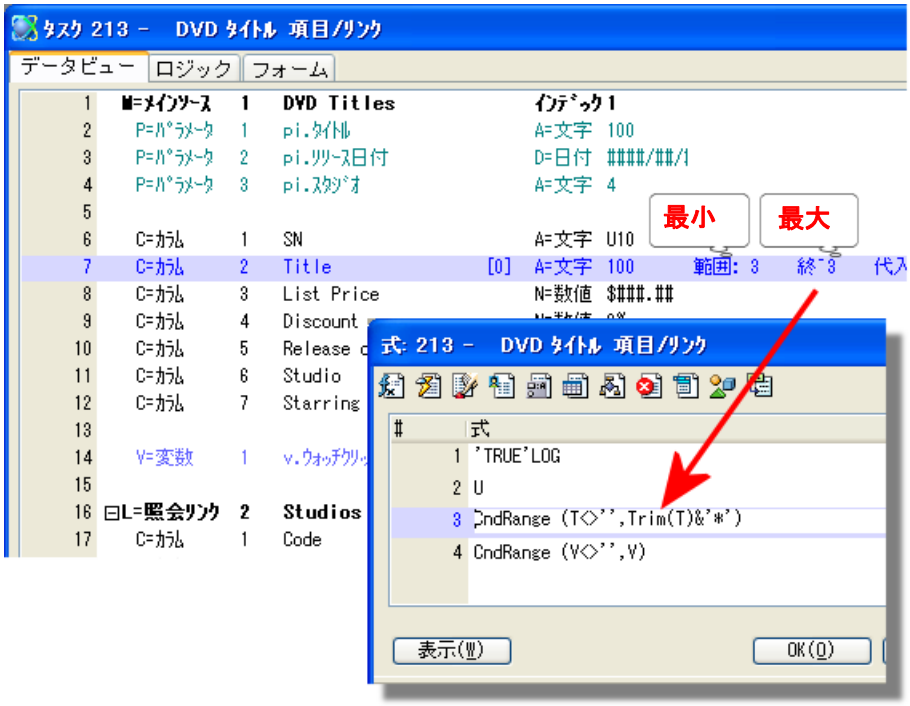
しかし、通常はすべてのレコードを表示することはありません。例えば、削除フラグが設定されたレコードを表示しないようにしたり、代理人による販売内容のみ表示させるようにすることができます。また、1 つのユニークなレコードを検索する場合もあります。

データビューでレコード数を制限するには、いくつかの方法があります。範囲カラムに直接範囲条件を設定する便利な方法もありますが、このようなオプションのほとんどは、範囲ウィンドウ（タスク→範囲 / 位置付または、*Ctrl+R*）にあります。

- 範囲：データ項目で範囲を定義します。
- 式：論理式で範囲を定義します。
- SQL Where 句 ：SQL Where 句を指定して範囲を定義します。

範囲設定についての詳細な方法を以下で説明します。

範囲の最大 / 最小を使用する

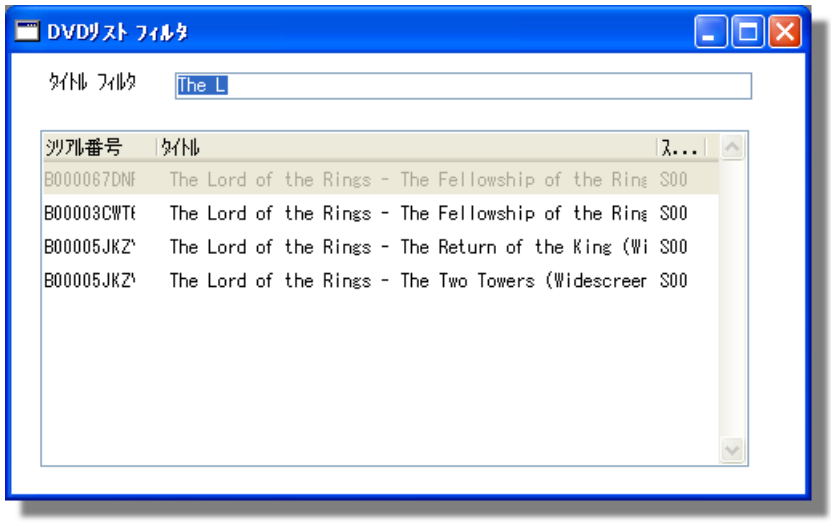


範囲指定の１番目の方法は、最大 / 最小の指定です。これは、最も簡単でよく利用される方法です。この場合、上限と下限のフィルタを指定し、指定された範囲内のレコードのみに絞ることを可能にします。文字型のカラムの場合、いくつかのマスク文字が指定できます。例えば、検索文字列の後に「*」が付加されている場合、移行の文字列を無視するように検索できます。

このような範囲指定方法は、一般的に、1 種類（同じ状態、同じ国、同じ親レコード ID）のレコードで絞り込んだり、最小と最大を同じ値に指定することで 1 つの特定のレコードを絞り込むために使用されます。

この例では、プログラムに渡された文字列で始まっているすべてのレコードを検索しています。The L という文字列で始まるレコードが存在する場合、結果として表示されるレコードには、映画「The Load of the Rings」の全てのレコードが含まれています。

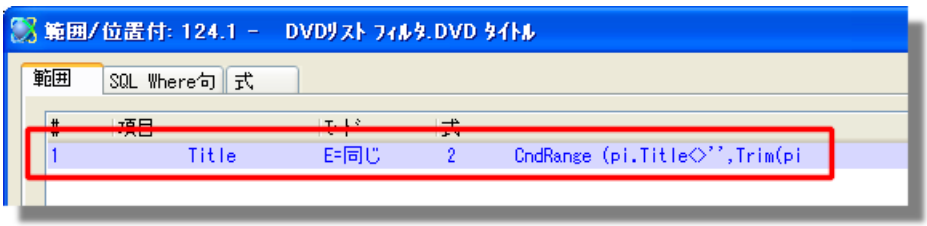
しかし、この方法では、文字列の中に存在するテキストを検索したり、より複雑な条件で検索することはできません。このような場合は、範囲式を使用する必要があります。



範囲指定を異なる場所から確認する

便宜上、範囲の最小 / 最大指定は、いくつか異なる場所で参照することができます。ここには、上記で設定された範囲条件をそれぞれの場所で表示させています。

データビュー内に表示される範囲カラム

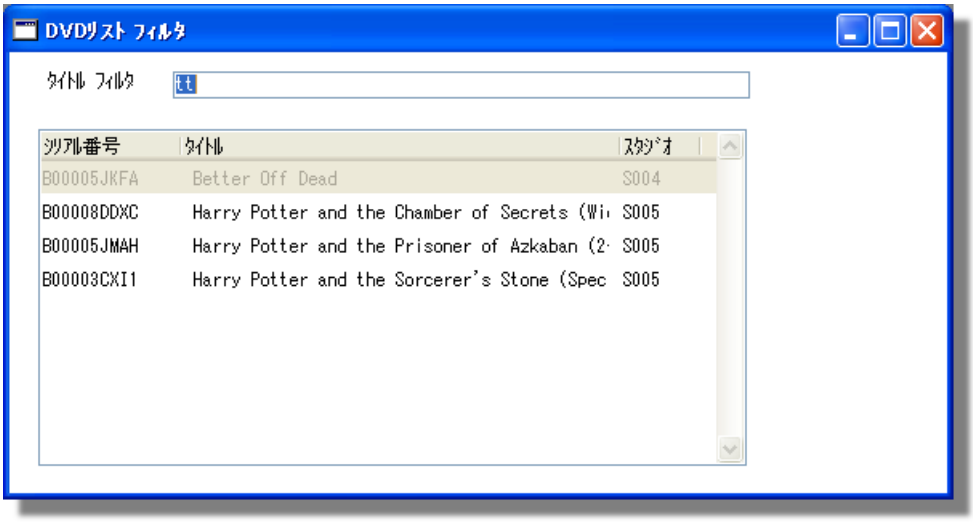


範囲ウィンドウ （*Ctrl+R*）で同じ範囲条件が指定されています。

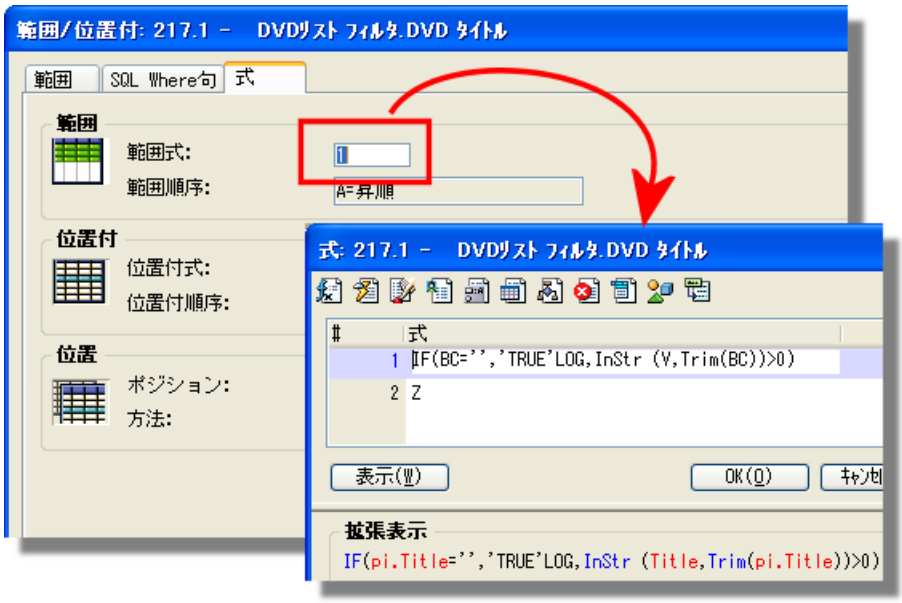
カラム特性 （*Alt+Enter*） での範囲指定

どのオプションを使用しても構いません。どこで指定しても同じ動作になります。

範囲式を使用する

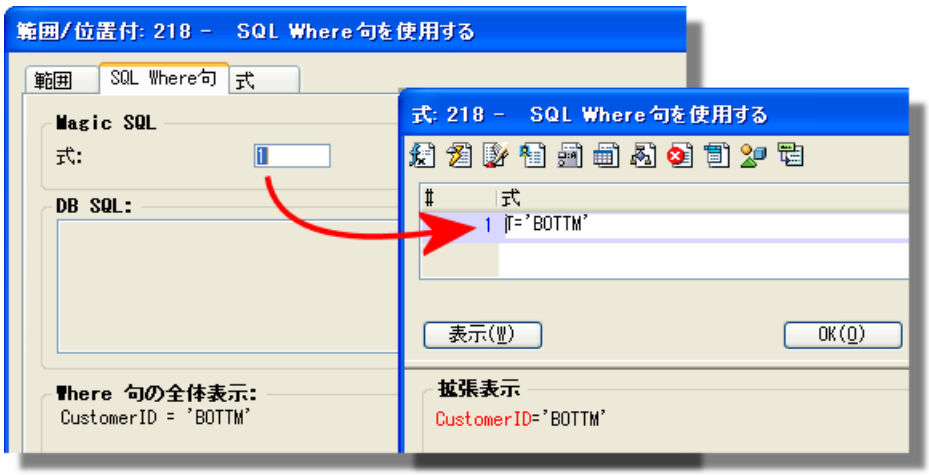


範囲式オプションを使用することで、より柔軟な範囲設定を行うことができます。この方法では、任意の式を入力することができ、式が True に評価された場合、レコードは選択されます。この例では、指定された値が文字列のどこに存在していても、True が返るような式を指定しています。



上記の例には、**範囲式**特性（**範囲 / 位置付→式タブ**）に入力された式が表示されています。パラメータが空白の場合、式は常に True を返すため、すべてのレコードが表示されます。空白でない場合、指定された文字列が現在のレコードの文字列内に存在するかどうかをチェックするために、**InStr()** 関数を使用しています。存在している場合、**InStr()** 関数は正の数値（0 より大きい数値）を返すため、該当するレコードはビューに含められます。

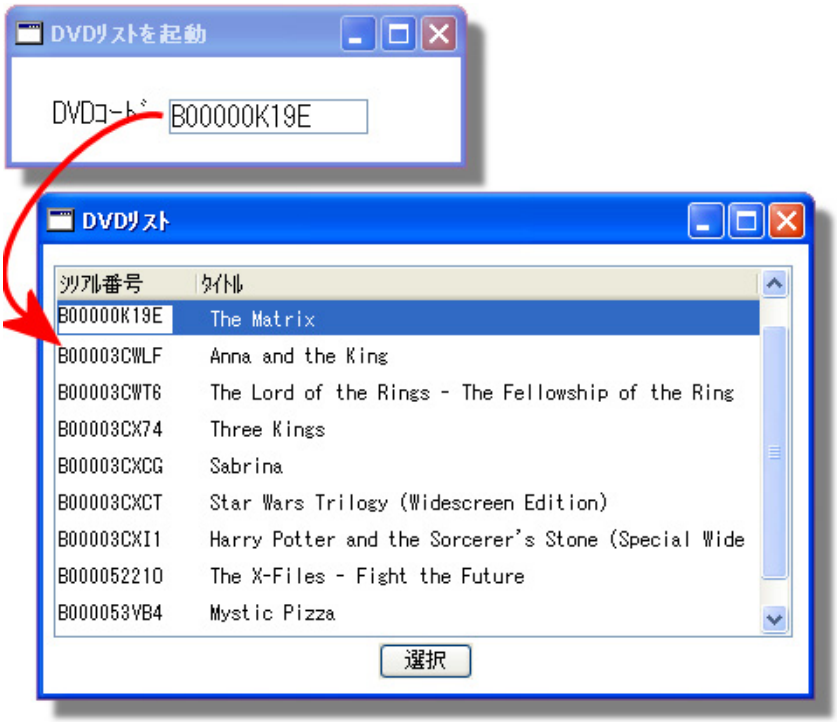
SQL Where 句を使用する



SQL テーブルを使用している場合、通常の範囲指定（最大 / 最小）オプションは実行時に SQL ステートメントに変換されます。しかし、SQL コードを **SQL Where** タブの **DB SQL** 特性に入力することもできます。この方法の不利な点は、SQL コードが使用する DBMS に依存する場合があるということです。このため、DBMS を切り換えて使用する場合、移植性がなくなる可能性が発生します。

その代わりに **Magic SQL 式**を入力することができます。この場合、実行時に使用する DBMS 用の SQL コードに変換されます。

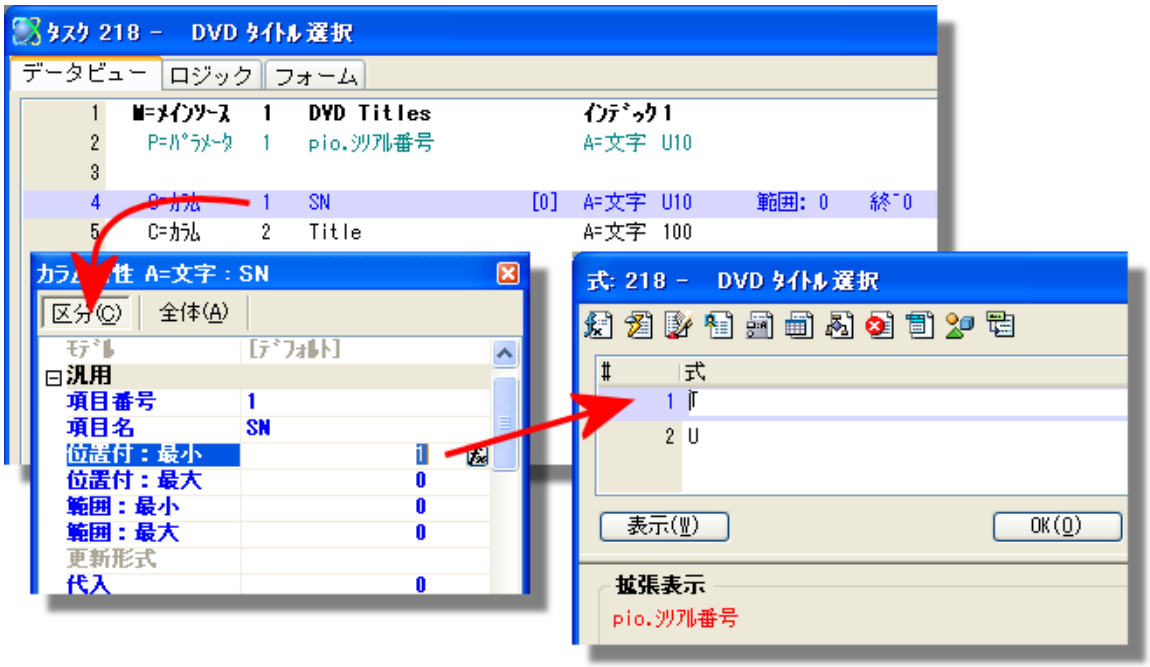
タスク起動時に特定のレコードに位置付けるには



テーブルを表示する際に、任意のレコードに位置付けたい場合があります。この例では、DVD タイトルを入力する項目でダブルクリックすると、カラムがそのそのタイトルに移動します。

これは、位置付特性を使用することで実行されます。範囲特性が、表示されるレコード数を限定する処理であるのに対し、位置付特性は現在のレコード位置が変わるだけです。

位置付特性を使用する



1. データビューエディタの位置付け条件を設定したいカラムにカーソルを移動します。この例では、シリアル番号（SN）カラムを表示し、パラメータとして渡されたシリアル番号と同じ番号の最初の DVD にレコードが位置付くようにしています。
2. *Alt+Enter* を押下して、カラム特性を開き、位置付 最小特性に移動します。
3. 位置付 最小特性でズームして式エディタを開きます。ここに、位置付け条件として評価される式を入力します。

これにより、プログラムが起動されると**位置付**特性で指定された値と同じ値を持つ最初のレコード上にカーソルが位置付きます。該当するレコードが存在しない場合、その次に相当するレコードに位置付きます。

位置付順序の効果

位置付特性のデフォルト設定は ***A= 昇順***です。従って、この例では、検索処理はレコードの先頭から最後まで実行されます。一致した最初のレコードに位置付くように、**位置付 最小**特性を使用しています。

しかし、**位置付順序**特性（**範囲 / 位置付→式タブ**）で降順を設定した場合、検索処理はレコードの最後から先頭方向に実行されます。この場合、**位置付 最小**特性に位置付条件を設定する必要があります。

位置付の最小 / 最大の両方を使用する

位置付 最小と**位置付 最大**の各特性に位置付け条件を設定した場合、レコードが見つからないと、エラーメッセージとして「レコードが見つかりません - 最初に位置付けます」が表示されます。

位置付式を使用する

レコードを検索するために論理式を使用する場合、**位置付式**特性（**範囲 / 位置付→式タブ**）に入力することができます。この場合、範囲式（「範囲式を使用する」（445 ページ）を参照してください）と同じように動作します。

テーブルの中央に位置付ける

デフォルトで、位置付けされたレコードはテーブルの先頭行として表示されます。この場合、上の行にレコードが存在しないものと受け取られる場合があります。しかし、**動作環境**ダイアログ（**オプション→設定→動作環境**）の**カーソルの画面中央位置付**（**動作設定**タブ）を ***Yes*** に設定することで、位置付けレコードがテーブルの中央に表示されるようになります。

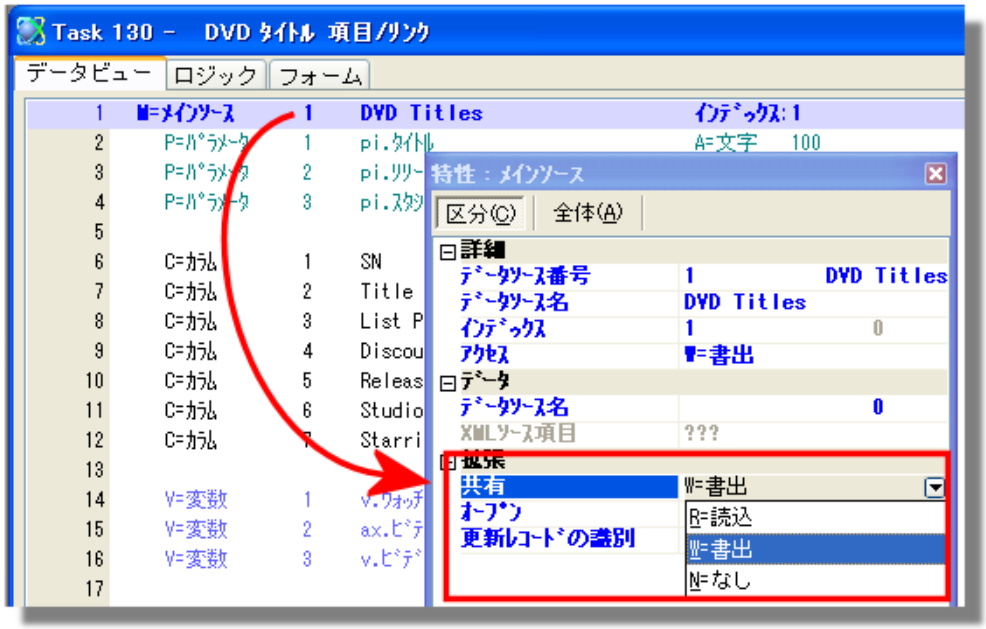
読み込み専用のテーブルにアクセスするには

テーブルを使用している時間のほとんどは、更新することがなく、他のテーブルの更新や表示、印刷を行うためにデータを読み込んでいるだけの場合が多いはずです。テーブルを更新する必要がない処理であれば、読み込み専用でテーブルをオープンすることを推奨します。これによって、レコードが競合する可能性を減らし、プログラムの処理が早くなります。

データソースを読み込み専用でオープンする

1. データビューエディタのメインソースまたはリンクテーブルのヘッダ行に移動します。
2. *Alt+Enter* を押下して、特性シートを開きます。
3. アクセス特性を、*R= 読込*に設定します。

データソースを複数のユーザ間でアクセスさせるには



複数のユーザがテーブルにアクセスしている時に、DBMS は Magic と連携して、通常は適切にレコードレベルでの処理を行います。すなわち、２人のユーザが、同時に同じレコードにアクセスしようとした場合、どちらか一方はロックアウトされ、エラーメッセージが表示されますが、テーブル内のデータが破損するようことはありません。

しかし、更新中に誰もテーブルにアクセスして欲しくない場合は、多少設定が必要です。例えば、経理用のテーブルを使用して月末調整を行っている場合や、古いレコードをの保管処理を行っているような場合などが、このようなケースに該当します。テーブルの**共有**特性を設定することによって、これらの処理を行うことができます。

共有特性には以下のオプションがあります。

- ***W= 書込*** : このタスクの実行中は、他のタスクでこのテーブルを書込モードでオープンすることができます。
- ***R= 読込*** : このタスクの実行中は、他のタスクでこのテーブルを読込モードでオープンすることができます。ただし、書込モードではオープンできません。
- ***N= なし*** : このタスクの実行中は、他のタスクでこのテーブルをオープンすることができません。

この特性は DBMS レベルで実行されるため、DBMS が Magic 以外のプログラムによってアクセスされる場合、これらのプログラムに対しても影響します。

タスクのデータ項目の値を設定するには

Magic のプログラムでデータ項目の内容を更新するには、いくつかの方法があります。これらの方法は、以下の種類で分けることができます。

- レコード作成時に自動的に設定される値。この場合は、**デフォルト値**特性や**代入**特性を使用します。
- プログラムの処理コマンドを使用して設定される値。この場合は、**項目更新**処理コマンドや **VarSet()** 関数、およびパラメータとして値を渡す方法があります。
- ユーザが入力したり、コントロールを操作することによって更新される値。

以下では、各更新方法についての詳細が説明されています。

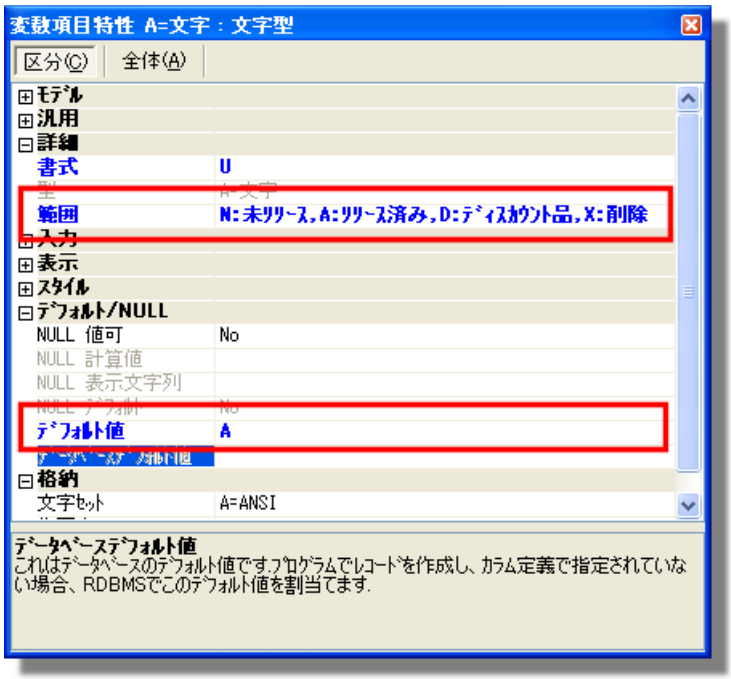
自動的に値を設定する

デフォルト値特性を使用する

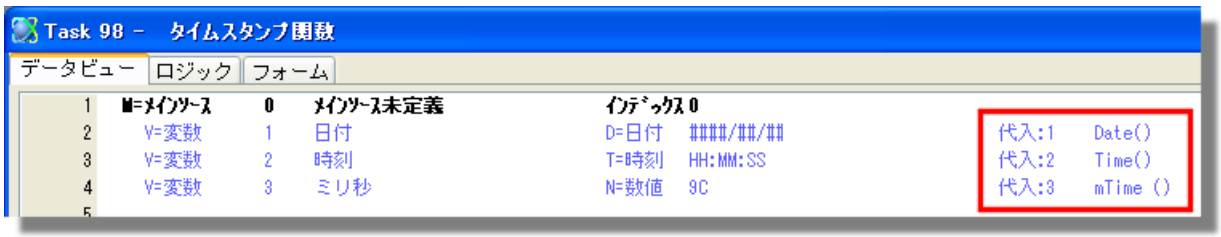
どのデータ項目にに対しても**デフォルト値**特性を設定することができます。この特性は、どのレベル（モデル、データソース、タスク）で指定できます。項目にデフォルト値が設定されている場合、ユーザがレコードを参照する前に、自動的にこの値が設定されます。

範囲特性は、この項目に入力できる値を制限する場合に指定します。この例では、有効な値は *'N'*、*'A'*、*'D'*、および *'X'* になります。

デフォルト値特性は、最初にオープンされた場合に初期設定する値を指定します。



代入特性を使用する



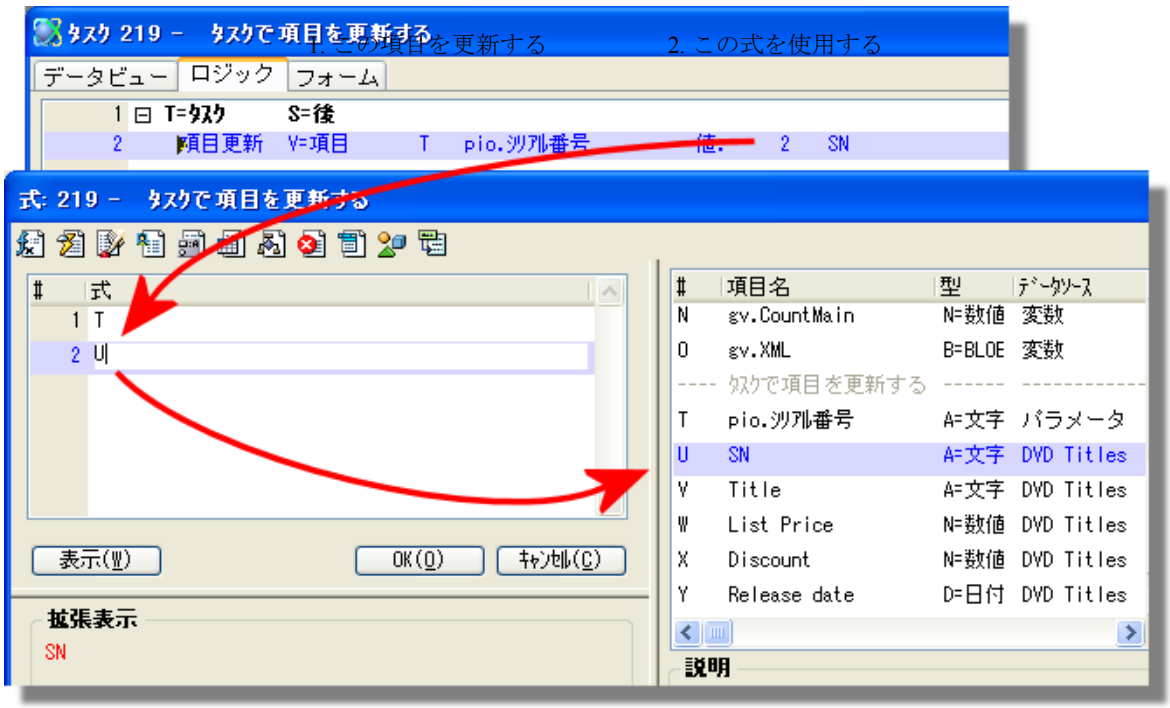
項目の**デフォルト値**特性に加え、**データビューエディタ**の項目定義の行の**代入**カラムを使用することもできます。

代入は**デフォルト値**とは少し異なった動作になります。

代入	デフォルト値
項目の初期設定	項目の初期設定
式で指定	値を直接入力
代入式の評価結果が変更されると、再計算される。	指定された値のみ有効

タスクで項目を更新する

項目処理コマンドを使用する



タスク内で項目を更新する最も一般的な方法は、**項目更新**処理コマンドを使用することです。この処理コマンドは、どのような**ロジックユニット**内でも使用できます。

項目更新処理コマンドの構文は上の図に表示されているとおりです。また、以下のようにも表すこともできます。

項目更新 〈項目〉 値 〈更新式〉 条件 〈条件式〉

ブラケット内の入力カラムでは、ズームすることで各々、項目一覧や**式エディタ**にアクセスすることができます。

VARSet() 関数を使用する

VarSet() 関数は、式の中で直接項目の値を更新することのできる関数です。様々な目的で使用することができます。関数の構文は以下の通りです。

VARSet(*variable*, *value*)

パラメータ：

- *variable*：更新対象の項目を **VAR** リテラルで指定します。**VAR** リテラルは、項目番号が P であれば、'P'VAR というように指定します。項目の定義位置が変更されても（'P' が 'Q' に変更されても）式の中で自動的に変更されます。
- *value* ：更新する値か更新値として評価される式を指定します。値を直接指定したり、項目を指定したり、関数を使用することもできます。

例：**VARSet(*'P'VAR, X+6*)**

項目 'P' の値を、項目 X の値 +6 の結果で更新します。

VarSet() 関数を使用すると、複数の項目を配列のように扱うことができます。例えば以下のような式を設定します。

VARSet(*'P'VAR+1, X+6*)

項目 'Q'（'P' の次）の値を、項目 X の値 +6 の結果で更新します。

ユーザによって項目を更新する

ユーザによって項目の値を変更することもできます。ユーザによって変更可能な項目は、**変更可**特性や**パーク可**特性が ***Yes*** に設定されているコントロールとしてフォームに配置されている必要があります。

1 つのタスク内で複数のテーブルからデータを検索するには

Magic でデータソースを利用する場合、以下の 2 つの方法でデータを取得することができます。

- メインソースから取得
- リンクテーブルから取得

メインソースは、タスク内で繰り返しアクセスすることのできるデータソースです。メインソースから取得されたデータはフォーム上のテーブルコントロールに表示させることができ、1 つのレコードやテーブル全体を表示させることができます。1 つのタスクに対して 1 つのメインソースのみが定義できます。

リンクテーブルは、1 度 1 レコードのみ表示することができます。リンクテーブルは、一般に、現在表示されているレコードに関連する情報を表示させるために使用されます。1 つのタスクには、必要に応じて複数のリンクテーブルを定義することができます。

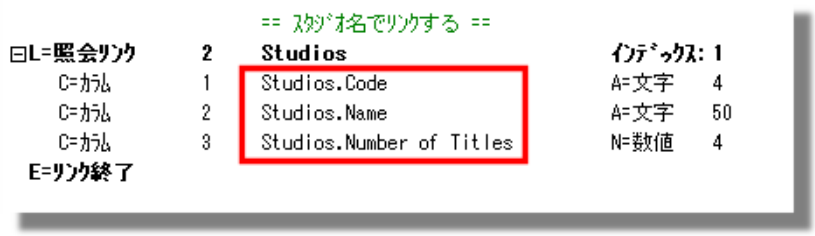
注：　ユーザに対して表示されるフォームに 1 つのテーブルしか表示できないという意味ではありません。1 つのタスクに対して 1 つのメインソースしか定義できませんが、サブフォームを使用することでフォーム上に複数のテーブルを表示させることができます。サブフォームは、親タスクの一部のように表示されますが、実際はサブタスクであり、これ自身でメインソースを定義することができます。

リンクコマンドを使用してリンクテーブルからデータを取得することができます。以下はこの方法について説明しています。

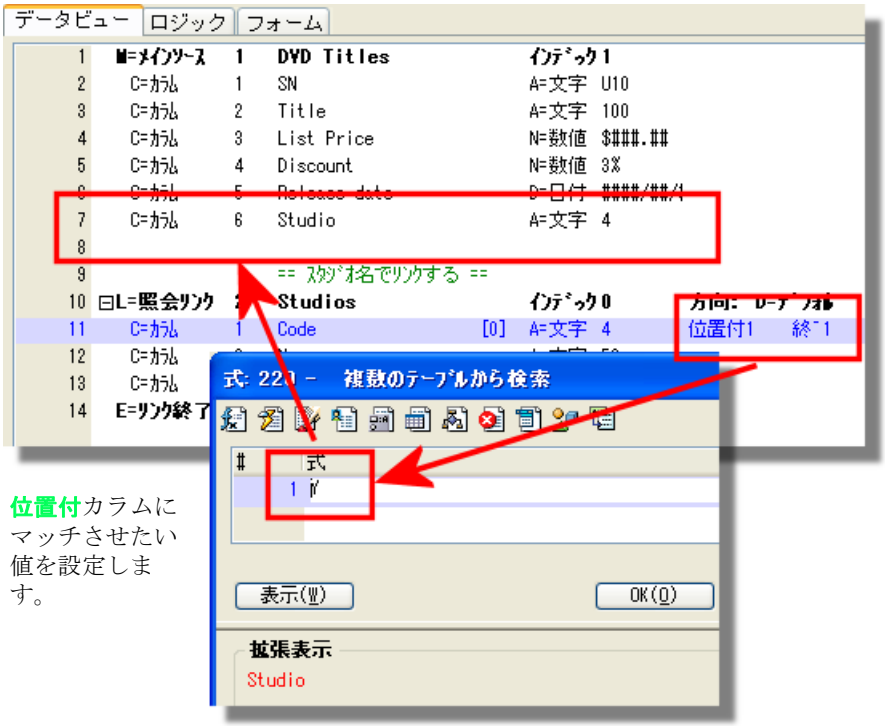
リンクコマンドを使用する

1. 最初に、**リンク / リンク終了**のペアの行を作成します。***F4*** を押下して 1 行追加します。***L*** を入力します。***照会リンク***と***リンク終了***の 2 つの行が作成されます。リンクさせたいデータソースを選択します。ここでは、#2 のデータソース（Studio テーブル）を選択しています。データソース番号を入力するか、ズームして一覧から選択できます。
2. 次に、**インデックス**カラムに ***Tab*** 移動します。インデックスは、テーブルの検索順序を決定します。指定するインデックスが、レコードの検索条件に合っているかどうかは重要な要因になります。例えば、この例では、インデックスがスタジオコードのため、スタジオコードを使用して位置付けしなければなりません。
   インデックスを選択すると、インデックスを構成するセグメントカラムが自動的に追加されます。
3. **位置付**カラムを設定します（設定方法は、次のセクションを参照してください）。
4. 最後に、リンクカラムとして含めたいカラムを選択します。選択するには以下のようにします。
   - ***F4*** を押下して 1 行追加します。項目のタイプは自動的に***カラム***になります。次のカラムに ***Tab*** 移動します。
   - ズームしてカラムを選択するか、カラム番号を直接入力します。
   - 必要に応じてカラム名を変更することができます。

ヒント：*カラム名を変更することで、どの項目がどのデータソースからリンクされたものなのかをわかりやすくすることができます。*



リンクの位置付カラムを設定する

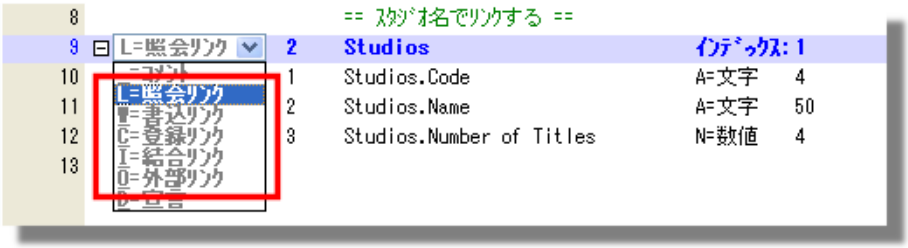


位置付カラムにマッチさせたい値を設定します。

リンクの**位置付**カラムは、SQL 系データベースの Select Where 句のように動作します。**位置付**と表示されているカラムは、位置付け条件の最小値を指定し、**終了**と表示されているカラムには、最大値を指定します。この例のように両方とも同じ値が設定されている場合、指定された条件値のレコードのみを取得しようとします。

従って、この例では、DVD タイトルに定義されているスタジオコードと合ったレコードが取得されます。

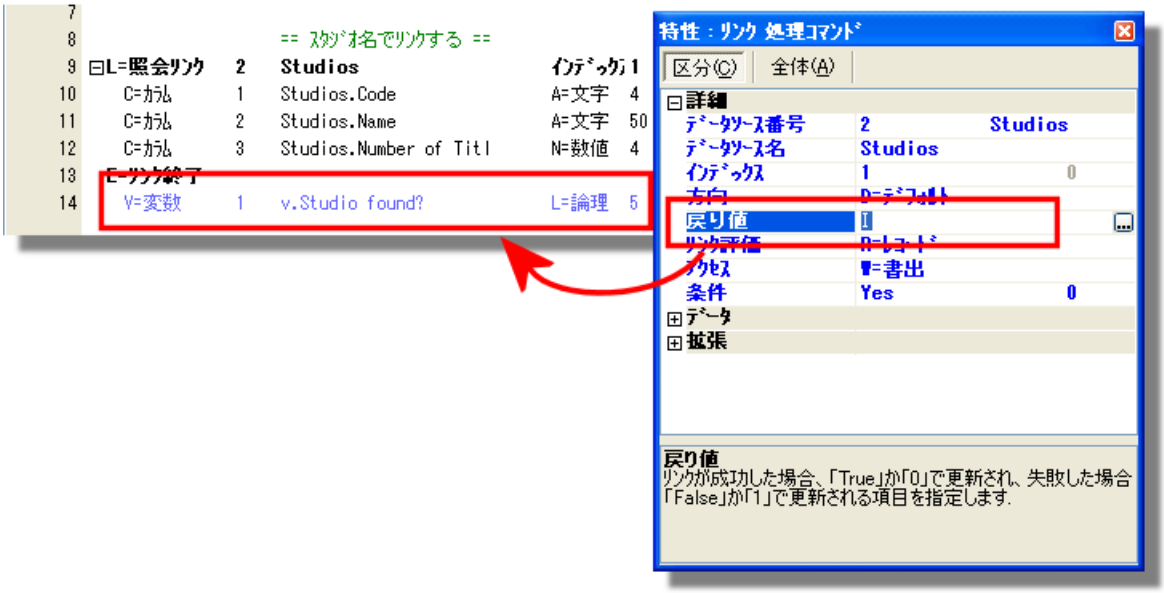
リンクの種類を設定する



リンクにはいくつかの異なるタイプがあり、それぞれ異なる動作をします。これらはすべて、現在のタスクにカラム項目を定義するものですが、書込リンクと登録リンクはレコードの作成も行います。以下は、各リンクのタイプとその特徴です。

リンクの種類	説明	利用目的
照会リンク	既存のレコードを取得します。	既存のレコードを取得したり、レコードが存在しているかどうかをチェックします。 例えば、顧客コードが顧客テーブル内に存在していない場合に、エラーメッセージを表示させたい場合、照会リンクを使用します。
書込リンク	既存のレコードを取得しようとします。レコードが存在しない場合は、そのレコードを作成します。	レコードの存在が不確定で、存在しなければ作成したい場合に使用します。 例えば、電話番号を作成したい場合に、入力された電話番号がすでに存在していなければ、自動的に登録するようにできます。
登録リンク	レコードを作成しようとします。そのレコードが存在しているかどうかは確認しません。	レコードが存在していない場合は、書込リンクより処理が早くなります。 例えば、ユーザが一定の画面を開いている場合に、常にログレコードを作成したいのであれば、ユーザ ID や日付、およびタイムスタンプを保存するためにリンク作成が使用できます。
結合リンク	データソースが両方とも SQL テーブルの場合、SQL の内部結合を実行します。	SQL テーブルを使用して内部結合を使用する場合は、SQL に対してより早くリンク処理が実行されます。
外部結合リンク	データソースが両方とも SQL テーブルの場合、SQL 外部結合を実行します。	SQL テーブルを使用して外部結合を使用する場合は、SQL に対してより早くリンク処理が実行されます。

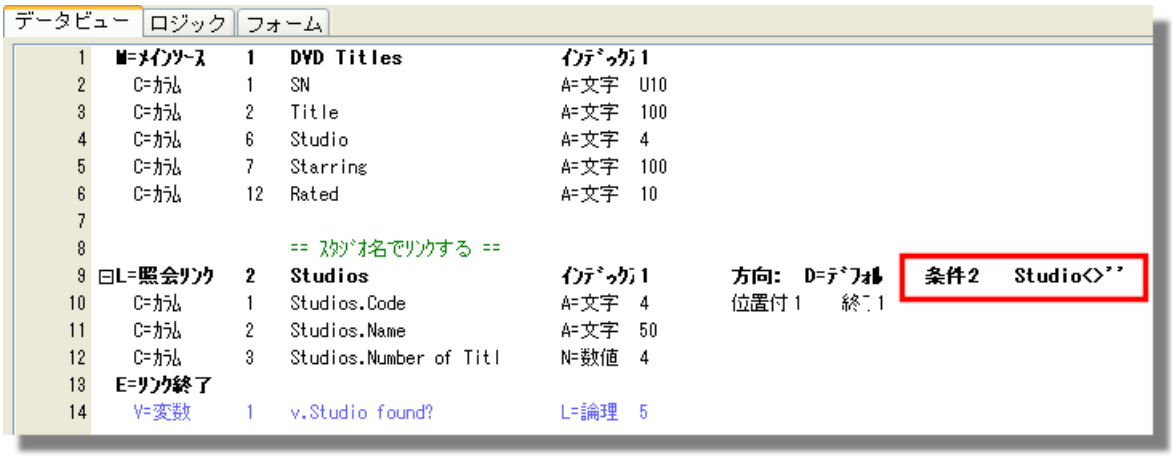
リンクの戻り値特性を設定する



リンクコマンドによって適合したレコードが存在したいるか否かは、戻り値のコードで確認できます。この値は、戻り値特性に設定されます。戻り値特性は、一般的にユーザによって入力されたデータの有効性を評価するために使用されます。

1. 論理型の変数項目を定義します（数値型も使用できますが、論理型の方が保守が容易です）。
2. 戻り値特性でズームしてこの変数項目を選択します。

リンク条件を設定する



リンクコマンドの動作を条件特性で制御することができます。リンク特性には、*Yes* または *No* を設定するか、式で設定できます。*No* または、実行時に式が False と評価された場合、リンクは実行されません。この例では、スタジオコードが空白の場合リンクが実行されない条件が設定されています。

リンクカラムの範囲を使用する

リンクカラムには、メインソースのカラムと同じように範囲条件を指定することができます。メインソースのカラムを使用することでデータビューとして展開するレコード数を制限することができます。例えば、メインソースのカラムで以下のような範囲設定を行った場合

- 範囲最小：*'2001/01/01'DATE*
- 範囲最大：*'2001/12/31'DATE*

これにより、2001/01/01 から 2001/12/31 までの日付で範囲が設定され、この範囲内のレコードのみ表示されます。

リンクテーブルに対して範囲指定を行った場合もそんなに明確ではありませんが、同じように動作します。すなわち、リンクテーブルで以下のように日付の範囲設定を行った場合。

- **範囲最小**：*'2001/01/01'DATE*
- **範囲最大**：*'2001/12/31'DATE*

リンクレコードにこの範囲の日付データが存在する場合のみ、メインソースのレコードが表示されるようになります。

これは便利な機能ではありますが、**範囲**特性に位置付用の値を設定しないように注意する必要があります。この 2 つの特性は、異なった動作をします。

テーブルの最初または最後のレコードを取得するには

タスク内で 1 つのレコード（テーブルの最初または最後）のみを取得する必要があるかもしれません。例えば特定のユーザの最後のログイン時間を更新したり、テーブル内の最大のレコード番号を取得したりする必要がある場合などです。

これらのことを行うには、以下のような手順でロジックを組み込みます。対象がメインソースの場合とリンクテーブルの場合では、方法が異なります。

メインソースの最初または最後のレコードの取得

1. インデックスを定義する

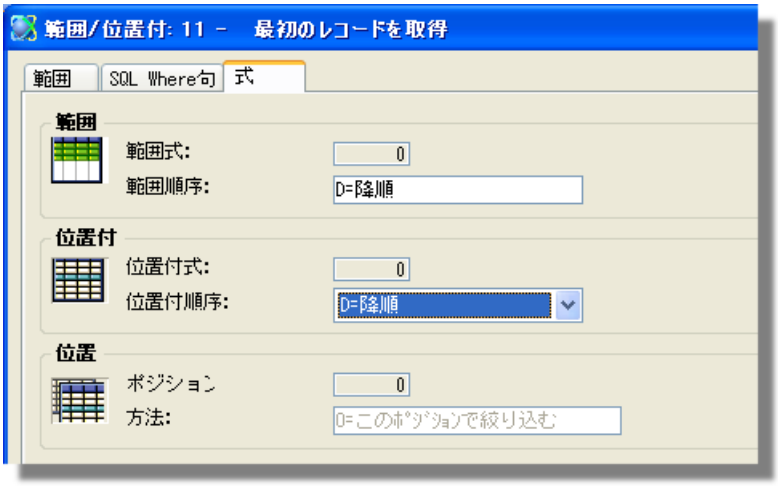
最初または最後のレコードを取得する場合に最初に考えることは、何をもとに最初と最後にするかということです。これは、テーブルには複数のインデックスが定義されている場合があるからです。例えば、顧客テーブルにおいて最初の顧客は、顧客 ID が最も小さい場合であったり、顧客名がアイウエオ順で先頭であったり、最も小さな郵便番号を持つ顧客であるなど色々考えられます。

Magic では、メインソース定義でインデックスを選択することでどのインデックスを使用しているかを決めることができます。

しかし、要求される検索順に対応したインデックスが定義されていない場合は、タスクのソートテーブルを使用することで実行時にレコードのソート処理が実行されます。

最初や最後のレコードを取得する場合は、データビューに範囲や位置付を指定する必要がありません。

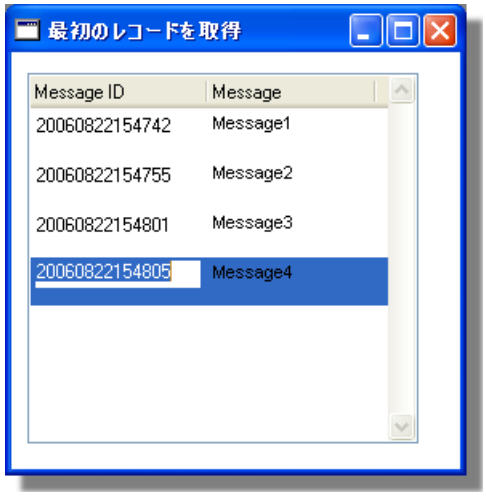
2. 検索順を設定する



次に、最初か最後かを決める必要があります。最初のレコードを取得する場合は何もする必要はありません。しかし、最後のレコードを取得したい場合は、テーブルの最後からレコードが並ぶように指定する必要があります（テーブルを最後まで検索するには非効率なためです）。

最後のレコードを取得するには、**範囲順序**か**位置付順序**（タスク環境→範囲／位置付→式）を ***D= 降順***に指定します。

範囲順序と位置付順序

範囲順	位置付順
	
テーブルは逆の順序で検索されるため、メッセージ 4 が先頭になります。	テーブルの表示順は昇順ですが、カーソルが最後のレコードにパークします。

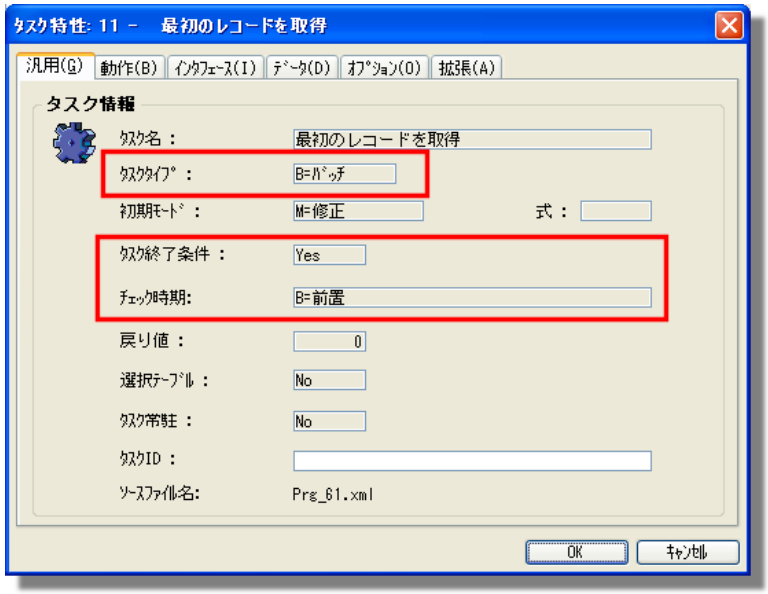
最後のレコードを取得する場合、**範囲順序**か**位置付順序**のどちらかを指定して行います。これらの機能は上図のように異なります。オンラインタスクの場合は、違いが明白になりますが、1 サイクルだけのバッチタスクとして実行すると同じ結果になります。

しかし、このような場合は、両方を降順にした方が理解し易くなります。

3. サイクル数をチェックする

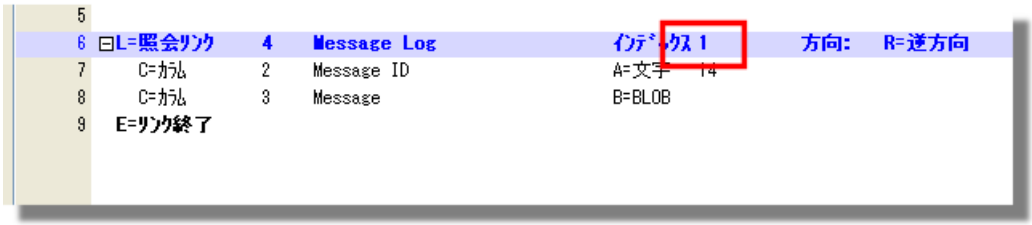
1 レコードのみを取得する場合、一サイクルのみ処理するバッチタスクを使用することになります。この場合、**タスク特性**（*Ctrl+P*）の**汎用**タブで以下の特性値を設定します。

- **タスクタイプ** ：***B= バッチ***
- **タスク終了条件** ：***Yes***
- **チェック時期**：***A= 後置***



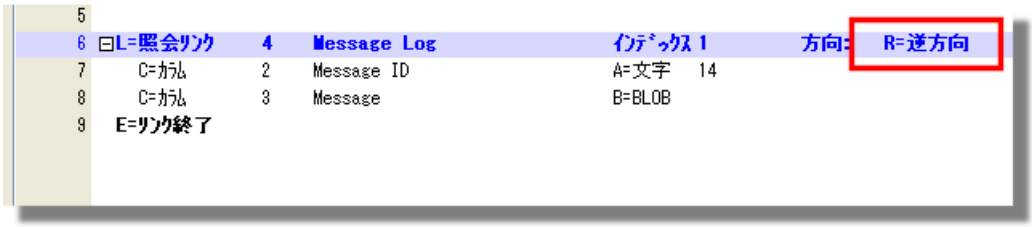
リンクテーブルの最初または最後のレコードの取得

1. インデックスを定義する



最初または最後のレコードを取得する場合に最初に考えることは、何をもとに最初と最後にするかということです。これは、テーブルには複数のインデックスが定義されている場合があるからです。例えば、顧客テーブルにおいて最初の顧客は、顧客 ID が最も小さい場合であったり、顧客名がアイウエオ順で先頭であったり、最も小さな郵便番号を持つ顧客であるなど色々考えられます。

2. 検索方向を設定する



次に、検索する方向を決める必要があります。最初のレコードを取得する場合は**方向**特性を ***D= デフォルト*** に設定します。最後のレコードを取得したい場合は、***R= 逆方向*** に設定します。

必要な設定はこれだけです。範囲や位置付の設定は必要ありません。

第 21 章： 式

式エディタ内で式の体裁を整えるには

式の内容によっては記述内容が長く複雑になる場合があります。例えば、ネストされた IF 文を定義する場合は以下のようになります。

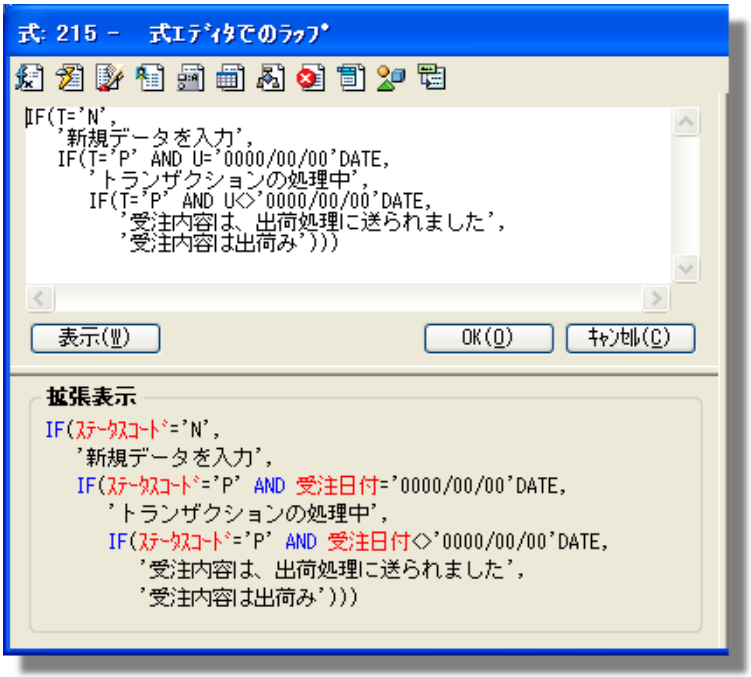
```
IF(I='N', 'Entering a new transaction', IF(I='P' and J='00/00/0000'DATE, 'Transaction is
being processed', IF(I='P' and J<>'00/0000'DATE,'Order has been sent to Shipping', 'Order
has shipped')))
```

構文上この式が正しくても、人が理解するには難しいものがあります。この式に改行や空白を追加することで、より理解しやすいものにすることができます。例えば以下のようになります。

```
IF(I='N',
   'Entering a new transaction',
    IF(I='P' and J='00/00/0000'DATE,
       'Transaction is being processed',
       IF(I='P' and J<>'00/0000'DATE,
         'Order has been sent to Shipping',
         'Order has shipped')))
```

ここでは、Magic の式エディタ内で式の体裁を整える方法について説明します。

式を改行入力する



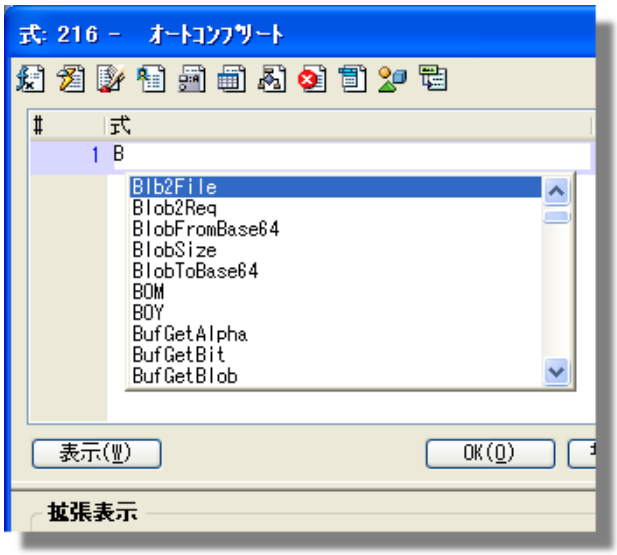
1. 式の定義欄から**ズーム**（*F5* または、*ダブルクリック*）して**式エディタ**を開きます。*Ctrl+E* を押下することでタスクのどこからでも開くこともできます。
2. *F6*（**編集→広域表示**）を押下します。この操作で現在パークしている式が拡張表示されます。
3. 広域表示されている場合、*Enter* を押下することで改行表示されます。また、インデントを加えるために空白や *Tab* を入力することもできます。
4. 編集が終了したら、OK ボタンや右上の  ボタンを**クリック**するか、*Esc* を２回押下します。*Enter* は改行として扱われるため、広域モード内では編集内容の確定処理には使用できません。

改行を入力すると式の中に CRLF コードが入力されます。これは Magic の式内であれば問題はありませんが、メッセージボックスなどでこの内容を出力すると改行されて出力されます。この機能は、開発者に対して式の内容を理解しやすくするためのものです。

関数名の入力を簡単に行うには

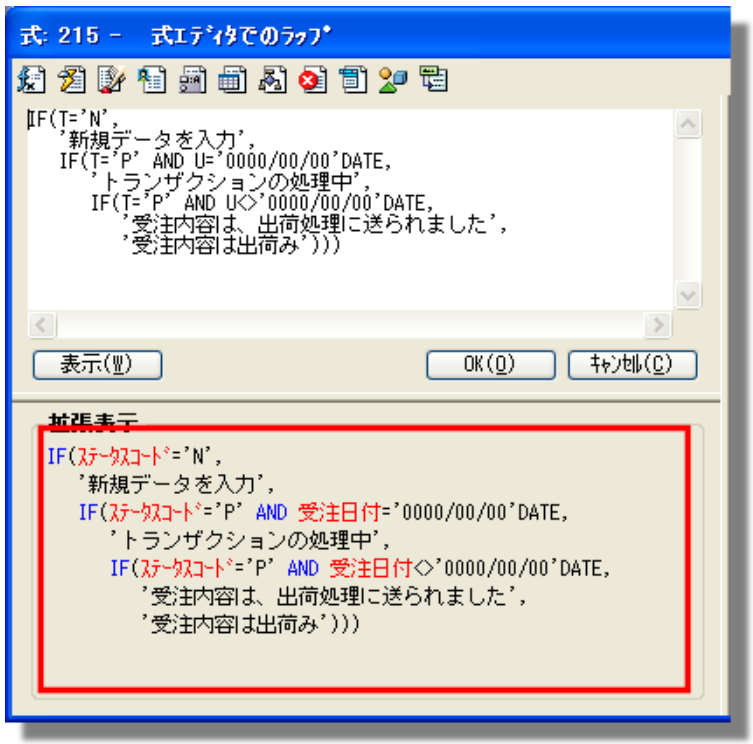
キー操作を減らすために、Magic には関数名に対してオートコンプリート機能が備わっています。関数名を入力する場合、入力操作のほとんどを Magic に依存させることができます。

オートコンプリート機能を使用する



1. 入力したい関数名の最初の数文字を入力します。この例では、B が入力されています。大文字小文字は区別しません。
2. ***Ctrl+Space*** を押下します。関数リストが表示され、名前の最初の文字が入力した文字と一致する関数に位置付けされます。
3. カーソルを下に動かして使用したい関数に位置付けます。この場合、カーソルキーを使用して動かすこともできますが、関数名を続けて入力することでも自動的にカーソルが移動します。
4. 使用したい関数名にカーソルが位置付いたら、***Enter*** を押下します。関数名が最初の括弧とともに追加されます。

式の拡張表示で色付けされる要素の色を設定するには

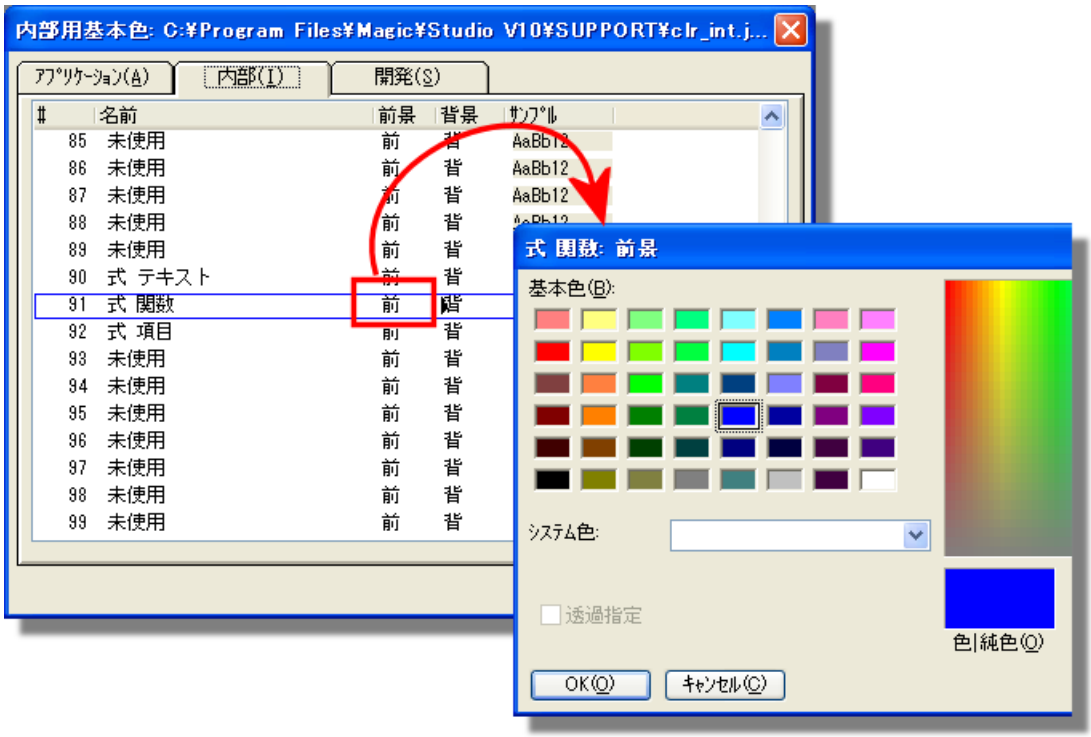


拡張表示領域に表示される式は、色分けされています。ここには以下の３つの色が使用されています。

- テキスト
- 関数名
- 項目名

これらの色は任意に設定することができます。ここでは設定方法について説明します。

式で使用する色を設定する

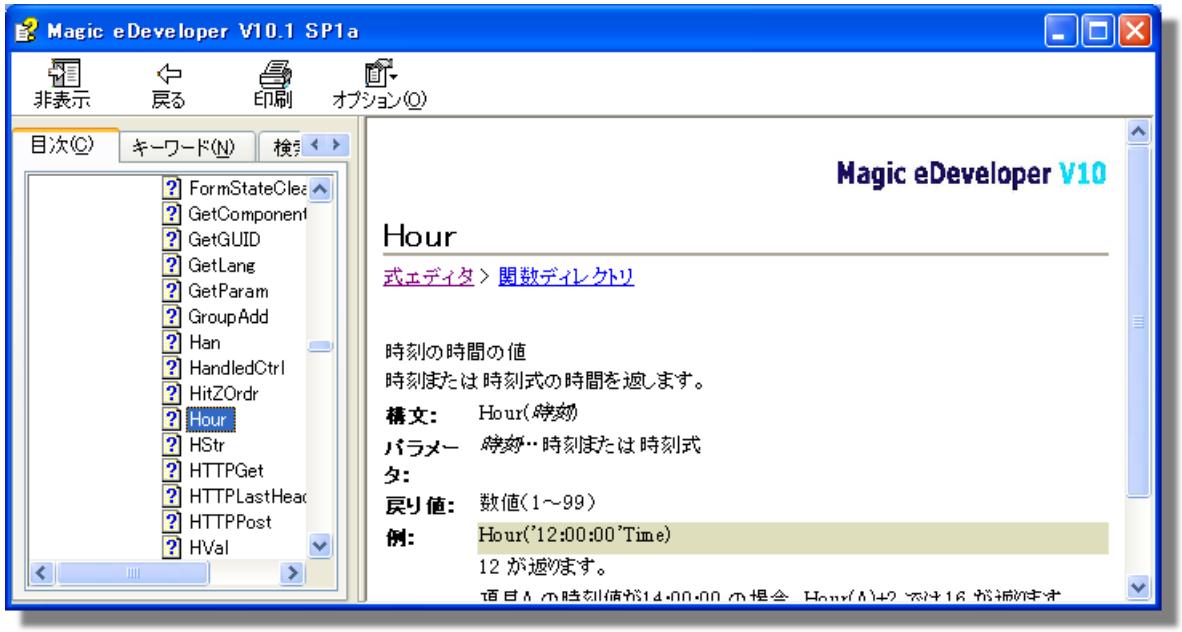


1. 基本色テーブル（オプション→設定→基本色）を開きます。

2. 内部タブをクリックします。
3. #90 ～ 92 の行に移動します。各色には、２つのカラム（前景と背景）があります。
4. 変更したいカラムに移動し、ズームします。基本色定義パレットが表示されます。ここで色を指定し、OK をクリックします。

基本色定義パレットには、２つの特別な色を指定するオプションがあります。透過指定は背景色でのみ指定できます。
システム色は Windows から色を引き継ぎます。

関数の関連ヘルプを表示させるには



どのような言語で開発する場合でも、全ての関数（メソッド）がどのように動作するものかをすべて覚えていることは難しいものです。Magic では、プログラムを作成中に該当するオブジェクトについて説明されたヘルプトピックを表示する機能があります。ヘルプを表示させるには、２つの方法があります。

任意の場所から関数のヘルプトピックを探す

開発環境ではどこからでも関数のヘルプトピックを表示させることができます。

1. *F1* を押下します。これによって現在のオブジェクトに対応したヘルプトピックが表示されます。
2. キーワードタブをクリックします。
3. 関数の名前を入力します。

入力した関数に対応するトピックが表示されます。また、検索タブを使用することでこの関数名を使用しているトピックを探すことができます。

ヒント : *開発作業中（特に Magic の勉強中）は、ヘルプを開いたままにしておくことで参照にかかる時間が早くなります。*

式エディタでヘルプを使用する

関数が含まれている式から、その関数に対応するヘルプトピックを表示させることができます。

1. 式エディタ（*Ctrl+E*）を開きます。
2. 関数の記述場所にカーソルを置きます。
3. *F1* を押下します。

これで、パークした関数に対するヘルプトピックが表示されます。

関数一覧からヘルプを表示する

関数を入力する便利な機能の１つに関数一覧があります。ここでは簡単にヘルプを表示させることができるだけでなく、関数を選択すると、括弧やパラメータのプレイスホルダが自動的に入り、関数入力が楽になります。

関数の入力方法については、「関数一覧から関数を選択する」（470 ページ）を参照してください。

長い式の編集を容易にするために式の入力行を拡張するには

式の編集中に、*F6* を押下すると編集領域が拡張表示されます。この状態の場合、ウィンドウには現在編集中の式のみ表示され、改行や空白の入力ができるようになります。詳細は、「式を改行入力する」（462 ページ）を参照してください。

広域モードの式をチェックするには

通常のモードで式を編集している場合、*Tab* を押下すると式がチェックされます。エラーが存在する場合、ビープ音が鳴り、ステータスラインにエラーメッセージが表示されます。これは、プログラムの構文チェックを行う前に式がチェックできるため、便利な機能です。

しかし、広域モードで編集している場合、*Tab* は式の中にタブコードを入力する処理になってしまいます。従って、広域モードの場合は、別の組み合わせキーとして *Ctrl+Enter*（または、**編集→式の確認**）を使用します。

文字型項目内で改行データを入力するには

式エディタ内で文字列を編集する場合、テキストエディタのように文字列を入力することができます。つまり、タブや改行および空白を入力することができることを意味しています。

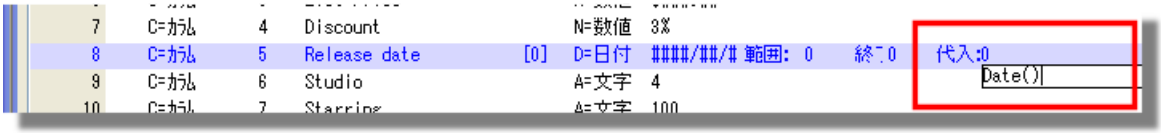
このようなテキスト入力を行うには、最初に広域モードにする必要があります。広域モードでは、小さなテキスト編集ボックスで式を編集することになります。詳細は、「式を改行入力する」（462 ページ）を参照してください。

タスクエディタで直接に式を編集するには

タスクのすべての式は**式エディタ**という 1 つのテーブルに定義されています。**式エディタ**を開くことで既存の式を探して再利用することができます。

しかし、簡単な式であればタスクの各エディタ上で、直接に入力することもできます。この機能を利用することで、短い式であれば入力する工数を削減することができます。

クイック式エディタを使用する



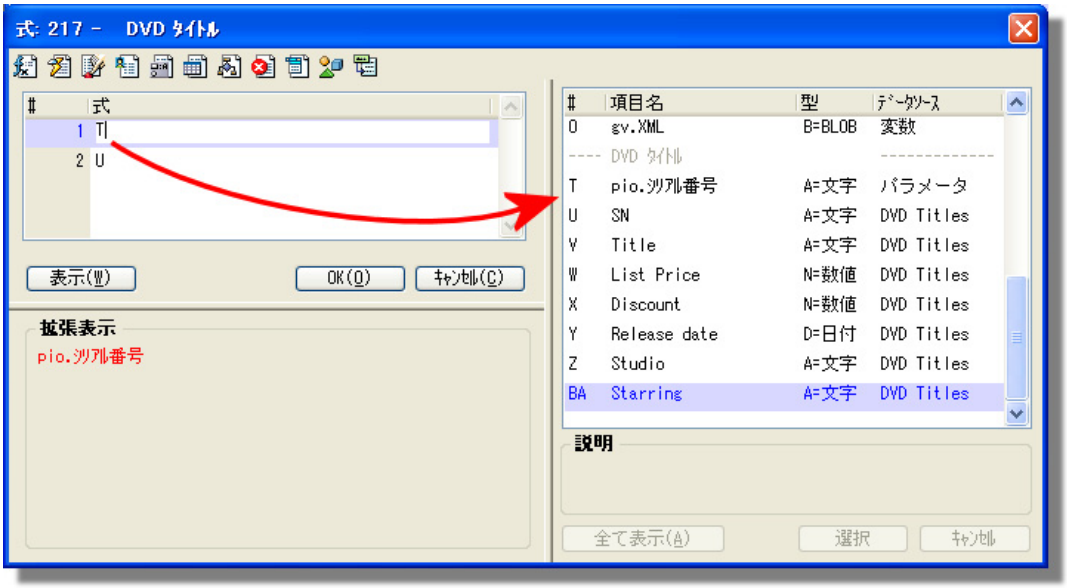
1. **データビューエディタ**や**ロジックエディタ**の**代入**特性や**条件**特性のような、式を定義する必要な場所に移動します。
2. 等号（=）を入力します。
3. 式の内容を入力します。
4. 入力を終了する場合、***Enter*** か ***Esc*** を押下すると式として保存され、***Ctrl+F2*** を押下すると入力内容がキャンセルされます。

入力した式が**式エディタ**に存在していない場合、その式は自動的に追加され、式番号が表示されます。既に存在している場合その式が使用されるようになります。

式エディタから項目一覧に移動するには

SP4 以降、式エディタのウィンドウの右側には、タスクがアクセスできる項目が一覧表示されています。ズームすることで項目一覧にカーソルを移動させることができます。

項目一覧からデータ項目を選択する



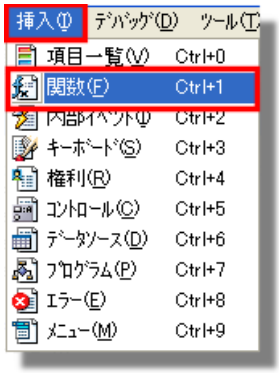
1. 式エディタの、項目を入力したい場所にカーソルを置きます。
2. *F5* を押下します。
3. 右側の項目一覧にカーソルが移動します。次のどちらかの方法で項目に位置付けます。
   - カーソルキーを使用して、カーソルを上下に移動します。
   - 位置付け機能（*Ctrl+L*）を使用して検索します。
   - 項目名の最初の文字を入力することで項目に位置付けます。

   式の入力行からマウスで直接、項目をクリックすることもできます。
4. 設定したい項目が見つかったら、*Enter* を押下するか選択ボタンをクリックすることで、式の中に項目が設定されます。

式エディタから関数一覧を開くには

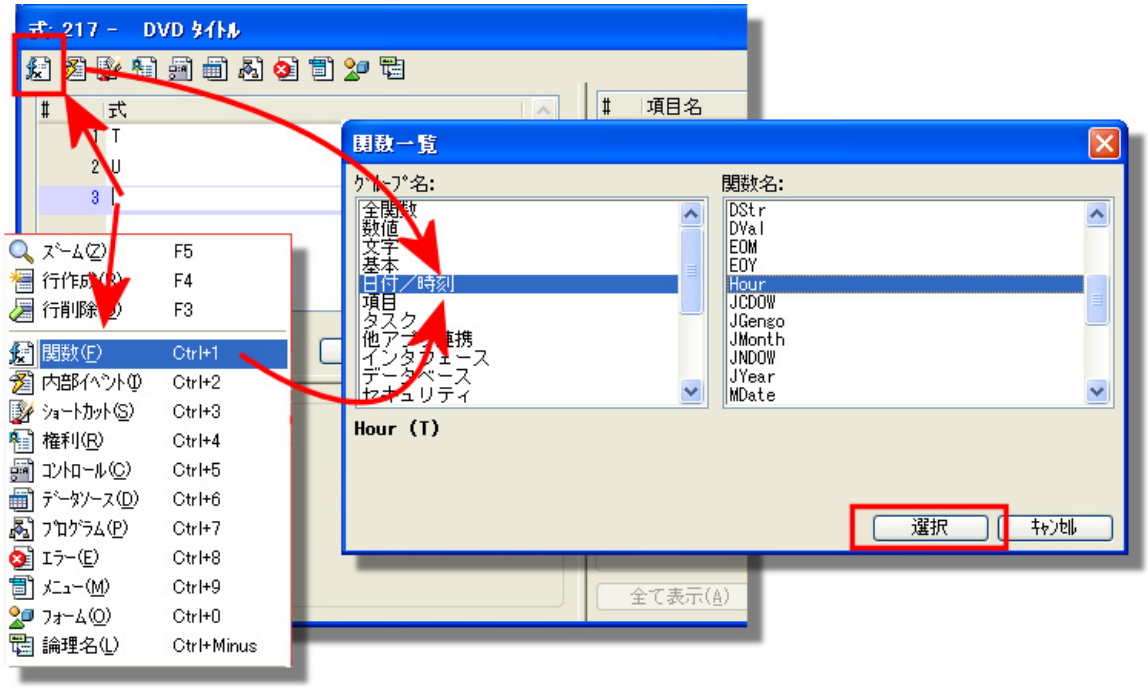
式エディタで編集中に、以下の操作を行うことで関数一覧を表示させることができます。

- *Ctrl+1* の押下
- 右クリックでコンテキストメニューを開き関数を選択
- プルダウンメニューから選択（挿入→関数）
- 式エディタに表示されているツールバーから  アイコンをクリック

コンテキストメニュー	挿入→関数
	

関数一覧を開くと、以下の手順で関数を選択することができます。

関数一覧から関数を選択する



1. 式エディタで、関数を設定したい場所にカーソルを置きます。
2. コンテキストメニュー（右クリック）を開き、関数を選択します。
3. 関数一覧では、グループ毎に関数名が表示されます。使用する関数名を知らない場合は、グループ名で機能を絞って表示させることで選択しやすくなります。左側のリストから区分を選択し、右側のリストから区分に関係する関数を選択することができます。
   関数名の先頭文字を入力することで関数に位置付けることもできます。

4. 関数名にカーソルがある状態で *F1* を押下すると、その関数に対応するヘルプトピックが表示されます。使用する関数がはっきりしない場合は、ヘルプを表示させて内容を確認しながらで探していく処理を繰り返すことになります。
5. 使用したい関数が見つかったら、*Enter* を押下するか選択ボタンをクリックします。
6. これで、指定された関数が式に追加されます。

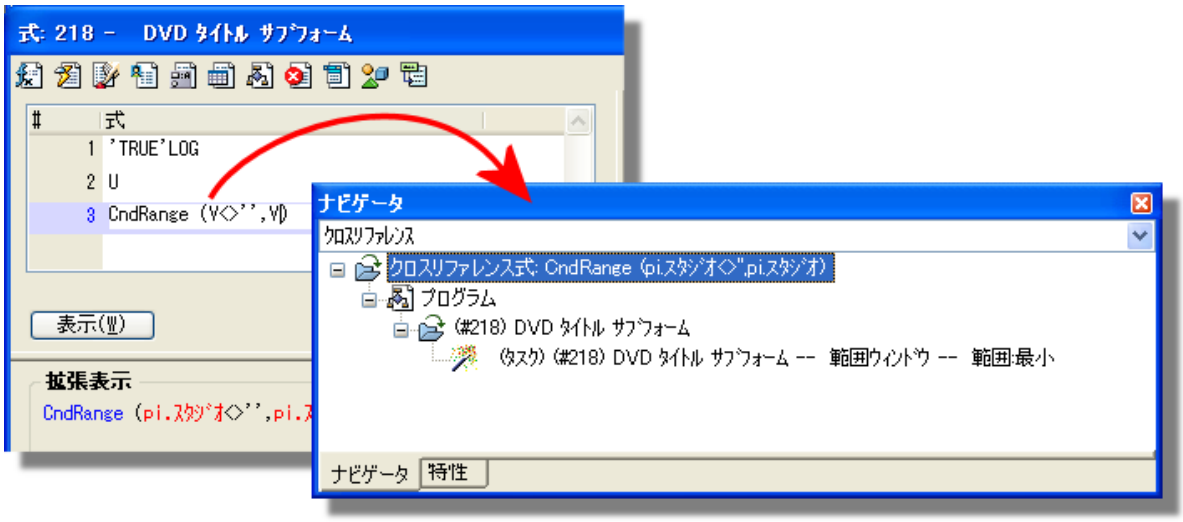
各選択一覧から選択する

関数以外の式で使用するオブジェクトを選択する場合も同じように、ツールバーやコンテキストメニューから一覧を開くことができます。

一覧名	ショートカット	アイコン
関数	*Ctrl+1*	
イベント	*Ctrl+2*	
ショートカット	*Ctrl+3*	
権利	*Ctrl+4*	
コントロール	*Ctrl+5*	
データソース	*Ctrl+6*	
プログラム	*Ctrl+7*	
エラー	*Ctrl+8*	
メニュー	*Ctrl+9*	
フォーム	*Ctrl+0*	
論理名	*Ctrl+-*（マイナス）	

式がどこで使用されているかを調べるには

式を変更する必要がある場合、それがどこで使用されるかを知る必要があります。クロスリファレンスユーティリティ（*Ctrl+F*）を使用することで、簡単に検索することができます。



式エディタ上でクロスリファレンスを使用する

1. 調べたい式にカーソルを置きます。
2. *Ctrl+F*（**編集→検索と置換→クロスリファレンス**）を押下します。
3. クロスリファレンスのダイアログボックスが表示されます。OK をクリックすると検索処理が実行されます。式を使用しているオブジェクトはプログラムだけなのでリポジトリ指定は必要ありません。
4. ナビゲータペインのクロスリファレンスの一覧に結果が表示されます。ツリーの最も低いレベルに、式を使用しているタスクが表示されています。このツリーをクリックすると、式を指定している設定欄にカーソルが移動します。

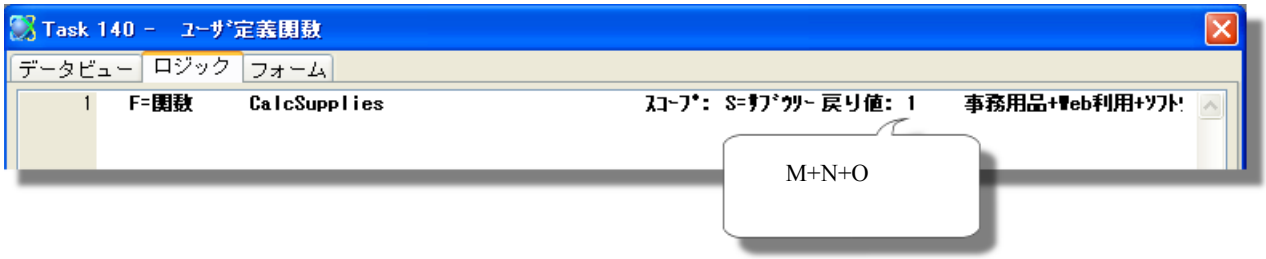
式の中で別の式を再利用するには

式エディタに定義された式は、同じタスク内で何度も使用することができます。同じ式は様々な計算処理（特に項目の初期設定）に使用されることが一般的です。特に、0、'TRUE'LOG、'FALSE'LOG、または **DATE()** などがよく使用されます。

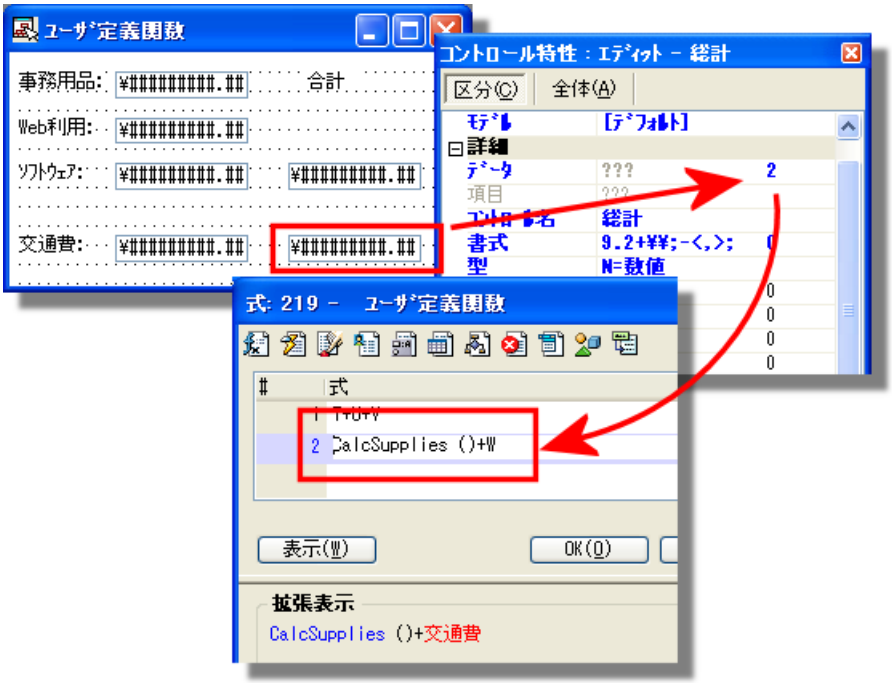
また、簡単なユーザ定義関数を作成したり、**ExpCalc()** 関数を使用することで 1 つの式を別の式の中で利用することができます。**ExpCalc()** は、複雑な計算式を 1 つ作成することで、これを他の式から再利用することを可能にするものです。

ここでは、式を再利用する 2 つの方法の例を紹介します。

ユーザ定義関数を使用する

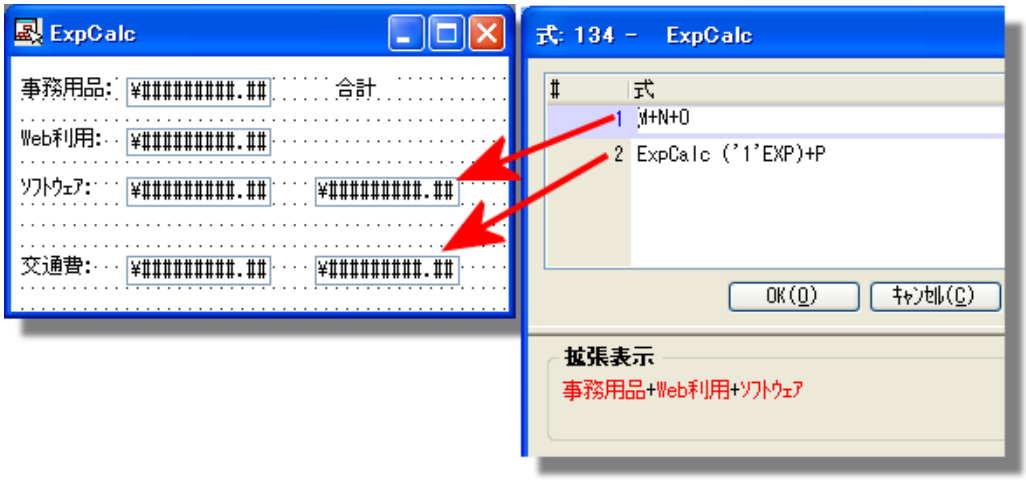


1. 最初に、式の値を返すユーザ定義関数を作成します。この例では、関数名を CalcSupplies として、式 M+N+O を戻すようにしています。



これで、作成されたユーザ定義関数は、式の中で使用することができます。

ExpCalc() 関数を使用する



ExpCalc() は、式の中で別の式の実行結果を使用することを可能にします。この例では、３つの項目が式 **#1** で合計され、小計として表示されています。小計は式 **#2** で使用され、４番目の項目が更に加算されます。

1. **ExpCalc(** を入力します。
2. **EXP** リテラルを使用して、実行させたい式番号を入力します。この例では、式 **#1** を使用するため '1'EXP が入力されています。
3. 関数の括弧を閉じます。

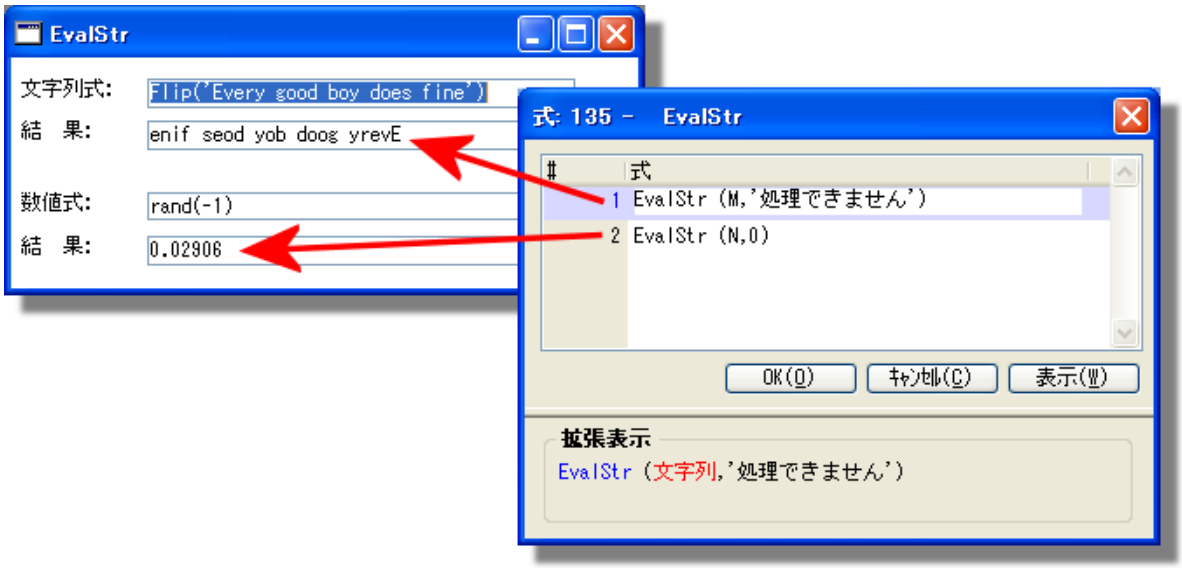
EXP リテラルを使用することで、式の位置が変更されても自動的に追随するようになります。例えば、**#1** の前に別の式を追加した場合、'1'EXP は自動的に '2'EXP に変更されます。

実行時に式の構文の確認と評価を行うには

実行時に Magic の式を構成し、実行させることができます。例えば DB テーブル内にマクロ命令を格納したい場合に便利です。命令は、プログラムによって作成され、別のプログラムで実行させることができます。

EvalStr() 関数を使用することで、このような処理を実現することができます。この関数は、他の Magic 関数を実行させることのできる強力な関数です。この例では、式としてして評価される任意の文字列を入力し、それを実行するものです。

EvalStr 関数を使用する



EvalStr() 関数の構文は以下の通りです。

EvalStr(*expression string*, *default value***)**

パラメータ：

- *expression string* ：評価される式を表す文字列です。シングルクォーテーションで囲まれた文字列や文字型項目を組み合わせて指定します。
- *default value* ：式の中に構文エラーが存在した場合に返される値です。この例では、文字列と評価されなかった場合に「処理できません」という文字列を返します。

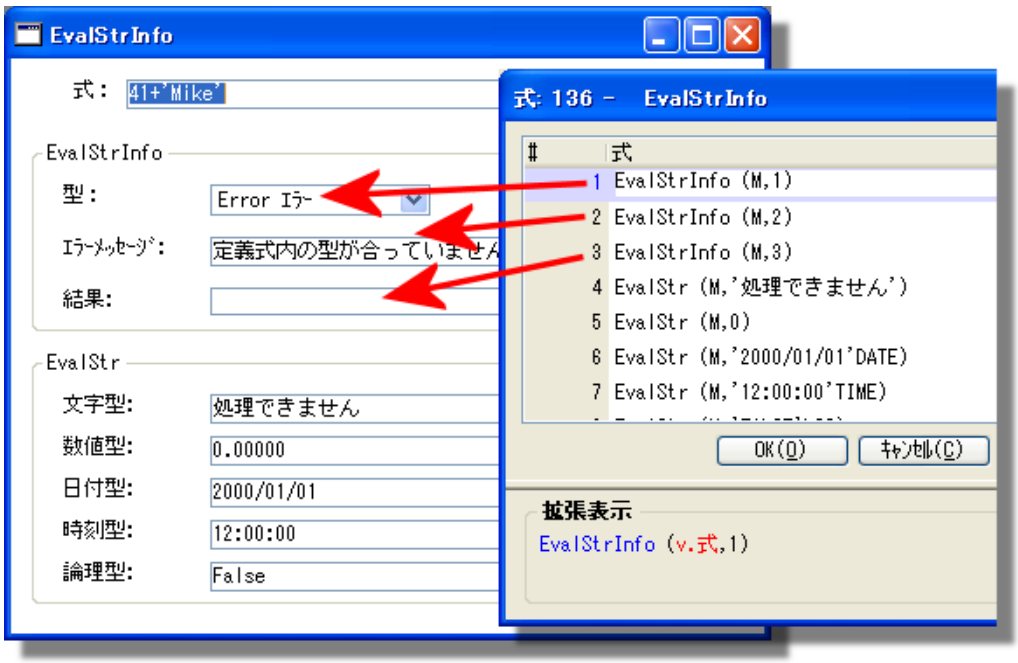
注： この関数を使用する上でいくつか注意することがあります。

第一に、結果として返るデータ型が、期待値と合っていることをあらかじめ確認しておく必要があります。Magic の構文チェックユーティリティは、この式が実行時にどのように評価されるかが分かりません。この例では、２つの文字列をパラメータとして使用し、一方の式では文字列を返し、もう一方では数値を返すようにしています。

また、プログラムが実行されるまでこの文字列はチェックされません。無効な式が入力された場合、デフォルト値が返されますが、どのようなエラーが発生したのか確認できません。

EvalStrInfo() 関数を使用することでこれらの問題を対応することができます。使用方法は以下で説明します。

EvalStrInfo 関数を使用する



EvalStrInfo() 関数の構文は以下の通りです。

EvalStrInfo(*expression string*, *option*)

パラメータ：

- *string*：評価される式を表す文字列です。文字列と文字型項目を組み合わせて指定します。
- *option*：返される情報のタイプを表す数値です。以下の値が指定できます。
  - 1 = 戻り値のデータ型
  - 2 = 実行時に発生するエラー内容
  - 3 = 式の内容
    - Option =1 の場合、指定された式が返す値のデータ型を文字で返します（A= 文字型、N= 数値型など）。式にエラーがある場合は、E が返り、データ型が不明の場合は、* が返ります。この例では、結果を表示させる項目を決定するためにこの関数を使用しています。
    - Option=2 の場合、構文エラーをチェックします。この例では、不正なエラーを入力すると「式のデータが正しくありません」というメッセージを表示させるようにしています。

既存の式を複写するには

式エディタに定義されている式を複写したい場合、以下の３つの方法があります。

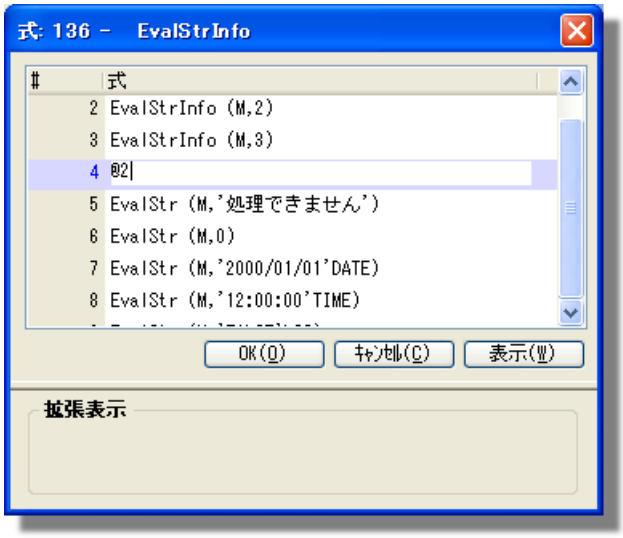
1. Windows で通常使用されるコピー & ペーストのショートカットキー（*Ctrl+C* と *Ctrl+V*）を使用します。
2. Magic 独自の行複写機能（**編集→登録→複写登録**または、*Ctrl+Shift+R*）を使用します。
3. 式エディタでは、**@Line** という特別な関数が使用できます。

入力内容の複写処理については、第 1 章：「入力行を複写するには」（10 ページ）も参照してください。

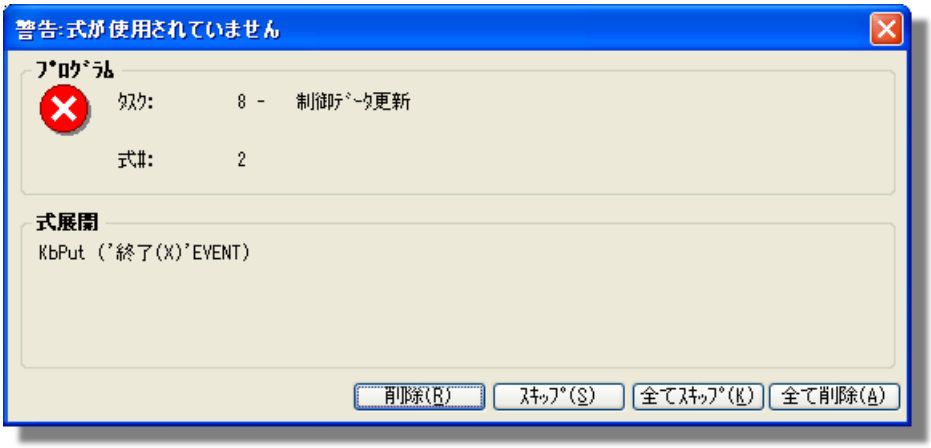
@Line を使用する

1. 式エディタ内で *F4* を押下して１行追加します。
2. @ を入力します。
3. 複写したい式番号を入力します。
4. *Tab* を押下します。

指定された行番号の式が複写されます。この例では、式 **#2** の内容が式 **#3** に複写されます。



使用していない式を検索するには



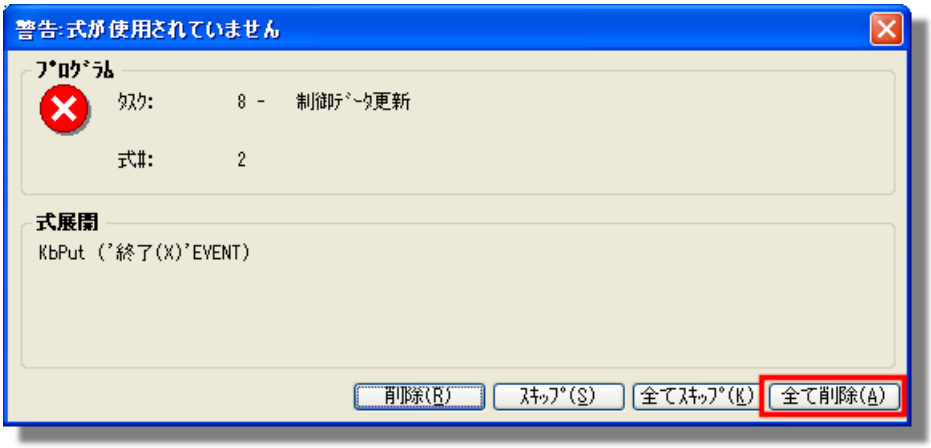
プログラム内に使用していない余分な式が定義されていると保守がしにくくなります。このため、これらを消去することを推奨します。Magic では簡単にこれらを見つけることがでできます。

未使用の式を検索する

1. **最小チェックレベル**（**設定→動作環境→動作設定**）を **W= 警告**か **R= 推奨**に設定します。
2. チェックしたいプログラムに移動します（すべてのプログラムをチェックする場合は、**プログラム**リポジトリのヘッダ行にカーソルを置きます）。
3. ***F8***（リポジトリ全体をチェックする場合は ***Alt+F8***）を押下します。
4. 未使用の式が見つかるたびに**警告**ダイアログが表示されます。各警告に対して、開発者が対応することになります。
   - この式を無視して次に進める場合は、スキップをクリックします。
   - これ以降、見つかった未使用の式に対して全てスキップする場合は、全てスキップをクリックします。
   - この式を削除する場合は、削除をクリックします。
   - 式を表示させる場合は、***Esc*** を押下します。この場合は、構文チェックを中断し、**式エディタ**が開いて使用していない式にカーソルが移動します。
   - すべての式を削除する場合は、全て削除をクリックします。

構文チェックの結果は、**チェック結果**ペインに表示されます。

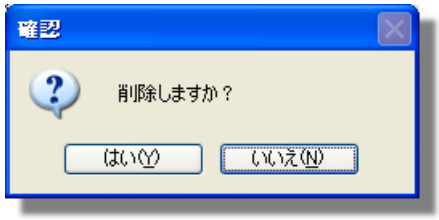
すべての未使用の式を削除するには



未使用の式を削除する場合は、「使用していない式を検索するには」（478 ページ）の説明に従って操作してください。

そして、最初の**警告**ダイアログが表示されたらに、全て削除を**クリック**します。すべての未使用の式が削除されます。

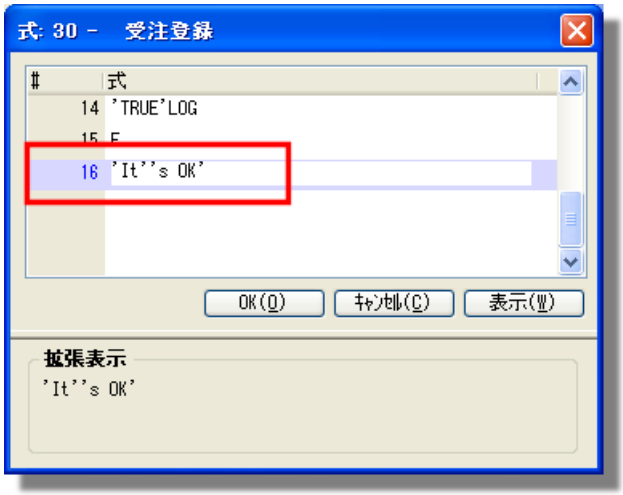
プログラムリポジトリ全体の構文チェック（***Alt+F8***）を行っている場合、右に表示されているような**確認**メッセージが表示されます。はいを**クリック**すると、未使用の式は自動的に削除されます。



式の中で文字列を定義するには

式を入力する際、すべての文字列はシングルクォーテーションで囲む必要があります。この指定をしない場合、文字列は項目のシンボル名（項目番号）か関数名として認識されます。

式の中で文字としてシングルクォーテーションを入力する必要がある場合は、２つのシングルクォーテーションを連続して入力します。



簡単な式が重複して定義されることを防ぐには

なるべく同じ式を共有することで、特に複雑なタスクを開発する場合などは管理が楽になります。例えば、'TRUE'LOG または 0 のような初期設定用の式などがこれに該当します。

式がすでに存在していて、**クイック式エディタ**で式を入力する場合は、このような式を入力する場合に使用できます。式がまだ定義されていない場合のみ、**式エディタ**内に追加されます。

クイック式エディタについては、「タスクエディタで直接に式を編集するには」（468 ページ）を参照してください。

［このページは意図的に空白にしています。］

第 22 章 : レポート

タスクのデータビューを外部ファイルに出力するには

ユーザから最も要求される機能の 1 つに、データを表計算ソフトやワードプロセッサ、または他のソフトウェアで読み込めるようなファイルとして出力することです。現在 Magic には、特別なプログラムを作成することなくこれらの処理を実現する機能が組み込まれています。

実現するには以下の 2 つの方法があります。

- データビュー関数 : プログラムで出力する機能として、**DataViewTo** という Magic の内部関数が使用できます。これらの関数は、出力内容と書式などを指定して定義します。「データを HTML ファイルとして出力するには」（491 ページ）を参照してください。
- データ出力ウィザード : エンドユーザに対して、オンラインタスク上からからデータ出力ウィザードを呼び出すことができます。このユーティリティはエンドユーザによって操作することができるものです。「動的にデータを出力させるには」（484 ページ）を参照してください。

どちらの方法も同じような出力オプション（テキスト、CVS、XML、または HTML）があります。

動的にデータを出力させるには

データ出力ウィザードを使用することで、ユーザが表示画面の内容を出力させることができます。このユーティリティはレポート作成プログラムのように動作します。このユーティリティは、ユーザが出力形式を選択することができます。表示されている項目が出力対象となり、表示順に出力されます。

参照： 「データを HTML ファイルとして出力するには」（491 ページ）

データ表示プログラムを作成する

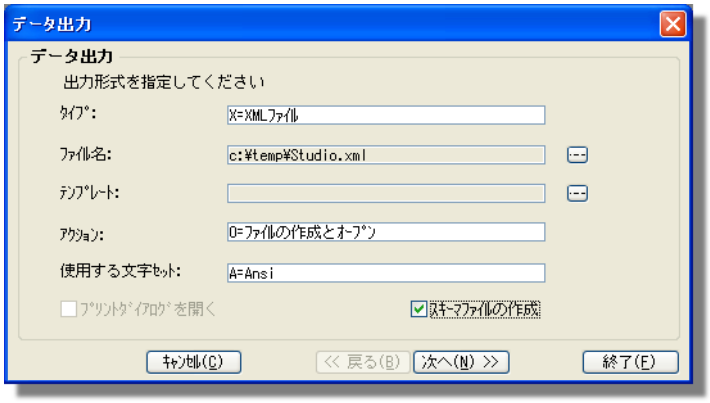
1. ユーザが参照したいと思われるすべてのレコードとカラムが一覧表示されるタスクを作成します。最も簡単な方法は、**APG**（*Ctrl+G*）を使用して照会プログラムを作成することです（「簡易参照用プログラムを作成するには」（486 ページ）を参照してください）。このタスクのデータ出力特性（タスク特性→オプションタブ）を *Yes* に設定します。
2. これで、プログラムを実行すると、以下に説明する手順でユーザは、データ出力ウィザード（*Ctrl+P* またはオプション→データ出力）を起動してデータを出力することができます。

データ出力ウィザードを使用する

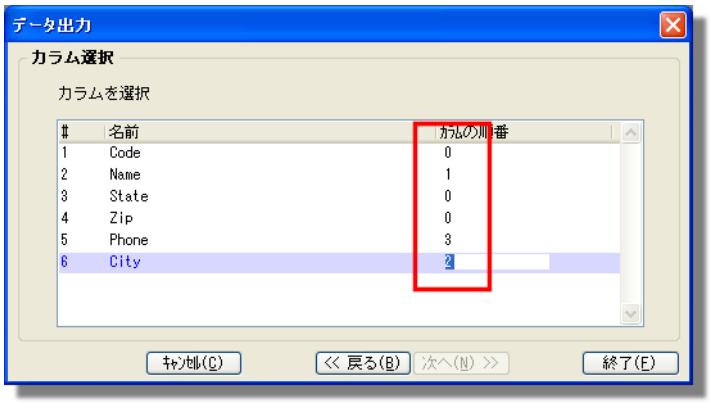
また、ユーザはプログラムのデータビューからデータを出力することができます。以下に、その方法について説明しています。

1. プログラムを実行します。
2. レコードが一覧表示されます。ソートや範囲オプションを利用することで必要に応じてレコードのソート順序を変更したり、表示されるレコードの内容を制限したりすることができます。

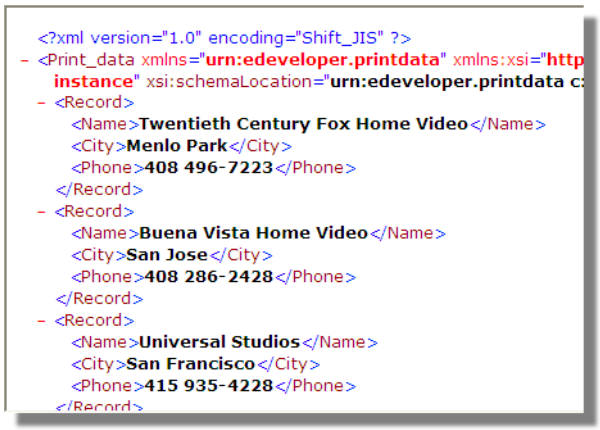
3. *Ctrl+P*（オプション→データ出力）を押下します。データ出力ウィザードのダイアログボックスが表示されます。出力したいデータ形式にもとづいて、オプションを選択し、次へをクリックします。



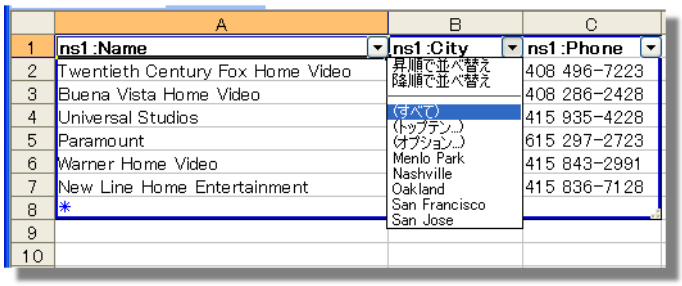
4. 次に、データビュー内のすべてのカラムの一覧がダイアログで表示されます。出力したくないカラムは、**カラムの順番**の値を *0* に設定します。また、**カラムの順番**の番号を付け替えることで出力する順番を変更することができます。
この例では、名前、都市、および電話をこの順番で出力するように設定しています。
次に、終了をクリックします。



5. 指定された内容にもとづきデータがファイルに出力されます。この例では、XML ファイルを選択したためファイルは XML 形式で出力されますが、HTML や CSV で出力することもできます。
作成と表示を選択すると、XML ファイルが PC に割り当てられたアプリケーションによってオープンされます。



ヒント：*データのタイプを XML としてデータを保存することができますが、ファイル名の拡張子を「.xls」に変更してみてください。これにより、Excel XML の形式として扱われ、Excel でオープンしやすくなります。*



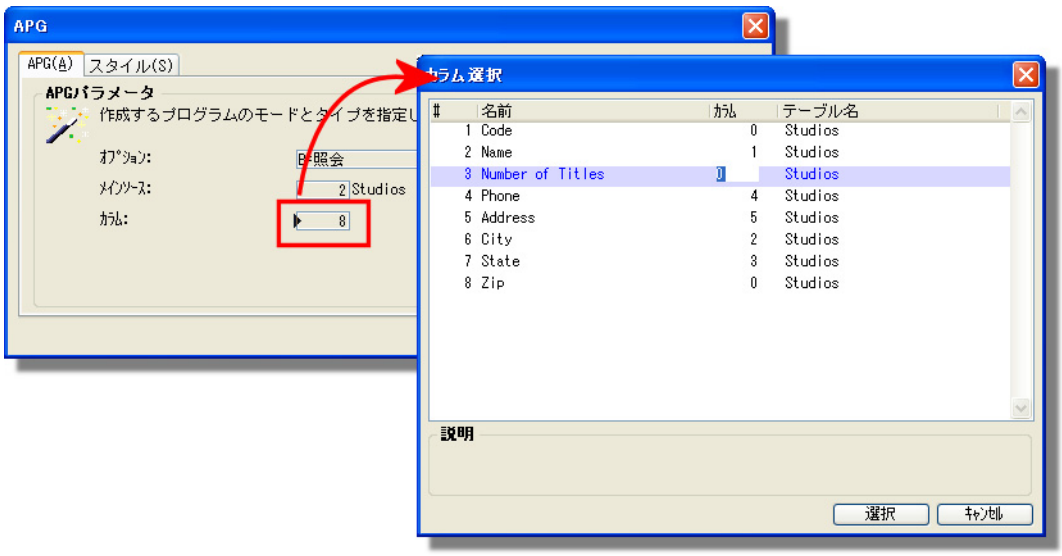
簡易参照用プログラムを作成するには

Magic プログラムは簡単に作成することができます。しかし、**APG**（*Ctrl+G*）ユーティリティを使用することでより簡単にプログラムを作成することができます。

レコードをファイルとして出力する前に、ユーザが範囲やソート機能を利用できるようにすることで、作成されるプログラムは、データビューを表示するプログラムとして利用しやすいものになります。

簡単な参照プログラムを作成する

1. *F4*（編集→行作成）を押下して、プログラムリポジトリに 1 行追加します。
2. *Ctrl+G*（オプション→ APG）を押下します。



3. **APG ダイアログ**が表示されます。以下のオプションを設定します。
   オプション：*Q= 照会*
   メインソース：出力したいデータソース。**ズーム**で一覧を開くことで選択できます。
   カラム：出力するカラムを選択するためにズームします。デフォルトは、すべてのカラムが定義順に指定されています。この例では名前、都市、州、電話、アドレスのみを出力するように設定しています。
4. **OK** を**クリック**します。

プログラムが作成されます。これを実行すると、**メインソース**のすべてのレコードが表示されます。このプログラムは必要に応じて修正できます。フォームを変更したり、範囲を指定したりすることができます。

現在のビューをテキスト出力するには

現在のデータビューをテキストファイルに出力する場合、**DataViewTo** 関数を使用することができます。DataView 関数には３つ関数があります。各関数についての説明は、該当するセクションを参照してください。

- **DataViewToHTML()** ：「データを HTML ファイルとして出力するには」（491 ページ）
- **DataViewToXML()** :「データを XML ファイルとして出力するには」（494 ページ）
- **DataViewToText()** ：「データをテキストファイルとして出力するには」（488 ページ）と「データを CSV ファイルとして出力するには」（490 ページ）

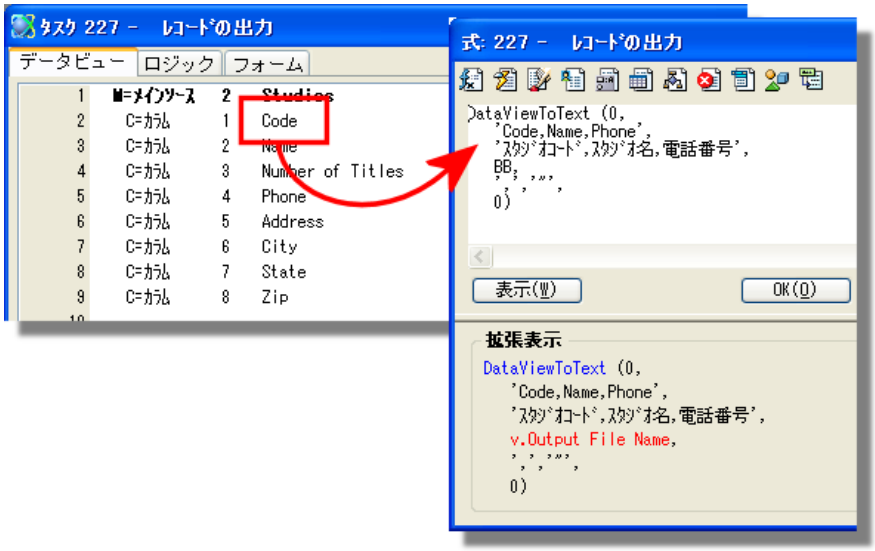
また、プログラムを作成することなく、エンドユーザに対して動的に現在のデータビューを出力させることもできます。「動的にデータを出力させるには」（484 ページ）を参照してください。

参照 : 「タスクのデータビューを外部ファイルに出力するには」（483 ページ）
第 5 章 :「データソースの内容をテキストファイルに出力したり、テキストファイルをデータソースに入力するには」（81 ページ）

データをテキストファイルとして出力するには

DtaViewToText() 関数を使用することで、データビューの内容をテキストファイルとして出力することができます。この関数を使用することで、出力したいレコードを制限させる等、出力内容を制御することができます。

DataViewToText() 関数を使用する



DataViewToText() 関数は、現在のデータビューの内容をテキスト出力します。構文は以下の通りです。

DataViewToText(*Generation*, *VariableList*, *HeaderList*, *OutputFileName*, *DelimiterString*, *StringIdentifier*, *CharSet***)**

パラメータ：

- *Generation* ：タスクの世代番号です。0 が現在のタスク、1 は親タスクになります。
- *VariableList* ：出力したい項目名のリスト。タスクのデータビューエディタに定義されている項目名を指定します。複数の項目名を、カンマ区切りで指定します。項目名の間には、空白は入れないでください。また大文字小文字を区別します。
- *HeaderList* ：出力ファイル内の項目タイトルのリスト。この例では、Code 項目に対してスタジオコード というタイトルが指定されています。このパラメータが指定されていない場合、ヘッダは出力されません。@ が指定された場合、*VariableList* で指定された名前がそのまま使用されます。
- *OutputFileName* ：出力ファイル名の名前。この例では、v.Output FileName 項目が指定されています。
- *DelimiterString*：出力される項目の区切り文字。この例では、タブコードとして **ASCIIChr(9)** が指定されています。
- *StringIdentifier* ：出力文字列の前後に付加する文字。この例では、各文字列をダブルクォーテーションで囲むように指定されています。
- *CharSet* ：使用する文字セットを表す番号。以下の番号が指定できます。
  - 0=ANSI
  - 1=Unicode
  - 2=UTF-8

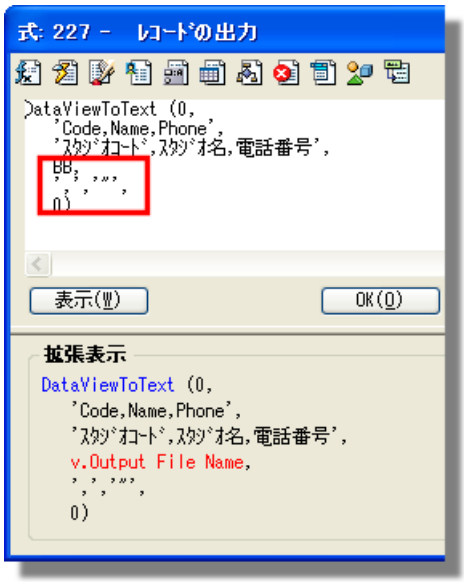
前述の例を実行した場合、以下のような内容のテキストが出力されます。タブコードは表示されませんが、これによってデータが区切られていることを確認できるはずです。

参照： 第 5 章：「データソースの内容をテキストファイルに出力したり、テキストファイルをデータソースに入力するには」（81 ページ）
「動的にデータを出力させるには」（484 ページ）

データを CSV ファイルとして出力するには

DataViewToText() 関数を使用することでデータを CSV ファイルに出力することができます。この例で示すように、必要な設定は、区切り文字としてカンマ指定し、文字列の認識用にダブルクォーテーションを使用することです。

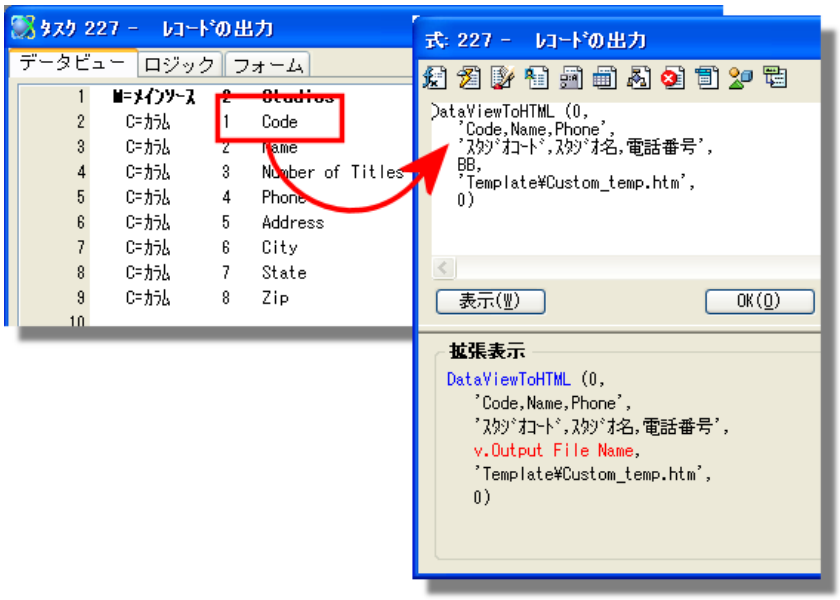
DataViewToText() 関数については、「データをテキストファイルとして出力するには」（488 ページ）を参照してください。



データを HTML ファイルとして出力するには

プログラムで **DataViewToHTML()** 関数を使用することで、データビューの内容を HTML 形式のテキストファイルとして出力することができます。この関数を使用することで、出力したいレコードを制限させる等、出力内容を制御することができます。

DataViewToHTML() 関数を使用する



DataViewToHTML() 関数は、現在のデータビューの内容をテキスト出力します。構文は以下の通りです。

DataViewToHTML(*Generation*, *VariableList*, *HeaderList*, *OutputFileName*, *TemplateFile*, *CharSet***)**

パラメータ：

- *Generation* ：タスクの世代番号です。0 が現在のタスク、1 は親タスクになります。
- *VariableList* ：出力したい項目名のリスト。タスクのデータビューエディタに定義されている項目名を指定します。複数の項目名を、カンマ区切りで指定します。項目名の間には、空白は入れないでください。また大文字小文字を区別します。
- *HeaderList* ：出力ファイル内の項目タイトルのリスト。この例では、Code 項目に対してスタジオコード というタイトルが指定されています。このパラメータが指定されていない場合、ヘッダは出力されません。@ が指定された場合、*VariableList* で指定された名前がそのまま使用されます。
- *OutputFileName* ：出力ファイル名の名前。この例では、v.Output FileName 項目が指定されています。
- *TemplateFile* ：HTML ファイルの作成時に使用するテンプレートファイル名（オプション）。
- *CharSet* ：使用する文字セットを表す番号。以下の番号が指定できます。
  - 0=ANSI
  - 1=Unicode
  - 2=UTF-8

上記のプログラム例を実行すると、以下のような結果になります。

スタジオコード	スタジオ名	電話番号
S001	Twentieth Century Fox Home Video	408 496-7223
S002	Buena Vista Home Video	408 286-2428
S003	Universal Studios	415 935-4228
S004	Paramount	615 297-2723
S005	Warner Home Video	415 843-2991
S006	New Line Home Entertainment	415 836-7128

HTML テンプレートファイルを使用する

必要であれば、データを出力する際に、HTML テンプレートファイルを指定することができます。HTML テンプレートファイルは、左側で示されている最小の HTML ファイル形式をもとにカスタマイズすることで利用することができます。**<MGTABLE>** タグは、HTML テーブルのテキストによってデータ出力時に置き換えられます。

最小のテンプレート

```
<html>
<head>
<title> </title>
</head>
<MGTABLE>
</html>
```

これは、テンプレートの最小構成です。**<MGTABLE>** は、実際のテーブルデータによって実行時に置き換えられます。

カスタマイズされたテンプレート

```
<html>
<head>
    <title> 私のDVDリスト</title>
    <meta http-equiv="Content-Type"
      content="text/html; Charset=Shift_JIS">
    <!-- This page should not be cached -->
        <meta http-equiv="Expires" content="Mon, 06 Jan 1990 00:00:01 GMT">
        <meta http-equiv="pragma" content="no-cache">
        <meta http-equiv="cache-control" content="0">
        <meta http-equiv="refresh" content="100">
        <link rel="stylesheet" type="text/css" href="DVDStyles.css">
</head>
<MGTABLE RowStyle=ALL ColumnStyle=ALL>
</html>
```

これは、カスタマイズされたテンプレートです。いくつかの Meta タグが追加され、スタイルシートにリンクされています。

MGTABLE スタイル

<MGTABLE> タグには、RowStyle と ColumnStyle というテンプレート内で使用できる 2 つのスタイルタグがあります。これらは、スタイルタグが作成される表をどのように表示させるかを指定させることができます。

RowStyle:

All	Magic は、各行とタイトルにスタイルを作成します。
EvenAndOdd	Magic は 2 つのスタイル（MG_Even_Row と MG_Odd_Row）を作成します。
Equal	Magic は、すべての行とタイトルに対して同じスタイルを作成します。

Column Style:

All	Magic は、各カラムに対してスタイルを作成します。
Equal	Magic は、すべてのカラムに対して同じスタイルを作成します。

従って、前述のテンプレート例を使用した場合、以下のような HTML テーブルが作成されます。class=tag は、すべて Magic のマージタグです。これらのタグは、テーブルの表示内容をカスタマイズするため使用することができます。

```
</head>
<table border="1" width="100%" class="MG_TABLE">
<THEAD>
 <tr>
  <td width="33%" class="MG_TITLE1">ｽﾀｼﾞｵｺｰﾄﾞ</td>
  <td width="33%" class="MG_TITLE2">ｽﾀｼﾞｵ名</td>
  <td width="33%" class="MG_TITLE3">電話番号</td>
 </tr>
</THEAD>
<TBODY>
<tr class="MG_ROW1">
  <td width="33%" class="MG_DATA1">S001  </td>
  <td width="33%" class="MG_DATA2">Twentieth Century Fox Home Video  </td>
  <td width="33%" class="MG_DATA3">408 496-7223  </td>
</tr>
<tr class="MG_ROW2">
  <td width="33%" class="MG_DATA1">S002  </td>
  <td width="33%" class="MG_DATA2">Buena Vista Home Video  </td>
  <td width="33%" class="MG_DATA3">408 286-2428  </td>
</tr>
<tr class="MG_ROW3">
  <td width="33%" class="MG_DATA1">S003  </td>
  <td width="33%" class="MG_DATA2">Universal Studios  </td>
  <td width="33%" class="MG_DATA3">415 935-4228  </td>
```

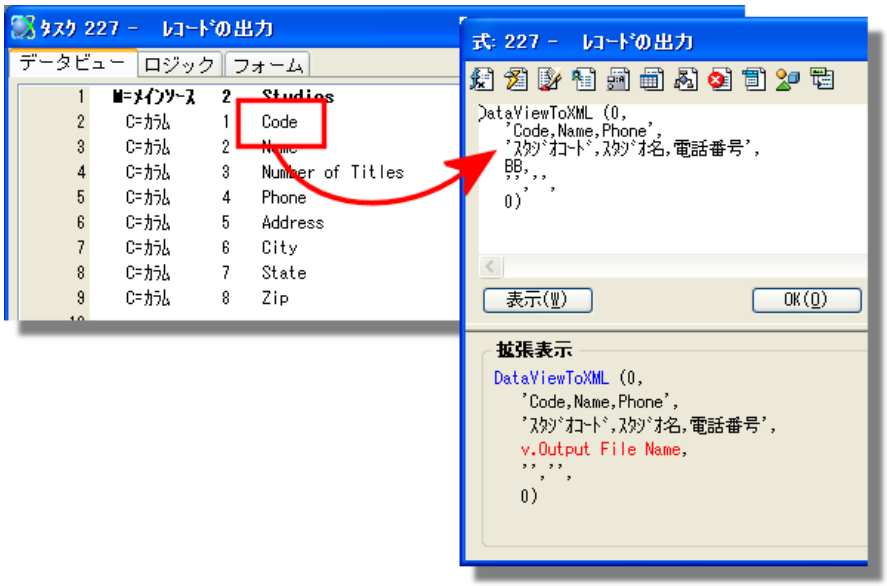
以下は、HTML 形式の DVD 一覧をカスタマイズしたものです。

ｽﾀｼﾞｵｺｰﾄﾞ	ｽﾀｼﾞｵ名	電話番号
S001	Twentieth Century Fox Home Video	408 496-7223
S002	Buena Vista Home Video	408 286-2428
S003	Universal Studios	415 935-4228
S004	Paramount	615 297-2723
S005	Warner Home Video	415 843-2991
S006	New Line Home Entertainment	415 836-7128

データを XML ファイルとして出力するには

プログラムで **DataViewToXML()** 関数を使用することで、データビューの内容を XML 形式のテキストファイルとして出力することができます。この関数を使用することで、出力したいレコードを制限させる等、出力内容を制御することができます。

DataViewToXML() 関数を使用する



DataViewToXML() 関数は、現在のデータビューの内容をテキスト出力します。構文は以下の通りです。

DataViewToXML(*Generation*, *VariableList*, *HeaderList*, *OutputFileName*, *SchemaFileName*, *TemplateFileName*, *CharSet***)**

パラメータ：

- *Generation* ：タスクの世代番号です。0 が現在のタスク、1 は親タスクになります。
- *VariableList* ：出力したい項目名のリスト。タスクのデータビューエディタに定義されている項目名を指定します。複数の項目名を、カンマ区切りで指定します。項目名の間には、空白は入れないでください。また大文字小文字を区別します。
- *HeaderList* ：出力ファイル内の項目タイトルのリスト。この例では、Code 項目に対してスタジオコード というタイトルが指定されています。このパラメータが指定されていない場合、ヘッダは出力されません。@ が指定された場合、*VariableList* で指定された名前がそのまま使用されます。
- *OutputFileName* ：出力ファイル名の名前。この例では、v.Output FileName 項目が指定されています。
- *SchemaFileName*（オプション）：作成されるスキーマファイル名。空白の場合は、スキーマを作成しません。
- *TemplateFileName*（オプション）：XML ファイルの作成時に使用するテンプレートファイル名。
- *CharSet* ：使用する文字セットを表す番号。以下の番号が指定できます。
  - 0=ANSI
  - 1=Unicode
  - 2=UTF-8

上記のプログラム例を実行すると右のような XML ファイルが作成されます。

```
<?xml version="1.0" encoding="Shift_JIS" ?>
- <Print_data>
  + <Record>
  - <Record>
      <ｽﾀｼﾞｵｺｰﾄﾞ>S002</ｽﾀｼﾞｵｺｰﾄﾞ>
      <ｽﾀｼﾞｵ名>Buena Vista Home Video</ｽﾀｼﾞｵ名>
      <電話番号>408 286-2428</電話番号>
    </Record>
  - <Record>
      <ｽﾀｼﾞｵｺｰﾄﾞ>S003</ｽﾀｼﾞｵｺｰﾄﾞ>
      <ｽﾀｼﾞｵ名>Universal Studios</ｽﾀｼﾞｵ名>
      <電話番号>415 935-4228</電話番号>
    </Record>
  - <Record>
      <ｽﾀｼﾞｵｺｰﾄﾞ>S004</ｽﾀｼﾞｵｺｰﾄﾞ>
      <ｽﾀｼﾞｵ名>Paramount</ｽﾀｼﾞｵ名>
      <電話番号>615 297-2723</電話番号>
    </Record>
```

スキーマファイルを作成する

DataViewToXML() 関数でスキーマファイル名を指定すると、スキーマファイルが生成されます。

XML テンプレートファイルを使用する

必要であれば、XML をフォーマットするために XML テンプレートファイルを指定することができます。テンプレートは、<Print_data> タグと </Print_data> タグの間にテキストを記述しない XML ファイルです。Magic は、このタグの間に <Record> タグを使用してレコードデータをマージします。テンプレートファイルに XSL スタイルシートを指定することにより表示内容をカスタマイズすることができます。

帳票を作成するには

データ出力ウィザード（「タスクのデータビューを外部ファイルに出力するには」（483 ページ）を参照）を使用することで、データを表計算ソフトや、レポートツール用に簡単に出力することができます。この機能は、エンドユーザに対してデータを動的に処理させたい場合に便利です。

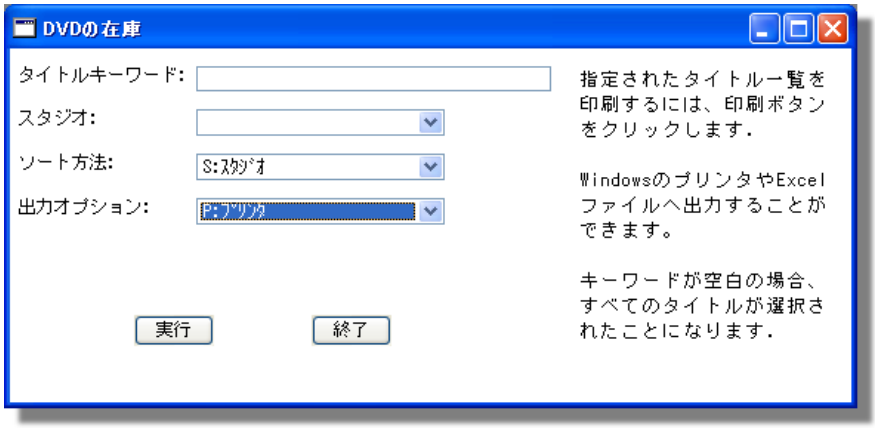
また、Magic にはこの他にもデータ出力用の機能が組み込まれています。毎月の請求書発行など、定型的な書式のデータを作成する場合、Magic のデータ出力機能を使用してたプログラムによって出力した方が便利です。

ここでは、簡単な帳票を作成する基本的な手順について説明します。同じような手続で、複雑な帳票も作成できますが、出力内容が増えたり以下で説明するような高度な機能を利用する必要が出てきます。

- 「帳票のテーブルに対して見出しを繰り返し出力させるには」（516 ページ）
- 「PDF ドキュメントを作成するには」（517 ページ）
- 「ページヘッダ / フッタの情報を定義するには」（503 ページ）
- 「帳票のブレイクレベルを作成するには」（511 ページ）
- 「帳票に複数行のコントロールを定義するには」（514 ページ）

Magic で帳票を作成する

1. “ 起動画面 ” を作成する

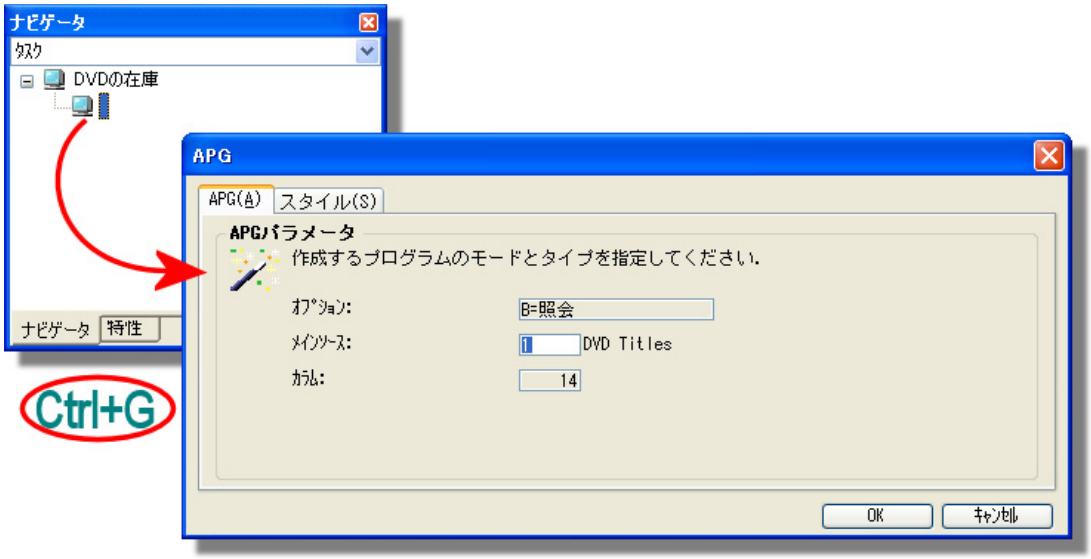


必ずしも必要ではありませんが、帳票のタイトルや何のための帳票か、またユーザに出力先を選択できるような起動画面を作った方が親切です。これは、簡単なオンラインタスクで実現できます（オンラインタスクの作成についての詳細は、第 5 章 :「簡単なプログラムを作成するには」（70 ページ）を参照してください）。

同じ画面を利用して、GUI 形式フォームでの印刷や Excel へのデータ出力を行うことができます。印刷プレビューやソート順序をユーザに選択させることもできます。

このような起動画面のデザインの標準化を検討することは有益です。標準化によってユーザ教育が容易となり、新規に作成する際も既存のタスクをコピーして利用することができます。

2. 簡単なテキスト出力プログラムを作成する



APG を利用して簡単な照会プログラムを作成します（「簡易参照用プログラムを作成するには」（486 ページ）を参照）。

このタスクを呼び出す起動ボタンを設定します。起動ボタンが正しく動作した場合、作成された照会プログラムが表示されます。

3. ソート順序と範囲を確認する

帳票出力プログラムを作成する際に考慮すべき点の 1 つに、範囲とソート処理が正しく動作するかどうかを確認することです。このような場合、オンラインタスクで確認した方が簡単です。このためここでは照会プログラムを最初に作成しています。

複数のソートや出力オプションがある場合、選択されたオプションにもとづいて異なる出力タスクを起動する必要があるかもしれません。しかし、1 つの出力プログラムを作成したら、次のバージョンのためにテンプレートとしてこのプログラムをコピーして使用することができます。

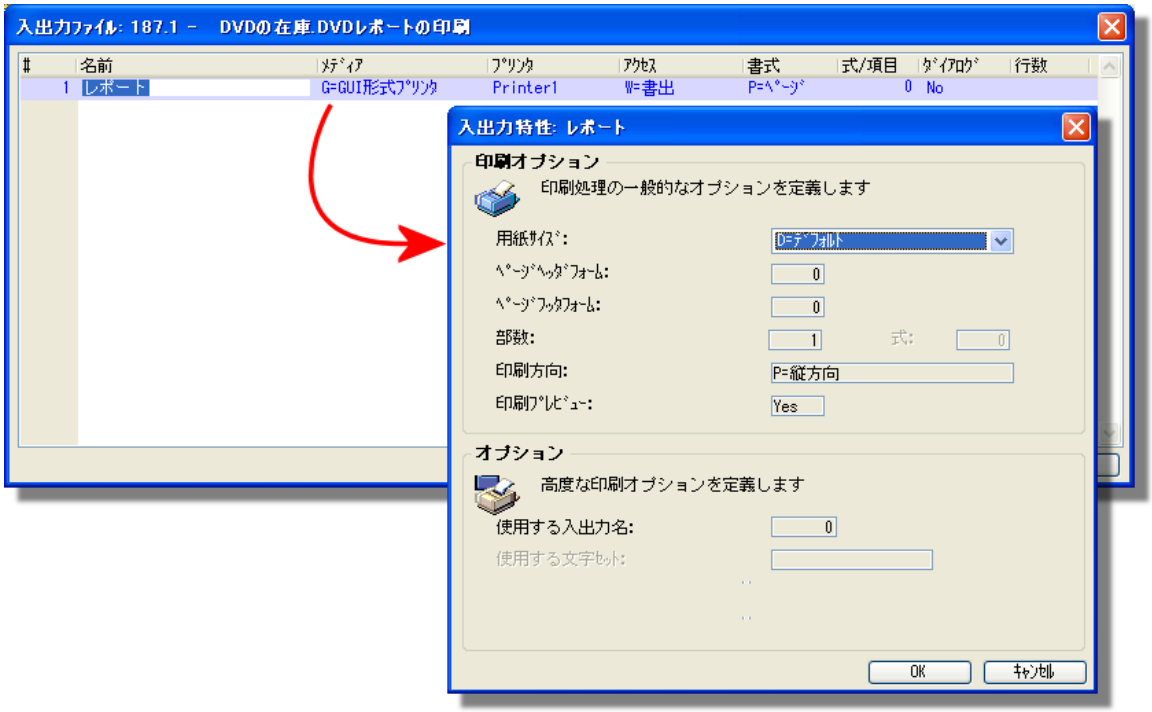
4. バッチタスクを定義する

デバッグを終了したら、**タスク特性→汎用タブ**で**タスクタイプ**特性を（オンラインから）***B= バッチ***に変更することにより、サブタスクをバッチタスクに変更します。

また、レコードの更新がない場合は、**タスク特性→データ**で**トランザクションモード**特性が ***P= 物理***で、**トランザクション開始**特性が、***N= なし***（トランザクションはタスクの処理速度を劣化させるため）になっていることを確認します。同様に、メインソースの**アクセス**特性が ***R= 読込***になっていることも確認します。

最後に、表示されるフォームをテーブル形式ではなく簡単なデータ表示になるように変更します。作成されたテーブルを削除し、ユーザに対して、動作状況が確認できるようなメッセージを表示するように、1 ～ 2 つのカラムのみ配置するようにします。フォーム上に様々な処理を実行させないようにすることが、処理速度を低下させないために必要です。

5. 入出力デバイスを設定する



次に、データ出力用の入出力ファイルを設定する必要があります。入出力ファイルを定義することでデータをどのように出力するかが決定されます。

1. データ出力用のサブタスクに移動します。
2. ***Ctrl+I***（**タスク環境→入出ファイル**）を押下して**入出力ファイル**テーブルを開きます。
3. ***F4*** を押下して、**入出力ファイル**テーブルに 1 行追加します。
4. ***メディア***カラムで ***G=GUI 形式プリンタ***を選択します。

5. 入出力特性（*Alt+Enter*）を開きます。印刷プレビュー特性を *Yes* に設定します。印刷プレビューは、デバッグの際に有効です。

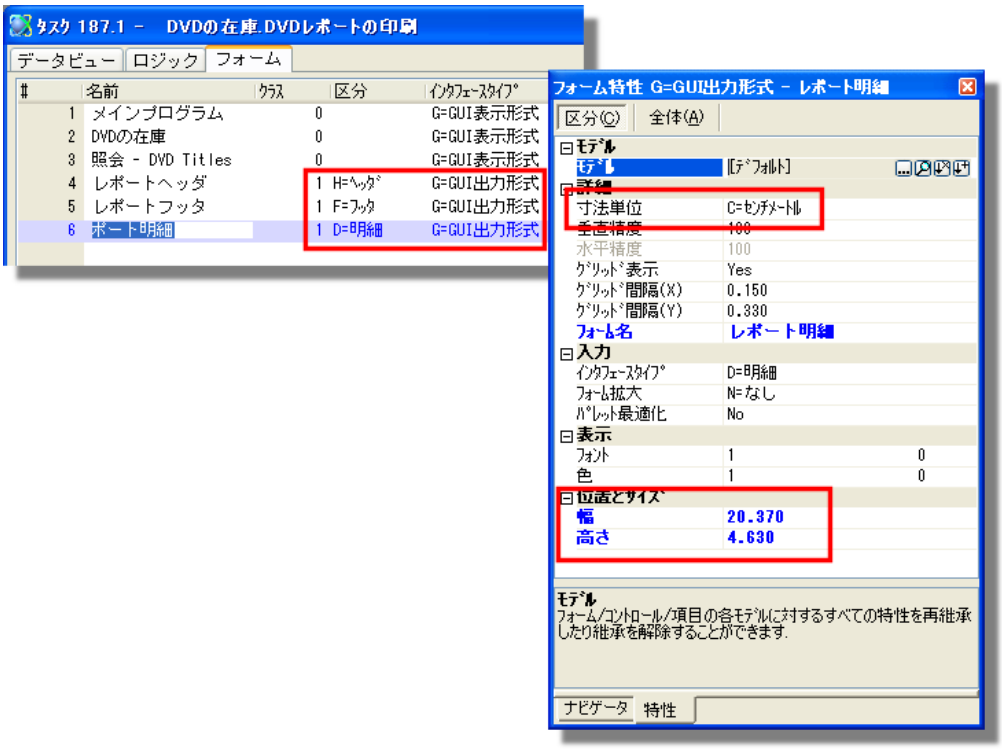
上記の図に表示されているように、入出力特性には様々なオプションがあります。各特性欄にカーソルを置いて F1 を押下すると該当するヘルプファイルが表示されます。

テストを行う場合は、印刷プレビューは非常に有益です。しかし、実際に作成するプログラムでは、印刷ダイアログを使用して直接プリンタに出力しないようにする場合があるかもしれません。印刷ダイアログを表示させたいのであれば、ダイアログカラムを *Yes* に設定します。

また、請求書のような定型の帳票を印刷する場合は、出力先を固定にする必要があります。その場合は、ダイアログカラムを *No* に設定します。この場合は、プレビュー特性も *No* に設定します。

これらのオプションのほとんどは、式で指定することもできます。

6. フォームを作成する



次に、フォームを作成します。基本的なステップは以下の通りです。

1. フォームタブに移動します。
2. *F4* を押下して 1 行追加します。
3. フォームに名前を入力し、クラスカラムに 1 以上の値を入力します。
4. フォームの機能にもとづいて区分カラムを入力します。この例では、*ページヘッダ*と *ページフッタ*、および*明細*を 1 つずつ定義します。
5. インタフェースタイプを *G=GUI 出力*に設定します。

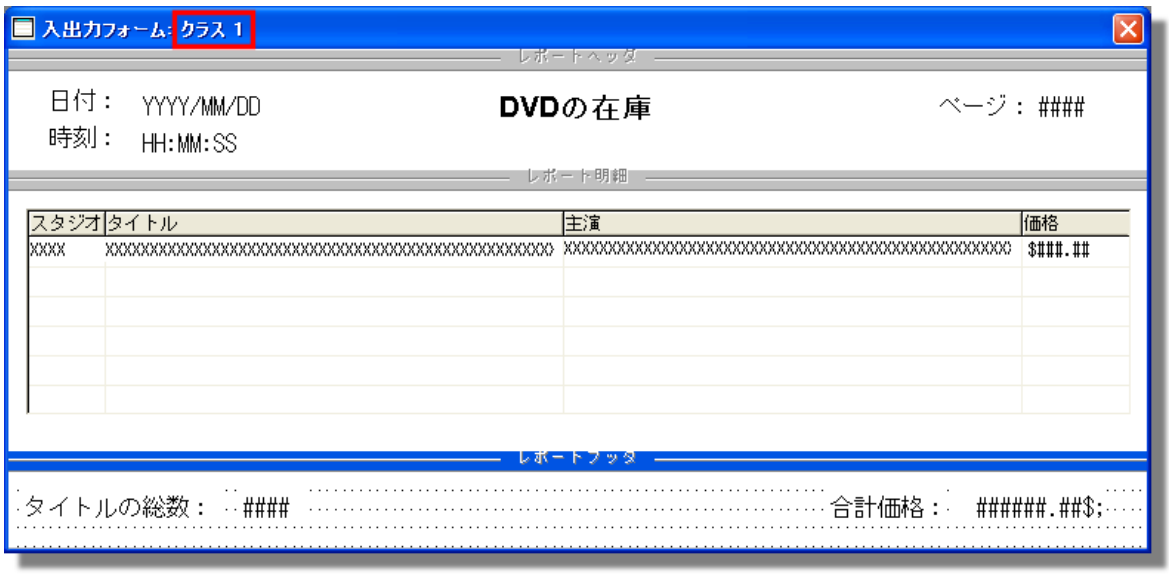
フォーム特性で、寸法単位特性は *I= インチ*か *C= センチ*に設定してください。*D= ダイアログ*に設定した場合、フォームは自動的にリサイズされます。幅特性は、用紙のサイズに合わせます。高さ特性は、各フォーム毎に異なり、マウスを使用することで簡単に変更できます。

注：　インターナショナル版（ダイアログ）の Magic と日本語版（センチ）の Magic は、寸法単位のデフォルト値が異なります。このため、日本語版の Magic Studio で開発したアプリケーションを英語版で実行した場合、フォームのサイズが変わる可能性があります。

帳票フォームに対応したモデルを作成して利用すると便利です。この場合、幅やフォント、寸法単位を自動的に一貫性を持って設定することができます。

ヒント：編集するフォームのサイズが非常に大きい場合、フォームの編集に入る前に、手動でサイズを設定することもできます。フォーム特性の幅特性と高さ特性に妥当なサイズを入力することで変更できます。

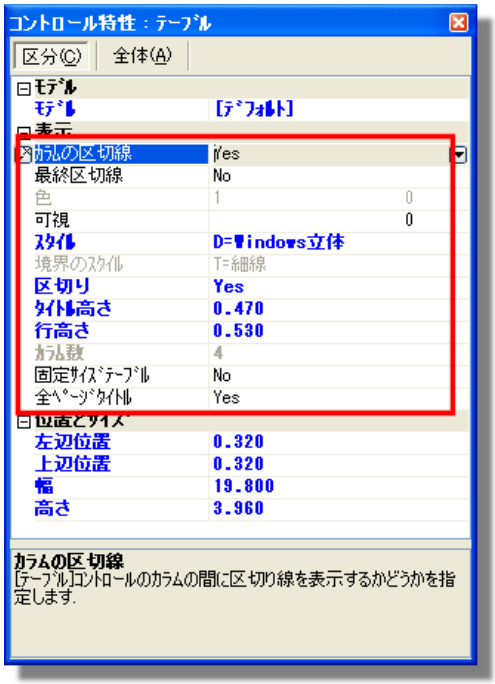
7. フォームを編集する



クラス特性が *1* のフォームのどれかに**ズーム**すると、同じクラスの３つのフォームがすべて表示されます。**フォームエディタ**上の順番にかかわらず、ヘッダは先頭に、フッタは最後に表示されます。1 つまたは 2 つのフォームを同時に編集する方が便利な場合があります。このような場合は、**クラス**特性を一時的に他の値に設定します。このようにすることで、これらのフォームを個別に編集することができます。

フォームを開くと、オンラインのフォームと同じように項目を選択してフォームに配置することでフォームの編集を行うことができます。項目の様々なタイプに対応した標準的なモデルをあらかじめ定義しておくと便利です。

明細フォームは基本的にテーブルです。帳票に境界線を出力させないようにする場合は、右に示すようにテーブルの**コントロール特性**を設定します。他のテーブルに対しても同じように定義します。フォーム上に**テーブル**コントロールを配置し、その上に項目を配置します。

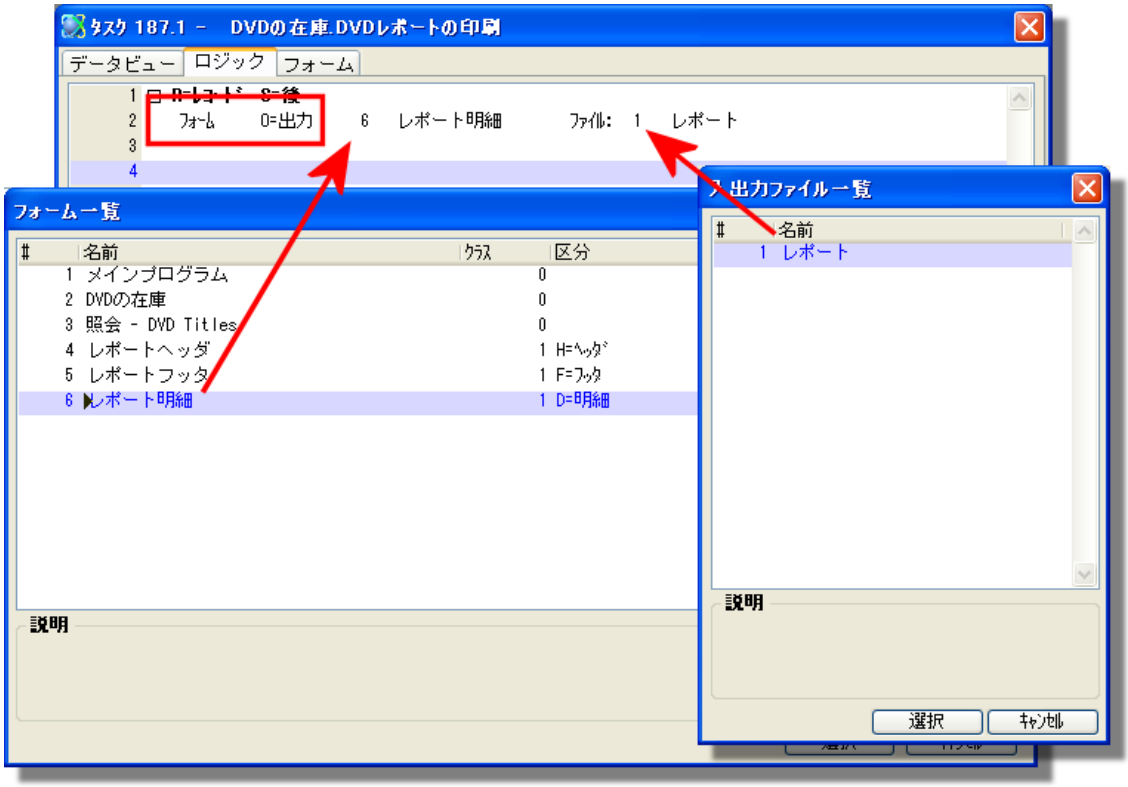


テーブルコントロールが帳票フォームに定義された場合、オンラインフォーム上に定義された場合と特性値が異なります。

全ページタイトル特性は、タイトル行が自動的に出力されるかどうかに関わらず、すべてのページに自動的にタイトルを出力するかどうかを定義します。通常は、*Yes* を設定します。

固定サイズテーブル特性は、テーブルの出力行数を固定にするかどうかを指定します。*Yes* の場合、**フォームエディタ**上に定義された行数分のレコードが 1 ページ内に出力されます。*No* の場合は、帳票のサイズの応じて 1 ページ内に出力される行数が変更されます。この場合、オンラインフォームと同じようにフォームの下の余白のサイズは維持されます。

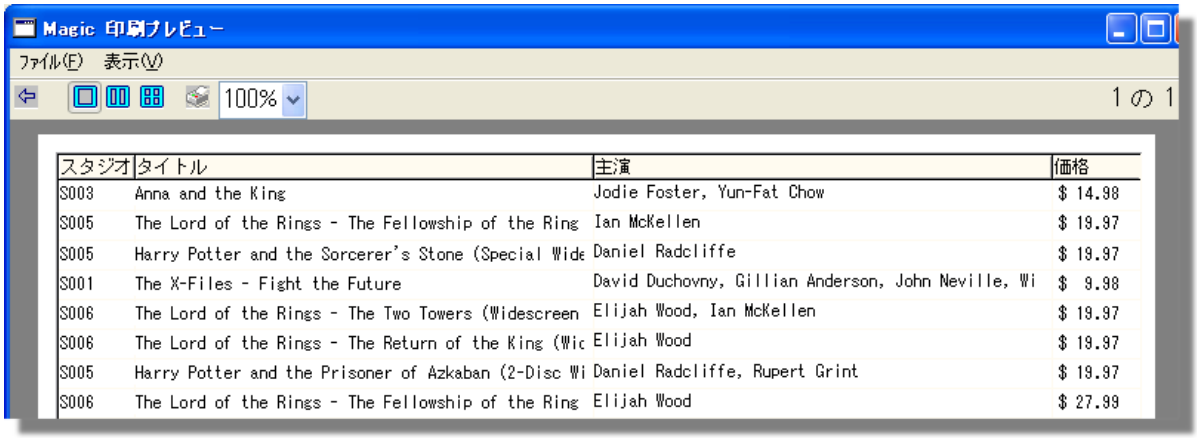
8. 明細フォームの出力処理を定義する



次に、帳票データを出力させる必要があります。通常、明細行は 1 レコード毎に出力されるので、上記の例のように**レコード後**ロジックユニットで出力処理を定義します。しかし、概要を出力する帳票の場合、**項目変更**ロジックユニットで定義します。意図する**ロジックユニット**で出力処理を定義することができます。

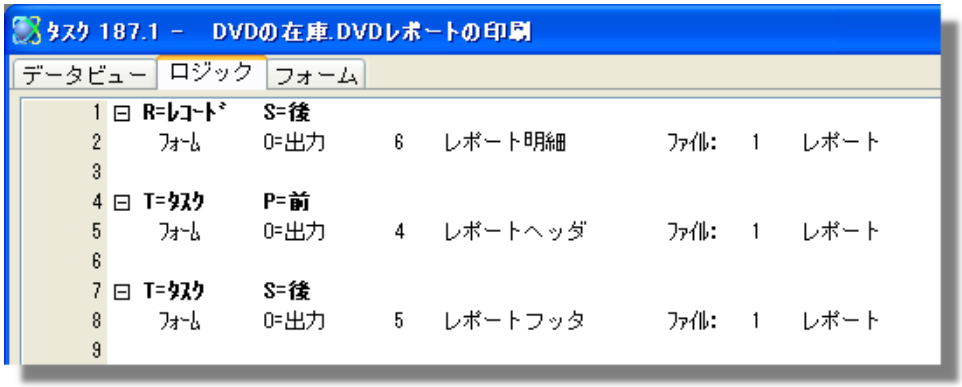
1. フォームを出力させたい**ロジックユニット**に移動します。上記の例では、**レコード後**です。
2. *F4* を押下して 1 行追加します。
3. *F4* を入力して**フォーム**処理コマンドを選択します。次のカラムには ***O= 出力***（デフォルト）が設定されます。
4. 次のカラムに移動し、**ズーム**して出力させたいフォームを選択します。
5. **ファイル**カラムには、デフォルトの入出力ファイル *1* が設定されます。入出力ファイルが複数定義されている場合、ここから**ズーム**して選択できます。

これでプログラムを実行すると、印刷プレビュアーが起動され、出力結果が表示されます。



次にヘッダとフッタを追加します。

9. ヘッダを出力する



帳票ヘッダを出力する場合は、明細行を出力する場合と同じような作業を行います。ただし、ヘッダは通常**タスク前**で最初に出力されます。その後、改ページが行われるたびに自動的に出力するようにします。ヘッダフォームは複数定義されている場合があり、これらのすべてが改ページされた時点で出力されます。

別の方法として、フォームをページ ヘッダとして定義することができます。この場合、あらゆるページの先頭で自動的に出力され、**タスク前**に出力処理を定義する必要がありません（「ページヘッダ / フッタの情報を定義するには」（503 ページ）を参照）。

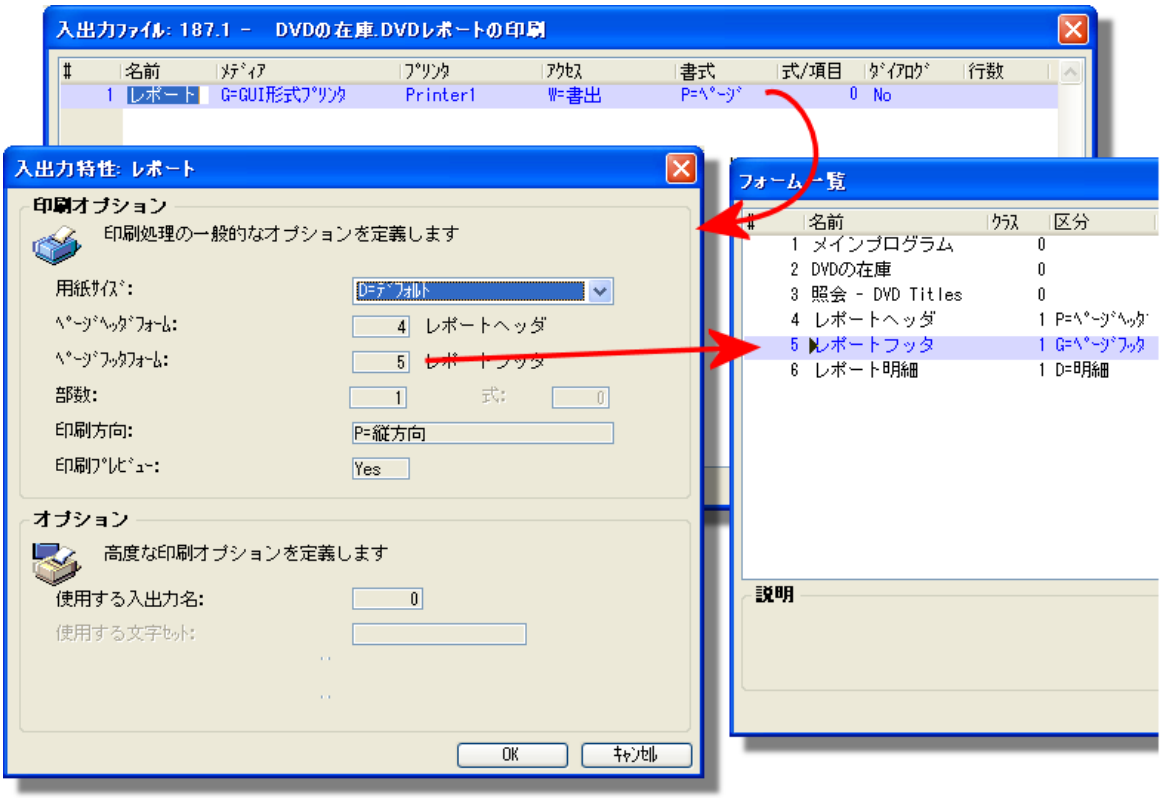
フッタを出力する

帳票フッタは最後のページが印刷された後に**タスク後**で出力されます。この場合（最後のレコードが印刷された後に）1 回だけ印刷されます。このため**タスク後**に処理を定義します。小計を印刷するためにブレイクが発生した場合にもフッタを出力することがあります。これについては、「ブレイクレベル毎の総計を定義するには」（513 ページ）を参照してください。

これでヘッダとフッタを含めた帳票出力プログラムが作成できました。

参照：
「帳票のブレイクレベルを作成するには」（511 ページ）
「ブレイクレベル毎の総計を定義するには」（513 ページ）
「帳票に複数行のコントロールを定義するには」（514 ページ）
「帳票のテーブルに対して見出しを繰り返し出力させるには」（516 ページ）

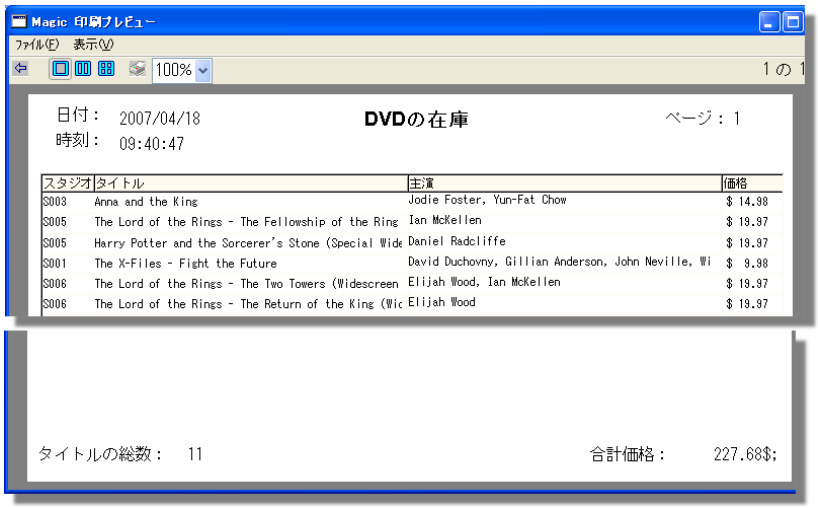
ページヘッダ / フッタの情報を定義するには



ページヘッダと ***フッタ***は入出力ファイルの入出力特性で指定されます。位置は自動的に処理されます。 ***ページヘッダ*** はページの先頭に、 ***ページフッタ***はページの最後に出力されます。

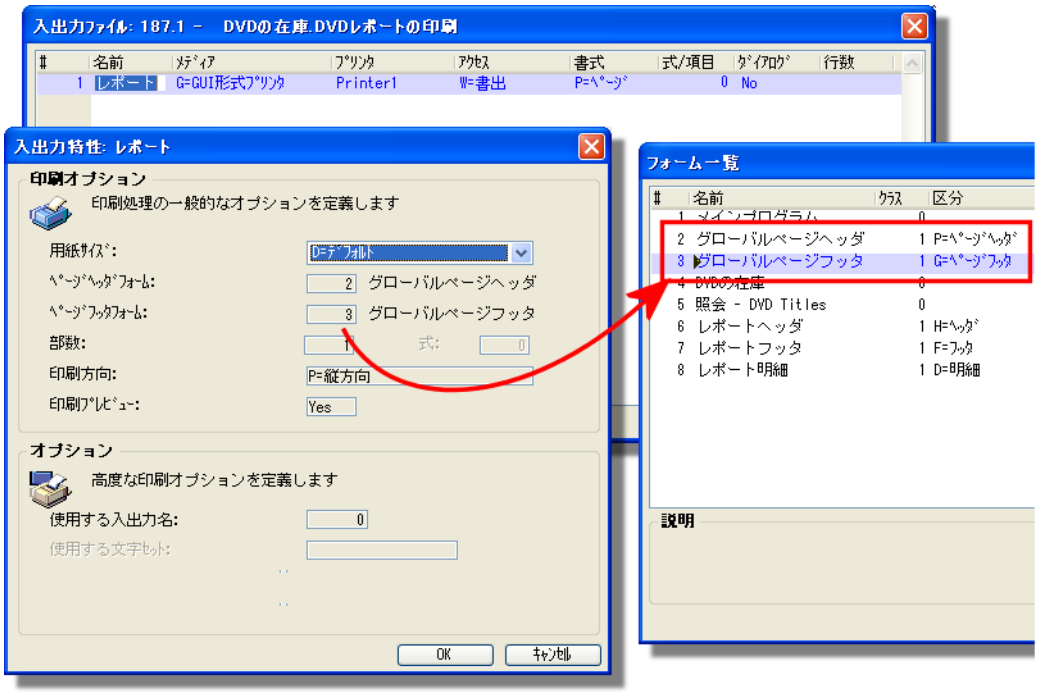
ページヘッダと ***フッタ***は、必ずしも帳票のヘッダやフッタと同じではないことに注意してください。 ***ページヘッダ***と ***フッタ***はすべてのページに出力され、すべてのページが同じ種類の帳票のものであるようにします。例えば、典型的な ***ページフッタ***は各ページの下部の同じ位置に出力され、罫線とページ番号のみが出力されるようにします。しかし、 ***レポートフッタ***は合計を出力したり、帳票の最終行のすぐ後に出力されたりします。

ページヘッダと ***フッタ***のために使用されるフォームは、現在のタスクに定義される必要はありません。 ***ページヘッダ***をメインプログラムで定義し、すべての帳票フォームで利用することもできます。この方法については、「グローバルなページヘッダ / フッタを定義するには」（504 ページ）を参照してください。



ヒント : *印刷などのページのタイプにもとづいたヘッダやフッタフォームの出力内容を変更するために可視特性を使用することができます。ある条件下で表示される項目と、その他の条件で表示される項目を組み合わせることで、条件内容にもとづいて出力されれる項目の内容を変えることができます。*

グローバルなページヘッダ / フッタを定義するには



帳票をページヘッダ付きで出力する方が便利な場合があります。帳票の書式の一貫性が保たれたり、新しい帳票を作成する際の手間を省く場合もあります。また、会社名が変更された場合に帳票の設定を簡単に変更することもできます。

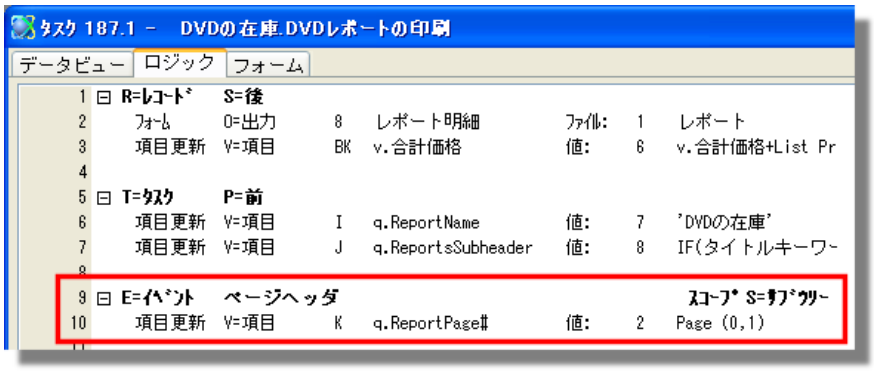
ページヘッダと ***フッタ***は、入出力特性で定義するため、タスクのローカルフォームからグローバルなフォームに切り替えることが簡単にできます。

グローバルフォームを作成し使用する

メインプログラムでフォームを作成することにより、グローバルなフォームを簡単に作成することができます。作業手順は、通常のタスクでフォームを作成する方法とまったく同じです。

フォームが作成されると、すべてのタスクのフォームエディタ上にローカルフォームより上に表示されます。上記の例では、2 つのグローバルなフォームが表示されています（フォーム #2 とフォーム #3）。このタスクで定義されているローカルなフォームは、フォーム #6 と #7 です。

グローバルなページヘッダ / フッタ上でデータ項目を扱う

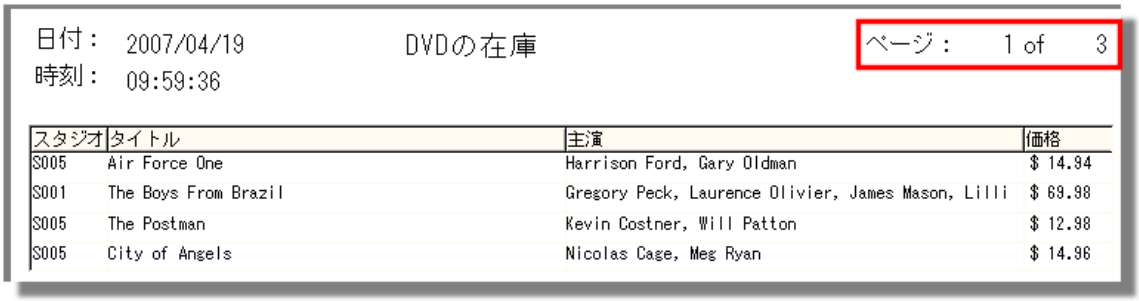


グローバルなヘッダとフッタを作成する際にいくつか考慮する点があります。

第一に、ヘッダには一般的に帳票名が出力されます。これは、何らかのフィルタ機能を使用して出力するようにします。グローバル変数をメインプログラムに定義し、タスク前でこれらを更新させることで簡単に処理することができます。

第二に、メインプログラムでは、帳票用に使用される **Page(*n,n*)** 関数が動作しないため、ローカルタスク内でページ数の値を更新させる必要があります。これは、ページヘッダイベントのロジックユニット内で処理を定義するで実現できます。ページヘッダイベントが実行されると、グローバル変数として定義されたページ番号を更新するようにします。上記の例では、入出力ファイルは現在のタスクの最初に定義されたもののため、**Page(*0,1*)** が定義されています。

帳票ページを列挙するには



日付： 2007/04/19　　DVDの在庫　　ページ： 1 of 3
時刻： 09:59:36

スタジオ	タイトル	主演	価格
S005	Air Force One	Harrison Ford, Gary Oldman	$ 14.94
S001	The Boys From Brazil	Gregory Peck, Laurence Olivier, James Mason, Lilli	$ 69.98
S005	The Postman	Kevin Costner, Will Patton	$ 12.98
S005	City of Angels	Nicolas Cage, Meg Ryan	$ 14.96

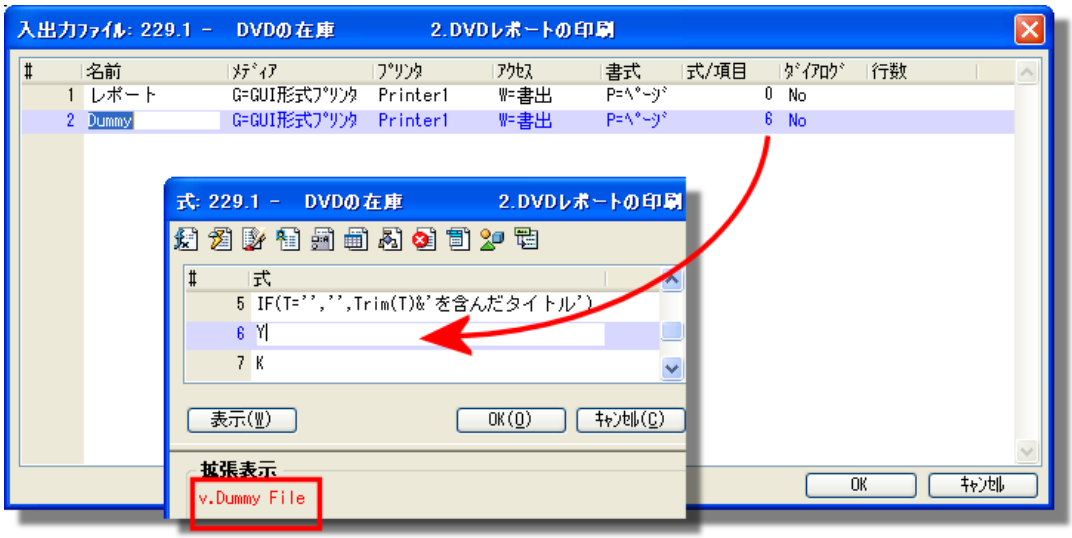
ユーザによっては、「*n of nn*」という付番方式を使用してページ番号を出力する場合があります。

しかし、出力するレコード数や指定されたプリンタをもとに全体のページ数を事前に取得することは不可能です。印刷マージンやフォントを変更するだけでもページサイズは変わります。このような処理を実現するには、出力処理を２回（1 番目：ページ総数の取得、２番目：実際の出力処理）繰り返す必要があります。

このようにすれば、Magic で実現することができます。以下でこの方法について説明します。

この例では、ヘッダかフッタに出力するページ番号をメインプログラム内で格納するようにします。従って、変数項目の g.Report Page# は現在のページ番号を保持し、g.Report Total Pages は出力するページの総数を保持するものとして定義します。グローバルなページヘッダ／フッタの使用方法については、「グローバルなページヘッダ / フッタを定義するには」（504 ページ）を参照してください。

1. ２つの入出力ファイルを定義する



帳票出力用のサブタスクには、2 つの入出力ファイルを定義します。最初は、実際に出力するプリンタです。2 番目は、ファイル（%TEMP%Dummy.txt のような）として出力されるものです。ファイル名を保持するために、親タスクで変数項目を定義し、**FileDelete()** 関数を使用してサブタスクが終了した時点でファイルを削除するようにします。

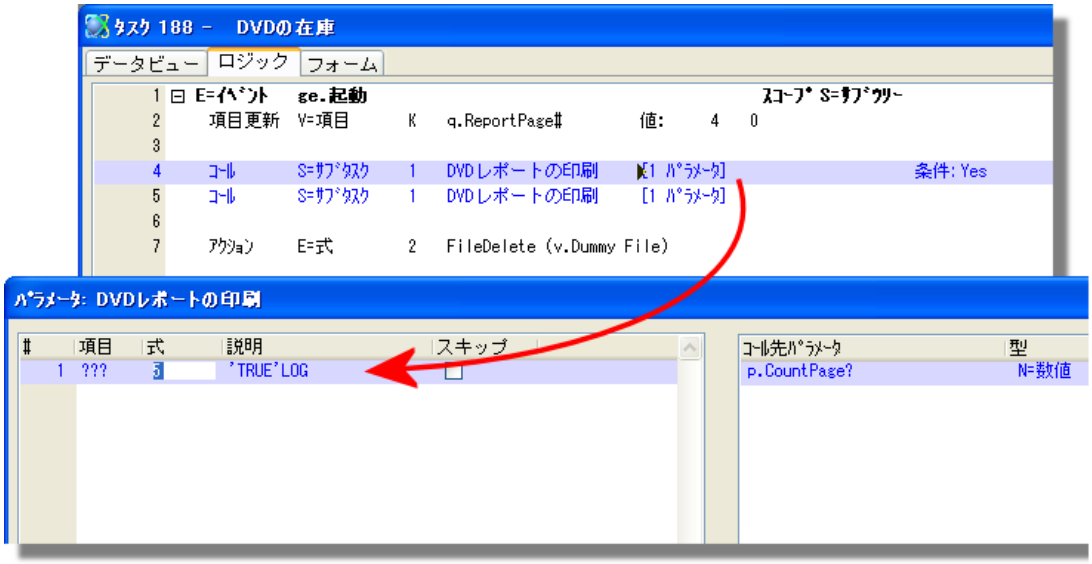
このファイル名の存在は、実際の印刷から出力内容を保護する働きをします。データが印刷されているかのようにフォーマットされますが、実際はファイルに保存されます。

これらは同じ出力先として定義されていることが重要な点です。例えば、同じヘッダとフッターを使用していても、出力先が異なる場合はページ数が同じにならない場合があります。

また、出力用のサブタスクでパラメータ項目（p.Count Page?）を作成します。これが True の場合、タスクはダミーを出力し、実際には印刷されません。

注： 通常は、両方の入出力ファイルで同じプリンタを定義することになります。しかし、Windows に複数のプリンタがセットアップされており、同じ処理を２回繰り返さないように警告を出すようにプリンタドライバが設定されている場合もあります。このような場合、プリンタテーブル（オプション→設定→プリンタ）で DummyPrinter を設定してください。

2. タスクの呼び出し処理を定義する



タスクの呼び出し処理では、4 つの処理を定義します。

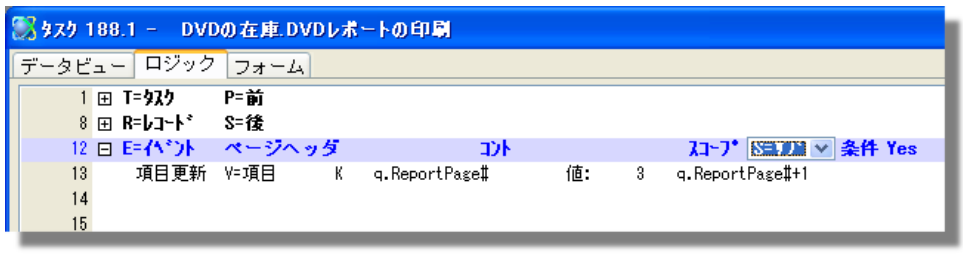
1. グローバル変数「g.ReportTotalPages」の値を 0 に設定します。
2. p.Count Pages? = TRUE としてパラメータを渡し、サブタスクを呼び出します。ここでは、ページ総数を算出します。
3. p.Count Pages? = FALSE としてパラメータを渡し、サブタスクを呼び出します。ここでは、実際に印刷処理を行います。
4. 一時ファイルを削除します。

3. タスク前で出力項目を初期化する

サブタスクでは、グローバル変数「g.Report Page#」やその他の演算用の変数項目の値を初期化します。

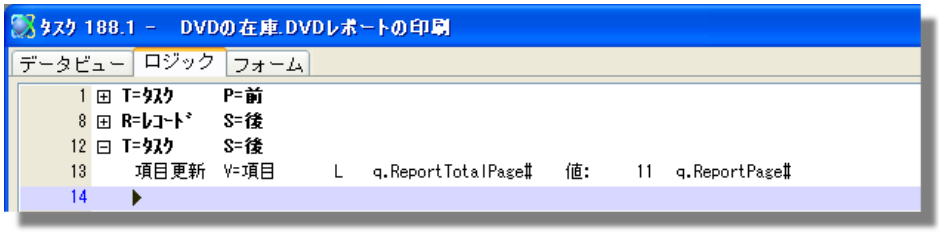
注： タスク実行される際に、自動的に値が初期設定されるため、総額のような項目を 0 に設定する必要はありません。しかし、時々タスクが常駐するように設定されている場合もあるため、とにかく 0 に設定しておいた方が無難です。このような場合、ユーザが印刷処理を２回実行した場合、合計が突然 0 に設定される場合があります。

4. ページヘッダイベントのロジックユニットを作成する



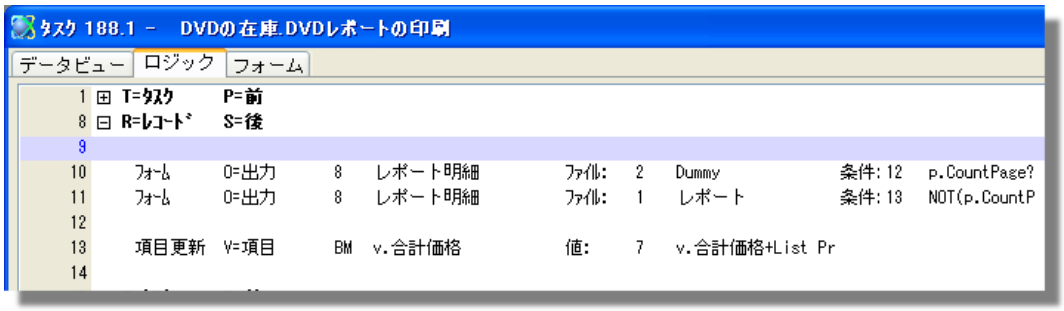
グローバルなページ番号を更新するために、ページヘッダイベントのロジックユニットを作成します。これは、どちらの入出力ファイルに対するページ番号かを注意して作成する必要があります。

5. 最初の処理で取得されたページ総数を格納する



次に、タスク後で、グローバル変数「g.ReportTotalPages」の内容を現在のページ番号によって更新します。この更新処理には条件が定義されており、最初の出力処理でのみ実行されます。結果として返る値は、各出力処理で同じになるため、この処理は厳密には不要ですが、ロジックを明確にするために定義しています。

6. ２つの入出力ファイルに出力する



さて、ここには出力処理に関するすべての処理が定義されています。各フォーム出力処理コマンド（ほとんどの帳票では、あまり多くはありませんが、これはそのほんの一例です）を複写します。p.Count Pages が True の場合、最初のフォーム出力はダミーファイルの出力処理です。p.Count Pages が False の場合、2 番目のフォーム出力はプリンタへの出力になります。

さて、このプログラムが実行されると、サブタスクが 2 回実行され、「*n of nn*」の正確なヘッダデータが取得できます。

複数のプログラムで同じ入出力ファイルを使用して出力するには

日付： 2007/04/19
時刻： 10:40:07

DVDの在庫

ページ： 1

スタジオ	タイトル	主演	価格
S005	The Lord of the Rings - The Fellowship of the Ring	Ian McKellen	$ 19.97
S006	The Lord of the Rings - The Two Towers (Widescreen	Elijah Wood, Ian McKellen	$ 19.97
S006	The Lord of the Rings - The Return of the King (Wic	Elijah Wood	$ 19.97
S006	The Lord of the Rings - The Fellowship of the Ring	Elijah Wood	$ 27.99

スタジオデータ

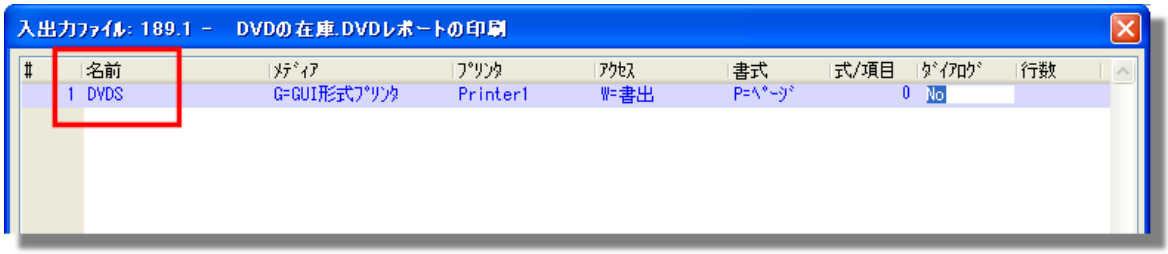
Code	Name
S001	Twentieth Century Fox Home Video
S002	Buena Vista Home Video
S003	Universal Studios
S004	Paramount

入出力ファイルの範囲は、項目やフォームとほぼ同じです。すなわち各タスクは上位タスクの入出力ファイルにアクセスすることができ、プログラム内で使用することができます。

しかし、印刷処理のために全く別のプログラムを呼び出すこともできます。例えば、要約情報を印刷したり、テキストをフォーマットするための汎用プログラムが必要な場合があります。このような場合、**使用する入出力ファイル名**と呼ばれる機能を使用することができます。

この例では、いくつかの帳票の最後に印刷するスタジオデータというプログラムを作成しています。これによって、新規ユーザがスタジオコードを把握することができるようになっています。コードを出力するプログラムをすでに持っている場合、他のプログラムからどのように出力させるようにするかを以下に示します。

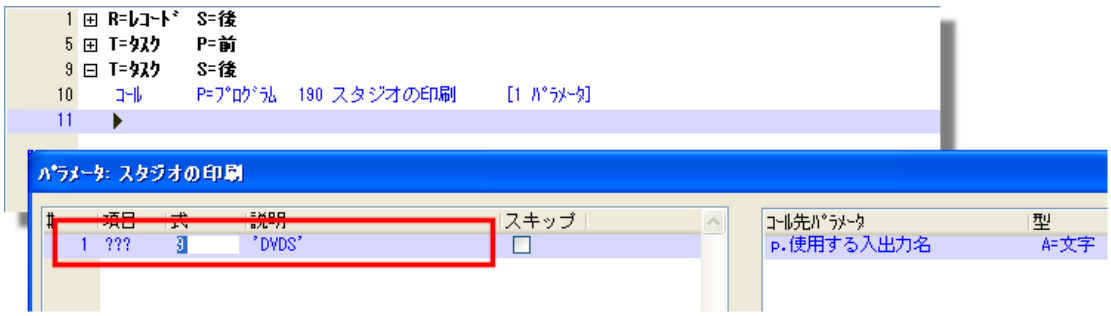
プログラムの呼び出し処理を設定する



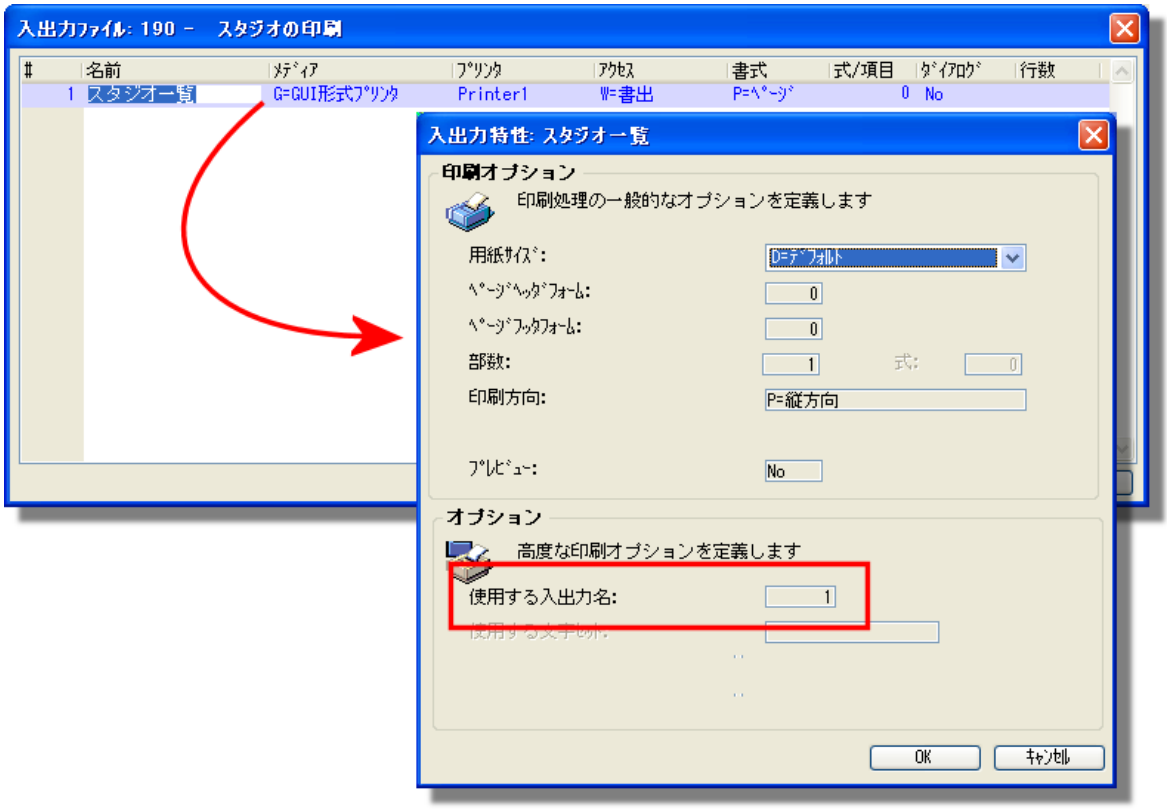
最初に、入出力ファイルを開くプログラムを作成する必要があります。ここでは、以下の２つの定義処理を行います。

1. 入出力ファイルを定義します。空白を含まない簡単な名前を設定します。
2. 入出力ファイル名を呼び出されたプログラムに渡します。プログラム内にハードコートされている場合は必要ありません。

この例では入出力ファイル名を DVDS としています。また、式エディタで DVDS という文字列を定義し、呼び出されたプログラムで渡すようにしています。



起動されるプログラムを設定する



さて、呼び出されるプログラムで入出力ファイルの定義を行います。入出力特性で、使用する入出力ファイル名特性に渡されたパラメータ値を設定します（実行時に DVDS と評価されます）。

これで呼び出されたプログラムの出力内容は、最初のプログラムでオープンされた入出力ファイルで指定されたデバイスに出力されます。

ヒント：*場合によっては、出力用に２つ以上の個別のプログラムを呼び出す「コントロールタスク」を定義する必要があるかもしれません。この場合は、コントロールタスク内で入出力ファイルをオープンします。そのタスクがオンラインタスクであったり、出力処理が定義されていなかったりしている場合、空白ページが出力されることを防止するために、設定→動作環境→動作設定の入出力デバイスのオープンタイミングを 0= 利用時に設定します。*

帳票のブレイクレベルを作成するには


日付： 2007/04/19
時刻： 15:20:44
DVDの在庫
ページ： 1
スタジオ S005 Warner Home Video

タイトル	主演	価格
Harry Potter and the Chamber of Secrets (Widescreer	Daniel Radcliffe, Rupert Grint	$ 19.97
Harry Potter and the Prisoner of Azkaban (2-Disc W	Daniel Radcliffe, Rupert Grint	$ 19.97
Harry Potter and the Sorcerer's Stone (Special Wid	Daniel Radcliffe	$ 19.97

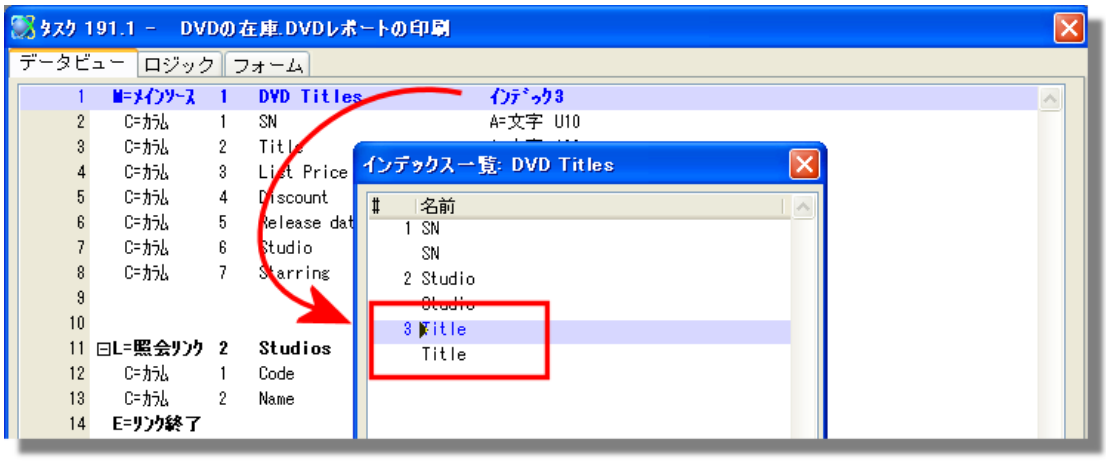
S005 のタイトル数 3 価格 59.91$

データがグループ分けされ、小計を印刷する場合、帳票プログラムではブレイクレベルを定義する必要があります。これは低レベルの言語で行うような処理ですが、Magic には複雑なブレイクレベルであっても簡単に処理できる機能が組み込まれています。このセクションでは、ブレイクレベルの設定方法について説明します。作業手順は以下の通りです。

1. ブレイクレベルに対応したレコードの並び順を設定します。
2. ブレイクの発生対象となるデータ項目を決めます。
3. グループレベルを設定します。
4. 合計を算出します。

次に、これらの手順について説明します。

1. ブレイクレベルに対応したレコードの並び順を設定する



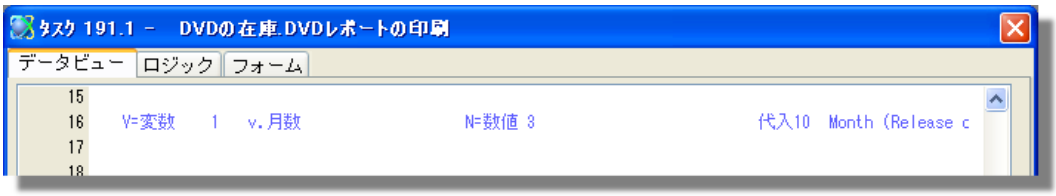
ブレイクレベル帳票を行う上で、最も一般的な間違いの 1 つは間違ったレコードの並び順を使用することです。この例では、Studio コードでレコードをグループ分けしています。従って、Studio インデックスを使用する必要があります。

最適なインデックスが定義されていない場合は、最適な並び順になるようにソート定義（**タスク環境→ソート**）を行う必要があります（第 18 章：「動的なインデックスを作成する」（413 ページ）を参照してください）。

2. ブレイクの発生対象となるデータ項目を決定する

通常、ブレイクレベルで使用するブレイク項目は、DB ソース内のカラムを使用します。この例では、レコードが Studio コード順に出力されるため、Studio カラムを使用してブレイクします。

しかし、適当なカラムが存在しない場合もあります。例えば、月毎にデータを要約する帳票を作成したくても日付カラムしか存在しない場合があります。レコードは日付カラムによって正しくソートされますが、月が変わった場合にブレイクさせるためのカラムはありません。



このような場合、例えばここで示されているように、v.月数と呼ばれる項目を設定し、**Month()** 関数を使用して日付の月の値を設定することでブレイクさせるようにします。または、項目のいずれか 1 つが変わる時だけブレイクが発生するように複数の項目を連結して 1 つのブレイク項目にすることもできます。変数項目は、カラム項目と同じようにブレイクレベルで使用することができます。

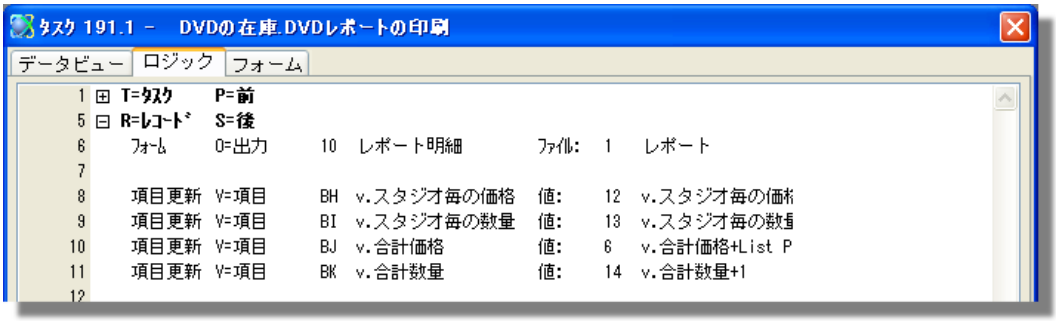
3. グループレベルを設定する

次に、グループレベルを設定します。グループレベルは、ブレイク処理で使用する項目を指定します（この例では、Studio カラムです）。

グループ前では、（必要であれば）ヘッダの出力と合計値の演算用変数の初期化を行います。

グループ後では、（必要であれば）フッタを出力します。グループ後では、フォームの印刷後に変数の初期化を行うこともできます。どちらの方法にしても、アプリケーションで一貫した方法で定義する必要があります。

4. 合計値の計算を行う



最後に、レコードを印刷する場合は常に、合計値を更新する必要があります。通常、このような処理はレコード後で簡単な更新処理によって行われます。

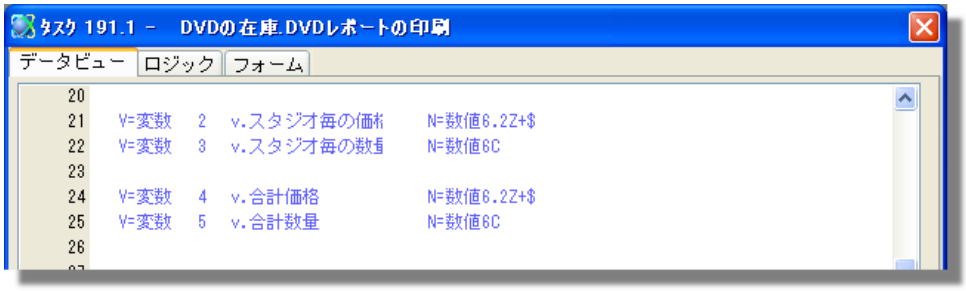
ブレイクレベル毎の総計を定義するには

ブレイクレベル毎の総計を定義するには、以下の３つの手順を実行します。

1. 総計用項目の定義
2. 総計用項目の更新
3. 総計の初期化

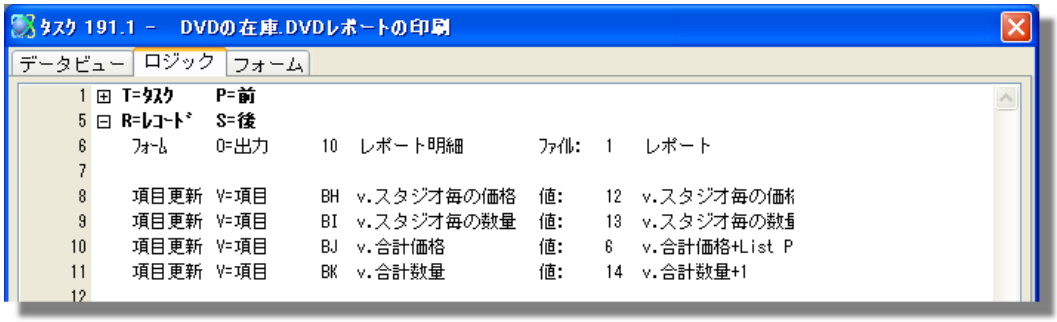
以下これらの手順について説明します。

1. 総計用項目を定義する



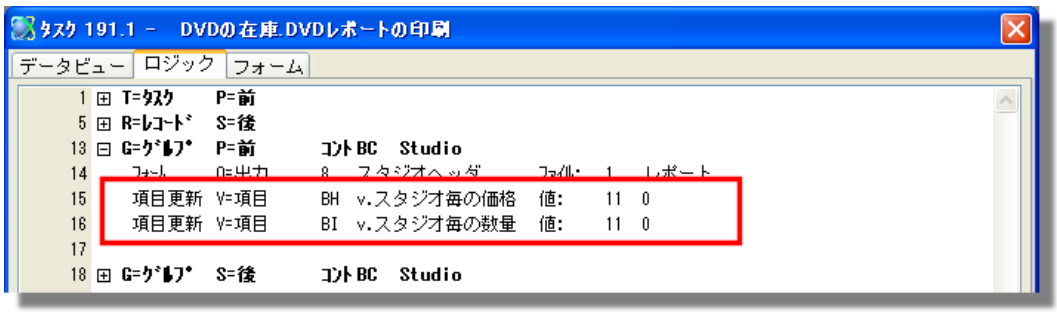
総計用に使用される項目は、通常は変数項目を使用します。これらは、データビューエディタで定義されます。これらの変数項目には目的がわかるような名前を定義しておきます。この例では、どの項目がどのブレイクレベルで使用されるかが分かるように、大文字の接頭辞を追加しています。

2. 総計用項目を更新する



各レコードが読み込まれるたびに、総計用項目を更新するための簡単な方法は、加算することです。この方法により、すべての更新処理は一箇所に定義されます。これによってデバッグしやすくなります。

3. 総計用項目を初期化する



最後に、総計用項目の初期化処理が必要になります。この処理は、グループ前かグループ後で印刷処理の終了後に実行します。

帳票に複数行のコントロールを定義するには

Date: 12/06/2008 DVDs in Stock Page: 1

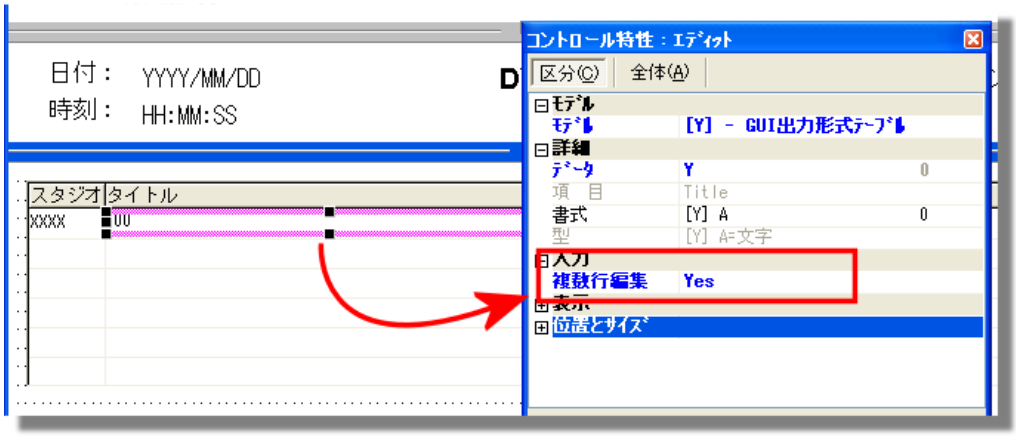
Time: 4:39 pm

Studio	Title	Starring	List Price	Release date
S001	The Boys From Brazil	Gregory Peck, Laurence Olivier, James Mason, Lilli Palmer, Uta Hagen	$ 69.98	09/21/2004
S001	Moulin Rouge (Single Disc Edition)	Nicole Kidman, Ewan McGregor, John Leguizamo, Jim Broadbent, Richard Roxburgh	$ 19.98	01/14/2003
S001	Mystic Pizza	Annabeth Gish, Julia Roberts, Lili Taylor, Vincent D'Onofrio, William R. Moses	$ 14.99	10/21/1988
S001	The X-Files - Fight the Future	David Duchovny, Gillian Anderson, John Neville, William B. Davis, Martin Landau	$ 9.98	01/23/2001
S001	Star Wars Trilogy (Widescreen Edition)	Harrison Ford	$ 69.98	09/21/2004

帳票のデータは、一定でない場合があります。例えば、この例では、タイトルと主演のデータが長すぎたり短すぎたりしています。このような場合、テーブルのカラム内容を折り返して出力するようにした方が効率よく収まります。

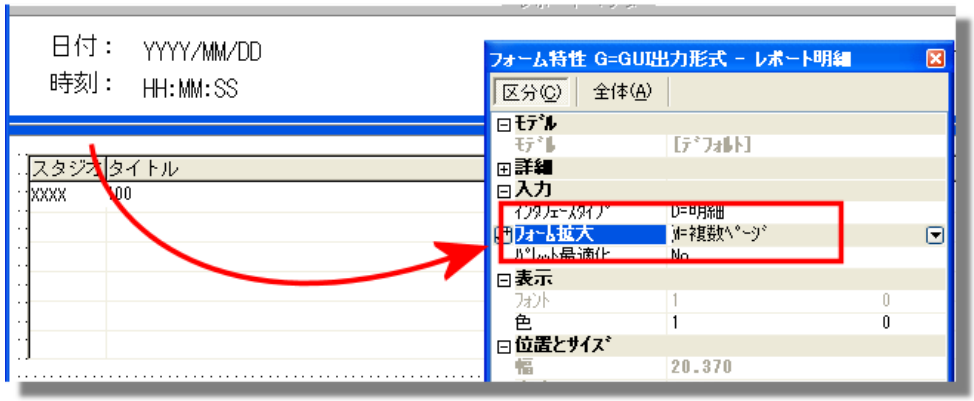
Magic では、**項目特性**を変更することで、データをより多く出力させることができます。

1. コントロール項目特性を変更する



出力フィールドを拡張するには、**コントロール特性**の**複数行編集**特性を *Yes* に設定します。

2. フォーム特性を変更する

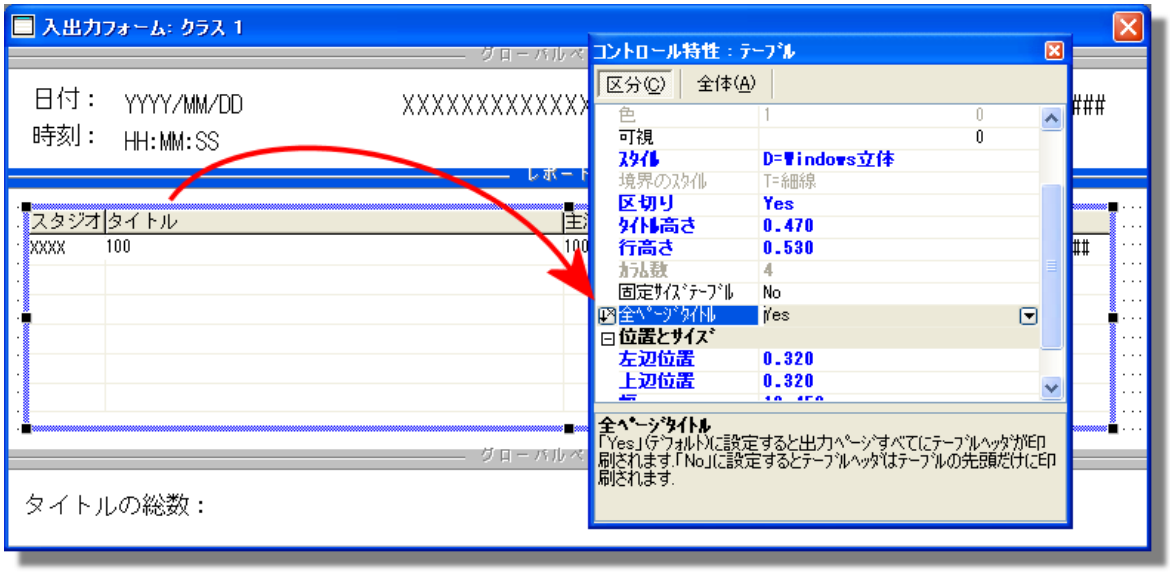


拡張フィールドを含んでいるフォームに対して、**フォーム特性**の**フォーム拡大**特性を *M= 複数ページ* に変更します。

これで、データが 1 行に納まらない場合、テーブル行が拡張して出力されます。

注： 印刷プレビュアーでは、折り返し行の最後の行が正しく表示されない場合があります。これは印刷プレビュアーの問題であり、プリンタには正しく出力されます。

帳票のテーブルに対して見出しを繰り返し出力させるには



テーブルの見出しは、通常各ページごとに先頭に一回出力されます。例えば、コントロールブレイクを持っている場合、テーブルが繰り返される度に出力されます。

この動作は、テーブルコントロールの全ページタイトル特性よって設定できます。ページごとに、見出しを出力させたい場合は、この特性値を *Yes* に設定します。最初のページのみに見出しを出力させたい場合は、*No* を設定します。

PDF ドキュメントを作成するには

以前は、プログラムで作成される帳票は紙に出力されていました。これにより、実際に使用する以上の多くの書類が生成されていました。最近では、PDF 形式で出力されることが標準化されてきました。

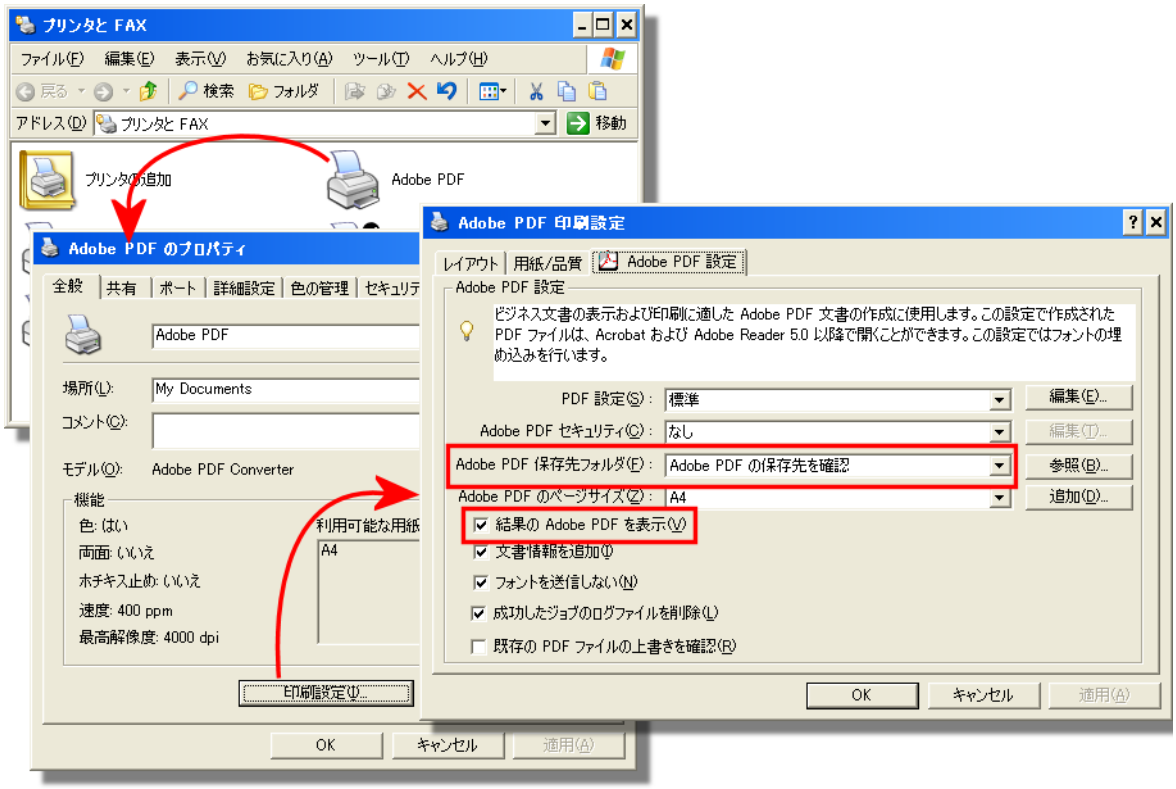
このセクションでは、基本的な PDF ファイルの作成方法について説明しています。PDF の操作に関して、市場でよく利用されている PDF 関連製品を使用して作成された PDF ドキュメントを読み込む方法も提案しています。

PDF が有益ないくつかの基本的な状況があり、それぞれ扱い方が異なります。

1) ユーザが実行時にプリンタを選択する
2) デフォルトの印刷プレビューとして PDF を指定する
3) ユーザの介入なしに PDF を出力するバッチ処理

これらの各状況ごとに対応方法を説明します。

ユーザが出力先を選択する際に PDF の設定を行う



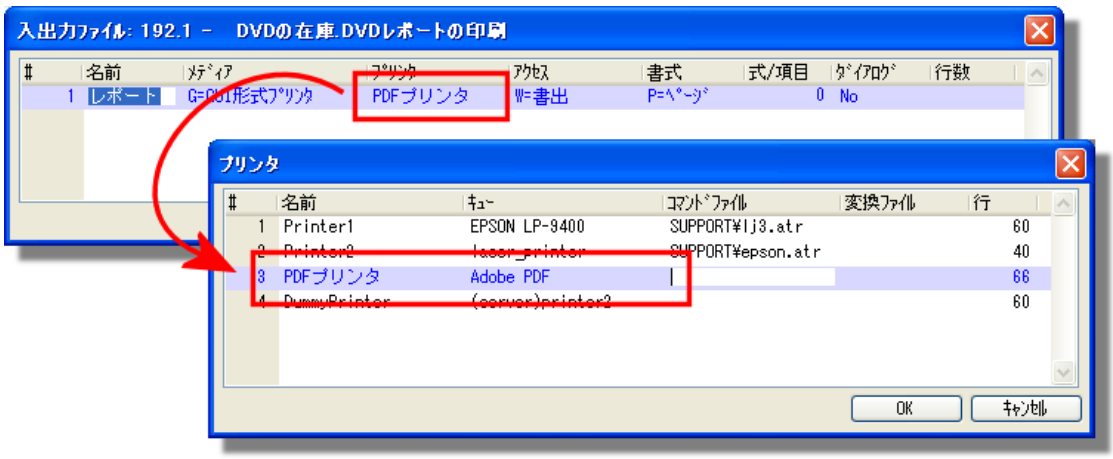
最初のケースは、簡単に設定できます。

1. 単に、出力先のプリンタとして PDF ドライバーを作成するだけです。通常は、PDF 製品がインストールされると自動的に作成されます。具体的な設定内容は、インストールする製品によって異なります。PDF が作成される際に、PDF ドライバーが自動的にオープンされます。
2. 入出力テーブルの**ダイアログ**カラムを *Yes* に設定します。
3. 入出力特性の**印刷プレビュー**特性を *No* に設定します。

これでユーザは、出力先を PDF に指定することができます。この場合、強制的に PDF プリンタを選択させるわけではありません。FAX ドライバーやその他の Windows にインストールされているデバイスドライバを任意に指定することができます。

Windows のプリンタ定義で、ファイル名や解像度、およびデフォルトの出力先の設定を選択することができます。この例では、Adobe PDF というドライバー名が表示されていますが、類似機能の製品を使用することもできます。

デフォルトの印刷プレビューアーとして PDF を設定する



1. 前述の例のように、Windows プリンタドライバを作成します。Windows プリンタドライバに、わかりやすい（空白を含めない、指定しやすい）名前を設定します。上の例では、Adobe PDF という名前で定義されています。
2. プリンタテーブルを開き（オプション→設定→プリンタ）、PDF として出力するために使用するプリンタを作成します。この例では、PDF プリンタという名前で定義されています。キューカラムでは、Windows に定義されているプリンタドライバの名前を入力します。この名前は正確に入力する必要があります。
3. 印刷用タスクに移動し入出力ファイルテーブルをオープンします。#2 で設定した PDF プリンタを選択します。

これで、印刷結果は直接 PDF に出力されます。Windows プリンタドライバが出力された後に自動的にオープンするように設定されている場合、ユーザは実質的に PDF 印刷プレビューを持つことになります。その際、ユーザは PDF にコメントを加えたり、ファイル名を変更して保存したり、別の人に e メールや FAX で送信したりすることができます。

注： コマンドファイルや**変換ファイル**の各カラムは通常空白にしておきます。これらは、特定のインスタンス（シリアルプリンタで専用の制御コードが必要な場合）でのみ使用されます。

ヒント：*印刷に関する社内標準がある場合、上記の方法が便利です。それは、すべての PC が同じ名前のプリンタドライバがインストールされている必要があり、インストール環境の設定を慎重に行う必要があるためです。ユーザは、勝手にプリンタドライバの名前を変更しないようにする必要があります。ドライバ名は、その機能にもとづいてわかりやすい名前（例：受注システムから出力される場合は、「受注」というドライバ名）に出力されるように検討する必要があります。*

バッチジョブ用に PDF を設定する

自動的に帳票内容を PDF に出力させたい場合があります。例えば、50 人の異なる店員のためのレポートを PDF で作成し、各店員にレポートのコピーをメールで送信する場合を考えます。エンドユーザは今まで通りの作業を開始しますが、プリンタを選択したり 50 人の店員用に PDF のファイル名を設定することは面倒な作業になります。

PDF 作成ツールによって設定内容が異なるため、バッチ処理で PDF を作成する方法の説明は多少複雑になります。ここでは、Acrobat を例にとって説明します。

1. 1 つの PDF 毎に入出力ファイルをオープンする処理を作成します。

この例では、DVD レポートを実行しようとしていますが、各スタジオ毎に 1 つの PDF ファイルを作成するようにしています。

これを実現するには、新しいスタジオデータが処理されるたびに、入出力ファイルをオープン／クローズする必要があります。

スタジオ毎にプリンタドライバにアクセスするタスクを作成し、各スタジオデータ毎にこのタスクを呼び出すようにします。

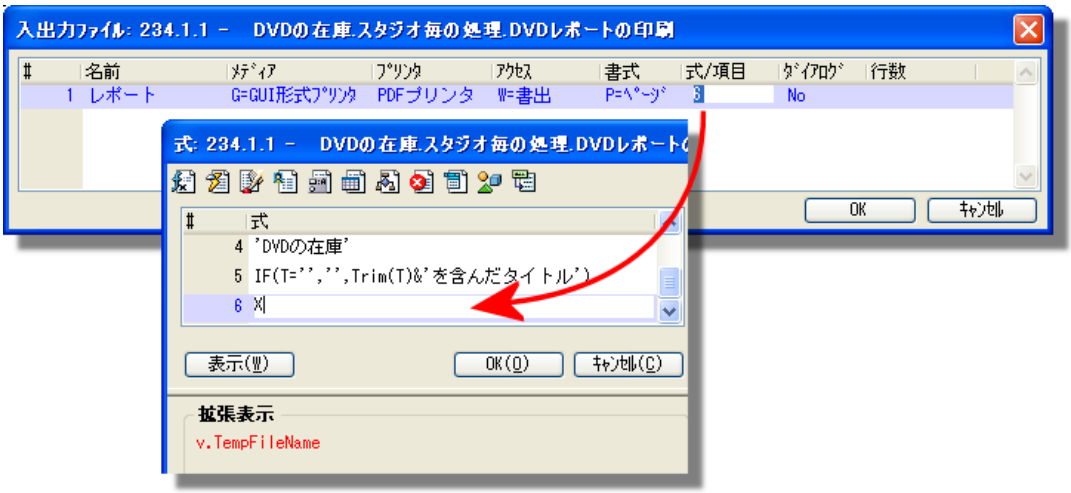
このタスクでは、各 PDF 毎にスタジオ名を含んだファイル名を指定しています。このため、例えばスタジオ S001 の場合、以下のようなファイル名になります。

```
C:¥Temp¥S001DVDS.PS
```

また、最終的な PDF リポートを以下のファイル名で作成します。

```
C:¥PDFReports¥S001DVDS.PDF
```

2. ポストスクリプトファイルを作成します。



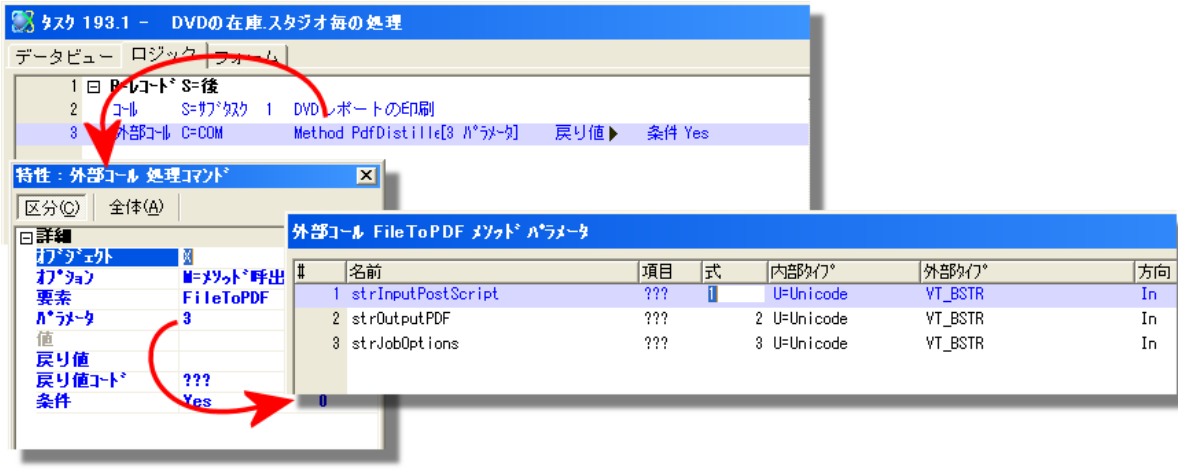
最初に、ポストスクリプトファイルを作成する必要があります。この処理は、入出力ファイルテーブルで指定されたプリンタ経由で自動的に行われます。この例で示されている入出力ファイルは、ファイル名が指定されている以外は、前のセクションで設定した内容とほぼ同じです。処理が実行されると、Acrobat は指定されたファイル名で帳票データを作成します。この例では、スタジオ S001 のための帳票となり C:¥Temp¥S001DVDS.PS と呼ばれるファイルが作成されます。

3. ポストスクリプトを PDF に変換する

次に、ポストスクリプトを PDF ファイルに変換する必要があります。Adobe Distiller のようなポストスクリプトファイルを PDF に変換するソフトを持っており、特定のディレクトリ内のポストスクリプトファイルを変換するために監視するように設定されている場合、何もする必要はありません。

Distiller は自動的に PDF に変換します。しかし、この方法の 1 つの問題は、Distiller の開始や設定内容がユーザに依存していることです。このことはまた、開発者が処理の手順をコントロールできないことを意味しています。例えば、(e メールで送信する処理などの) PDF 作成後に実行されるべき処理が、いつ作成されるか分からないことになります。

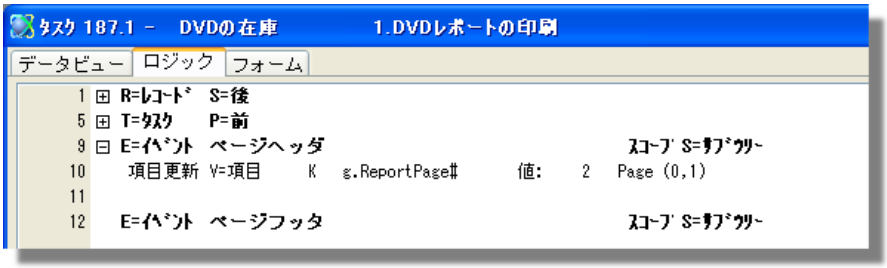
Delay() 関数とブロックループを組み合わせることで PDF が作成されるまで待つようにすることで解決できます。しかし以下に示すように、より洗練された方法として、PdfDistiller の COM オブジェクト（Acrobat Distiller（Ver1.0））のような COM オブジェクトを使用することです。



他の方法でもコントロールすることができますが、COM オブジェクトを呼び出す場合は、入力ファイル（最初のパラメータ）と出力ファイル（２番目のパラメータ）を指定し、３番目のパラメータに空の文字列を指定するだけで済みます。

この例では、スタジオ毎の処理が 1 回処理するたびに、1 つの PDF ファイルが作成されます。この後、PDF を e メールで送信したり、他の処理を行うことができます。

ページ毎に計算処理を実行させるには



ページ毎に合計値やその他の計算結果を算出する必要があるかもしれません。例えば、「ページ毎の総費用」だったり、ページ番号を加算したりする場合などが考えられます。しかし、GUI 形式の帳票ではページ毎に出力する行数は可変のため、印刷中に手動で算出することは非常に難しい処理になります。従って、Magic ではページヘッダとページフッタの２つの内部イベントを提供することで対応しています。これらのイベントを利用することで、ページヘッダやページフッタを出力する前に処理を実行させることができます。

ページヘッダやページフッタ自体は、自動的に出力されるため、これらのイベントを使用する必要はありません。

参照： 「ブレイクレベル毎の総計を定義するには」（513 ページ）
「ページヘッダ / フッタの情報を定義するには」（503 ページ）
「グローバルなページヘッダ / フッタを定義するには」（504 ページ）

帳票にマージンを設定するには

出力するプリンタや帳票の種類によって、周辺の余白のサイズが異なる場合があります。このような場合、印刷時にマージンを指定することで調整することができます。ここでは、マージンの設定方法について説明します。

注： 印刷マージンの設定は、日本語版の Magic でのみ有効です。

プリンタテーブルに設定する

1. **プリンタ**テーブル（**オプション→設定→プリンタ**）を開きます。
2. マージンを設定するプリンタの**キュー**を以下のように設定します。

   Windows のプリンタ名 /M ＜上マージン＞ X ＜左マージン＞ X ＜下マージン＞ X ＜右マージン＞

例えば、Adobe PDF に対して、上マージン：1mm、左マージン：2mm、下マージン：3mm、右マージン：4mm の場合は、以下のようになります。

```
Adobe PDF/M10X20X30X40
```

スタイル設定ユーティリティを使用する

1. **スタイル設定**ユーティリティ（Setstyle.exe）を起動します。
2. **編集対象 MAGIC.INI** 欄に設定したい Magic.ini を選択します。
3. **プリンタ名**欄で、設定したいキューを選択します。
4. スタイル設定を**クリック**し、**スタイル設定**ダイアログを開きます。
5. **マージン設定**のチェックボックスをオンににし、下の表示されている**マージン**設定欄を有効にします。
6. 上下左右のマージン（0.1mm 単位）を設定し、実行を**クリック**します。

全てのキューで同じマージンを設定する

1. **プリンタ設定**ユーティリティ（Mgprn.exe）を起動します。
2. **編集対象ファイル**欄に設定したい Magic.ini を選択します。
3. **印刷形式**を ***GUI*** に設定し、編集を**クリック**し ます。**パラメータ設定**ダイアログが表示されます。
4. **プリンタ名**欄で、設定したいキューを選択します。
5. **マージン設定**のチェックボックスをオンににし、下の表示されている**マージン**設定欄を有効にします。
6. 上下左右のマージン（0.1mm 単位）を設定し、実行を**クリック**します。

［このページは意図的に空白にしています。］

第 23 章 : マージ

テキストファイルにデータをマージするには

マージは、Magic の機能の中でも有益で多用途な機能の 1 つです。Web アプリケーションを作成する場合に利用されるだけでなく、XML や RTF、または HTML などのテキストタイプのファイルにデータをマージさせる場合にも使用できます。

マージを行うタスクは、データを使用してループし、データを出力するための**フォーム出力**処理コマンドを使用するため、帳票の出力用タスクに似ています。帳票用タスクの作成方法については、第 22 章 :「帳票を作成するには」（496 ページ）を参照してください。

このセクションでは、マージ機能の基本的な作成方法について説明します。より詳細な内容は、以降のセクションで説明いたします。

データをマージするのための基本的な手順は以下の通りです。

1. テキストのテンプレートを作成します。
2. 出力ファイルを指定します。
3. マージフォームを作成します。
4. タグを選択します。
5. 各タグをデータと関連付けます。
6. マージフォームを出力します。

次に各ステップの詳細を説明します。

1. テキストのテンプレートを作成する

```
1
2  Dear <!$MG_Guest>
3
4  これは、在庫のあるDVDのリストです。<!$MGIF_KeywordSpecified>次のキーワードが含まれています <!$MG_Ke
5
6  もしどちらの注文かを伝えた場合、電子メールで回答します。
7  AAA DVDレンタルへお掛けくださいましたありがとうございます。
8
9  <!$MG_Salesrep>(<!$MG_Salesphone>)
10
11 ========================================================================
12      ID番号          タイトル                        価格
13 ========================================================================
14 <!$MGREPEAT>
15      <!$MG_SN>      <!$MG_MovieTitle>              <!$MG_Cost>
16 <!$MGENDREPEAT>
17 ========================================================================
```

マージを作成する上での最初の作業は、テンプレートを作成することです。テンプレートファイルは、編集可能なテキストファイルです。この例では、簡単なプレーンテキストを使用しています。

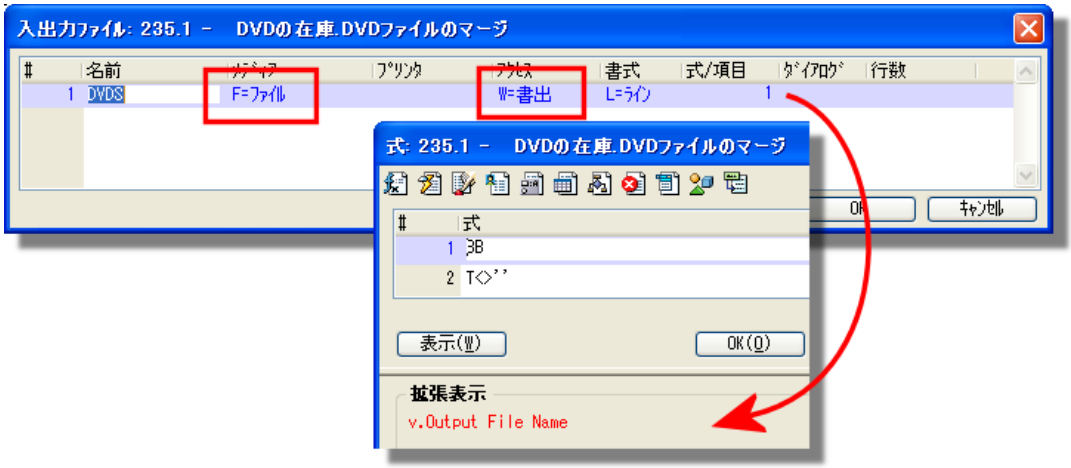
このテンプレートを作成するには、テキストエディタで内容を入力し、DVDMerge.txt という名前で作業ディレクトリに保存します。また、Magic でテンプレートを作成することもできます（「好みの HTML エディタを利用するように設定するには」（536 ページ）を参照してください）。

マージしたいデータ項目がある場合、実行時にその項目と置き換えるためにタグを使用します。各タグは、接頭辞（*!<$MG_*）、タグ名、および接尾辞（>）から構成されます。これらのタグは、大文字小文字を区別しています。

テンプレート内では、**<!$MGREPEAT>** タグを使用することで、データの繰り返しを指定したり、**<!$MGIF>** を使用して、条件付きのテキストを指定することができます。

参照： 「テンプレートに置き換え可能なトークンを設定するには」（534 ページ）
「テンプレートへのデータ挿入を調整するには」（533 ページ）
「HTML テンプレートのテーブルに繰り返し可能なデータを挿入するには」（532 ページ）

2. 出力ファイルを指定する



マージを行うタスクは、データを使用してループし、データを出力するための**フォーム出力**処理コマンドを使用するため、帳票の出力用タスクに似ています。ただし、入出力ファイルやフォームの定義方法が異なります。

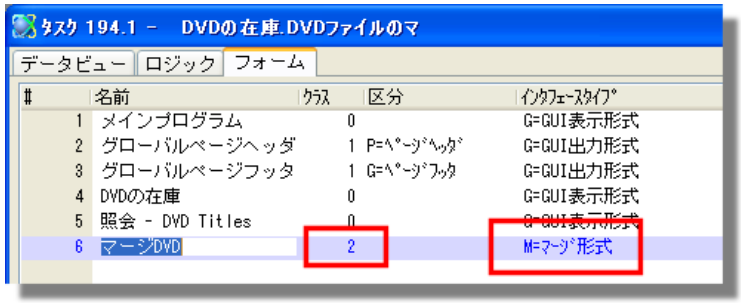
入出力ファイルの**メディア**カラムを ***F= ファイル***に設定し、**アクセス**カラムは ***W= 書出***に設定します。**式 / 項目**カラムにはファイル名を指定します。

注： Web アプリケーションでマージを利用する場合は、**メディア**カラムには ***R= リクエスタ***を設定したり、**メディア**カラムを ***V= 項目***に設定した場合は、BLOB 型項目に格納するように設定するなど、色々な組み合わせがあります。ただし、基本的な手順に違いはありません。

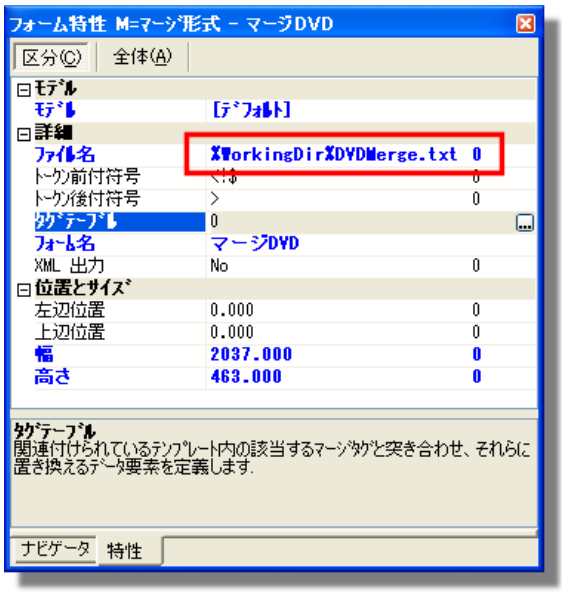
3. マージフォームを作成する

次に、マージフォームを作成します。このフォームは他のフォームと同じように**フォームエディタ**で作成します。この場合、**インタフェースタイプ**は ***M= マージ形式***に設定します。

また、他のフォームとは異なるクラスを設定する必要があります。この例では、既にグローバルな ***ページヘッダ***と ***フッタ***がクラス =1 で定義されているため、マージフォームのクラスは *2* に設定されています。



4. マージフォームの特性を設定する



次に、マージ形式フォームのフォーム特性（Alt+Enter）を開きます。

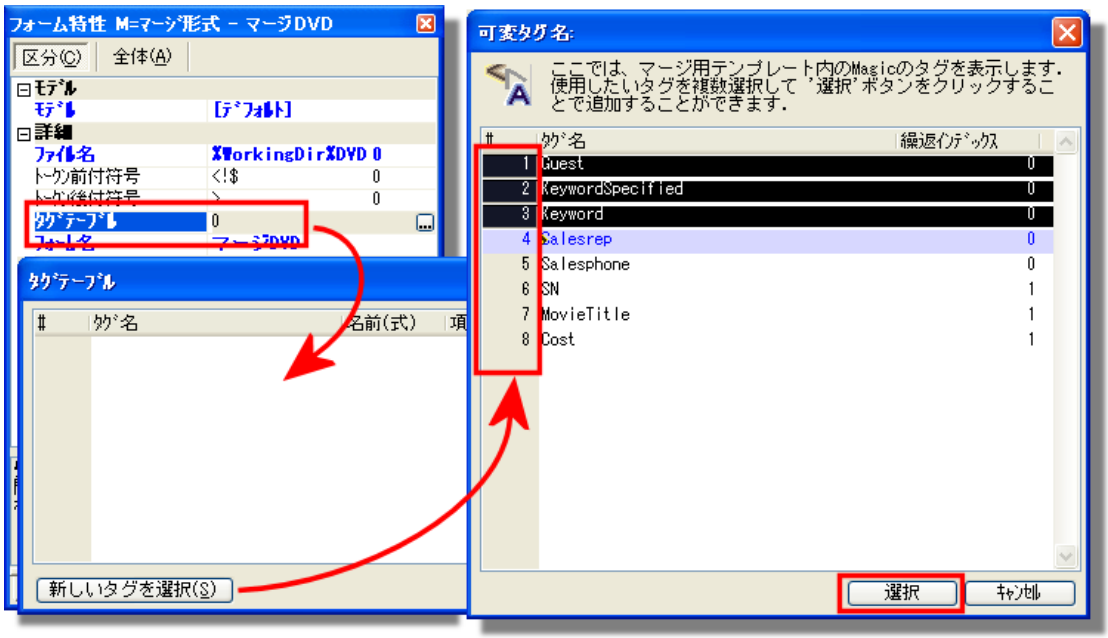
ここには、ファイル特性にテンプレートファイル名を設定します。テンプレートのパスに論理名を使用することで、システム環境に依存しないようにすることができます。

ここにテンプレートファイルを指定すると、フォームエディタの名前カラムでズームすることでテンプレートを編集することができます。この操作は、テンプレート名が正しく設定されているかどうかを確認することにもなります。

また、トークン前付符号特性とトークン後付符号特性はここで変更することができます。変更した場合、テンプレート内に定義されたタグは異なるものとして扱われます。例えば、トークン前付符号特性を *<！$* から *<#%* に変更した場合、Cost タグは、**<!$MG_Cost>** から **<#%MG_Cost>** に変更する必要があります。保守の観点から、本当に必要でない限り、トークン前付符号特性とトークン後付符号特性を変更しないようにしてください。

ファイル名を指定したら、タグテーブル特性から、ズームしてください。

5. タグを選択する



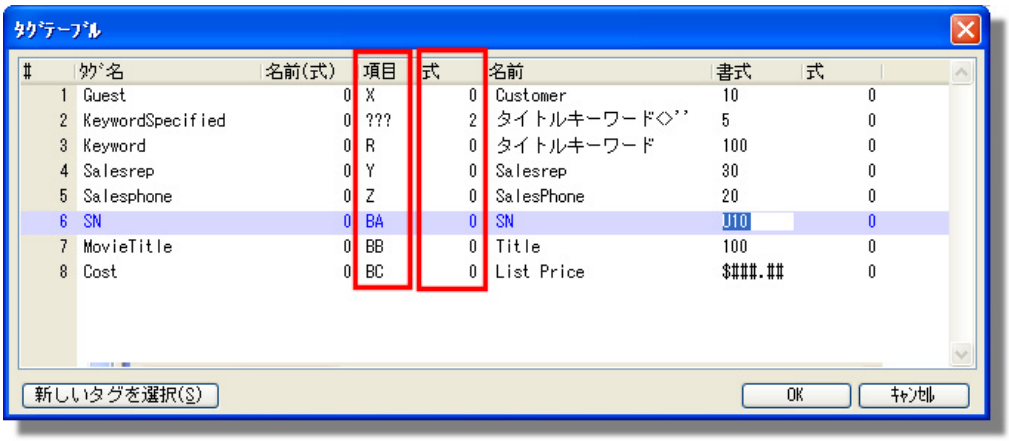
次に、タグを選択する必要があります。以下の手順で選択します。

1. フォーム特性のタグテーブル特性に移動し、ズーム（*F5* または、ダブルクリック）します。新規フォームの場合、タグテーブルは空のままです。
2. 新しいタグの選択ボタンをクリックし、テンプレートに定義されているタグ名のリストを表示します。
   タグのリストが表示されない場合、以下のような原因が考えられます。
   - すでにすべてのタグが選択されている。この場合は、ビープ音が鳴り、画面の下に「テンプレートファイルに新しいタグが見つかりませんでした」というメッセージが表示されます。
   - タグ名に誤りがある（トークン前付符号特性やトークン後付符号特性がテンプレートに定義されたタグ名と合っていない）。
   - テンプレートファイル名に誤りがある。ファイル名が正しい場合、フォームエディタからズームすることでファイルが表示されるため、この方法で確認できます。
3. 最初のカラム（行番号が表示されています）をクリックすることで、タグを選択できます。タグをクリックすると黒く反転表示されます。*Ctrl+* クリックを使用して複数のタグを選択することができます。すべてのタグが選択されたら、選択ボタンをクリックします。

4. 選択されたすべてのタグは**タグ**テーブルに表示され、次の手順に進むことができます。

注： タグ名を直接入力することもできますが、選択した方が簡単で確実です。

6. 各タグをデータと関連付ける



前の手順が正しく行われた場合、**タグ**テーブルにタグが定義されます。次に、各タグと値と関連付ける必要があります。値は項目または式のどちらかで指定します。

項目を指定する場合は、**項目**カラムから**ズーム**し、**項目**テーブルから選択します。

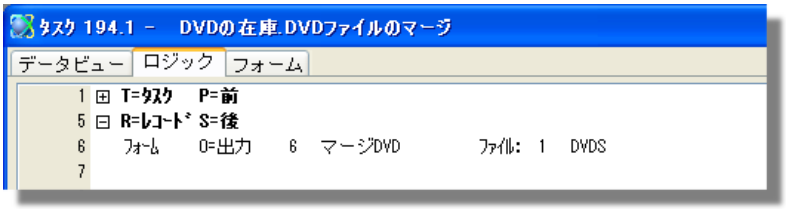
式を指定する場合は、**式**カラム（**項目**カラムのとなり）から**ズーム**します。

この作業は、帳票用にデータを選択する場合と同じではありますが、いくつか異なる点もあります。大きな違いの 1 つは、データが自動的に調整されることです。

例えば、タイトル長が 100 文字であっても、それがマージされると、100 桁の文字として出力されません。

別の違いは、マージの場合、一般にヘッダやフッターフォームがないことです。すべてのデータは 1 つのテンプレートとマージされます。MGREPEAT タグは、繰り返す要素がどこにあるかを指定します。

7. マージフォームを出力する



最後に、フォームの出力処理を定義する必要があります。通常は、処理される各レコード毎に 1 回のデータ出力が行われるため、**レコード後**で定義します。

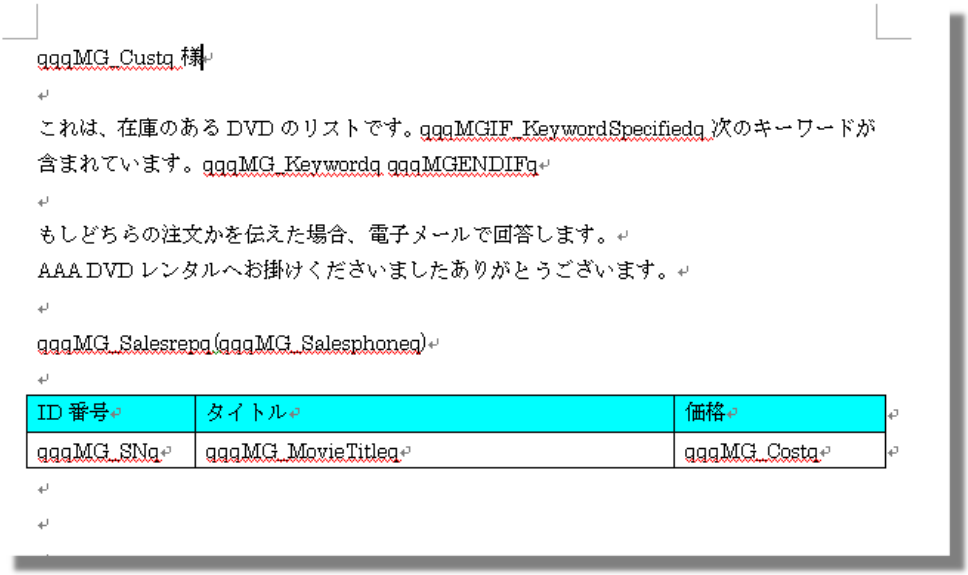
これにより以下のような結果が出力されます。

```
Frank Smit様

これは、在庫のあるDVDのリストです。

もしどちらの注文かを伝えた場合、電子メールで回答します。
AAA DVDレンタルへお掛けくださいましたありがとうございます。

Jill(555.555.1212)

================================================================================
     ID番号        タイトル                          価格
================================================================================
     0784012717     The Boys From Brazil              $ 69.98
     B00003CWLF     Anna and the King                 $ 14.98
     B00003CXCT     Star Wars Trilogy (Widescreen Edition)                $ 69.98
     B000052210     The X-Files - Fight the Future                $  9.98
     B000053VB4     Mystic Pizza                      $ 14.99
     B000077VR3     Moulin Rouge (Single Disc Edition)                $ 19.98
     B00009AOBK     Gotcha!                 $ 14.99
     B0000DZ3GQ     Midnight Madness                  $ 14.99
     B0003JAONG     Cloak & Dagger                 $  9.99
     B0006H32DY     The Palm Beach Story              $ 12.99
================================================================================
```

フィールドがどのように自動的にトリミングされているかを確認してください。顧客名項目は 30 文字の長さですが、コロンがタグのすぐ隣のため、名前の後のコロンはトリミングされた名前のすぐ隣になります。

しかし、タイトルフィールドがトリミングされると、価格カラムは整列されません。価格カラムを整列させるには、テンプレート内でタブ文字を使用するか、異なる種類のフォーマット（HTML など）を使用する必要があります。

動的に Word ドキュメントを作成するには



qqqMG_Custq 様

これは、在庫のある DVD のリストです。qqqMGIF_KeywordSpecifiedq 次のキーワードが含まれています。qqqMG_Keywordq qqqMGENDIFq

もしどちらの注文かを伝えた場合、電子メールで回答します。
AAA DVD レンタルへお掛けくださいましたありがとうございます。

qqqMG_Salesrepq(qqqMG_Salesphoneq)

ID 番号	タイトル	価格
qqqMG_SNq	qqqMG_MovieTitleq	qqqMG_Costq

1. 必要な文書を Word で作成します。
2. タグを追加したい場所に、特殊な文字列（qqqMG_Custq など）を追加します。特殊文字は次の手順で Word によって変換されるため、**<!$MG_Cust>** というような書式は使用できません。
3. 文書を HTML 形式で保存し、Word を終了します。

```
</tr>
<!$MGREPEAT>
 <tr style='mso-yfti-irow:1;mso-yfti-lastrow:yes'>
  <td width=119 valign=top style='width:89.6pt;border:solid windowtext 1.0pt;
  border-top:none;mso-border-top-alt:solid windowtext .5pt;mso-border-alt:solid windowtext .5pt;
  padding:0mm 5.4pt 0mm 5.4pt'>
  <p class=MsoNormal><span class=SpellE><span lang=EN-US><!$MG_SN></span></span></p>
  </td>
  <td width=384 valign=top style='width:288.0pt;border-top:none;border-left:
  none;border-bottom:solid windowtext 1.0pt;border-right:solid windowtext 1.0pt;
  mso-border-top-alt:solid windowtext .5pt;mso-border-left-alt:solid windowtext .5pt;
  mso-border-alt:solid windowtext .5pt;padding:0mm 5.4pt 0mm 5.4pt'>
  <p class=MsoNormal><span class=SpellE><span lang=EN-US><!$MG_MovieTitle></span></span></p>
```

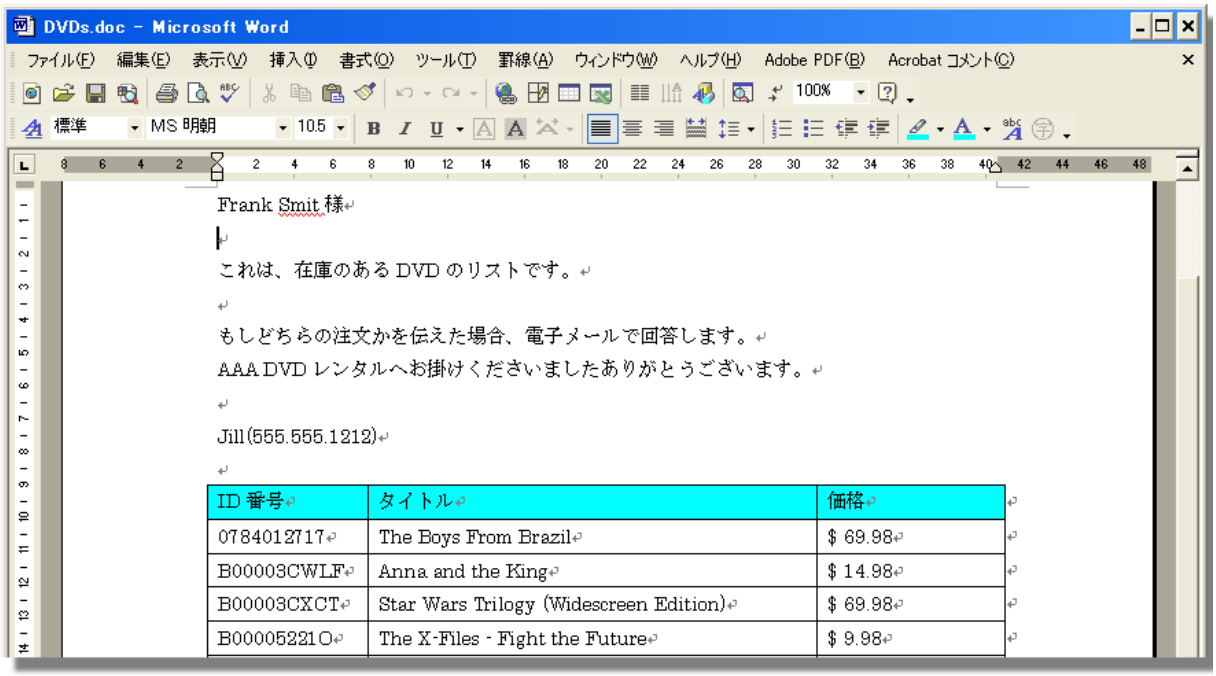
4. テキストエディタで HTML を編集します。qqqMG_Custq のような特別なタグを **<!$MG_Cust>** などの Magic タグに置き換えます。必要に応じて、MGREPEAT と MGIF を追加します。

ヒント : *qqq のようなユニークな文字列を使用した場合、検索 / 置換コマンドを使用して作業することができます。*

5. マージテンプレートとしてこの HTM ファイルを使用します。
6. 出力ファイル名の拡張子に、DOC を使用します。

これで文書が作成すると、Word の文書として Microsoft Word でオープンできるようになります。

注： この方法は、RTF 形式に対しても利用できます。



動的に Excel ドキュメントを作成するには

1. 必要な文書を Excel で作成します。
2. タグを追加したい場所に、特殊な文字列（qqqMG_Custxx など）を追加します。特殊文字は次の手順で Excel によって変換されるため、**<!$MG_Cust>** というような書式は使用できません。
3. 文書を HTML 形式で保存し、Excel を終了します。

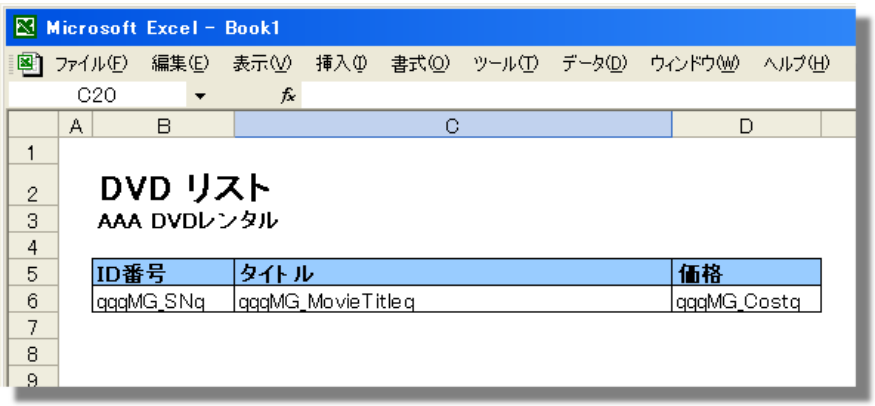
```
 </tr>
<!$MGREPEAT>
 <tr height=18 style='height:13.5pt'>
  <td height=18 style='height:13.5pt'></td>
  <td class=xl24 style='border-top:none'><!$MG_SN></td>
  <td class=xl24 style='border-top:none;border-left:none' x:str="<!$MG_MovieTitle> "><!$MG_MovieTitl
  <td class=xl24 style='border-top:none;border-left:none'><!$MG_Cost></td>
  <td></td>
 </tr>
<!$MGENDREPEAT>
 <tr height=18 style='height:13.5pt'>
  <td height=18 colspan=5 style='height:13.5pt;mso-ignore:colspan'></td>
```

4. テキストエディタで HTML を編集します。qqqMG_SNxxx のような特別なタグを **<!$MG_SN>** などの Magic タグに置き換えます。必要に応じて、MGREPEAT と MGIF を追加します。

ヒント :*qqq のようなユニークな文字列を使用した場合、検索 / 置換コマンドを使用して作業することができます。*

5. マージテンプレートとしてこの HTM ファイルを使用します。
6. 出力ファイル名の拡張子に、XLS を使用します。

これで文書が作成されると、Excel の文書として Excel でオープンできるようになります。



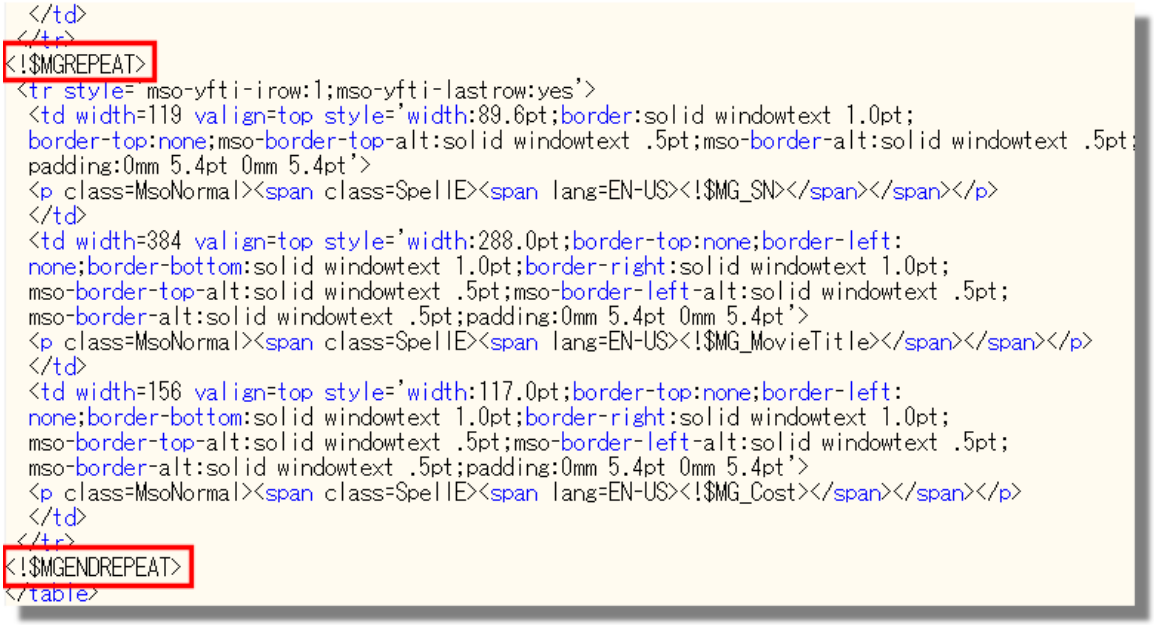
テンプレートに繰り返し可能なデータを挿入するには

```
10
11 ================================================================================
12     ID番号          タイトル                              価格
13 ================================================================================
14 <!$MGREPEAT>
15     <!$MG_SN>      <!$MG_MovieTitle
16 <!$MGENDREPEAT>
17 ================================================================================
18
```

```
11 ================================================================
12     ID番号          タイトル                          価格
13 ================================================================
14     0784012717      The Boys From Brazil                $ 69.98
15     B00003CWLF      Anna and the King                 $ 14.98
16     B00003CXCT      Star Wars Trilogy (Widescreen Edition)
17     B000052210      The X-Files - Fight the Future
18     B000053VB4      Mystic Pizza                $ 14.99
19     B000077VR3      Moulin Rouge (Single Disc Edition)
20     B00009AOBK      Gotcha!                $ 14.99
21     B0000DZ3GQ      Midnight Madness                  $ 14.99
22     B0003JAONG      Cloak & Dagger                  $  9.99
23     B0006H32DY      The Palm Beach Story                $ 12.99
```

繰り返し可能なデータは **<!$MGREPEAT>** タグによって処理されます。**<!$MGREPEAT>** と **<!$MGENDREPEAT>** 間にある内容は、フォームが出力されるたびに繰り返されます。従って、この例において、レコード後の中でフォーム出力処理コマンドが実行されています。フォームは、各タグ（SN、タイトル、および価格）を更新するため、これらの内容が繰り返し出力されます。これは、**GUI 出力形式**フォームでのテーブルコントロールの動作に似ています。

HTML テンプレートのテーブルに繰り返し可能なデータを挿入するには



```
  </td>
 </tr>
<!$MGREPEAT>
 <tr style='mso-yfti-irow:1;mso-yfti-lastrow:yes'>
  <td width=119 valign=top style='width:89.6pt;border:solid windowtext 1.0pt;
  border-top:none;mso-border-top-alt:solid windowtext .5pt;mso-border-alt:solid windowtext .5pt;
  padding:0mm 5.4pt 0mm 5.4pt'>
  <p class=MsoNormal><span class=SpellE><span lang=EN-US><!$MG_SN></span></span></p>
  </td>
  <td width=384 valign=top style='width:288.0pt;border-top:none;border-left:
  none;border-bottom:solid windowtext 1.0pt;border-right:solid windowtext 1.0pt;
  mso-border-top-alt:solid windowtext .5pt;mso-border-left-alt:solid windowtext .5pt;
  mso-border-alt:solid windowtext .5pt;padding:0mm 5.4pt 0mm 5.4pt'>
  <p class=MsoNormal><span class=SpellE><span lang=EN-US><!$MG_MovieTitle></span></span></p>
  </td>
  <td width=156 valign=top style='width:117.0pt;border-top:none;border-left:
  none;border-bottom:solid windowtext 1.0pt;border-right:solid windowtext 1.0pt;
  mso-border-top-alt:solid windowtext .5pt;mso-border-left-alt:solid windowtext .5pt;
  mso-border-alt:solid windowtext .5pt;padding:0mm 5.4pt 0mm 5.4pt'>
  <p class=MsoNormal><span class=SpellE><span lang=EN-US><!$MG_Cost></span></span></p>
  </td>
 </tr>
<!$MGENDREPEAT>
</table>
```

HTML 内でのテーブルの基本的なフォーマットは以下の通りです。

```
<table>
  <tr >
    <td> ..Header stuff ..... </td>
  </tr>
  <tr >
    <td> ..Detail line stuff ... </td>
  </tr>
</table>
```

テーブルをマージテンプレートに変換するための秘訣は、2 つの行を定義することです。一つはヘッダ用で、もう一つは MGREPEAT エリア用です。

最も簡単な方法は、現在使用しているツール（Dreamweaver や Excel、Word）で HTML を編集することです。この 2 つの行以外を削除します。

そして、テキストエディタ（または Dreamweaver のソースコードエディタ）を使用して、**<!$MGREPEAT>** と **<!$MGENDREPEAT>** で最後のテーブル行を囲みます。これにより以下のようになります。

```
<table>
  <tr >
    <td> ..Header stuff ..... </td>
  </tr>
  <!$MGREPEAT>
  <tr >
    <td> ..Detail line stuff ... </td>
  </tr>
  <!$MGENDREPEAT>
</table>
```

テンプレートへのデータ挿入を調整するには

一定の条件が満たした場合だけ、データを挿入したい場合があります。このような場合、MGIF タグを使用します。MGIF タグは条件付きのデータを取り囲むように定義します。例えば以下のように定義します。

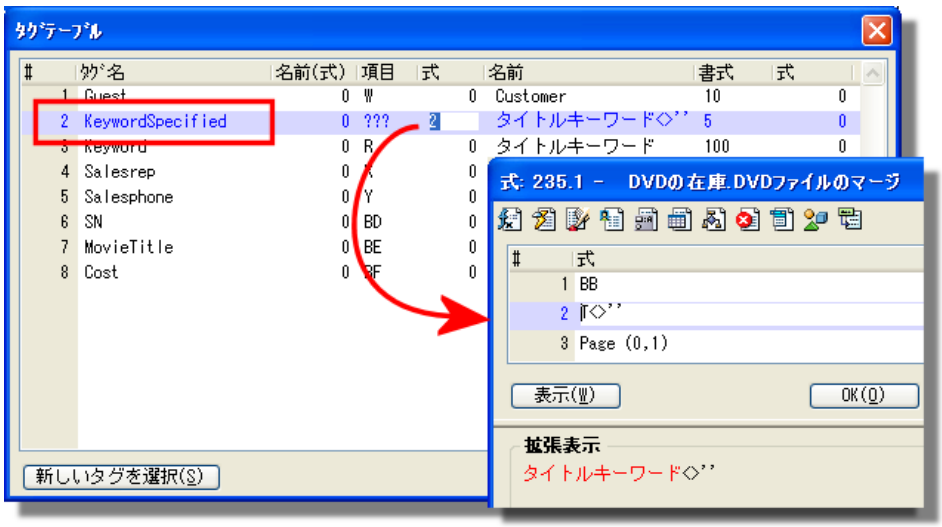
> これは、在庫のある DVD のリストです。<!$MGIF_KeywordSpecified> 次のキーワードが含まれています <!$MG_Keyword><!$MGENDIF>。

KeywordSpecified タグが True の場合、テキスト「次のキーワードが含まれています <!$MG_Keyword>」がテキスト内に含まれ、False の場合は無視され、以下のテキストとして扱われます。

> 「これは、在庫のある DVD のリストです。」

条件を指定する

Magic で、KeywordSpecified が True かどうかを判定するデータを定義する必要があります。これは*タグ*テーブルで行います。



タグ名は、MGIF タグを使用します。MGIF タグは以下のテキストを読み込みます。

```
<!$MGIF_KeywordSpecified>
```

式は、論理値として扱います。これにより、エンドユーザがキーワードを入力したかどうかを確認することができます。キーワードが入力された場合、項目「***K***」は空白にならず、式は True として評価されます。

テンプレートに置き換え可能なトークンを設定するには

置き換え可能なトークンの最も簡単な設定方法は以下の通りです。

1. よく利用しているエディタ（例えば Dreamweaver や Excel または Word）を使用して、テンプレートを編集します。
2. 置き換え可能なトークンを設定したい場所に、qqqMG_Custxx のような簡単に分かる文字列を入力します。
3. Excel か Word を使用している場合、結果を HTML 形式で保存します。
4. 文字列を **<!$MG_Cust>** などのトークンと変更します。

この例は、「動的に Word ドキュメントを作成するには」（528 ページ）を参照してください。

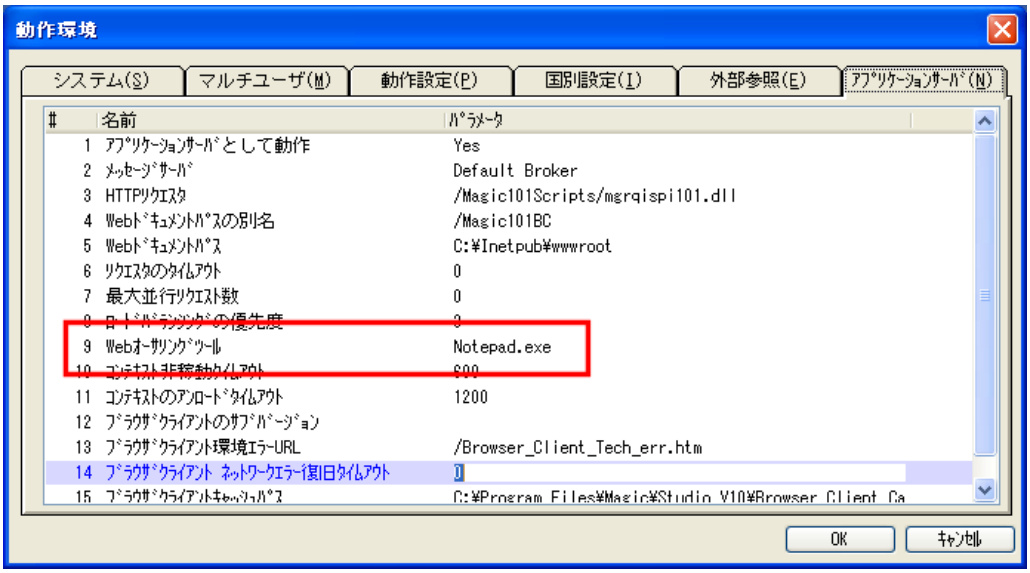
HTML タグとマージのトークンを区別するには

```
<!$MGREPEAT>
 <tr height=18 style='height:13.5pt'>
  <td height=18 style='height:13.5pt'></td>
  <td class=xl24 style='border-top:none'><!$MG_SN></td>
  <td class=xl24 style='border-top:none;border-left:none' x:str="<!$MG_MovieTitle> "><!$MG_MovieTitle>
  <td class=xl24 style='border-top:none;border-left:none'><!$MG_Cost></td>
  <td></td>
 </tr>
<!$MGENDREPEAT>
```

HTML タグとマージトークンは、同じように前後を **< >** によって囲まれています。しかし、マージトークンは特定の文字列（通常は、**<!$MG_**）から始まります。**フォーム特性**の値を変更することで最初の 3 文字を指定することができますが、MG_ は Magic で使用する固定の文字として設定されています。

従って、上記のコードでは、マージトークンがすべて **<!$MG** で始まるため、**<tr>** と **<td>** などの HTML タグとマージトークンを区別することができます。

好みの HTML エディタを利用するように設定するには



フォームエディタ上でマージフォームの名前カラムでズームすると、あらかじめ指定されたフォームテンプレートがオープンされます。タグを指定している際に、テンプレートを編集することができるため便利な機能です。

利用できるオーサリングツールにはたくさんの種類があります。このツールは、オプション→設定→動作環境→アプリケーションサーバの Web オーサリングツールで設定します。この例では、*Notepad.exe* が設定されています。

複数のタスクで 1 つのドキュメントにデータをマージするには

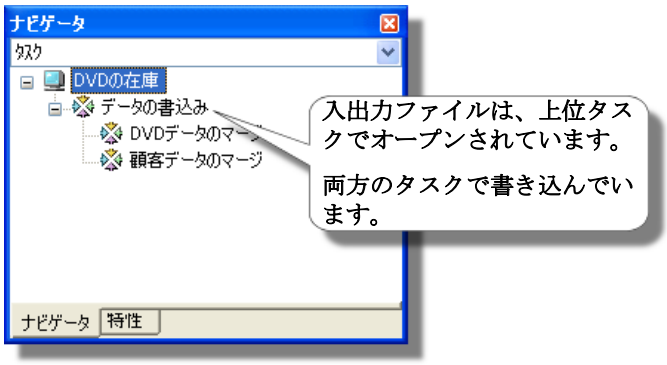
この場合、以下のようなタスクの定義方法によって異なる対応が必要です。

- 各タスクが同じプログラムに定義されており、同じタスクのサブタスクとなっている場合
- タスクが異なるプログラムで定義されている場合

これらについて以下で説明します。

注： この方法は、帳票を印刷するために使用された方法と同じです。第 22 章：「複数のプログラムで同じ入出力ファイルを使用して出力するには」（509 ページ）を参照してください。

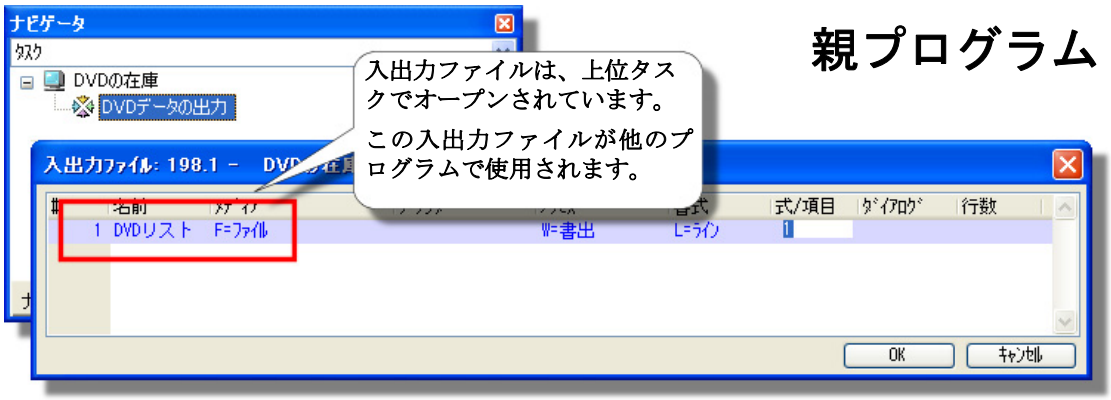
同じプログラム内の２つのタスクからのデータをマージする



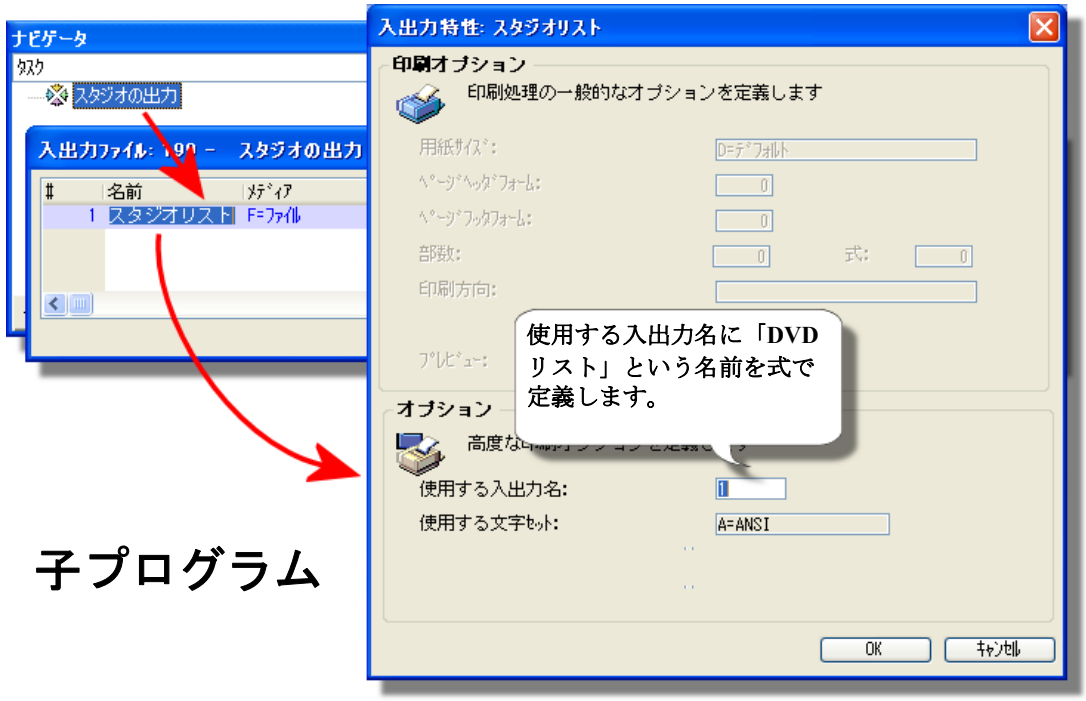
1 つのドキュメントにデータをマージさせる必要がある２つのタスクが同じプログラム内に定義されている場合、帳票処理と同じように対応することができます。つまり、親タスクで定義された入出力ファイルを２つのタスクで共有して使用することです。

２つの異なるプログラムのデータをマージする

２つの異なるプログラムでデータをマージする場合は、入出力特性の使用する入出力名特性を使用します。以下に示すように、親タスクで入出力名を設定します。



これで、同じファイルやリクエスタに送られる出力内容を持つプログラムで、この同じ名前を使用することができます。



子プログラムは、異なるファイル名が指定されていても、親プログラムでオープンされたファイルに出力します。

注： 両方のプログラムで同じメディアタイプを指定するようにしてください。一方のプログラムでファイルを指定し、他方がプリンタになっている場合、構文チェックではエラーにはなりませんが、実行時に一貫性のない結果となります。

テンプレートを使用して、データをグループ化して出力するには

HTML テンプレートを使用してデータを出力し、データをグループ化したい場合、最もよい方法は HTML テーブルを使用することです。1 つのファイル内に複数の HTML テーブルを定義し、スタイル定義を行うことで見栄えよく出力することができます。

例えば、1 つの HTML フォーム内でグループ分けされ、２つの個別のリストがあります。

S/N	タイトル
B00003CWT6	The Lord of the Rings - The Fellowship of the Ring (Widescreen Edition)
B00005JKZV	The Lord of the Rings - The Two Towers (Widescreen Edition)
B00005JKZY	The Lord of the Rings - The Return of the King (Widescreen Edition)
B000067DNF	The Lord of the Rings - The Fellowship of the Ring (Platinum Series Special Extended Edition)

スタジオコード	スタジオ名
S001	Twentieth Century Fox Home Video
S002	Buena Vista Home Video
S003	Universal Studios
S004	Paramount
S005	Warner Home Video
S006	New Line Home Entertainment

２つの異なるタスクが、同じファイルにデータを出力するために使用されます（「複数のタスクで 1 つのドキュメントにデータをマージするには」（537 ページ））。

ここで示されるように、使用されたテンプレートは、MGREPEAT タグを使用して HTML テーブルを作成しています。

最初のタスクでは、DVD データを出力しています。ここでは、最初の 2 つのタグ（MG_SN、および MG_Title）を参照しているだけです。

２番目のタスクでは、スタジオデータを出力しています。ここでは、MG_Studio と MG_StudioName の行を書き込んでいます。

```
<table class="MGW_TableControl">
  <tr class="MGW_TableHeader">
    <td ><b>S/N</b></td>
    <td ><b>タイトル</b></td>
  </tr>
   <!$MGREPEAT>
  <tr>
    <td ><!$MG_DVD_SN></td>
    <td ><!$MG_TITLE></td>
  </tr>
   <!$MGENDREPEAT>
</table>
<p></p>
<table  class="MGW_TableControl">
  <tr class="MGW_TableHeader">
    <td ><b>スタジオコード</b></td>
    <td ><b>スタジオ名</b></td>
  </tr>
   <!$MGREPEAT>
  <tr>
    <td> <!$MG_STUDIO_CODE></td>
    <td ><!$MG_STUDIO_NAME></td>
  </tr>
   <!$MGENDREPEAT>
</table>
```

また 1 つのデータソースにあるデータをグループ化するようなことも可能です。この例では、スタジオコードにもとづいて DVD データをグループ化しています。

ここでは、個別の HTML テーブルが各グループ毎に作成されています。

これで、どのようにして作成されたか理解できると思います。

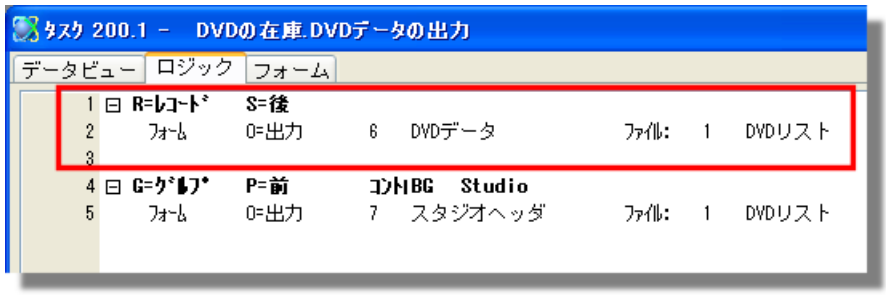
1. テンプレートを設定する

```
14 <FORM Name="Studios">
15   <!$MGREPEAT>
16   <H3> スタジオ： <!$MG_STUDIO_CODE> <!$MG_STUDIO_NAME> </H3>
17   <table class="MGW_TableControl">
18     <tr class="MGW_TableHeader">
19       <td >S/N</td>
20       <td >タイトル</td>
21     </tr>
22     <!$MGREPEAT>
23     <tr>
24       <td ><!$MG_DVD_SN></td>
25       <td ><!$MG_TITLE></td>
26     </tr>
27     <!$MGENDREPEAT>
28   </table>
29 <!$MGENDREPEAT>
```

テンプレートはネストされた MGREPEAT タグを使用しています。内側の MGREPEAT は、帳票の場合と同じように**詳細**行を作成するものです。MG_DVD_SN と MG_TITLE の 2 つのタグは、**レコード後**で出力されるため、1 レコードあたり 1 行が出力されます。

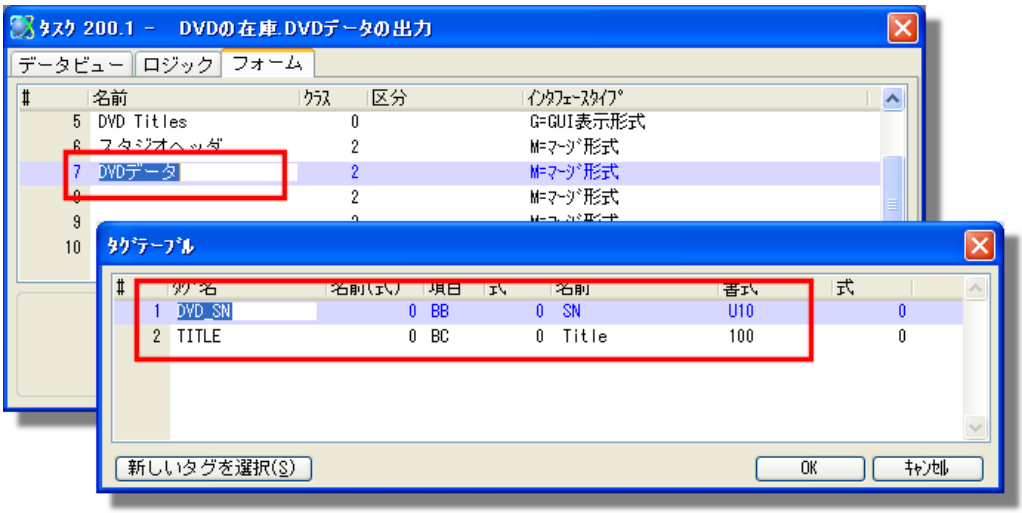
外側の MGREPEAT はスタジオ名を大文字で出力し、HTML テーブルとテーブルヘッダーの定義を処理します。

2. 詳細行を書き込む

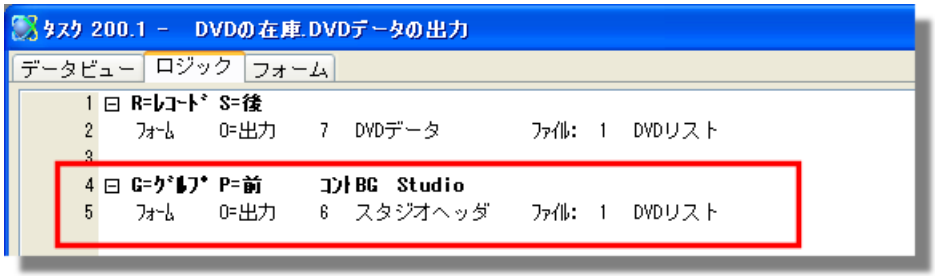


帳票の出力処理と同じように、明細行は**レコード後**で出力されます。

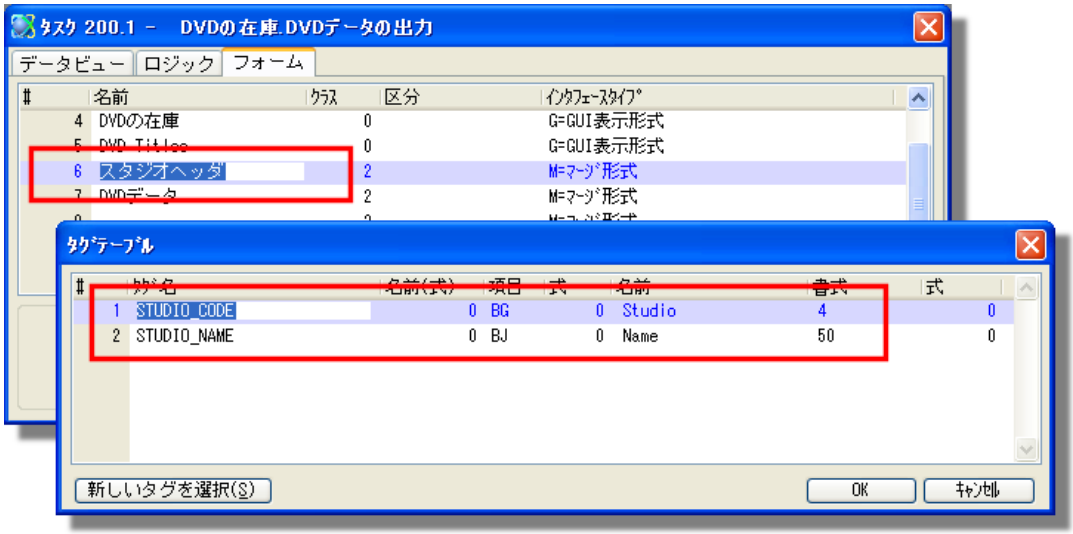
しかし、出力されるフォーム（DVD データ）は、内部の MGREPEAT タグのみ参照します。従って、フォーム出力処理コマンドは 1 レコード分の DVD データフォームのみを書き込みます。



3. ヘッダを書き込む



ヘッダは、グループ後ロジックユニットで出力します。参照する項目の内容が変更された場合のみ、このロジックユニットは実行されます。データビューで Studio インデックスを使用しているため、スタジオ毎にスタジオヘッダが出力されます。GUI 出力フォームで帳票を出力する場合と同じように動作します。



スタジオヘッダのフォームは、外側の MGREPEAT タグ内のタグのみ参照します。従って、フォーム出力処理コマンドが実行されると、新しい **<H3>** ヘッダと HTML テーブルが書き込まれます。

テンプレートにファイルを埋め込むには

```
1  <html>
2
3  <head>
4  <!$MGINCLUDE>C:¥MAGIC¥Projects¥Exsample¥StyleSheet.htm<!$MGENDINCLUDE>
5  </head>
6
```

```
1  <head>
2    <meta http-equiv="Content-Language" content="ja">
3    <meta http-equiv="Content-Type" content="text/html; charset=shift_jis">
4   <style>
5    .MGU_Body {BACKGROUND-COLOR: White; COLOR: #000000;}
6    .MGU_BCBody {BACKGROUND-COLOR: White; COLOR: #000000; filter;
7     progid:DXImageTransform.Microsoft.Gradient(GradientType=0, StartColorStr='#410099FF',
8     EndColcrStr='#00000000');}
9    .MGU_TableControl{BACKGROUND-COLOR: #EBEBEB; BORDER: 1px groove Black;}
10   .MGU_TableHeader {COLOR: #000000; Font-family: "ＭＳ Ｐゴシック";BACKGROUND-COLORR:
11    #FFE8C6;Font-Size:10pt;Font-weight:bold;}
12   .MGU_TableRow {COLOR: #000070; Font-family: "ＭＳ Ｐゴシック";BACKGROUND-COLORR: #FFE8C6;
13   Font-Size:10pt;}
14    .MGU_HypertextControl{COLOR: #F000F7;}
15   </style>
16 </head>
```

<!$MGINCLUDE>** タグを使用することで、ファイル全体を埋め込むことができます。この例では、実行時にファイルの中にスタイルシート（StyleSheet.htm）を埋め込むために、**<!$MGINCLUDE> を使用しています。

<!$MGINCLUDE> の構文

<!$MGINCLUDE> の構文は以下の通りです。

<!$MGINCLUDE>***<file name>***<!$MGENDINCLUDE>

<file name> には、埋め込むファイルの名前を定義します。ファイル名のためにタグを使用することができます。ファイルが完全にマージされた後で、埋め込み処理が実行されます。

マージ Web アプリケーションを作成するには

ここでは、マージ機能を利用した Web アプリケーションについて説明いたします。Magic は、マージ機能使用して Web アプリケーションを開発することができます。

基本的なマージ Web アプリケーション

マージ形式を使用したプログラムの例として、**商品一覧**というデータテーブルの内容を表示する簡単な Web プログラムを紹介します。このプログラムは、「商品番号」をパラメータとして渡されるとその番号以降を表示します。

表示するデータソースの内容

データソースのカラムは以下の通りです。

名前	型	書式
商品番号	N= 数値	5Z
商品名	A= 文字列	20
単価	N= 数値	N7CZ

HTML ファイルを作成する

HTML オーサリングツールを使用して HTML ファイルを作成します。このファイルに Magic のデータがマージされます。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=shift_jis">
</head>
<body>
<p align="center"><font size="6"> 商品一覧 </font></p>
<hr>
<div align="center">
   <table border="1">
      <tr>
         <td> 商品 #</td>
         <td> 商品名 </td>
         <td> 単価 </td>
      </tr>
      <!$MGREPEAT>
      <tr>
         <td><!$MG_ID></td>
         <td><!$MG_Name></td>
         <td><!$MG_Price></td>
         </tr>
         <!$MGENDREPEAT>
      </table>
   </div>
</body>
</html>
```

マージタグ

データ置換タグ

書式 ： <!$MG_ID>

ID は任意の項目名です。この部分に Magic がデータを埋め込みます。**タグ**テーブルには ID のみが表示されます。

繰り返しタグ

書式 ： <!$MGREPEAT> ～ <!$MGENDREPEAT>

<!$MGREPEAT><!$MGENDREPEAT> に囲まれた部分が繰り返されます。このタグは、テーブルを一覧形式で表示させる場合に使用します。**タグ**テーブルには表示されません。

条件タグ

書式　：<!$MGIF_Active?＞～ <!$MGELSE>～ <!$MGENDIF>

この記述で、「IF then else endif」の構文を実現します。Active? は任意の名称で、「True」かどうかの判定条件として扱われます。**タグ**テーブルには Active? が表示され、論理型の値を割り当てます。

マージプログラムを作成する

HTML ファイルにテーブルのデータをマージするプログラムを作成します。

タスク特性

- タスクタイプ……B= バッチ
- メインテーブル……商品一覧テーブル

フォームテーブル

- フォーム形式……HTML マージ形式
- HTML ファイル……作成した HTML ファイルを指定
- トークン前付符号……<!$
- トークン後付符号……>

タグテーブル

タグテーブルには、HTML ファイルに定義されたタグ（**!$MG_** の後に記述された文字列）とデータを割り当てます。

- ID……カラム「商品番号」
- Name……カラム「商品名」
- Price……カラム「単価」

入出力テーブル

- メディア……リクエスタ

タスクを定義する

データビューエディタ

1. パラメータ項目「変数：商品番号」を定義します。書式は、テーブルのカラムの 「商品番号」と同じです。
2. メインソースののカラムを全て定義します。
3. カラム「商品番号」の**範囲：最大値**特性に「変数：商品番号」を**式エディタ**で割り当てます。

レコード後

以下の内容で**フォーム出力**処理コマンドを定義します。

- 内容……マージフォーム
- ファイル……**入出力**テーブルに定義されたリクエスタ

マージプログラムを実行する

マージプログラムは、開発モードで実行確認することができません。実行モードにして、Web ブラウザより呼び出します。呼び出しには、Web ブラウザから URL を入力する方法と、他の Web ページから Form タグで指定する方法の２種類があります。

URL で呼び出す

Web ブラウザより以下のような URL を入力して呼び出します。

```
http://［サーバー名］/［スクリプトパス］/mgrqispi101.dll?APPNAME=［アプリケーション名］
&PRGNAME=［公開プログラム名］&ARGUMENTS=［パラメータ］
```

例えば以下のように指定します。

```
http://localhost/Magic101Scripts/
mgrqispi101.dll?APPNAME=MergeProg&PRGNAME=Test&ARGUMENTS=-N1013
```

これによって、プログラムが呼び出され、数値「1013」がパラメータとして渡されます。

JavaScript の利用

URL での呼び出し方法を JavaScript で関数化することができます。この場合、事前に Script タグ内で関数を定義する必要があります。

```
<HTML>
......
<script>
function mgcall(prog,arg)
{
url = 'http://localhost/Magic101Scripts/mgrqispi101.dll?APPNAME=MergeProg&PRGNAME=' +
prog + '&ARGUMENTS=' + arg ;
window.open (url,'MainFrame');
}
</script>
.......
<a href="javascript:mgcall('Test','-N1013')">MergeSample</a></font></p>
```

Form タグの送信で呼び出す

別の Web ページ（HTML ファイル）に **Submit ボタン**を配置し、ボタンが**クリック**された時に呼び出すようにすることもできます。

```
<form action="http://localhost/Magic101Scripts/mgrqispi101.dll" method="post">
<input type="submit" value=" 送信 " name="B1">
<input type="hidden" name="APPNAME"value="MergeProg">
<input type="hidden" name="PRGNAME"value="Test">
<input type="hidden" name="ARGUMENTS"value="-N1013">
</form>
```

これらの内容をマージ機能で動的に変更することも可能です。

Cookie を使用するには

Cookie は、Web ブラウザを実行している PC 上にファイルとして情報を保存するものです。Cookie を使用することで以下のような処理を実現させることができます。

- 訪問者がそのページに何回訪れたか記録して表示する。
- 訪問者の好みの表示方法を記録しておき、次回訪問時にそのモードで表示する。
- 掲示板やチャットで入力したユーザー名を記録しておき、次回訪問時にユーザー名の入力を省略する。
- ログインによるセッションを確立する。

ここでは、Web アプリケーションで Cookie に情報を書き込む方法と Cookie から情報を読み込む方法を説明します。

Cookie の書き込み

マージプログラムの場合、HTML ファイルに以下のような方法で Cookie の書き込み処理を記述します。

- HTML ファイルの先頭に書き込み処理を記述する。
- META タグを使用する。
- JavaScript を使用する。
- RQHTTPHEADER 関数を使用する。

HTML ファイルの先頭に 書き込み処理を記述する

HTML ファイルの先頭に以下の記述を追加します。

```
<!$MG_CookiesString>
<html>
…
</html>
```

この場合、マージ形式フォームのタグテーブルに以下のように設定します。

- タグ名……CookiesString
- 式……'set-cookie: NAME=VALUE; expires=Fri, 05-Mar-2010 04:00:00 GMT;'

META タグを使用する

META タグの中に書き込み処理を記述します。

```
<html>
<head>
<meta http-equiv="set-cookie" content="<!$MG_CookiesString>">
…
</html>
```

この場合、マージ形式フォームのタグテーブルに以下のように設定します。

- タグ名……CookiesString
- 式……'NAME=VALUE; expires=Fri, 05-Mar-2010 04:00:00 GMT;'

JavaScript を使用する

Scripts タグの中に JavaScript 形式で記述します。

```
<html>
…
<script type="text/javascript">
document.cookie = "<!$MG_CookiesString>";
</script>
…
<body>
```

- タグ名……CookiesString
- 式……'NAME=VALUE; expires=Fri, 05-Mar-2010 04:00:00 GMT;'

RqHTTPHeader 関数を使用する

タスク前にアクション処理コマンドで **RqHTTPHeader** 関数を以下ように実行します。

```
RqHTTPHeader ('set-cookie: NAME=VALUE; expires=Fri, 05-Mar-2010 04:00:00GMT;')
```

Cookie の読み込み

書き込まれた Cookie の情報を読み込むには、以下の２つの方法があります。

- **GetParam** 関数を使用する。
- HTTP ヘッダ情報から取り出す。

GetParam 関数を使用する

Cookie に設定された内容は **GetParam** 関数を使用することで取得することができます。例えば、以下のように Cookie が書き込まれた場合。

```
Set-Cookie: NAME=Tanaka;TIME=10:12:00; expiers=Fri, 31-Dec-2030 23:59:59;
```

GetParam(*NAME*) では、「Tanaka」が返ります。

Cookie の書き込みの際に、日付制限（expires）を指定しないとブラウザ終了時に Cookie の情報が削除されます。

HTTP ヘッダ情報から取り出す

また、HTTP ヘッダ情報を取り出すことで、まとめて取得することもできます。

Mgreq.ini に以下の記述を加えます。

```
HttpVarts = HTTP_COOKIE
```

この状態で、**GetParam(*HTTP_COOKIE*)** を実行すると「NAME=tanaka;TIME=10:12:00」が返ります。

セッション管理を行うには

セッション管理の必要性

HTTP プロトコルは Web サーバーとの接続を保持しません。このため、アプリケーションによっては、ブラウザからの呼び出しが同じユーザからなのかどうかをその都度チェックする必要があります。この処理をセッション管理といいます。

マージを使用した Web アプリケーションでは、セッション管理をアプリケーション側でサポートする必要があります。

セッション管理の方法

セッション管理は、セッション ID を使用して管理します。セッション ID を次のページに渡し、データベースと照合することでユーザを識別します。

セッション ID を生成する

Web サーバー側でセッション ID を発行する際、ランダムで容易に推測できない値にしてください。規則性を持った（例えば 0001、0002、0003 のような）セッション ID を発行すると、他のユーザによるなりすましが発生する危険性があります。完全にユニークなセッション ID を生成し、さらにセッション ID を暗号化するように検討してください。

例えば、「最終アクセス日時」と「ランダム値」、「ホストの IP アドレス」を組み合わせた文字列を暗号化するなどして算出します。

セッション ID を実装する

セッション ID を実装する方法として、以下の２つがあります。

- Cookie を使用する
- Hiden フィールドに埋め込む。

Cookie を使用する

Cookie にセッション ID を書き込むことで、同一 PC 上のセッション ID を管理することができます。

Hiden フィールドに埋め込む

入力フォームを Web サーバーに送信する際に利用する FORM タグの hidden フィールドにセッション ID を持たせることで、セッション管理を実現できます。Cookie を使用できない場合はこちらを利用します。

セッション ID を保存する

セッション ID は、データベースに保存します。データベースには以下の情報を保存します。

- セッション ID
- ログイン日時
- 最終更新日時
- ユーザー ID

その他の項目はアプリケーションの要求に応じて管理します。

コンテキスト管理を行うには

コンテキストオブジェクトを使用すると、アプリケーション情報を保管したり、アプリケーションのさまざまなコンポーネント間で情報を共有ですることができます。

たとえば、アプリケーションが複数の HTML ページなどで構成されているとします。これらのアプリケーションコンポーネント間でのやり取りを可能にするために、アプリケーションコンテキストオブジェクトを使用して、その情報を保管したり、取り出したりすることができます。

Magic では以下の情報をコンテキスト毎に保持することができます。

- メモリテーブル
- グローバル値 （**GetParam()** 関数で取得できます。）
- **IniPut()** 関数で設定した値
- メインプログラムで定義されたグローバル変数の内容

コンテキスト管理の有効化

ブラウザクライアントでは、コンテキスト管理を Magic 側の機能で行っていますが。マージアプリケーションの場合は、アプリケーション側で以下のような設定が必要になります。

タスク特性

拡張タブの**コンテキストの保持**を ***Yes***（または、「True」になる式）に設定します。これにより、バッチタスクが終了してもコンテキストは破棄されなくなります。

コンテキスト ID の取得

一番最初に自分のコンテクスト ID を取得する必要があります。このとき、**CtxGetdId()** 関数を実行することで取得できます。

プログラム起動時の URL

プログラムを実行させるための URL にコンテキスト ID（CTX=）を明示的に入れるようにします。これにより、このリクエストは指定されたコンテキストで実行されます。

（例）

```
http://localhost/Magic101Scripts/
mgrqispi101.dll?appname=Samples&prgname=Test&CTX=123456789123456789
```

または、

```
<FORM Name="Items" action="http://localhost/Magic101Scripts/mgrqispi101.dll">
<input type="Hidden" name="APPNAME" value="Samples">
<input type="Hidden" name="PRGNAME" value="Test">
<input type="Hidden" name="CTX" value="123456789123456789">
<input type=submit value=" 送信 "
</FORM>
```

コンテキスト管理プログラム

以下の図は、コンテキスト管理を行った場合のプログラムのフローを表しています。「ログインチェックプログラム」で**コンテキスト保持**を ***Yes*** にすることで以降の処理中の同じコンテキストを使用することができます。これにより、プログラム間のパラメータは、コンテキスト ID のみとなり。

グローバル値やグローバル変数、メモリテーブルに値を設定することで、コンテキスト内のデータの一貫性が保たれます。

クライアントの認証

コンテキストにクライアントの認証情報がない場合、クライアント（ユーザ）が異なっていても、リクエストに指定されたコンテキスト ID さえ合っていれば（割り込みして）処理を継続させることができます。

このようなことを防ぐためには、一連の処理が同じクライアントで行われたことを識別する情報が必要になります。そのための情報として HTML 環境変数を利用します。

HTML 環境変数

HTML 環境変数とは、Web サーバやリクエストを送信したクライアントの情報を Web サーバが設定する環境変数です。この値をもとにクライアントを識別することができます。クライアントに関する環境変数には以下のものがあります。

注： 「REMOTE_USER」は、エイリアスの［匿名アクセス］が有効になっている場合は取得できません。

Magic プログラムで HTML 環境変数の内容を取得するには、**GetParam()** 関数を使用します。

認証項目を指定する

クライアントの認証項目の指定は、インターネットリクエスタが参照する（同じディレクトリ内にある）MGREQ.INI ファイルで行います。

設定例

```
[REQUESTER_ENV]
...
HttpSigVars = REMOTE_ADDR
...
```

ISAPI 用のインターネットリクエスタの場合、MGREQ.INI の変更を有効にするには、IIS を再起動する必要があります。

確認

一方のクライアントの Web ブラウザからアプリケーションを呼び出します。

```
http://localhost/Magic101Scripts/mgrqispi101.dll?appname=Samples&prgname=Test
```

この時のコンテキスト ID を「123456789123456789」とします。別のクライアントの Web ブラウザから同じサーバの同じアプリケーションをコンテキスト ID を指定して呼び出します。

```
http://localhost/Magic101Scripts/
mgrqispi101.dll?appname=Samples&prgname=Test&CTX=123456789123456789
```

HttpSigVars を指定した場合、以下のようなエラー画面が表示されます。

第 24 章 : メッセージング

メッセージを MSMQ に送るには

MSMQ コンポーネントを使用することで、Magic プログラム内で簡単にメッセージを MSMQ に送ることができます。メッセージを送るために、以下の 2 つの基本的なオプションが用意されています。

- ３つのプログラム（**Open Queue**、**Send Message**、**Close Queue**）を使用することができます。送信するメッセージがたくさんある場合、これらを使用することで効率よく処理ができます。
- **Quick Send** を使用することで、上記の処理を 1 つのプログラムを呼び出すことで実行させることができます。

ここでは、両方の方法について説明します。

必要条件 : 利用する PC にあらかじめ MSMQ をインストールしておく必要があります。また、Magic の MSMQ コンポーネントがアプリケーションで利用できるように設定する必要があります。「PC に MSMQ を設定するには」（563 ページ）を参照してください。

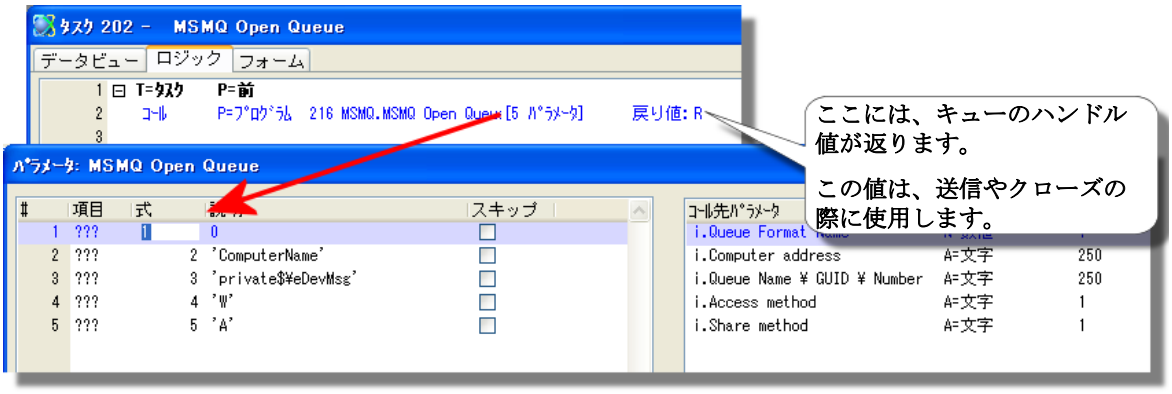
Open/Send/Close を使用する

最初の方法は、以下の 3 つの MSMQ プログラムを使用します。

- Open Queue：送信キューをオープンし、そのハンドル値を返します。返されたハンドル値を使用してメッセージを送信します。
- Send Message：キューのハンドル値を指定してメッセージを送信します。
- Close Queue：キューをクローズします。

これらの 3 つのプログラムに関する詳細情報は、Magic の『リファレンスヘルプ』を参照してください。ここでは、簡単な例を紹介します。

1. メッセージキューをオープンする



メッセージングキューを使用する前に、オープンする必要があります。キューをオープンする際に、そのキューは送信用なのか受信用なのかを指定します。同時に両方の処理が必要な場合、2 つの異なるキューを用意する必要があります。

同様に、各宛先毎にユニークなキューをオープンする必要があります。この例で示すように、1 台の特定の PC を表すアドレスや具体的な IP アドレスを指定してキューをオープンするか、パブリック、または専用（プライベート）キューを使用することができます。

最初の 3 つのパラメータはキューを指定します。

キューには様々なタイプがあり、指定される PC のアドレスとキュー名は、どんな種類のキューにアクセスするかに依存します。

例：

キューの書式名	PC のアドレス	キュー名
0: PC のホスト名	Windows に設定されているコンピュータ名	Windows の MSMQ コンポーネントに設定されたキュー名
1: IP アドレス	IP アドレス	
2: パブリック名	不要	キューの GUID
3: プライベート名	キュー番号	キュー番号

4 番目のパラメータはアクセス方式を指定します。もしそれが送信用のキューの場合、「書込」を表す ***W*** を指定します。他のプログラムとキューの書込を共有させるため、５番目のパラメータでは共有指定として ***A*** を設定します。

Open Queue プログラムがキューのオープンに成功したら、正の整数（キューのハンドル値）を返します。この値は、メッセージの送信やキューをクローズする際に使用するため、記憶しておく必要があります。

Open Queue プログラムが処理に失敗した場合、それは負の整数を返します。エラーメッセージを取得する場合は、この値を使用します。**MSMQ.PublicError** イベントを使用することでエラーメッセージを取得することができます（「メッセージングのエラーを捕捉するには」（561 ページ）を参照）。

ヒント : ***この例では、読みやすさを考慮してコンピュータ名を直接設定していますが、OSEnvGet('COMPUTERNAME') 関数を使用して実行中の PC のコンピュータ名を取得したり、データ ソースにコンピュータ名や IP アドレスを格納して使用してください。***

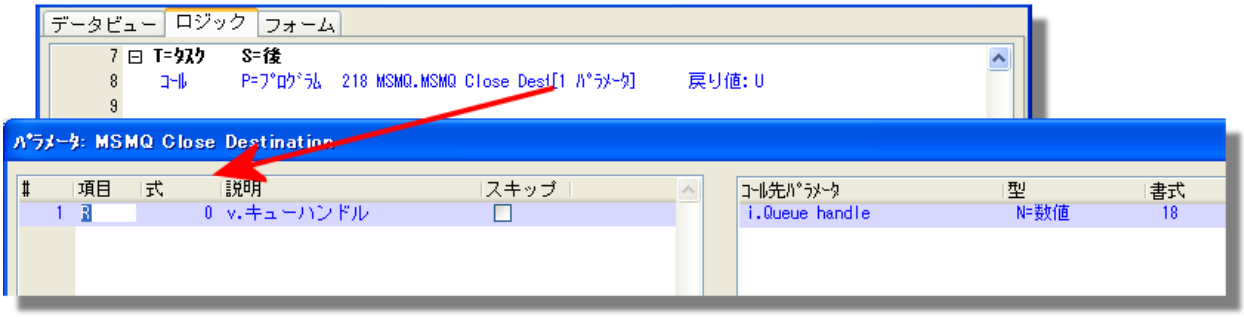
2. メッセージを送信する

簡単なメッセージを送信するために、以下のパラメータを使用します。

- ***Queue handle*** : キューをオープンした際に取得されたハンドル値
- ***Message*** : 送信したいメッセージ。BLOB 型のデータを指定します。
- ***Data type*** : 簡単なテキストメッセージとして、***A*** を指定します。
- ***Picture*** : このパラメータは、数値データの場合のみ必要です。
- ***Transaction mode*** : トランザクションを使用していないため、***N*** を指定しています。
- ***Transaction handle*** : トランザクションを使用していないため必要ありません。

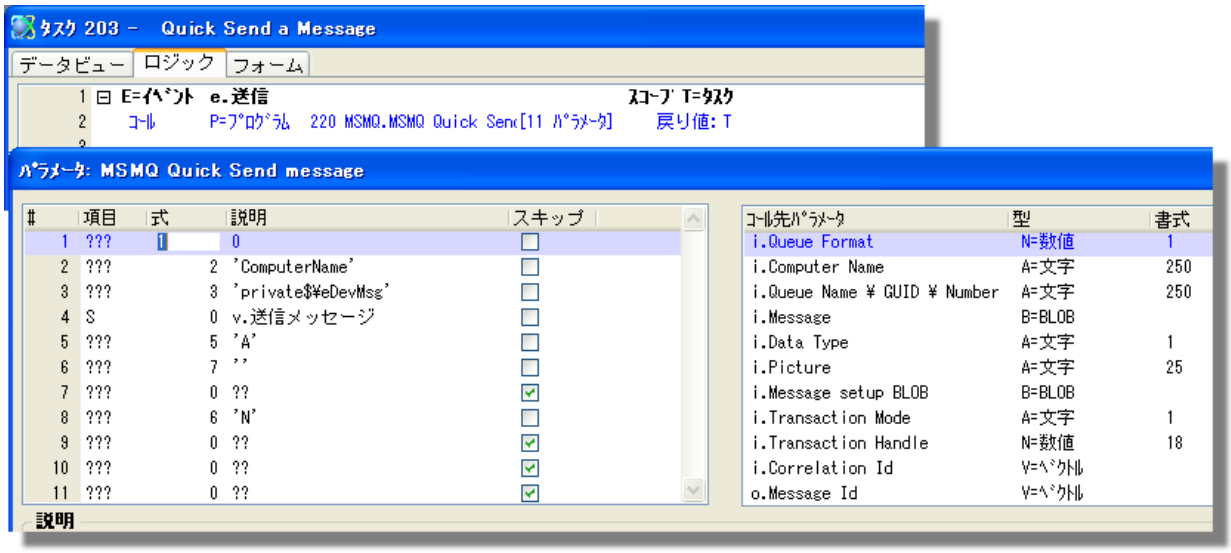
再度キューをオープンし直すことなく複数のメッセージを送信することができます。

3. キューをクローズする



メッセージの送信後にキューをクローズします。キューをオープンした際に取得したハンドル値を使用します。

Quick Send を使用する



Quick Send プログラムを使用することで、キューのオープン、送信、キューのクローズといった一連の手順を実行することなく、簡単にメッセージを送信することができます。1 つのメッセージを送信するだけであれば、こちらの方がより簡単に行うことができます。

Quick Send プログラムは、前述の 3 つのプログラムの処理を実行させるため、これらと同じぐらいの多くのパラメータを使用しています。ただし、簡単なテキストメッセージを送信する場合は、これらのパラメータの全部を指定する必要はありません。

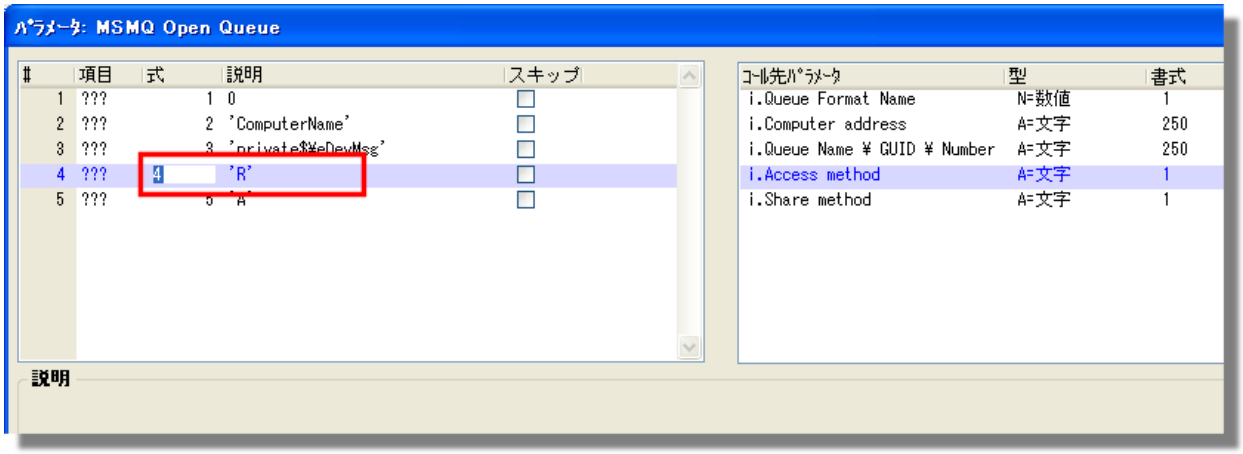
MSMQ メッセージを受信するには

MSMQ メッセージを受信するには、コンポーネントから 3 つの MSMQ プログラムを呼び出す必要があります。

- Open Queue：受信キューをオープンし、そのハンドル値を返します。返されたハンドル値を使用してメッセージを受信します。
- Read Message：キューのハンドル値を指定してメッセージを受信します。
- Close Queue：キューをクローズします。

これらの 3 つのプログラムに関する詳細情報は、Magic の『リファレンスヘルプ』を参照してください。ここでは、簡単な例を紹介します。

1. メッセージキューをオープンする

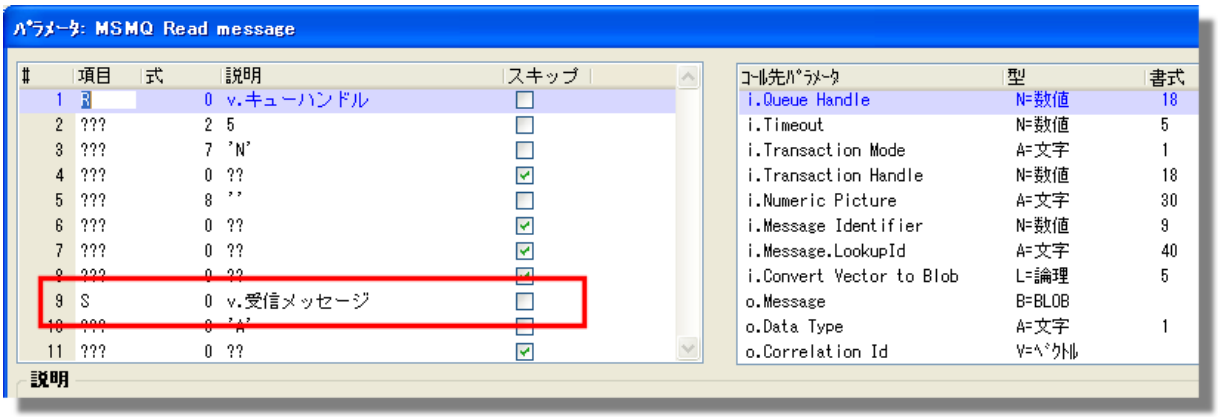


メッセージキューをオープンするには、MSMQ.Open Queue プログラム呼び出します。4 番目のパラメータで ***R*** を指定することでキューを受信用としてオープンすることができます。この指定を行うことで、読み込まれたメッセージはキューから削除されます。***V*** と指定した場合は、メッセージを読み込んでもキューからは削除されません。

これ以外のパラメータの説明は、「1. メッセージキューをオープンする」（551 ページ）を参照してください。

Open Queue プログラムがキューのオープンに成功したら、正の整数（キューのハンドル値）を返します。この値は、メッセージの受信やキューをクローズする際の最初のパラメータとして使用されます。

2. メッセージを読み込む



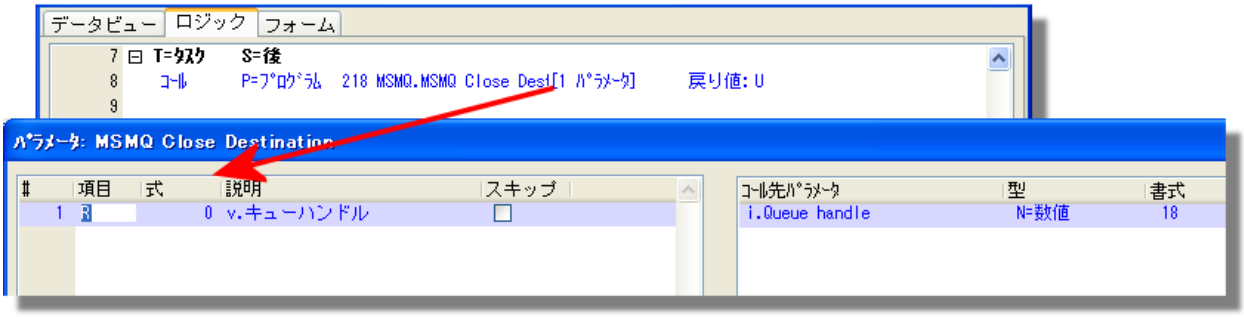
キューがオープンされると、MSMQ.Read messages プログラムを使用して、メッセージを受信することができます。プログラムが呼び出されるたびに、1 つのメッセージがメッセージキューから読み込まれ、9 番目のパラメータで指定される BLOB 項目にコピーされます。

この例のように、トランザクションを使用しない簡単なテキストメッセージの場合、パラメータのほとんどを空白（またはスキップ）にしておくことができます。

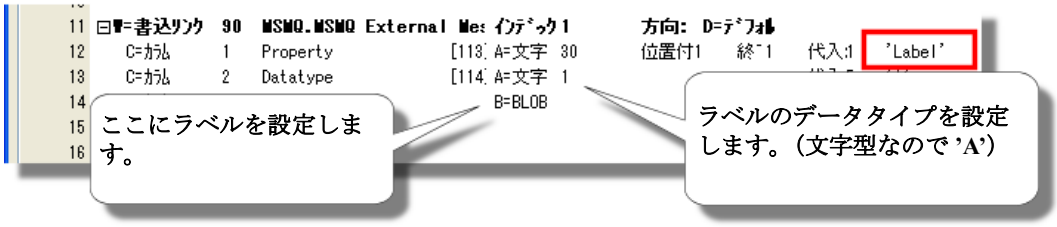
Timeout パラメータは、キューのメッセージを待つ時間を示します。メッセージがキューに送信されるまで永久に待つようにする場合、このパラメータに -1 を設定します。これで、メッセージが到着するまで、タスクの処理が停止します。これはリスナー用のタスクをどのように実現するかを指定することになります。

3. キューをクローズする

メッセージを受信したら、キューをクローズする必要があります。その際、キューをオープンした時に取得されたハンドル値を使用します。



送信メッセージにラベルを設定するには



MSMQ メッセージは、Microsoft によって定義された一連の特性を持っています。メッセージを送信する際に、Magic の MSMQ コンポーネントはこれらの特性を設定する必要があります。特性は、コンポーネント内に設定する必要があります。MSMQ.External Message Set を使用することでこれらの処理を行うことができます。

MSMQ のラベル特性を設定する

1. データビューエディタで書込リンクコマンドを定義します。
2. Property カラムで文字列「Label」を使用してリンク定義を設定します。代入特性にも同じ値を設定します。
3. Datatype カラムを、設定するラベルのデータタイプ（通常は文字型のため、A）で更新します。
4. Data カラムを、設定するラベル値で更新します。

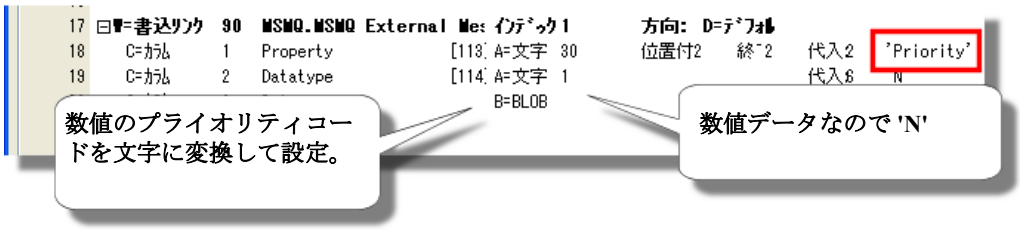
テーブルをコンポーネントに送る

次に、すべての特性を持つテーブルをコンポーネントに送る必要があります。この方法については、「外部メッセージテーブルをセットアッププログラム に送るには」（560 ページ）を参照してください。

送信メッセージにプライオリティを設定するには

MSMQ メッセージは、Microsoft によって定義された一連の特性を持っています。メッセージを送信する際に、Magic の MSMQ コンポーネントはこれらの特性を設定する必要があります。特性は、コンポーネント内に設定する必要があります。MSMQ.External Message Set を使用することでこれらの処理を行うことができます。

MSMQ のプライオリティ特性を設定する

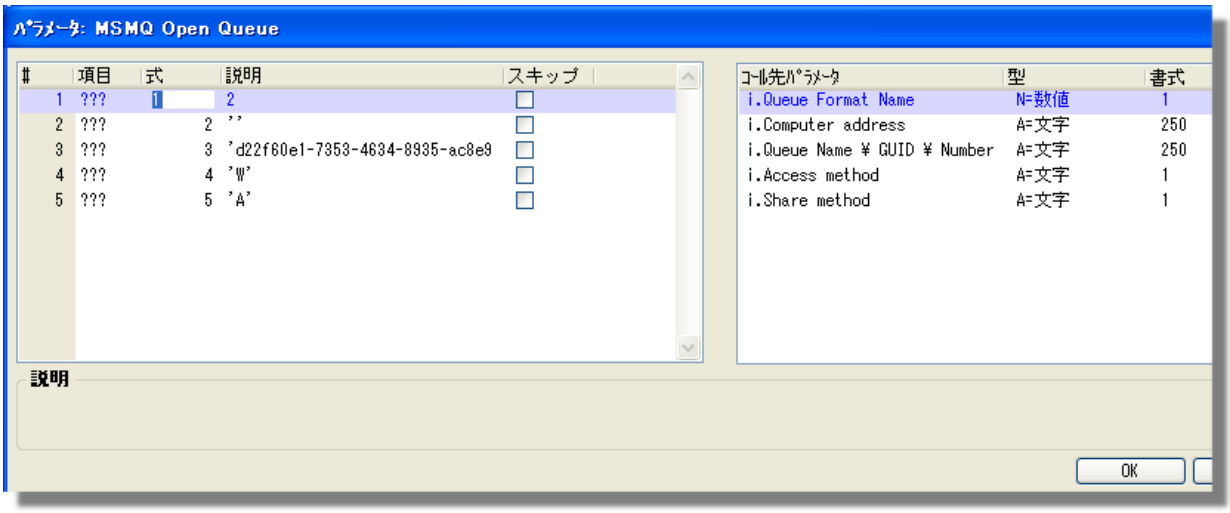


1. データビューエディタで書込リンクコマンドを定義します。
2. Property カラムで文字列「Priority」を使用してリンク定義を設定します。代入特性にも同じ値を設定します。
3. Datatype カラムを、設定するラベルのデータタイプ「N」で更新します。
4. Data カラムを、（文字列書式の）プライオリティコードで更新します。必要であれば、**Str()** 関数を使用してコードを文字列に変換することができます。

テーブルをコンポーネントに送る

次に、すべての特性を持つテーブルをコンポーネントに送る必要があります。この方法については、「外部メッセージテーブルをセットアッププログラム に送るには」（560 ページ）を参照してください。

MSMQ のパブリックキューにアクセスするには



MSMQ のパブリックキューにアクセスしたい場合、以下を使用してキューをオープンする必要があります。

- *Queue Format Name*：*2*（パブリック）
- *Computer Address*：*''*（空白）
- *Queue Name*：キューの GUID

指定された GUID を持つパブリックキューがネットワーク上に存在する場合、送信されたメッセージはそのキューに書き込まれます。

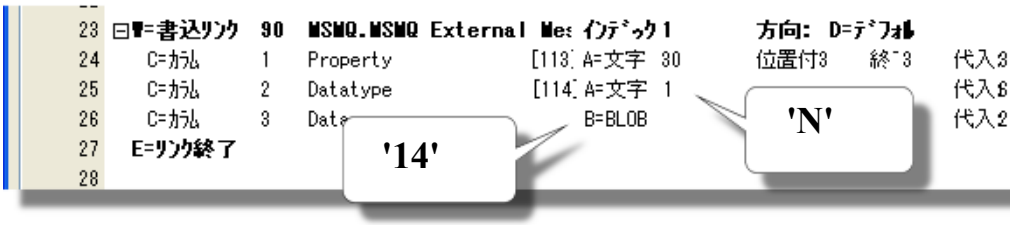
注： パブリックキューにアクセスするには、MSMQ をディレクトリモードでインストールする必要があります。

メッセージが読み込まれたことを確認するには

MSMQ オブジェクトには、メッセージがキューに到達したりするなどの状態を確認することが可能なシステムが含まれています。

MSMQ メッセージは、Microsoft によって定義された一連の特性を持っています。メッセージを送信する際に、Magic の MSMQ コンポーネントはこれらの特性を設定する必要があります。特性は、コンポーネント内に設定する必要があります。MSMQ.External Message Set を使用することでこれらの処理を行うことができます。

MSMQ の Ack 特性を設定する



1. データビューエディタで書込リンクコマンドを定義します。
2. Property カラムで文字列「Ack」を使用してリンク定義を設定します。代入特性にも同じ値を設定します。
3. Datatype カラムを、設定するラベルのデータタイプ「N」で更新します。
4. Data カラムを、（文字列書式の）Ack コードで更新します。必要であれば、**Str()** 関数を使用してコードを文字列に変換することができます。

MSMQ の Ack コードには、様々なものがあります。詳細は、『リファレンスヘルプ』を参照してください。この例では、14 の Ack コードを指定しています。これはキュー内のメッセージの受信状態（時間内に受信できた場合は肯定応答、それ以外は否定応答）にもとづいて Ack を送信する指定です。

MSMQ の AdminPath 特性を設定する

Ack メッセージは、MSMQ の管理キューに送られます。管理キューは、MSMQ の AdminPath 特性で設定します。設定は、同じように MSMQ.External Message Set を使用して行います。

1. データビューエディタで書込リンクコマンドを定義します。
2. Property カラムで文字列「Adminpath」を使用してリンク定義を設定します。代入特性にも同じ値を設定します。
3. Datatype カラムを、設定するラベルのデータタイプ「A」で更新します。
4. Data カラムを、管理キューのパスを表す文字列で更新します。パスは以下のとおりです。

```
<ServerName>¥<QueueType>$¥<QueueName>
例：serverPC¥Private$¥Magic
```

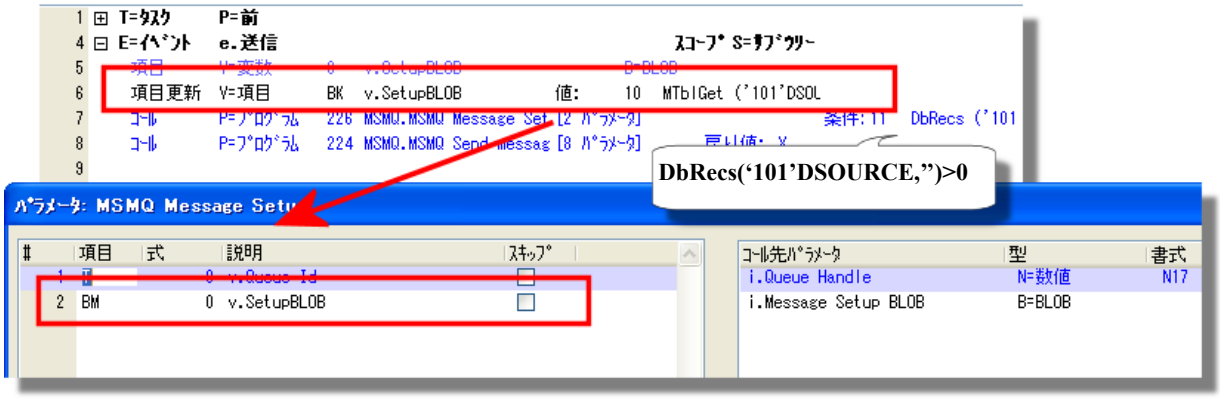
注： 管理キューは、トランザクションキューでなければどのキューも指定可能です。

テーブルをコンポーネントに送る

次に、すべての特性を持つテーブルをコンポーネントに送る必要があります。この方法については、「外部メッセージテーブルをセットアッププログラム に送るには」（560 ページ）を参照してください。

これで、キュー内のメッセージが受信されると、指定された Ack のタイプにもとづいて MSMQ は応答を返します。アプリケーションは、管理キューを読み込み、必要に応じて応答を返す必要があります。

外部メッセージテーブルをセットアッププログラム に送るには



MSMQ.External Message Set テーブルに MSMQ の各特性値を設定したら、この内容を有効にするには、このテーブルをコンポーネントに送る必要があります。**MSMQ.External Message Set** テーブルをコンポーネントに送るには、BLOB 項目内にテーブルをパッケージ化し、コンポーネントに送ります。これを行うには、以下の手順を実行します。

1. **MTblGet()** 関数を使用して、BLOB 項目内にテーブルをパッケージ化します。
2. MSMQ.Message Setup プログラムを呼び出します。２番目のパラメータには送信する BLOB 項目を設定します。
3. 常にテーブルを設定するわけではない場合は、セットアップテーブル内にレコードがあるかどうかにもとづいて MSMQ.Message Setup プログラムの呼び出しを調節してください。

これで、メッセージが送信される、**ラベル**特性が設定されます。

メッセージングのエラーを捕捉するには

MSMQ プログラムを使用する場合、エラーが発生するす可能性を考慮する必要があります。エラーが発生すると、各プログラムは、エラー内容を確認するためのエラー番号を戻り値として返します。また、自動的に MSMQ エラーを捕捉するためにイベントロジックユニットを定義することもできます。このロジックユニットを追加するには、以下の手順を実行します。

1. MSMQ エラーを処理したいタスクを開きます。定義する場所は、以下の３つのうちどれかになります。
- メインプログラム。スコープ特性は、*G= グローバル*に設定します。
- タスクツリー上の上位タスク。スコープ特性は、*S= サブツリー*に設定します。
- MSMQ プログラムを呼び出すタスク。スコープ特性は、*T= タスク*に設定します。

2. ロジックタブをクリックします。
3. *Ctrl+H*（編集→ヘッダ行作成）を押下して、新しいヘッダ行を作成します。
4. *E* を入力して*イベント*を選択します。イベントダイアログが表示されます。
5. イベントタイプで *U= ユーザ*を選択します。
6. イベントからズームして *MSMQ.Public error* を選択します。
7. OK をクリックします。「このイベントに対応するパラメータを作成しますか？」という確認のメッセージボックスが表示されます。Yes をクリックします。
8. イベントロジックユニットにパラメータ項目が作成されます。
9. #1 の内容に応じてスコープ特性を設定します。
10. 作成されたパラメータ項目は、必要に応じてエラーの処理で使用します。この例では、ユーザ用のエラーメッセージを表示させるために、メッセージテキスト（pO.Error message）を使用しています。

このイベントには、以下の４つのパラメータが作成されます。

パラメータ名	内容
pO.Messaging system	• M …… MSMQ • J …… JMS • W …… WebSphere MQ
pO.Error code	エラーコード
pO.Validation Error?	• True ……エラーが MSMQ コンポーネントの確認機能で発生したも • False ……実行時に発生したエラーの場合
pO.Error message	エラーメッセージ

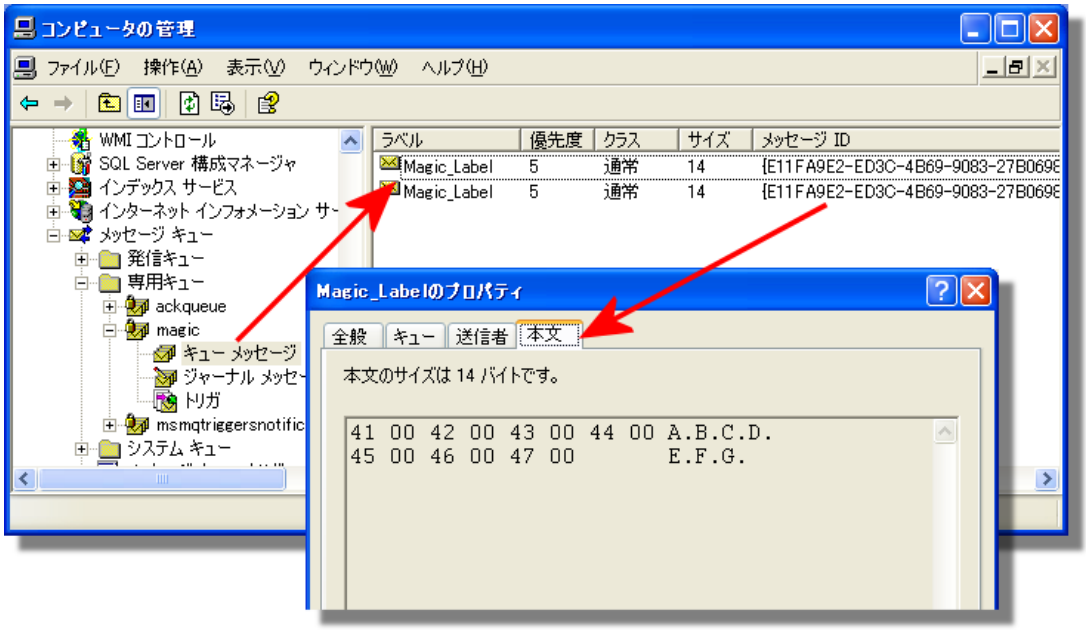
注： ここですべてのエラーメッセージを捕捉した場合、それらはメッセージエラーログにも書き込まれます。このログは、論理名の MessagingComponentDir で指定された場所に作成されます。

MSMQ アプリケーションをデバッグするには

メッセージングは複数のアプリケーションとサーバが関連するため、MSMQ をデバッグすることは Magic のアプリケーションのみの場合より複雑になります。MSMQ のデバッグには以下のようなツールを使用することができます。

- デバッガ（第 29 章 :「デバッガを使用してデバッグするには」（607 ページ）を参照してください）は、Magic の内部処理を確認する場合に非常に有効です。
- MSMQ のエラーは、ログファイルに書き込まれます。ログファイルの位置とファイル名は、論理名 MessagingErrorLogFile で設定されます。デフォルトは、%MessagingComponentDir%MG_message_err.log です。
- Windows 上で MSMQ のメッセージを参照することができます。以下で説明します。

Windows 上でメッセージを確認する



1. コンピュータの管理（スタート→設定→コントロールパネル→管理ツール→コンピュータの管理）ウィンドウを開きます。
2. サービスとアプリケーションのツリーノードを開きます。
3. メッセージングキューを開きます。これで登録されているメッセージキューが表示されます。
4. メッセージを送信したキューに移動します。キューメッセージセクションで、キューから読み込み処理が実行されていないメッセージを参照することができます。
5. 参照したいメッセージにカーソルを置き、右クリックでコンテキストメニューを開き、プロパティを選択します。上図のようにメッセージの内容やの他の特性を参照することができます。

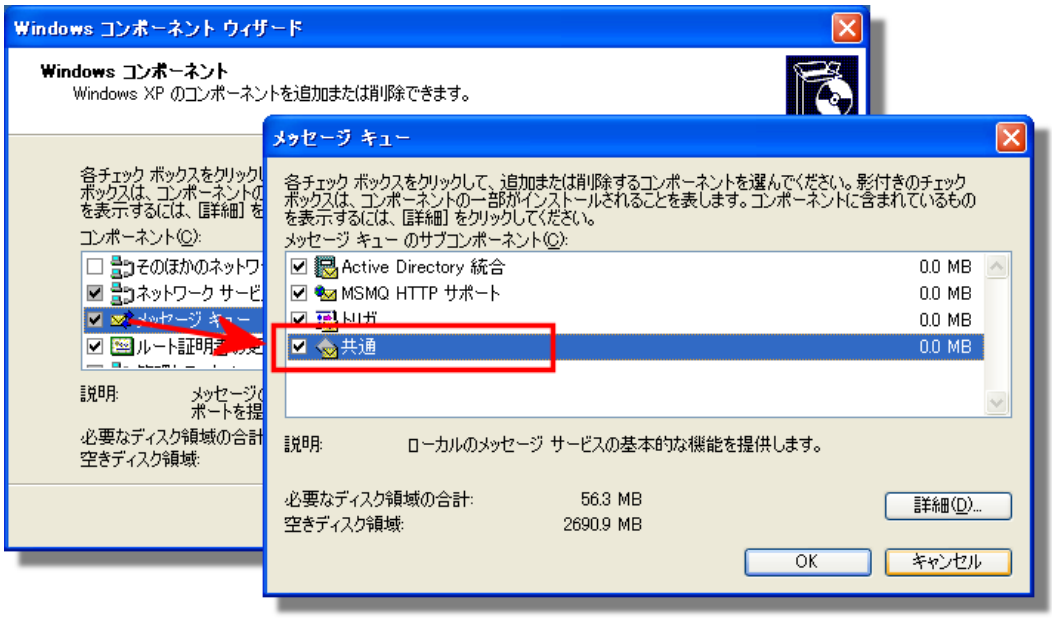
注： コンピュータの管理ウィンドウが開いている状態で操作→最新の情報に更新を選択すると、新しいメッセージを表示させることができます。

PC に MSMQ を設定するには

MSMQ は、Windows に標準で装備されているメッセージングシステムです。ただし、デフォルトではインストールされません。Windows ユーザが MSMQ コンポーネントを選択する必要があります。どのようにインストールするかを説明します。

注： コンポーネントの名前やダイアログのイメージは、OS によって異なる場合があります。

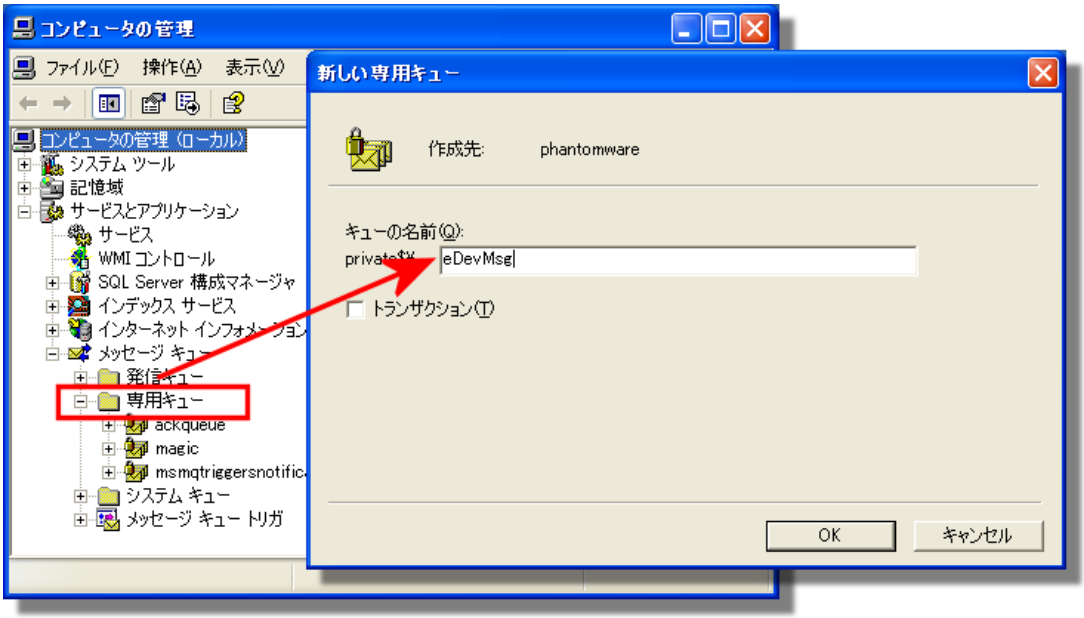
MSMQ をインストールする



MSMQ を利用する前に、あなたの PC にインストールする必要があります。以下の手順で実行します。

1. Windows のメニューから**スタート→コントロールパネル**を選択します。
2. **プログラムの追加と削除**を**クリック**します。
3. **Windows コンポーネントの追加と削除**のアイコンを**クリック**します。
4. **メッセージキュー**を選択します。チェックボックスをチェックします。
5. 詳細ボタンを**クリック**します。
6. **共通**を選択します。
7. OK を**クリック**して選択内容を確定します。

キューを追加する



1. コンピュータの管理（コントロールパネル→管理ツール→コンピュータの管理）ウィンドウを開きます。
2. サービスとアプリケーション→メッセージキューを開きます。キューをパブリック、または専用キューとして追加することができます。パブリックキューはドメインに参加している PC でのみ利用できます。
3. キューを追加するには、コンテキストメニューから新規→専用キュー（必要であればパブリックキュー）を選択します。
4. 新しい専用キューダイアログが表示されます。キューに名前を定義します。この例では、$¥eDevMsg になっています。この名前は、Magic でキューをオープンする際に使用されます。
5. 必要であれば、トランザクションのチェックボックスをチェックします。MSMQ のバージョンによっては、トランザクションでないメッセージをトランザクションキューに送信することを許可していないものもあります。
6. OK をクリックすると、キューが作成されます。

メッセージングコンポーネントをインストールする

Magic のインストールの際に、カスタムインストールでメッセージングコンポーネントを選択するとインストールされます。インストールされていない場合は、インストール処理を実行し直し、追加 / 削除オプションで選択します。

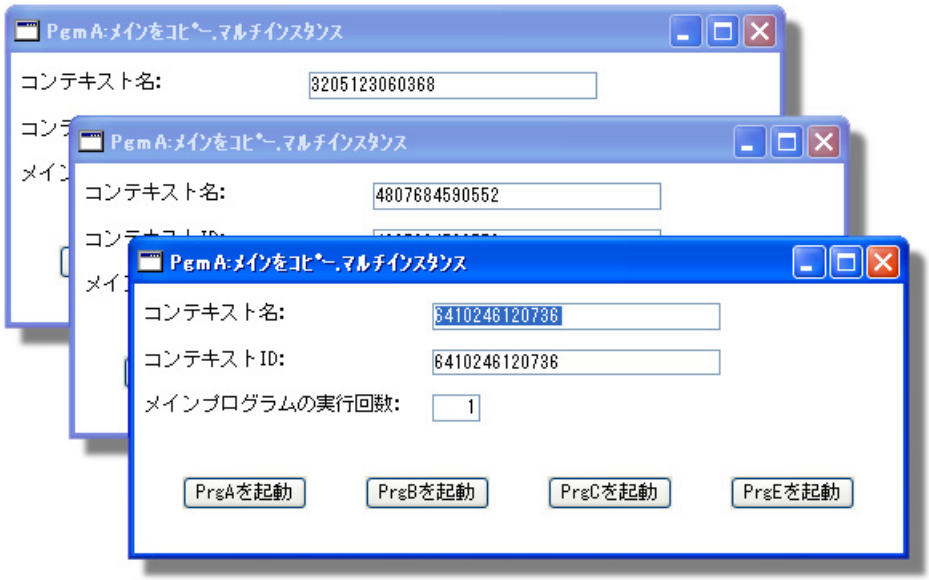
メッセージングコンポーネントを利用する上で、２つの論理名を使用されます。

- MessagingComponentDir ……これはメッセージングコンポーネントが格納されているディレクトリを指定します。コンポーネントは ***.ecf*** ファイルで、メッセージングを利用する上で使用されるプログラムやハンドラが含まれています。***.eci*** ファイルも含まれており、このコンポーネントをアプリケーションに追加することができます。.
- MessagingErrorLogFile ……これはエラーログファイルの位置とファイル名を指定します。

コンポーネントの設定については、第 16 章：「プロジェクトにコンポーネントを読み込むには」（362 ページ）を参照してください。

第 25 章 : マルチタスク

複数の対話型タスクを同時に実行させるには



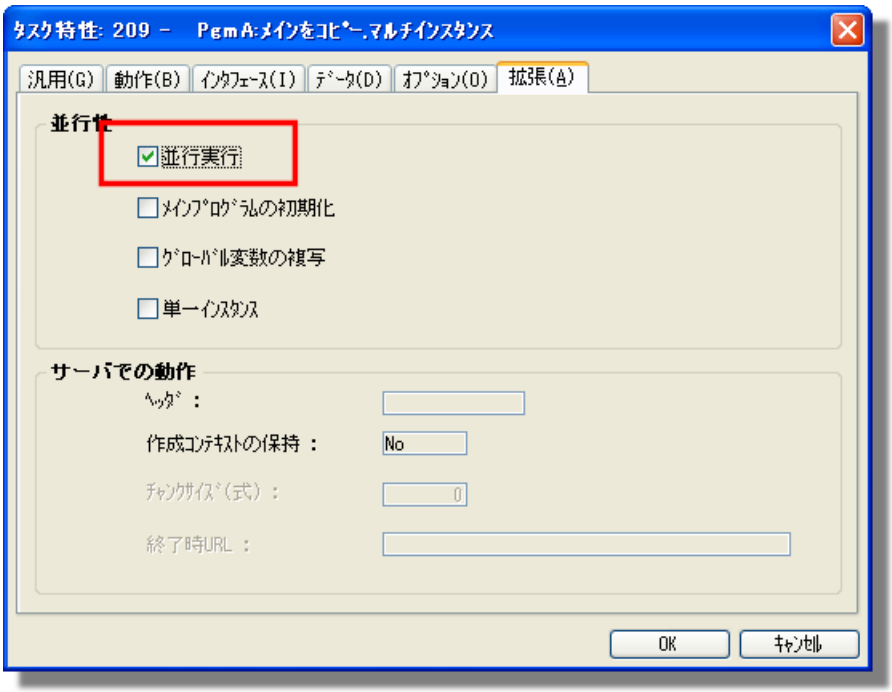
Magic プログラムは、デフォルトでは 1 度に 1 つのランタイムツリーを実行させることができます。つまり、メニューからプログラムを開くと、実行中の他のランタイムツリーは自動的に終了します。プログラムは他のタスクを呼び出しますが、このタスクは階層的なツリー構造内で動作します。グローバルイベント使用することで、階層上に存在しないタスクを起動させることができます。これらのタスクはメインタスクと一緒に同時に実行させることができますが、同じプログラムをマルチウィンドウでオープンさせたままにすることはできません。

しかし、プログラムを並行モードで動作させることができます。プログラムが上の図のように並行に実行している場合、各プログラムは独自の階層ツリーで実行させたり、より多くの並列プログラムを呼び出すことができます。

各並列プログラムは、ユニークな**コンテキスト ID** を持っています。**コンテキスト ID** は、自動的に割り当てられる文字列で表された 15 桁の番号です。必要であれば変更することができます（「現在のコンテキストに異なる名前を設定するには」（572 ページ）を参照してください）。

この例では、Pgm A が 3 回呼び出されています。メニューや別のプログラムから呼び出されたかどうかにかかわらず、プログラムはユニークな**コンテキスト ID** を持っており、他のウィンドウと無関係に動作します。

タスクを並行に実行させる



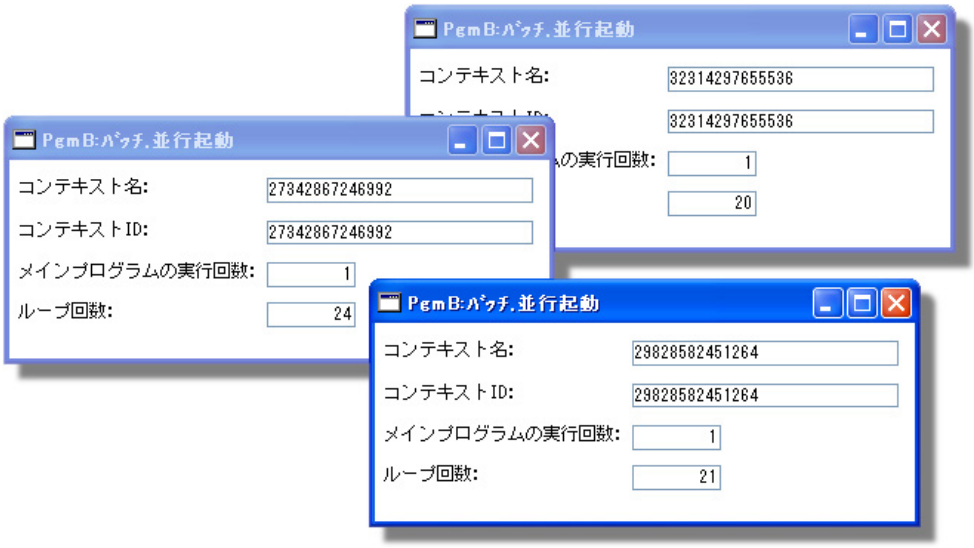
タスクを並行に実行させるには以下の手順を実行します。

1. **タスク特性**（*Ctrl+P*）を開きます。
2. **拡張**タブを**クリック**します。
3. **並行実行**のボックスをチェックします。

これで、プログラムは並行に実行されます。

プログラムが並行に実行するように設定された場合、どのように実行するかを定義するためのオプションを指定することもできます。詳細は、「並行タスクの初期設定をコントロールするには」（577 ページ）を参照してください。

並行処理のバッチプログラムを実行させるには



バッチタスクは、対話型タスクと同じ方法で実行させることができます。並行に実行する各バッチタスクは、独自のコンテキストを取得し、タスクが終了するとクローズします。

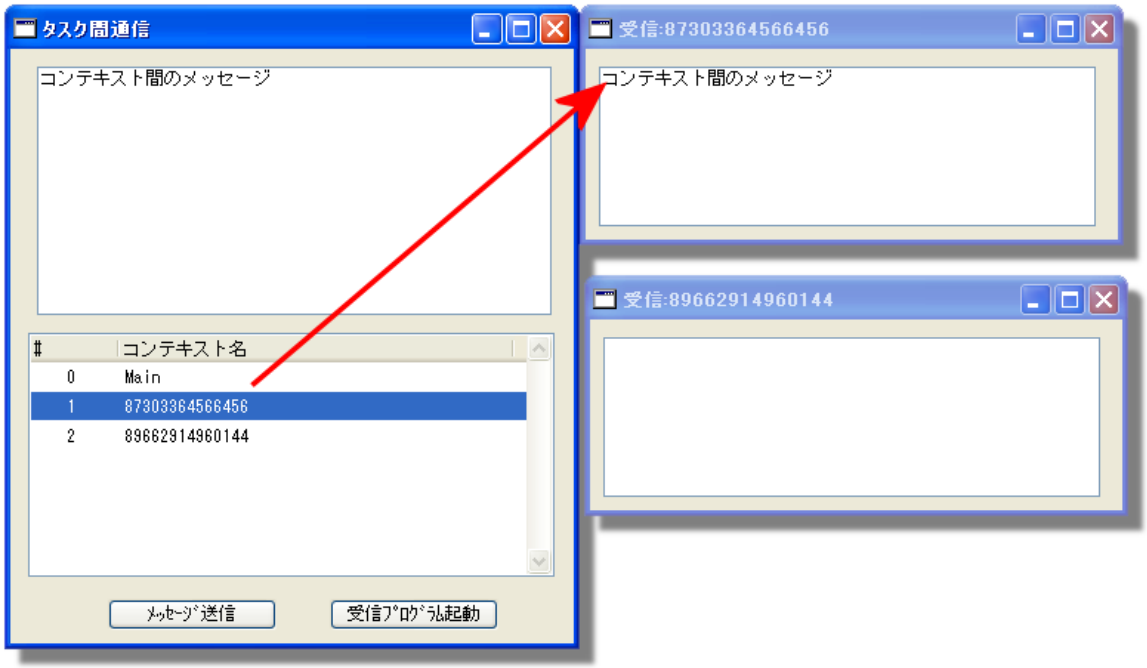
バッチタスクを並行に実行させる

バッチタスクを並行に実行させるには、以下の手順を実行します。

1. **タスク特性**（*Ctrl+P*）を開きます。
2. **拡張**タブを**クリック**します。
3. **並行実行**のボックスをチェックします。

これで、プログラムは並行に実行されます。

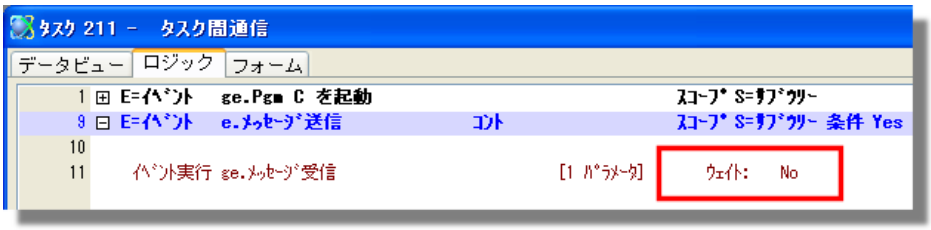
並行実行しているタスク間の通信を管理するには



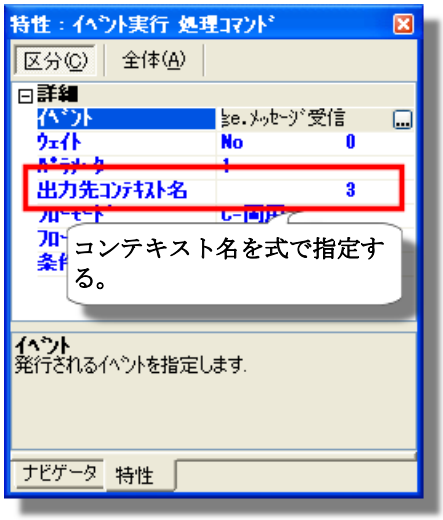
並行タスクは、個別のメモリ空間で実行しているため、それらの間で直接パラメータを渡したり、メインプログラムに定義されたグローバル変数を共有することができません。従って、並行タスクを「呼び出す」には、イベントをタスクに送る必要があります。イベントは、並列タスクにデータを渡すためのパラメータを定義することができます。ここでは、その方法について説明します。

必要条件： 通信するプログラムのコンテキスト ID を知る必要があります。コンテキスト ID を取得する方法は、を参照してください。

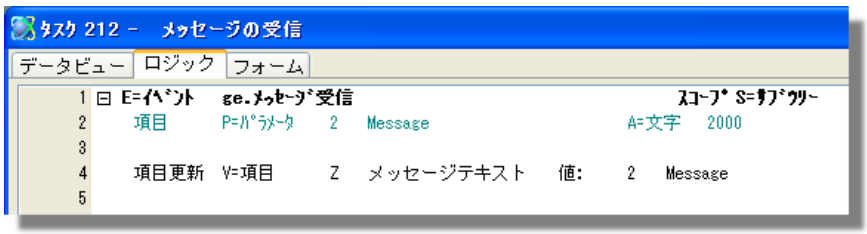
1. プログラム間で渡す必要があるパラメータを持つグローバルイベントを定義します。この例では、**ge. メッセージ受信**と呼ばれるイベントが定義されています。このイベントには、1 つのパラメータ（渡したいメッセージ）が定義されています。**ウェイト**特性は ***No*** に設定する必要があります。
2. 最初のプログラムでは、ここに示されているように、パラメータ渡してイベントを実行します。**ウェイト**特性は ***No*** に設定しなければなりません。***No*** に設定しないとエラーが発生します。



3. イベント特性で、出力先コンテキスト名特性に、*通信するコンテキスト名*を設定します。ほとんどの場合、1493034580378080 というような Magic で生成された文字型の数値となります。しかし、Main と呼ばれるメインのコンテキストもあるため、必要であればコンテキストに名前を設定することもできます。



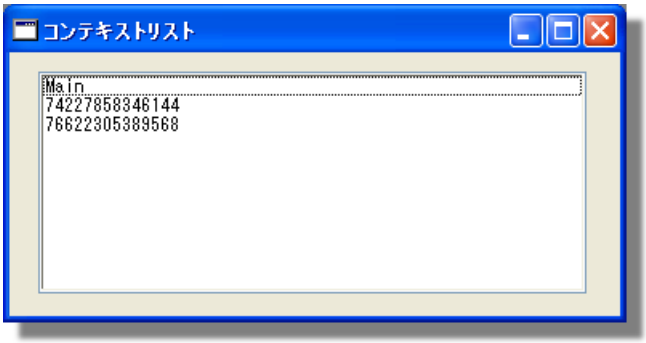
これでイベントが実行されると、指定されたコンテキストで実行されることになります。この例では、メッセージを送る間、ge. メッセージ受信イベントが実行されています。



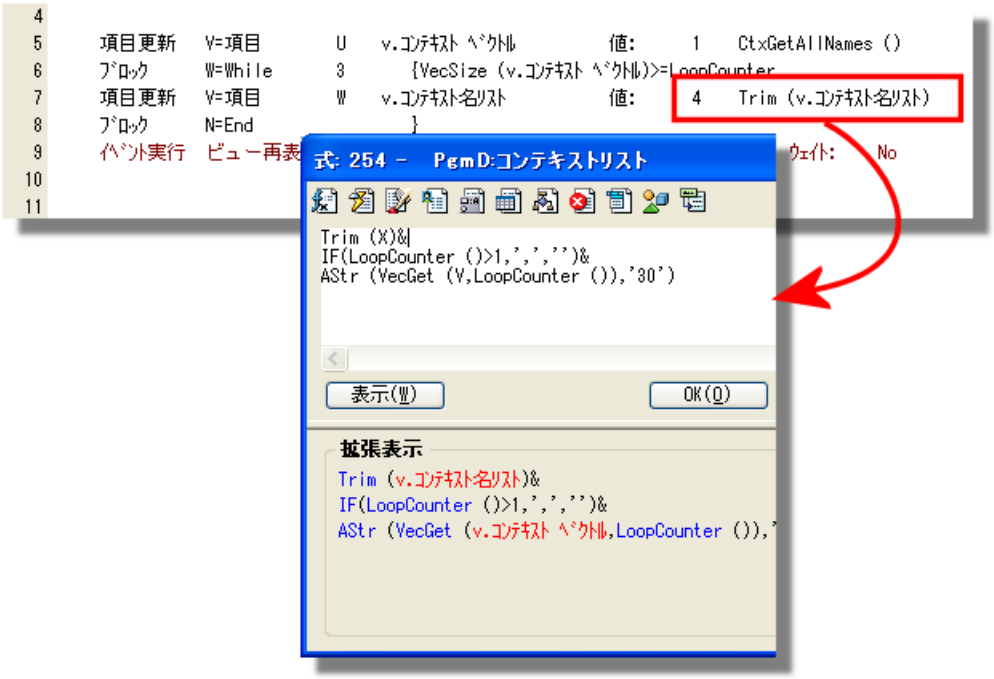
他のコンテキストでは、発生したイベントを受け取り適切に処理を実行します。イベントを受け取ったタスクは、イベントの発生元を認識することはありませんし、その必要もありません。メッセージが到着し表示されます。

しかし、このタスクが送信元のプログラムにデータを送り返す必要がある場合、送信元プログラムのコンテキスト名を知る必要があります。このような場合は、単にパラメータを追加してコンテキスト名を受け渡しするようにすることで対応できます。

実行中のコンテキストのリストを取得するには



デバッガを有効にしている場合、デバッガの実行コンテキストペインに既存のコンテキストがすべて表示されます。しかし、実行環境で実行中のコンテキストのリストを表示させたい場合は、**CtxGetAllNames()** 関数を利用することで実現できます。この関数は、コンテキスト名が格納されたベクトルデータを返します。Magic のベクトル関数を使用することでこのデータを処理することができます。



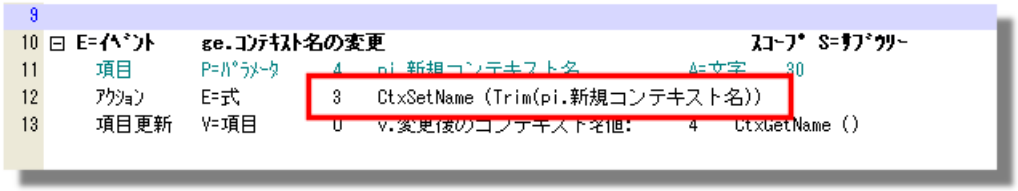
画面にコンテキストを表示させたい場合、ここに示したように、取得したデータをカンマ区切りで結合させることでリスト表示させることができます。

現在のコンテキスト名を取得するには

コンテキスト間で通信する場合、送信元のプログラムを相手先が認識できるようにするため、自身のコンテキスト名を知る必要があります。

コンテキスト名を取得するには **CtxGetName()** 関数を使用します。パラメータはありません。この関数は文字列（最長 128 文字）を返します。Magic によって割り当てられたコンテキスト名は 16 文字の文字列ですが、プログラムによって最高 128 文字までの長さのコンテキスト名を指定することができます。

現在のコンテキストに異なる名前を設定するには



コンテキストに名前を設定する必要がある場合は、**CtxSetName()** 関数を使用します。名前は静的で認識しやすいため、コンテキストに名前を設定することは、単一インスタンスのコンテキストを扱う場合に便利です。

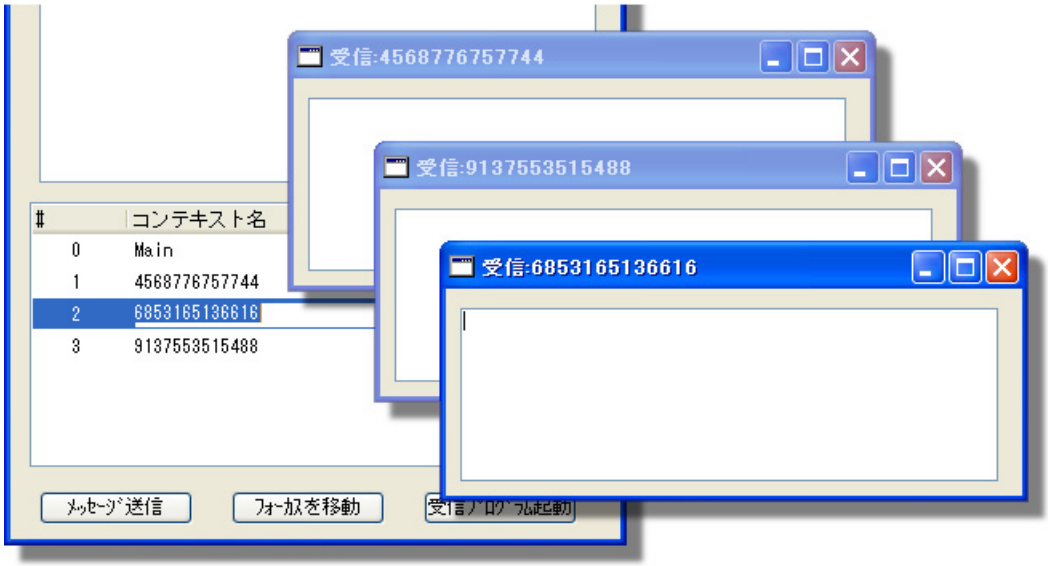
関数の構文は以下の通りです。

CtxSetName(*new name*)

パラメータ：

- ***new name***：指定するコンテキスト名です。最高 128 文字の文字列が指定できます。

任意のコンテキストにフォーカスを移すには



ユーザが対話型のコンテキストにアクセスしている場合は、任意のコンテキストをクリックすることができます。しかし、**SetContextFocus()** 関数を使用することでプログラムによって任意のコンテキストにフォーカスを移すことができます。

```
12
13 ⊟ E=ｲﾍﾞﾝﾄ     e.ﾌｫｰｶｽを設定                                    ｽｺｰﾌﾟ S=ｻﾌﾞﾙｰﾁﾝ
14     ｱｸｼｮﾝ      E=式          4    SetContextFocus (Trim(コンテキスト名))
15
16
```

関数の構文は以下の通りです。

SetContextFocus(*name*)

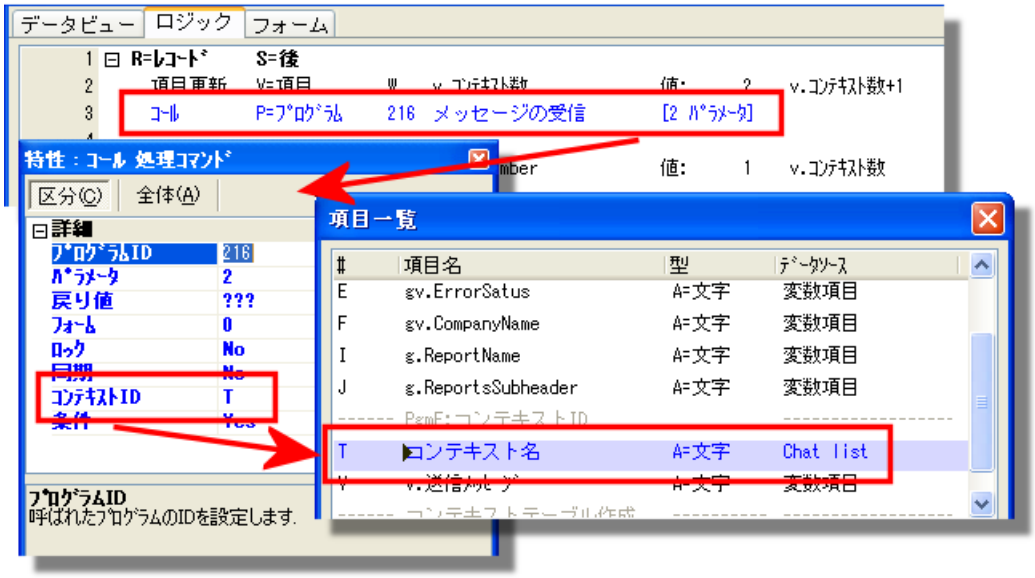
パラメータ：

- *name*：コンテキスト名を表す文字列です。**Trim()** 関数を使用して余分な空白を削除するようにする必要があります。

関数が実行され、指定されたコンテキストが存在していれば、そのコンテキストにフォーカスが移ります。

コンテキストが存在しており、フォーカスがそのコンテキストに切り替わった場合、関数は True は返します。それ以外は、False が返ります。

呼び出された並列プログラムのコンテキスト ID を取得するには



並行プログラムを呼び出す場合、コール処理コマンドのコンテキスト ID 特性に項目を指定することで簡単に呼び出されたプログラムのコンテキスト ID を保存することができます。設定するには以下の手順で実行します。

1. 並行プログラムを呼び出す、コールプログラム処理コマンドの定義行に移動します。
2. *Alt+Enter* を押下してコール特性を開きます。
3. コンテキスト ID 特性に移動します。
4. ここからズームして、返されるコンテキスト ID を保持するための文字型項目を選択します。

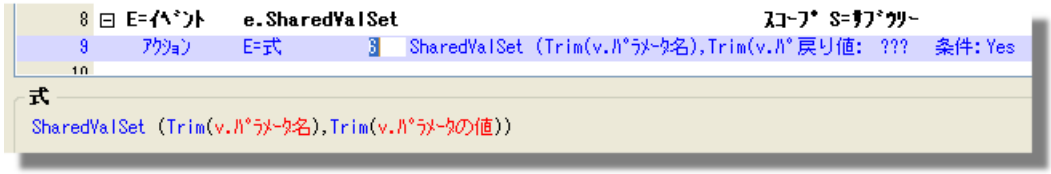
これで、プログラムが呼び出されると、項目にコンテキスト ID の値が格納されます。

注： 呼び出されたプログラムには、呼び出し元プログラムのコンテキスト ID を認識するための機能は持っていません。従って、必要であればパラメータとして自分のコンテキスト ID を渡してください。

コンテキスト間で項目を共有するには

単一コンテキスト環境では、通常メインプログラムや **SetParam()** 関数を使用することでプログラム間のデータ共有を実現することができます。しかし、これらはどちらもコンテキスト間では利用できません。上の例では、各ウィンドウが **SetParam()** と **GetParam()** を使用して、自身のコンテキスト内に値を格納することができます。しかし、複数のコンテキスト間で値を共有させるには、**SharedValSet()** と **SharedValGet()** 関数を使用する必要があります。

SharedValSet() 関数を使用する



関数の構文は以下の通りです。

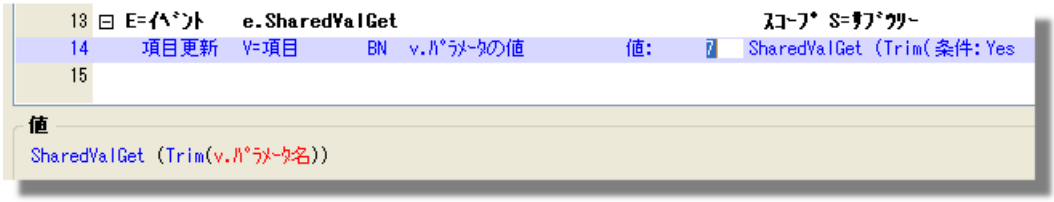
SharedValSet(*name,value*)

パラメータ：

- ***name***：項目名です。上記の例のように項目に格納して使用したり、直接項目名を指定することができます。
- ***value***：項目に格納される値です。データの型は任意ですが、データを取得する際に同じ型になるように開発者の責任で定義する必要があります。例えば、日付データを格納した場合、別のプログラムでは日付型データとして取り出す必要があります。

この関数は、常に True を返します。

SharedValGet() 関数を使用する



関数の構文は以下の通りです。

SharedValGet(*name*)

パラメータ：

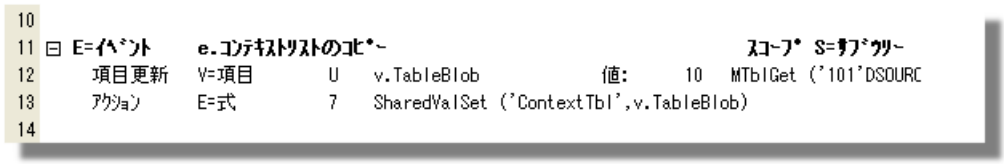
- ***name***：項目名です。上記の例のように項目に格納して使用したり、直接項目名を指定することができます。

この関数は、Magic のメモリに格納されている項目の値が返ります。

ヒント : *共有された値は、デバッガの変数リストの最上位に表示されます。*

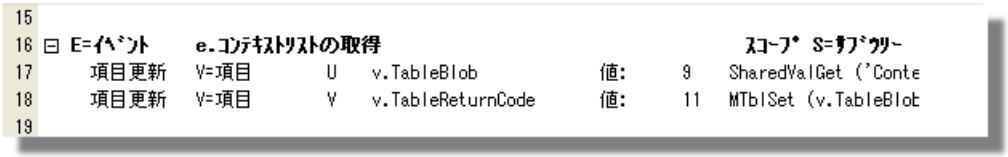
コンテキスト間でメモリテーブルを共有するには

コンテキスト間で直接メモリテーブルを共有することはできません。しかし、BLOB 項目にメモリテーブルを格納し、**SharedVal** 関数を使用して BLOB 項目を格納することで共有することが可能になります。



共用メモリにメモリテーブルを格納する例を説明します。

1. 最初に、**MTblGet(*'101'DSOURCE、''*)** を使用して、データソース **#101**（メモリテーブル）を BLOB 項目（b.TableBlob）に格納します。
2. 次に、**SharedValSet(*'ContextTbl'、b.TableBlob*)** を使用して BLOB 項目を共用メモリ（ContextTbl）に格納します。



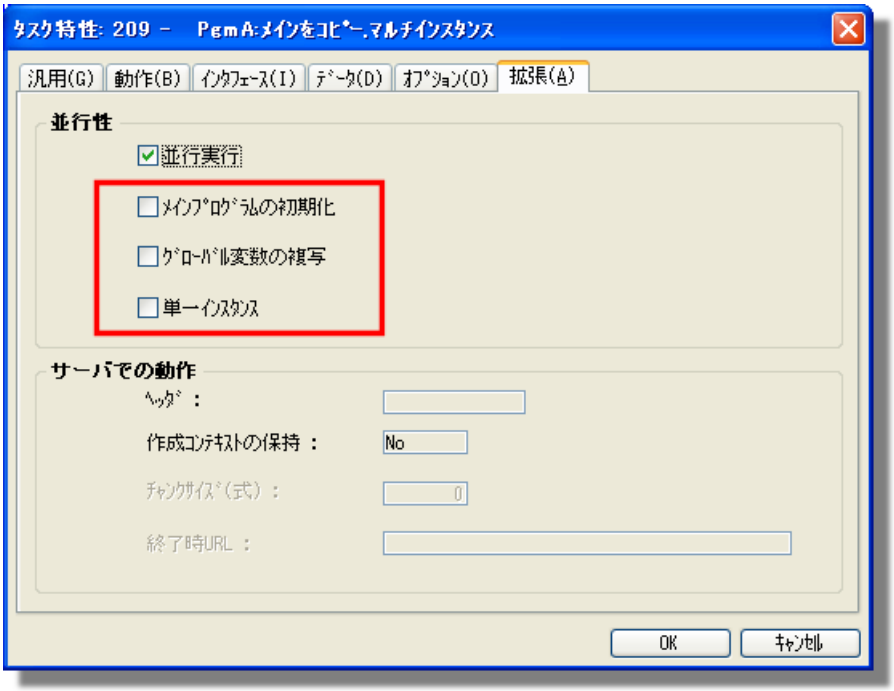
逆の処理は以下のように実行します。

1. 最初に、**SharedValGet(*'ContextTbl'*)** を使用して、共用メモリから BLOB 項目を取得します。
2. 次に、**MTblSet(*b.TableBlob,'101'DSOURCE、'',1*)** を使用して BLOB 項目をデータソース **#101**（メモリテーブル）に展開します。

MTbl() 関数の詳細は、『リファレンスヘルプ』を参照してください。

SharedVal() 関数の使用方法については、「コンテキスト間でメモリテーブルを共有するには」（576 ページ）を参照してください。

並行タスクの初期設定をコントロールするには



タスク特性で並行実行特性をチェックすると、他の３つのオプションが利用可能になります。３つのオプションのうち２つは並行プログラムの初期設定をコントロールするために使用できます。

メインプログラムの初期化

この特性がチェックされた場合、並行プログラムが実行される前に、メインプログラムが再度実行されます。メインプログラムのタスク前が実行され、変数項目の値は新しいコンテキストで初期化されます。

この特性がチェックされない場合、メインプログラム内の項目の値はオリジナルのコンテキストでの内容がコピーされます。その後、新しいコンテキスト内で発生した変更内容はそのコンテキストでのみ有効であり、オリジナルのコンテキストのメインプログラムの内容には影響しません。

グローバル変数の複写

この特性がチェックされた場合、既存のグローバルパラメータのコピーが作成されます。新しいコンテキストはグローバルパラメータのスナップショットを取得します。この場合、グローバルパラメータが更新された場合、新しいコンテキストにコピーされたパラメータが更新対象となります。

この特性がチェックされない場合、新しいコンテキストのグローバルパラメータは初期化されます。

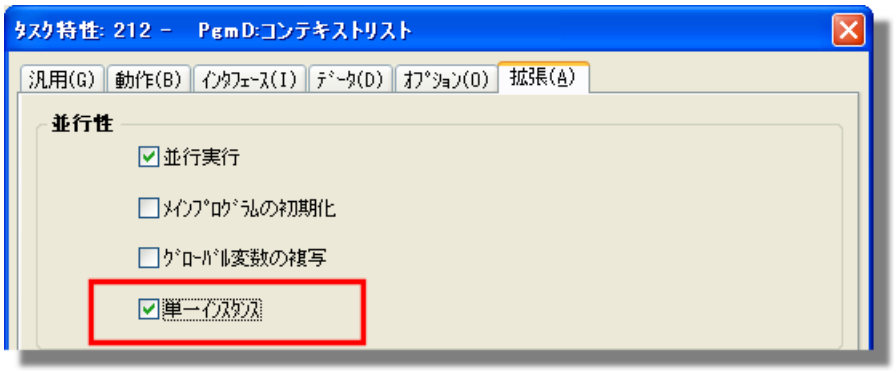
これらのグローバルパラメータは、**SetParam()** 関数で扱うことのできる項目です。**SharedValSet()** と **SharedValGet()** 関数を使用することで、コンテキスト間で常にパラメータを共有することができます。

実行プログラムの単一インスタンスを強制するには

並行タスクには、２種類の基本的なタイプがあります。

最初のタイプは、同時に複数の異なるウィンドウをオープンします。例えば、同時に複数の異なるドキュメントを処理するために複数のチャットウィンドウでこれらをオープンするような場合に使用します。これは、Magic の並行プログラムのデフォルトタイプとなります。

２番目のタイプは、常に同じウィンドウで処理する場合です。例えば、ドキュメントを開くメニューやチャットウィンドウを開くリスト等がこのケースに当たります。このタイプの並行プログラムは、**単一インスタンス**プログラムと呼ばれます。



プログラムを単一インスタンスプログラムで動作させたい場合以下の手順で設定します。

1. **タスク特性**（*Ctrl+P*）を開きます。
2. **拡張**タブを選択します。
3. **単一インスタンス**特性をチェックします。

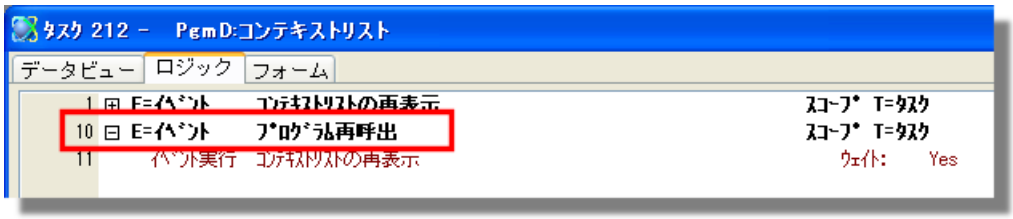
これで、メニューからや、別のプログラムから起動された場合、このプログラムは独自のコンテキストを開きます。しかし、このプログラムがすでに実行中に、同じように起動された場合、新しいコンテキストを作成せず、既存のコンテキストが切り替わります。

単一インスタンスプログラムの再起動を識別するには

インスタンスプログラムが最初に起動されると、新しいコンテキストを作成します。その際、タスク前が実行されます。

単一インスタンスプログラムが２回目に起動されると、既存のインスタンスが使用されます。この場合、フォーカスは移動しますが、タスク前は実行されません。

従って、このプログラムが再度呼び出されたことを認識させるための処理が必要になります。



このような場合、プログラム再呼出という内部イベントが発生します。このイベントは、単一インスタンスプログラムが、再起動された場合のみ実行されます。最初にプログラムが起動された場合は、タスク前を利用します。

［このページは意図的に空白にしています。］

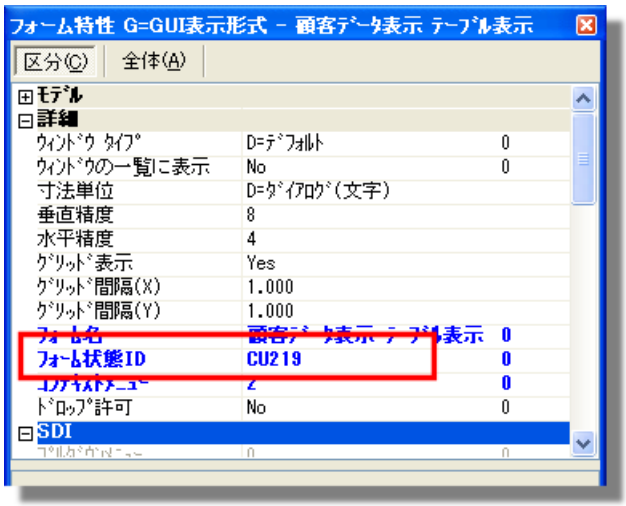
第 26 章 : ウィンドウインタフェース

エンドユーザがカスタマイズしたフォーム状態を保持するには

Windows 上で実行する Magic は、ウィンドウ表示の内容を様々にカスタマイズすることができます。例えば、テーブル上のカラムのサイズを変更したり、ソート順序を変更したり、カラムの配置を変更したりすることができます。ウィンドウの位置やサイズの変更も可能です。

ユーザは、セッション間に保存されたこれらのカスタマイズ情報を保持することを考えると思います。しかし、これを手動で実現するには、それはたくさんのプログラミング作業が必要になります。Magic には、このような情報を保持するための機能が備わっています。この設定方法について説明します。

フォーム状態 ID 特性を使用する

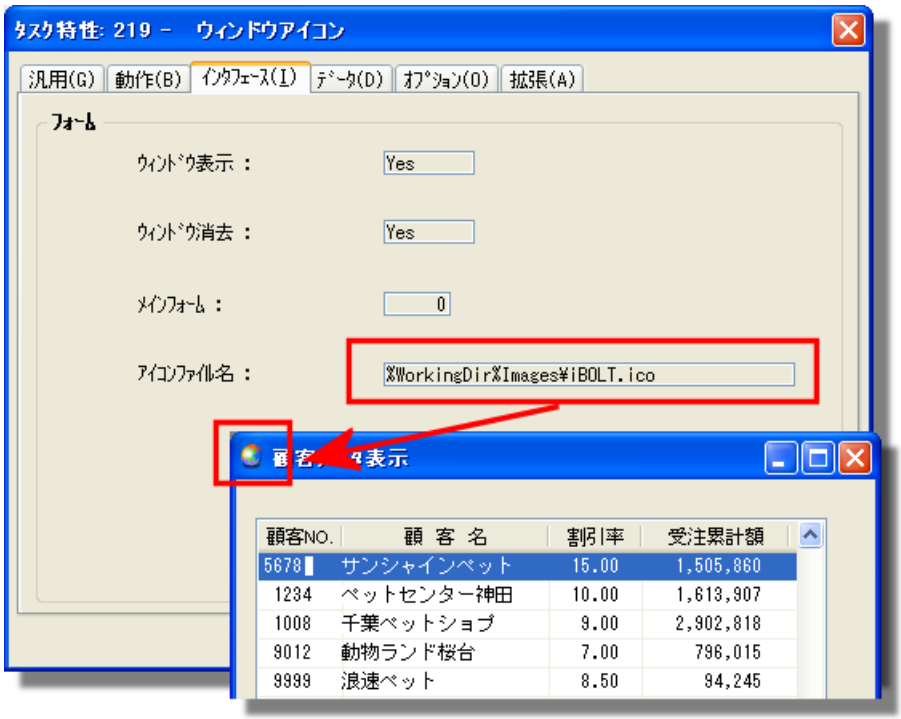


1. タスクの**フォームエディタ**を開きます。
2. ***Alt+Enter*** を押下して**フォーム特性**を開きます。
3. **フォーム状態 ID** 特性に移動します。
4. ユニークな識別子を入力します。ID には任意の文字列が指定できますが、ここでは簡単に参照できるように、各プログラムに割り当てプログラム ID を使用します。

これで、ユーザがウィンドウに入り直す際に、ユーザによってカスタマイズされたウィンドウ情報が記憶されます。

注 : この機能は、開発エンジンで実行している場合は無効です。つまり、**フォーム状態 ID** 特性を設定し、***F7*** によってプログラムを実行するしても、フォームのカスタマイズ情報は保持されません。

ウィンドウにアイコンを設定するには



アプリケーション特性のアイコンファイル名特性にて、アプリケーション全体に反映されるデフォルトアイコンファイルを指定することができます。しかし、ウィンドウ毎に個別のアイコンを設定することもできます。

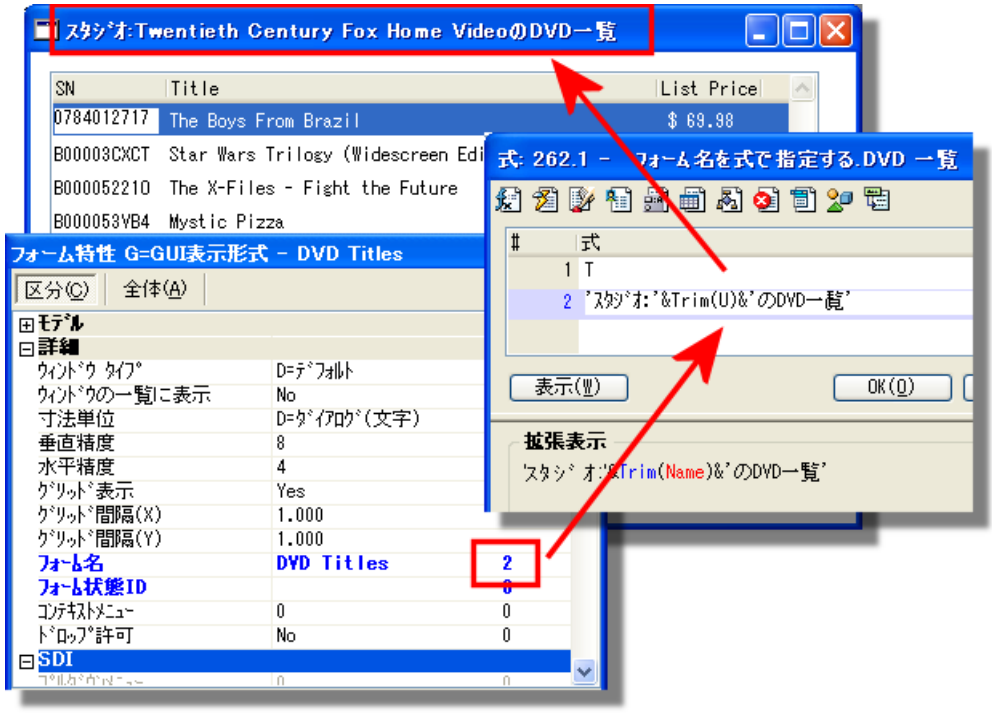
1. アイコンを設定したいタスクを開きます。
2. *Ctrl+P* を押下してタスク特性ダイアログを開きます。
3. インタフェースタブをクリックします。
4. アイコンファイル名特性でズームしてアイコンファイルを選択するか、相対パスを入力するか、論理名を使用してアイコンファイル名を指定します。

アイコンは実行時にウィンドウの左上に表示されます。

動的にウィンドウタイトルを設定するには

デフォルトでは、**フォーム名**特性はタスクの名前から引き継がれます。**フォーム特性**の**フォーム名**特性を変更することで簡単にこれを変更することができます。しかし、実行時に動的に名前を設定することもできます。

フォーム名特性を式で設定する



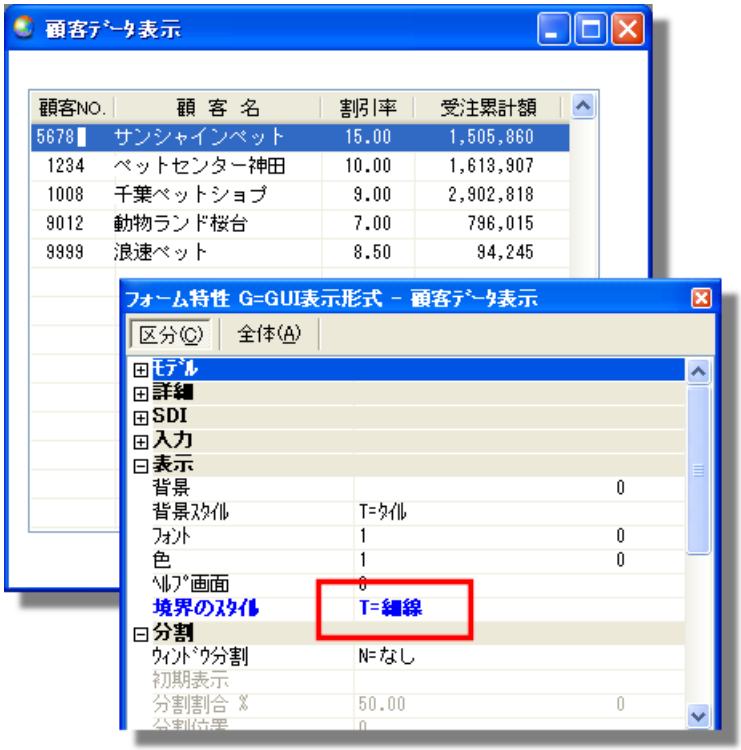
1. タスクの**フォームエディタ**を開きます。
2. *Alt+Enter* を押下して**フォーム特性**を開きます。
3. **フォーム名**特性に移動します。
4. **式**欄からズームしてフォームタイトルとして評価される文字型データを定義します。

開発時は、**フォーム名**特性のテキスト指定がタイトルとして表示されます。しかし、実行時は、式が評価されタイトルバーに表示されます。

ユーザがウィンドウのサイズ変更をできないようにするには

デフォルトでは、ユーザがウィンドウの端をドラッグすることでウィンドウのサイズを変更するこができます。しかし、必要であれば、**境界のスタイル**特性を ***T= 細線***または ***N= なし***に設定することで、この操作ができないようにすることが可能です。

サイズ変更を防止する



1. タスクの**フォームエディタ**を開きます。
2. ***Alt+Enter*** を押下して**フォーム特性**を開きます。
3. **境界のスタイル**特性に移動します。
4. **式**欄から**ズーム**してフォームタイトルとして評価される文字型データを定義します。
5. ***T= 細線***または ***N= なし***を選択します。

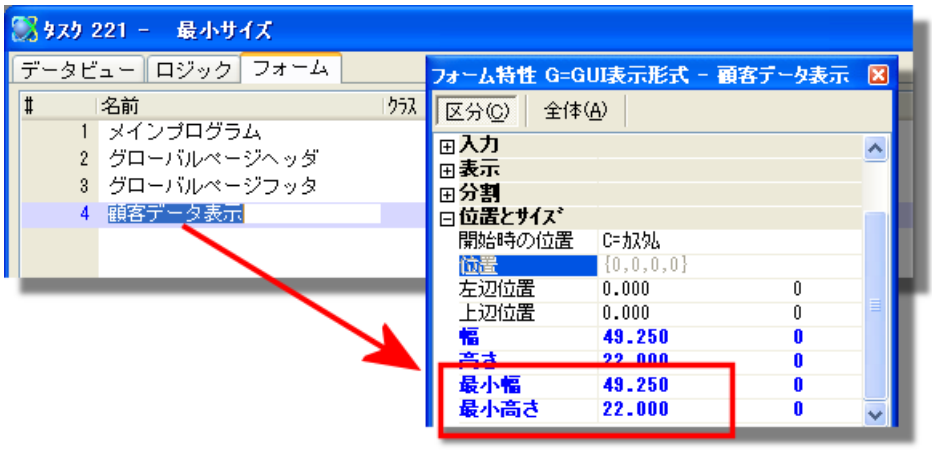
これで、カーソルの形状はウィンドウの端に移動しても変更されず、ウィンドウのサイズを変更することができなくなります。

ウィンドウの最小サイズを設定するには

Magic で作成されるウィンドウは、デフォルトではユーザによってサイズが変更可能になっています。位置特性の指定にもとづいて、ウィンドウのサイズが変更されると、テーブルなどのコントロールもサイズが変更されます。しかし、フォームのサイズを小さくし過ぎると非常に見づらい表示になる場合があります。

これを防止するには、フォームの最小サイズを設定することです。フォームは、開発時に設定されたサイズ以上に大きくすることはできますが、指定されたサイズより小さくすることはできなくなります。

最小サイズを設定する

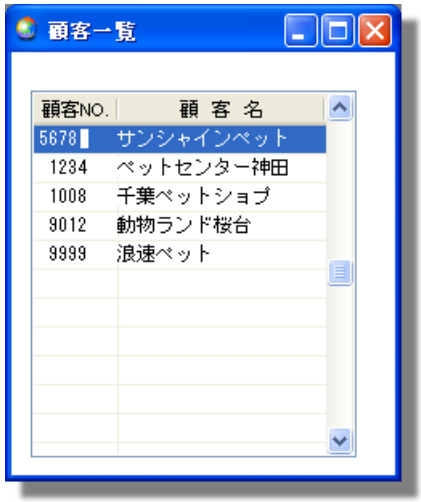
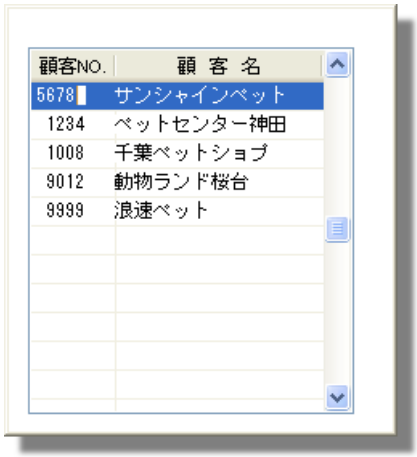


1. タスクのフォームエディタを開きます。
2. *Alt+Enter* を押下してフォーム特性を開きます。
3. 最小幅特性と最小高さ特性の値を設定します。
4. これらの特性値を式で指定することもできます。

これで、この例のように、フォームは幅 49.25（ダイアログ）、高さ 22（ダイアログ）以下には小さくすることはできません。

ヒント :*最小サイズを設定する場合の適切な値を求める方法として、手動でウィンドウのサイズを変更し、その時の幅と高さの特性値をもとに設定します。*

ウィンドウタイトルを表示させないようにするには

タイトルを表示	タイトルを表示しない
	
フォーム特性→タイトルバー→ Yes	フォーム特性→タイトルバー→ No

デフォルトでは、すべてのウィンドウは標準の Windows タイトルバーを持っています。フォーム特性のタイトルバー特性を *No* に設定することで、タイトルを表示させないようにすることができます。

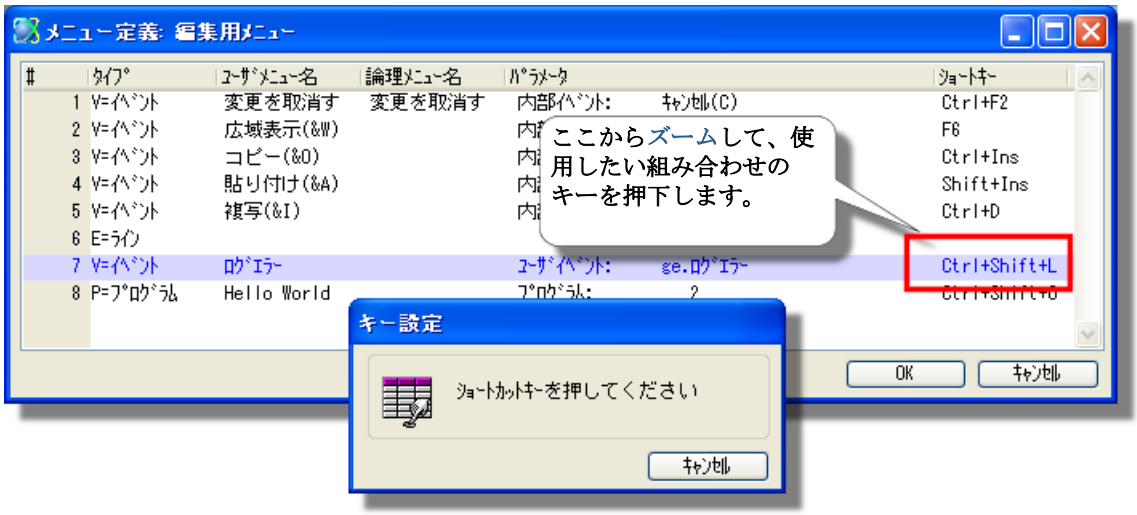
タイトルバーは表示させ、タイトルのみ表示させないようにするには、フォーム名特性を空白にします。

第 27 章 : メニュー

メニューにショートカットキーを定義するには

多くのユーザは、頻繁に使用する機能に対して***ショートカットキー***（または、***ホットキー***）と呼ばれる組み合わせキーを割り当てることがあります。アプリケーションを作成する際に、メニューに対して独自のショートカットキーを必要に応じて定義することができます。

内部イベント以外に対してショートカットキーを割り当てる



1. ショートキーカラム上にカーソルを置きます。
2. ここらかズームします（*F5* または、ダブルクリック）。キー設定ウィンドウが表示されます。
3. 割り当てたい組み合わせキーを押下します。この例では、*Ctrl+Shift+L* を押下しています。*Alt* や *Enter* を含めた、任意のキーの組み合わせを使用することができます。
4. 組み合わせキーを押下すると、キー設定ウィンドウがクローズされ、ショートキーカラム上に設定した組み合わせキーが表示されます。

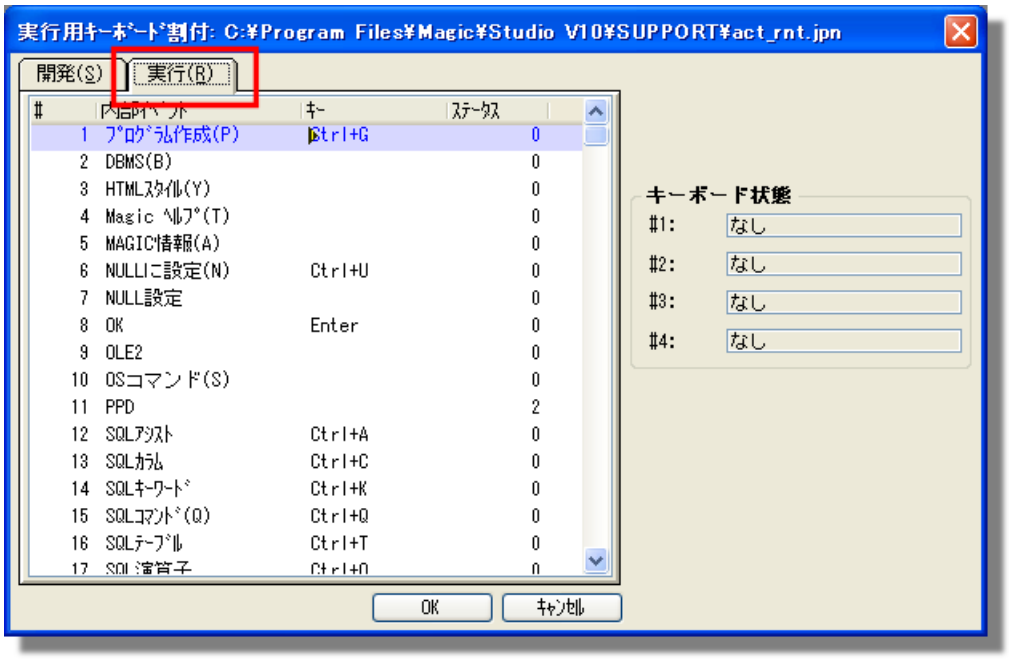
これで、アプリケーションが実行されると、定義されたショートカットキーを押下することで、メニューに定義された処理が実行されます。

内部イベントに対してショートカットキーを割り当てる

内部イベントに対してショートカットキーを割り当てようとすると、以下のメッセージが表示されます。

内部イベントのショートカットキーは、キーボード割付（オプション→設定→キーボード割付）にもとづいて、自動的に設定されます。

キーボード割り当てを変更する場合は、以下に示されるように、実行タブ上の内容のみ変更しなければなりません。
キーボード割り当てを変更すると、変更内容はメニュー表示に自動的に反映されます。



実行プログラムのメニューを変更するには

メニューリポジトリを使用することで、Magic アプリケーションのメニュー構造を設計することができます。しかし、実行しているプログラムや処理内容にもとづいてメニュー内容を変更することもできます。

メニュー表示の変更

メニュー上の項目の有効 / 無効を指定することができます。以下の関数は、メニュー上の項目自体を変更することはありませんが、メニュー項目を非表示にしたり、無効にしたりすることができます。

- **MnuCheck()** ：メニュー項目にチェックマークを付けたり外したりすることができます。
- **MnuEnable()** ：メニュー項目の有効／無効を指定できます。無効にした場合、メニュー名は灰色で表示されます。
- **MnuShow()** ：メニュー項目の表示／非表示を指定できます。

これらは、メニュー項目に依存します。詳細は、「メニューの表示／非表示を行うには」（595 ページ）を参照してください。

サブメニューの追加

メニュー関数は、メニュー項目を追加したり、削除したりすることを可能にします。これらの関数は、メニュー項目の変更は行いません。メニュー項目を既存のメニュー構造に追加して、メニュー表示を動的に変更するために使用されます。この関数を使用することで、メニュー表示に柔軟性を与えることができます。

- **MnuAdd()** ：メニュー項目を追加します。
- **MnuRemove()** ：メニュー項目を削除します。
- **MnuReset()** ：メニューをデフォルト状態に戻します。

注：　セキュリティ上の理由でメニューを変更するのであれば、そのメニュー項目に対して権利設定をを行うことで対応できます。ユーザが、メニュー項目を使用する権限を持っていない場合、プログラミングで操作することなく自動的に表示されなくなります。

MnuAdd() 関数を使用する

MnuAdd() 関数は、メニュー項目をメニューシステムのどの中にでも追加することができます。構文は以下の通りです。

MnuAdd(*MenuNumber*, *MenuPath*)

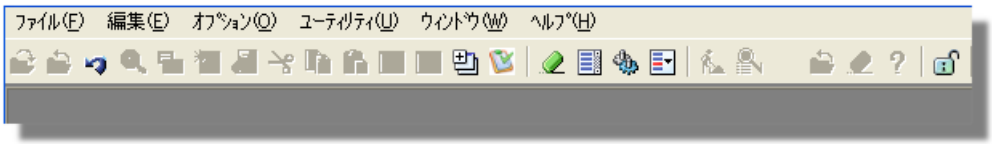
パラメータ：

- ***MenuNumber*** ：追加対象となるメニューの番号。ここには、メニューリポジトリの行番号を指定します。**MENU** リテラルを指定する必要があります。この例では、***'3'MENU*** を使用しています。
- ***MenuPath*** ：この後に挿入したいメニュー項目のパス。パスの最後にバックスラッシュ (¥) が指定され場合は、サブメニューとして追加されます。

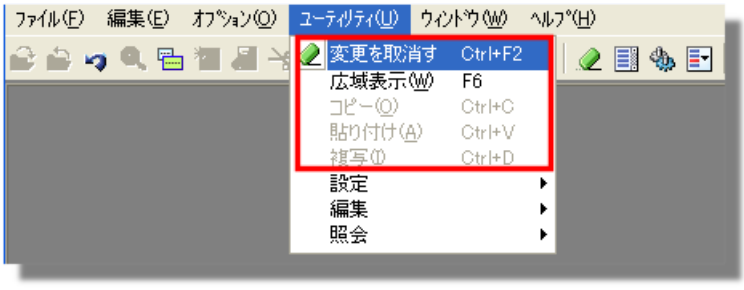
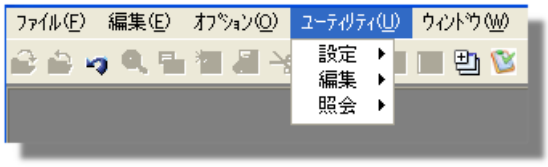
以下は設定例と、それによる表示結果を示しています。

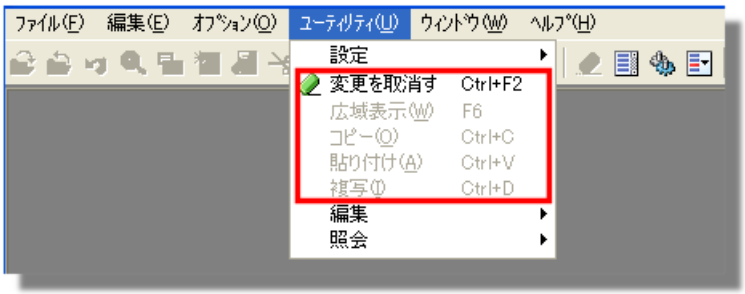
メニューリポジトリは、以下のように定義されています。メニュー **#3** は、プルダウンメニューに追加しようとしているものです。

以下は、**MnuAdd()** の 3 つの異なる指定方法によってどのように表示が変わるかを示しています。

関数の指定例
変更前のメニュー

MnuAdd(*'3'MENU,''***)** メニュー **#3**（編集用メニュー）がデフォルトプルダウンメニューの最後の位置に追加されます。

MnuAdd(*'3'MENU,'UtilityMenu'***)** デフォルトプルダウンメニューのユーティリティメニューの後にメニュー **#3**（編集用メニュー）が追加されます。

MnuAdd(*'3'MENU,'UtilityMenu\'***)** デフォルトプルダウンメニューのユーティリティメニューのサブメニューとしてメニュー **#3**（編集用メニュー）が追加されます。

変更前のメニュー


関数の指定例
MnuAdd(*'3'MENU,'UtilityMenu\SetupMenu'***)** ユーティリティメニューのサブメニューである**設定**メニューの後にメニュー **#3**（編集メニュー）が追加されます。

MnuAdd(*'3'MENU,'UtilityMenu\SetupMenu\'***)** ユーティリティメニューのサブメニューである**設定**メニューのサブメニューとしてメニュー **#3**（編集メニュー）が追加されます。


MnuRemove() 関数を使用する

MnuRemove(*MenuNumber*, *MenuPath***)**

パラメータ：

- *MenuNumber* …… 削除対象となるメニューの番号。ここには、**メニュー**リポジトリの行番号を指定します。**MENU** リテラルを指定する必要があります。
- *MenuPath* …… 削除するメニュー項目のパス。**MnuAdd** 関数で指定したパスと同じ値を指定する必要があります。

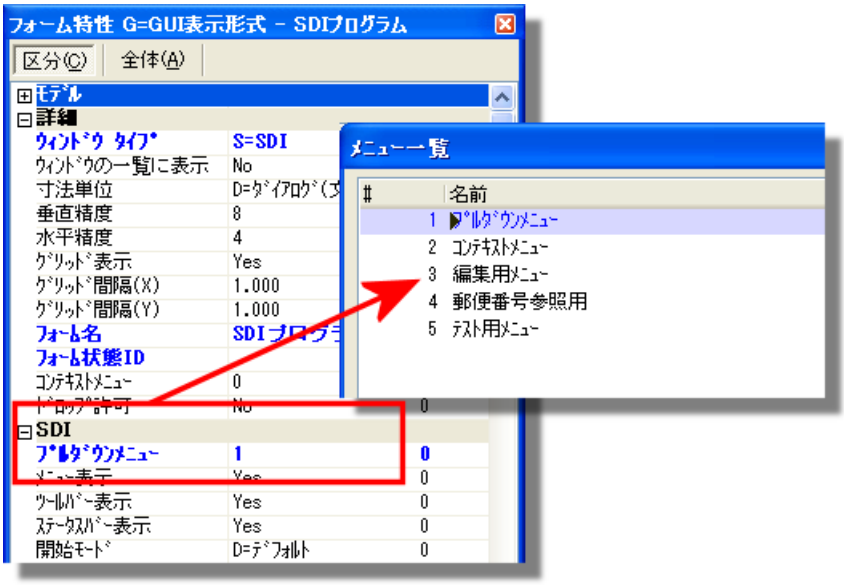
MnuRemove() 関数は、**MnuAdd()** 関数と同じようなパラメータですが、動作は逆になり、追加されたメニューを削除します。

MnuAdd(*'3'MENU,'UtilityMenu\SetupMenu'***)** や **MnuAdd(***'3'MENU***)** などは、前述で追加されたメニューを削除します。

MnuReset() 関数を使用する

関数は、**MnuAdd()** 関数が実行される前のメニュー状態に戻します。**MnuRemove()** の代わりに **MnuReset()** を使用することもできますが、**MnuRemove()** が、追加された 1 部分のみを削除できるに対し、**MnuReset()** は追加されたメニューがすべて削除されます。

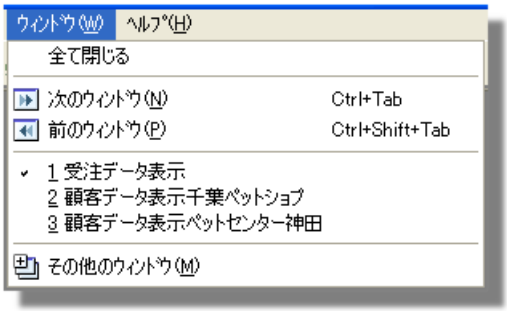
SDI プログラムのメニュー変更



SDI プログラムは、独自のプルダウンメニューを持っています。SDI フォーム上の**プルダウンンメニュー**特性を指定することで指定することができます。式で指定することもできますが、式はプログラムが起動される前に 1 回だけ評価されるため、プログラムの実行中に式の評価結果が変更されてもメニューには影響しません。プログラムの実行中に変更したい場合は、メニュー関数を使用することで変更することができます。

キーボードでウィンドウの切り替えを有効にするには

Magic には、ユーザに対して複数のウィンドウを一度にオープンしておくことを許可させるためのオプションがあります。ユーザがこれらのウィンドウ間の移動を容易にするために、ウィンドウメニュー機能を使用することができます。

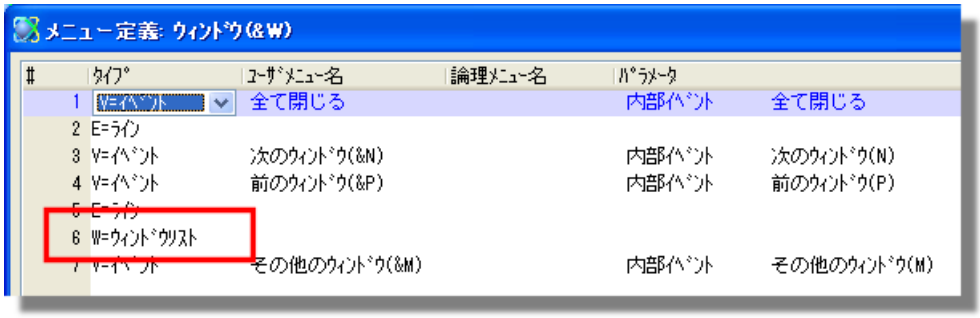


ウィンドウメニュー機能を使用して、複数のウィンドウ間を移動することができます。この例では、同時に開いている 3 つのウィンドウがあります。 1 つのウィンドウは、注文データを表示しています。他の 2 つのウィンドウは、2 つの異なる顧客用に起動された同じ顧客情報表示のプログラムです。

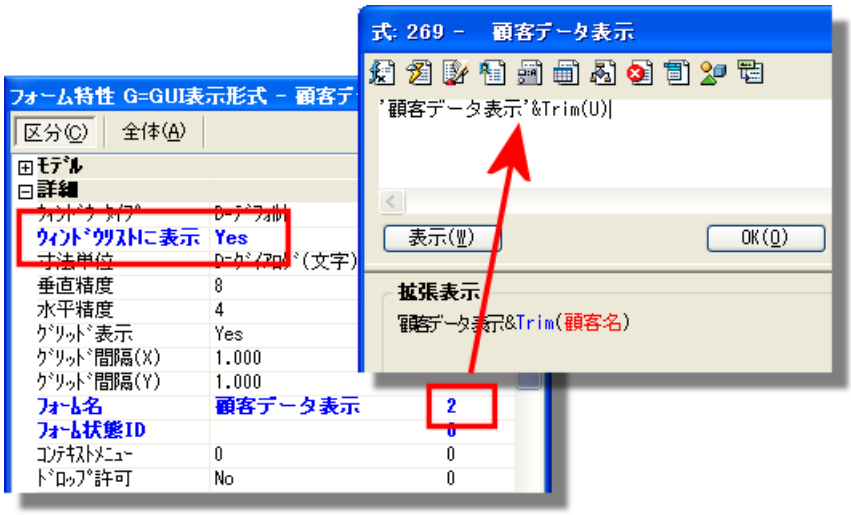
- **ウィンドウ操作**：ユーザは、オープンされたウィンドウ間で移動するための**次のウィンドウ**（*Ctrl+Tab*）または**前のウィンドウ**（*Ctrl+Shift+Tab*）を使用することができます。また、メニュー上にあるウィンドウのリストからウィンドウを選択することができます。メニュー項目の最初の文字を入力することでカーソルをその項目に移動します。
- **その他のウィンドウ**：9 つを超えるウィンドウがオープンされている場合、リスト全体を表示させるためのウィンドウを選択することができます。
- **リストの順番**：使用された順番や作成された順番にもとづいて、最新のウィンドウのリストを表示します。表示基準は、**アプリケーション特性**の**ウィンドウリストの表示順序**特性によってアプリケーション単位で指定することができます。この設定は、メニューリストやウィンドウ間で移動する場合に使用する *Ctrl+Tab* の両方に影響します。

次に、この機能の使用方法について説明します。

ウィンドウメニューを使用する



1. 最初に、必要な場所にウィンドウメニューを定義する必要があります。デフォルトでは、上記で示されるように、**ウィンドウ（&W）**と呼ばれるプルダウンメニューオプションがあります。オープンされているウィンドウの一覧は、**ウィンドウリスト**と呼ばれる入力タイプで表示されます。必要に応じて別の場所に設定することもできます。



2. 次に、このメニューに表示させるようにプログラムを設定する必要があります。フォーム特性のウィンドウリストに表示特性を *Yes* に設定します。ウィンドウタイプ特性が *デフォルト*、*フローティング*、*ツール*、*MDI 調整*の場合に対してのみ、この設定を行うことができます。
3. フォーム名がウィンドウリスト上で表示されるため、各ウィンドウ毎にユニークなフォーム名を定義するようにしてください。この例では、各ウィンドウ毎に顧客の名前をフォーム名として表示させるように、パラメータを使用して式でフォーム名を指定しています。

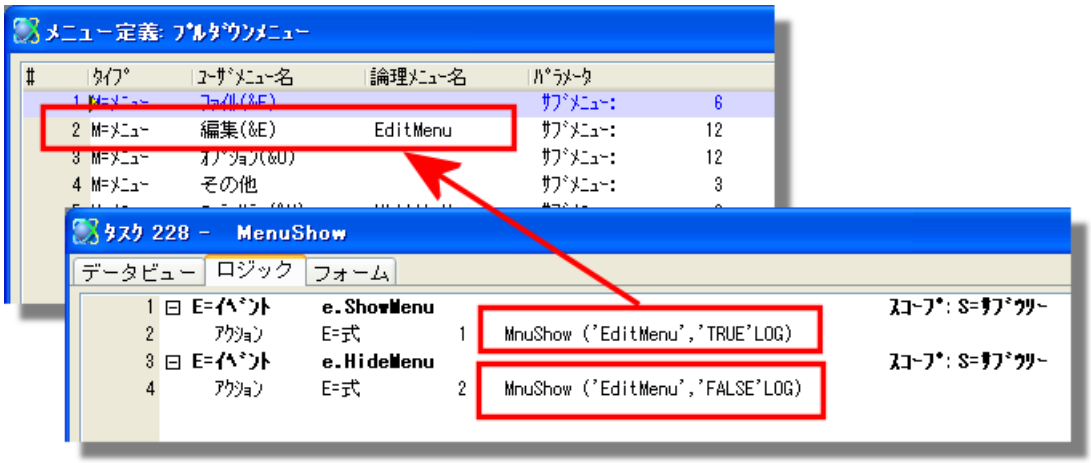
これでプログラムが実行されると、ウィンドウメニューに表示されます。

メニューの表示／非表示を行うには

MnuShow() 関数を使用することで、実行時にメニュー項目を表示させないようにすることができます。また、**MnuEnable()** または **MnuCheck()** を使用することで、表示されているメニューを機能を制御することができます。

注： メニュー特性の権利特性が設定されており、ユーザはその権利を持っていない場合、メニューは表示されません。

MnuShow() 関数を使用する



構文は以下の通りです。

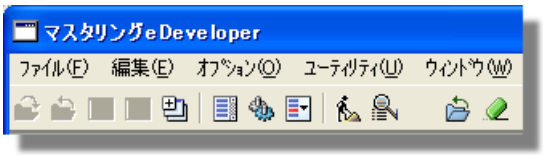
MnuShow(*MenuName*, *True/False*)

パラメータ：

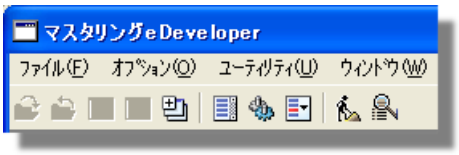
- *MenuName* ：表示させたいメニュー項目の**論理メニュー名**。この例では *EditMenu* が指定されています。
- *True/False* ：指定されたメニュー項目を表示させたい場合は、*'TRUE'LOG* を指定します。表示させない場合は、*'FALSE'LOG* を指定します。この例では、２つのボタンに対応したイベントが定義されています。１つは表示用、もう１つは非表示用です。

注： 論理メニュー名は、実行時に評価されます。このため、式では固定の名前を指定する必要はありません。

次にどのような結果になるかを紹介します。



MnuShow(*'EditMenu','TRUE'LOG*)



MnuShow(*'EditMenu','FALSE'LOG*)

メニューバーを削除するには

MDI または SDI アプリケーションからメニューバーを削除することができます。以下にその方法を示します。

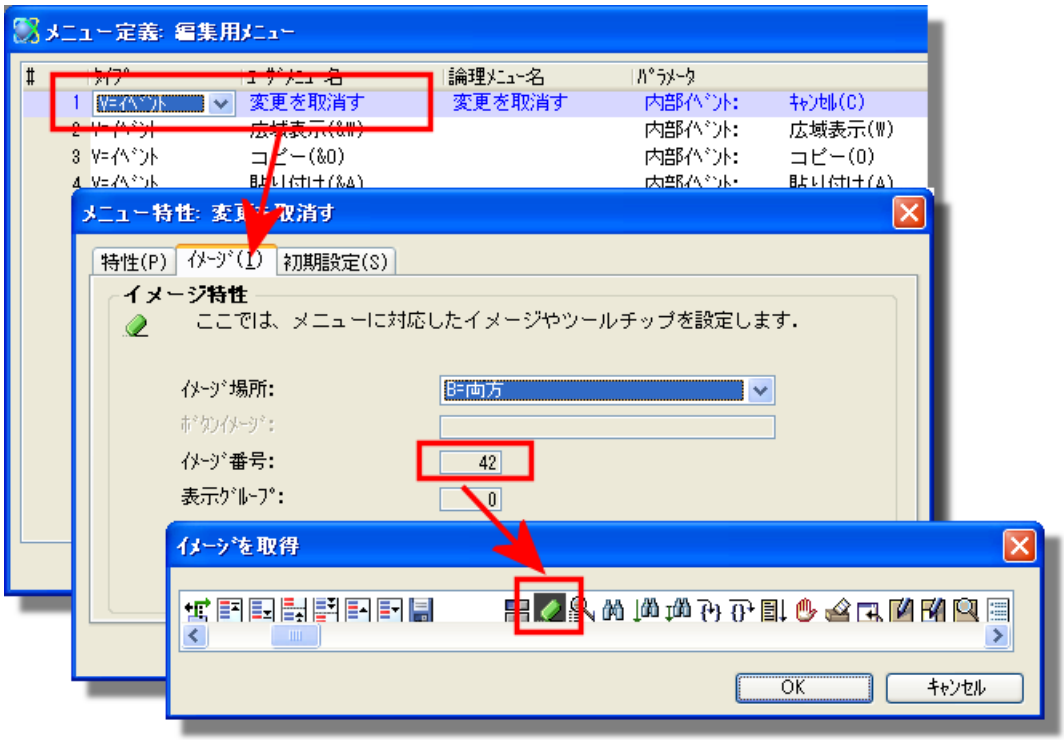
メニュー	MDI	SDI
プルダウンメニュー	アプリケーション特性のシステムプルダウンメニュー特性を *0* に設定します。	フォーム特性のプルダウンメニュー特性を *0* に設定します。 または、 フォーム特性のメニュー表示特性を *No* に設定します。
ツールバー	オプション→動作環境のツールバーの表示を *No* に設定します。	フォーム特性のツールバーの表示特性を *No* に設定します。
ステータスバー	ステータスバーを削除する必要がある場合、SDI プログラムに変更します。	フォーム特性のステータスバーの表示特性を *No* に設定します。
タイトルバー	タイトルバーを削除する必要がある場合、SDI プログラムに変更します。	フォーム特性のタイトルバーの表示特性を *No* に設定します。

メニューにアイコンを追加するには

プルダウンメニューまたはコンテキストメニューにアイコンを追加することができます。同じアイコンをツールバーに表示させることもできます（「ツールバーにアイコンを追加するには」（599 ページ）を参照）。

アイコンは、Magic が提供するリソースファイルから選択したり、ビットマップファイルを指定することができます。

メニューアイコンを指定する



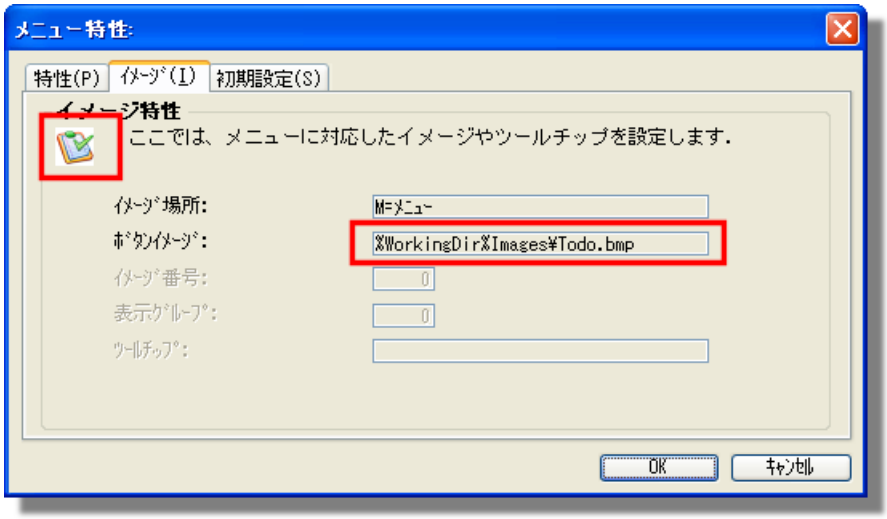
メニュー特性で指定することによりメニューにアイコンを追加することができます。

1. メニューリポジトリ（*Shift+F6* または、プロジェクト→メニュー）に移動します。
2. アイコンが必要なメニュー項目に移動します。*Alt+Enter* を押下して、メニュー特性を開きます。
3. イメージタブをクリックします。
4. イメージ場所特性を*メニュー*（メニューにのみアイコンを表示させたい場合）または *B= 両方*（ツールバーにも表示させたい場合）のどちらかを設定します。
5. Magic のリソースファイルからアイコンを選択する場合は、イメージ番号特性からズームします。表示させたいアイコンを選択し、OK をクリックします。アイコン番号がイメージ番号特性に設定され、選択されたアイコンがツールタブの上部に表示されます。

これで、実行時にアイコンがメニューに表示されます。

アイコンとしてビットマップファイルを選択した場合は、以下のようになります。

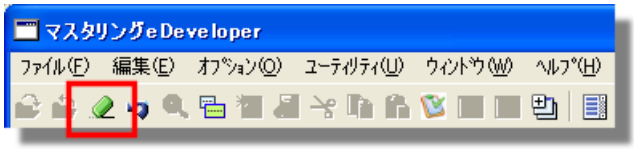
独自のアイコンを指定する



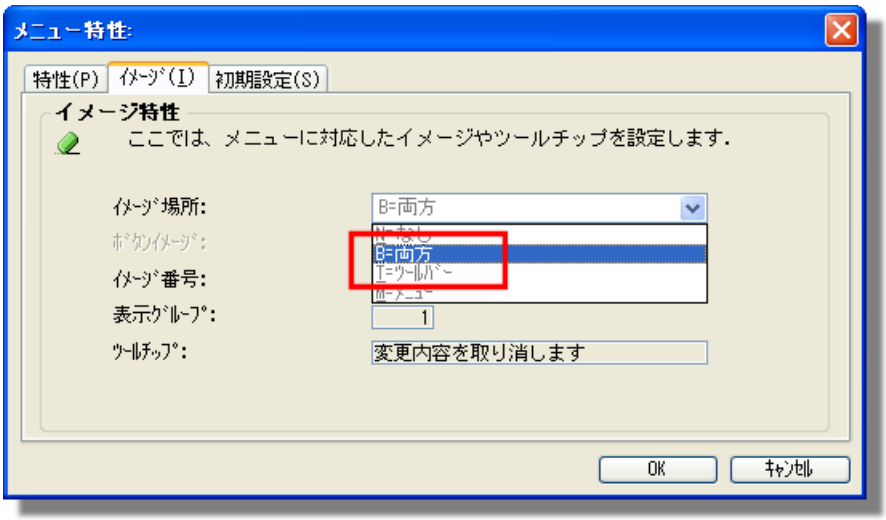
独自のアイコンを指定することもできます。アイコン用のイメージファイルは、16 × 16 ～ 24 × 24 ピクセルの bmp ファイルにしてください。

1. 前述のように、**メニュー特性**を開きます。ここでは、**ボタンイメージ**特性を指定します。ここから**ズーム**してイメージファイルを選択することもできますが、上の図のように論理名を使用した方が汎用性が高くなります。
2. アイコンが、**イメージ**タブの上部に表示されます。

ツールバーにアイコンを追加するには



メニューに対応したツールバーアイコンを追加することができます。



ツールバーにアイコンを追加するには、アイコンをメニューに設定する手順に従って操作します（「メニューにアイコンを追加するには」（597 ページ）を参照）。ただし、**イメージ場所**特性は、***B= 両方***または、***T= ツールバー***を選択します。

［このページは意図的に空白にしています。］

第 28 章 : Unicode

多言語データをサポートできるようにするには

Magic は、多言語データをサポートしています。Magic は、内部で Unicode 形式のデータ書式をサポートしており、また Unicode と他のデータ形式間で変換するためのツールを持っています。

しかし、PC 上の Unicode で動作するため、使用する PC が Unicode を正しくサポートしているかどうかを確認する必要があります。

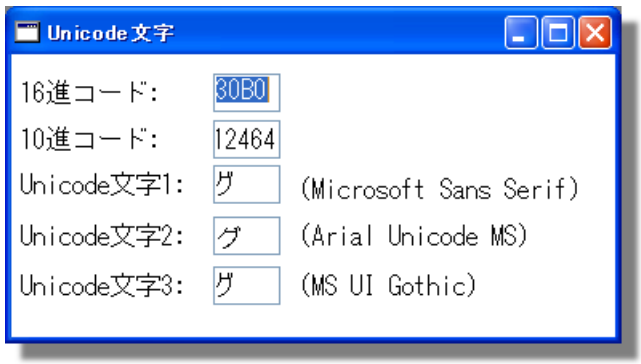
また、Unicode データは、Unicode フォントが設定された Unicode 型の項目を使用して表示させる必要があります。ここでは、これらの内容について説明します。

Unicode フォント

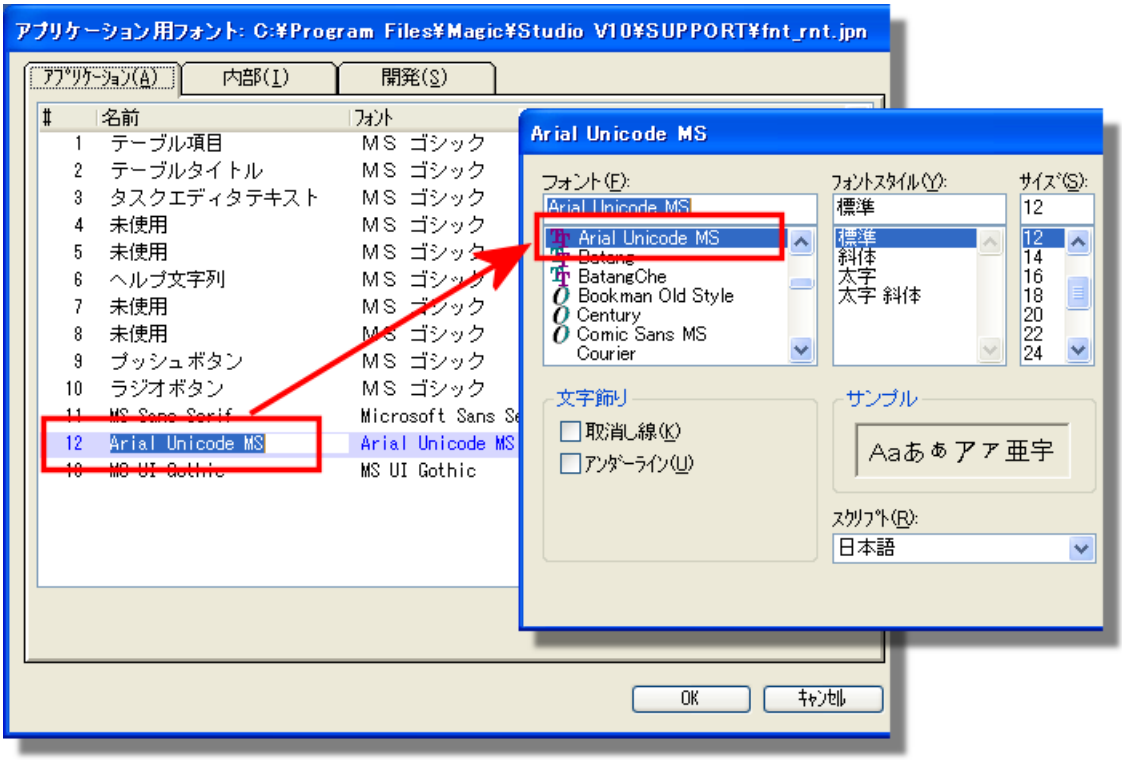
Ascii（JIS）コードを使用していた当初は、文字を 1 バイトで納めることができました。これは、一般的にプログラム言語は、英語を基準にして作成されていたためで、特殊文字を除き、アルファベットの 26 個の文字で十分な状態でした。しかし、" 特殊文字 " が、用法に応じた異なる利用方法が出てきたため、Ascii コード 210 は、使用される ***コードページ***にもとづいて、変形した E や O、またはグラフィックライン文字を表すために使用されるようになりました。

この問題を回避するために、Unicode が開発されました。Unicode は、基本的には、世界中のあらゆる文字を 1 つのコードページに統合したものです。www.unicode.org のサイトには、Unicode と文字の割り当てに関する情報が記載されています。ここには、オリジナルの Ascii フォントが 0 から 128 までに割り当てられていますが、更に様々な国の文字が追加されています。例えば、X'30B0' は日本語の「グ」が割り当てられています。歴史的な文字や架空の文字（クリンゴン文字も存在します）も割り当てられています。

これは考え方としては問題ないのですが、フォントデータが非常に大きなものになってしまいます。このため、多くの Windows システムは、すべての Unicode 文字を参照できるようにフォントデータがインストールされていません。他言語用の文字を表示させたい場合は、必要に応じてフォントをインストールする必要があります。

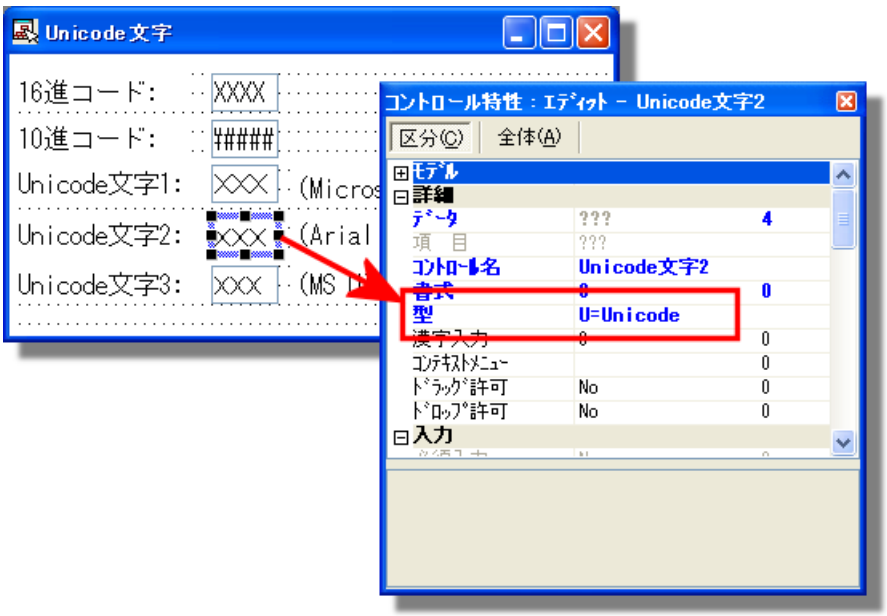


特定の言語で文字を表示している場合は、その言語用の Unicode フォントが必要になります。上記の例では、日本語の Unicode 文字 H'30B0' で表示されていますが、図に表示されているほど綺麗には表示されません。日本語の専用フォントを使用した方が綺麗に表示されます。



使用したい Unicode フォントを PC にインストールしたら、Magic でそのフォントを使用できるように設定する必要があります。フォント定義テーブル（オプション→設定→フォント）でそのフォントを選択し、Unicode フォントが必要な場合は、これを使用するようにします。この例では、フォント **#12** は Unicode になっています。

Unicode 型

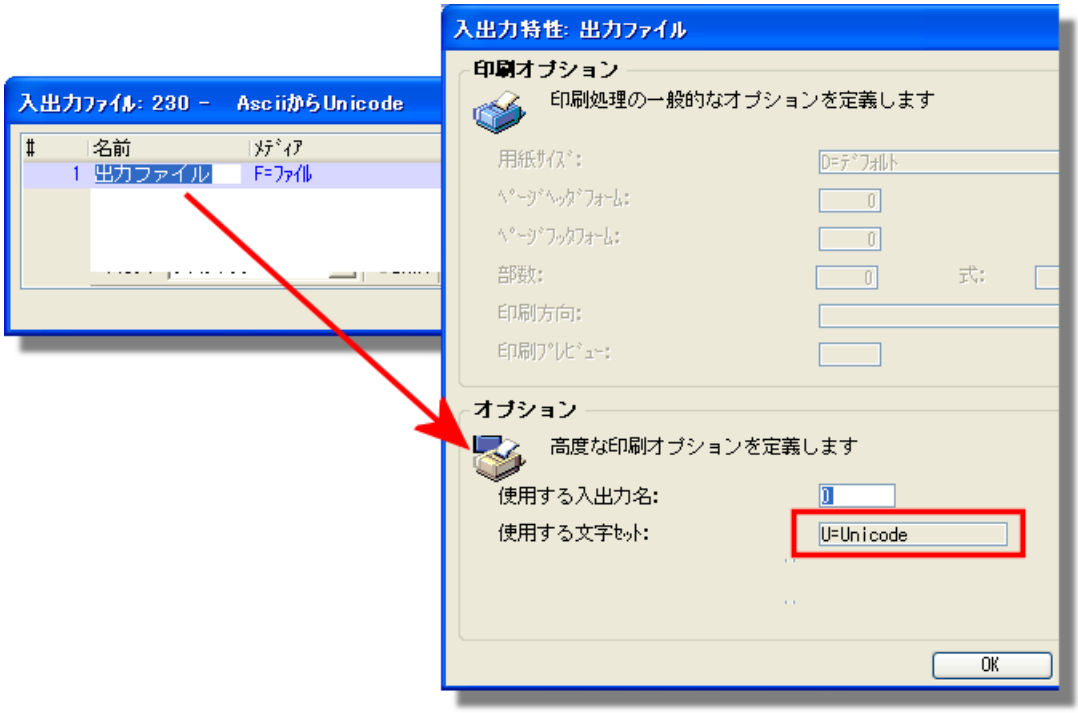


また、Unicode 文字が Unicode 型の項目で処理されていることを確認する必要があります。Unicode データを文字型や BLOB 型（RTF）に変更する場合、正しく表示されない場合があります。

Unicode のテキストファイルの作成 / 読み込みを行うには

Magic では、Unicode 形式のテキストファイルを他の形式と同じように読み書きすることができます。この場合、以下の２つの処理が必要です。

- Unicode データを Unicode 型の項目に格納します。ANSI（JIS）コードから変換する必要がある場合、**UnicodeFromANSI()** 関数を使用します。
- 以下に示すように、入出力特性の使用する文字セット特性で ***Unicode*** を選択します。

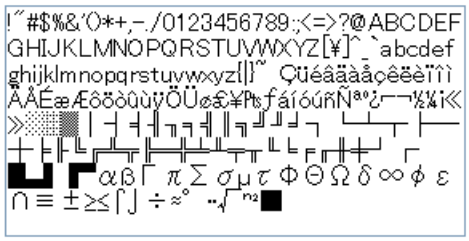


自動的に Unicode でフォーマットされます。

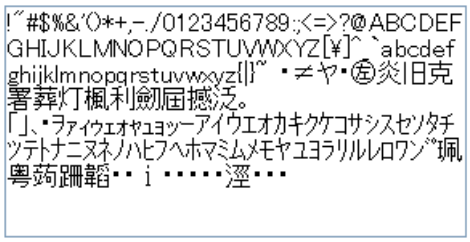
ANSI と Unicode 間でデータを変換するには

Ansi と Unicode 間で変換する場合、異なる ANSI のコードページが存在することを認識しておく必要があります。0 から 255（x'00' から x'FF'）の間の数字コードのみが使用されている場合、それを解釈するために使用されたコードページにもとづいて、異なる文字が表示されます。異なるコードページは、現在サポートされている言語にもとづいて使用されます。

例えば、#00 から #255 までの文字コードをコードページ 437（IBM PCM 基本）と 932（Shift-JIS）で表示させた場合、以下のようになります。



コードページ 437



コードページ 932

Unicode は、これらのすべての文字に対して文字変換が可能ですが、変換をする際にコードページによって変換される文字が異なることを知っておく必要があります。

文字を ANSI から Unicode に変換する関数は、以下の通りです。

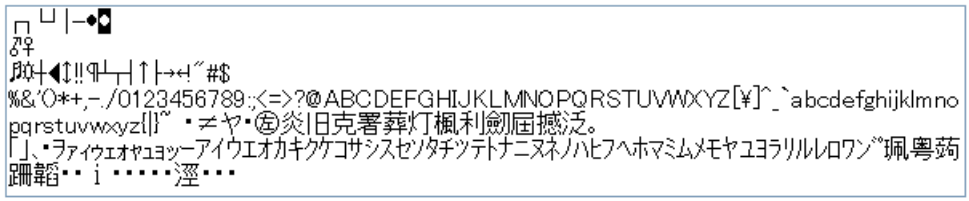
UnicodeFromANSI (*String*,*CodePage*)

パラメータ：

- *String* ：変換元の ANSI（JIS）文字列
- *CodePage* ：変換の際に使用するコードページ

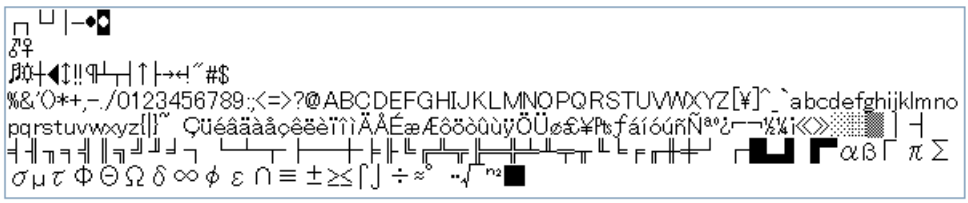
従って、M がすべての Ascii コードで共通な文字列の場合、**UnicodeFromANSI (*M*, *0*)** と指定できます。

Unicode(デフォルトのコードページ)



UnicodeFromANSI (*M*, *437*) を実行した場合、以下のような結果になります。

Unicode(コードページ:437)



Unicode コードを ANSI に変換する

以下の関数を使用して Unicode を ANSI 変換することができます。

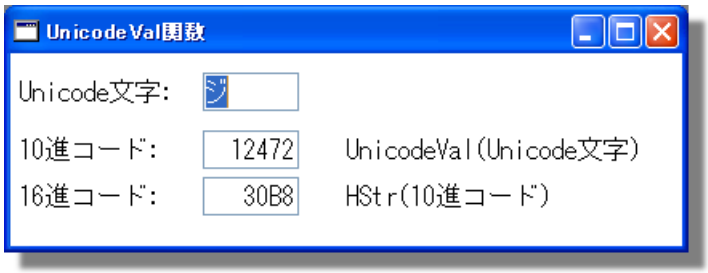
UnicodeToANSI (*String*,*CodePage*)

パラメータ：

- *String* ：変換元の Unicode 文字列
- *CodePage* s：変換の際に使用するコードページ

Unicode 文字の文字コードを取得するには

UnicodeVal() 関数を使用して Unicode 文字を 10 進数の値に変換することができます。この例では、カタカナ文字を 10 進値 12,472 に変換しています。



UnicodeVal(*UnicodeCharacter*)

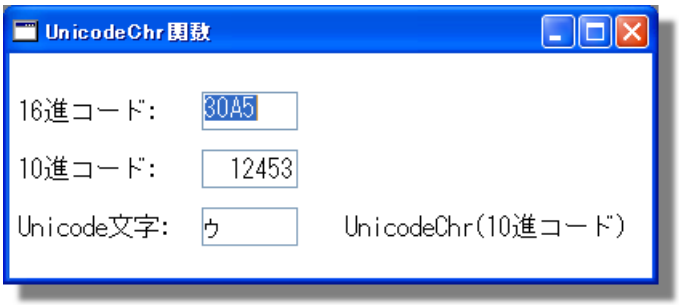
パラメータ：

- *UnicodeCharacter* ：Unicode 型の文字

指定された文字に対応した 10 進数の値が返ります。

Unicode の文字コードを文字に変換するには

UnicodeChr() 関数を使用して、10 進数の数値を Unicode 文字に変換することができます。



UnicodeChr(*Number*)

パラメータ：

- ***Number*** ：Unicode 値を表す 10 進数の数値

指定された数値に対応した Unicode 文字が返ります。

注： 指定する数値は 10 進数にする必要がありますが、Unicode のコードページは一般的に 16 進数で指定します。このため、**HVal()** 関数を使用することで、10 進数に変換するすることができます。例えば、**UnicodeChr(HVal(*30A5*))** は、上記の例と同じ結果 **UnicodeChr(*12453*)** になります。

第 29 章 : アプリケーションのデバッグ

デバッガを使用してデバッグするには

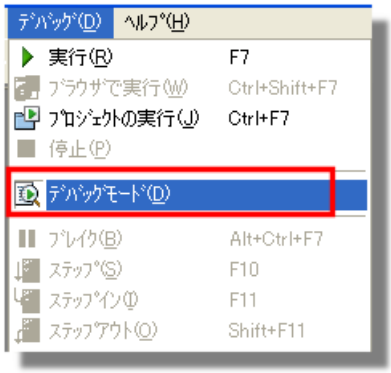
Magic には、デバッグツールが組み込まれています。このデバッガを使用すると以下のことが可能になります。

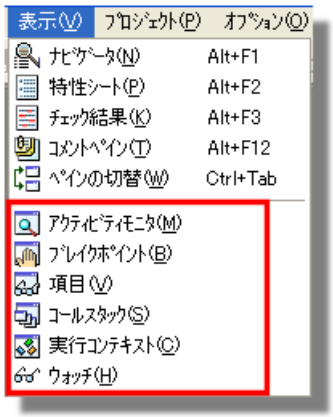
- エンジンの動作内容の確認
- コールスタックやデータベースアクセスの表示
- プログラム実行時に処理される内容の表示
- すべてのコンテキストの表示
- メモリ上のデータ項目の内容を表示／更新

ここでは、デバッガの使用方法について説明します。

1. デバッグモードを有効にする

最初にデバッグモードを有効にする必要があります。デバッグ→デバッグモードを選択するか、ツールバー上のアイコンをクリックすることで有効になります。デバッグモードが選択されると、アイコンが押下された状態で表示されます。





2. モニタ用ウィンドウを開く

デバッガが実行されると、対応するウィンドウ内にデータが表示されます。これらのウィンドウは、1 つに統合して表示させたり、個別に表示させたりすることができます。これらウィンドウのうちどれを表示させ、どれを表示させないかを指定することができます。

プルダウンメニューの表示メニューから表示させたいウィンドウをクリックすることで、該当するモニタ用ウィンドウが開きます。またデバッグ処理中にウィンドウを開くこともできます。

3. ブレイクポイントとウォッチ設定する

次に、何処で停止させるかを指定する必要があります。これには２つの方法があります。

- ブレイクポイントを設定します。処理コマンドや項目の値が更新される場所に設定できます。
- ブレイクの使用します。

これらの使用方法については、「アプリケーションのブレイクポイントを設定するには」（610 ページ）を参照してください。

項目やコールスタック、およびコンテキストはブレイクポイントでのみ参照できるため、ブレイクポイントの設定やブレイクの使用は重要になります。しかし、プログラムの実行中にこれらを設定することも可能です。

4. プログラムやプロジェクトを実行する

次に、プログラムまたはプロジェクトを実行します。1 つのプログラムをデバッグしているだけなら、そのプログラム上にカーソルを置き *F7* を押下します（デバッグ→実行）。開発中のタスクを閉じる必要はありません、Magic はプログラムを実行する前に変更内容を自動的に保存します。

プロジェクト全体をテストしたい場合、*Ctrl+F7*（デバッグ→プロジェクトの実行）を押下してください。

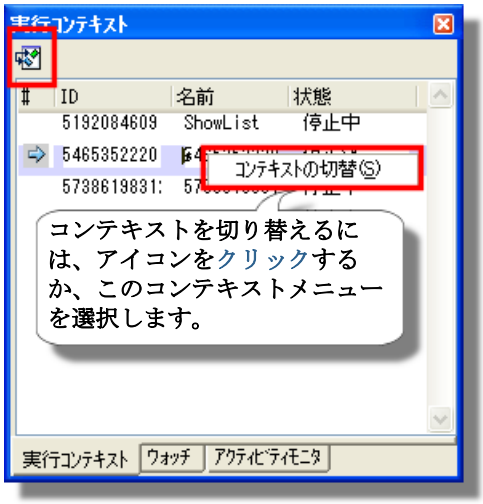
5. プロジェクトをステップ実行する

ブレイクポイントが設定されたら、実行ウィンドウ内で実行内容を参照しながら、処理コマンド毎にプログラムの処理ステップを進めることができます。以下の表に示すように、いくつかのステップオプションがあります。

コマンド	ショートカットキー	処理内容
実行	*F7*	次のブレイクポイントに到達するまで、プログラムを実行させます。
ステップ	*F10*	現在のプログラムの次の処理コマンドに進み、停止します。しかし、処理コマンドが別プログラムを呼び出す場合、コール処理コマンドで停止しません。
ステップイン	*F11*	ステップコマンドと同じように動作しますが、コール処理コマンドやイベント実行処理コマンドの場合、呼び出されたタスク／プログラム内に入った状態で処理が停止します。
ステップアウト	*Shift+F11*	現在のハンドラ内で処理が順番に停止しますが、ハンドラの起動内で処理を再開します。
ブレイクポイント	*F9*	ブレイクポイントやステップコマンドによって処理が停止している間に、既存のブレイクポイントを解除したり設定を追加したりすることができます。

デバッグ処理を終了する場合は、デバッグ→停止を選択するか、ツールバーの  をクリックします。プログラムに不具合がある場合、*Ctrl+Shift+F9*（デバッグ→実行エンジンのリセット）を押下することで実行エンジンを再起動させることができます。

並行プログラムの実行中にデバッガを使用するには



複数のコンテキストを同時に実行させている場合、デバッガは便利です。コンテキストビューは、既存のすべてのコンテキストのリストを表示します。

実行中のコンテキストリストから、コンテキストメニューの**コンテキストの切替**を選択するか、  アイコンを使用することでコンテキスト間の切り換えを行うことができます。ステータスが停止済のコンテキストのみこの機能は有効です。このため、コンテキストを切り替えたい場所にブレイクポイントを追加する必要があります。

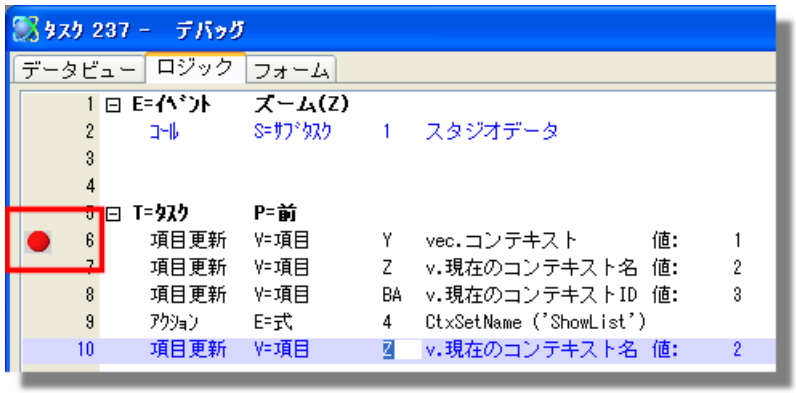
アプリケーションのブレイクポイントを設定するには

プログラムの実行中に処理内容を参照するためにブレイクポイントを設定します。ブレイクポイントには、以下の３つの異なる設定方法があります。

- 処理コマンドに設定する
- データ項目に設定する
- ブレイクを使用する

ここでは、ブレイクポイントの設定方法について説明します。

処理コマンドでの設定



ブレイクポイントは、開発エンジンで設定します。デバッガがブレイクポイントに到達した場合、処理が停止します。デバッガによって処理が停止すると、項目やコールスタック、およびコンテキストの内容を参照することができます。

ブレイクポイントを設定する

ブレイクポイントを設定するには以下のようにします。

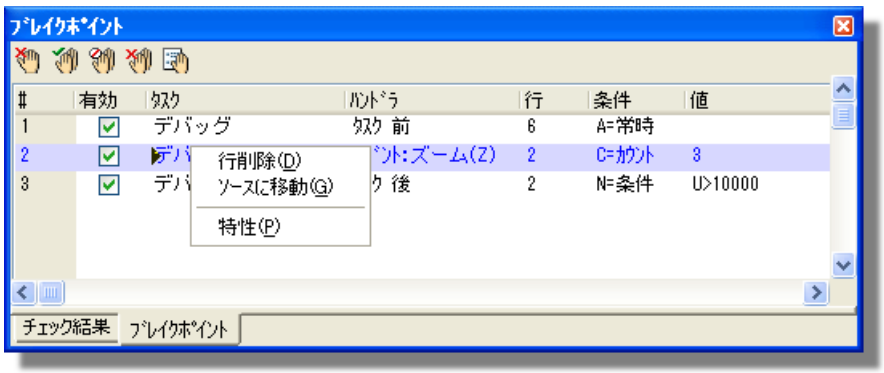
1. ロジックエディタを開き、停止させたい処理コマンドの設定行に移動します。
2. *F9*（デバッグ→ブレイクポイント）を押下します。

同じ場所で再度 *F9* を押下するとブレイクポイントは解除されます。しかし、ブレイクポイントペインから削除することでも解除できます。この方法は、デバッグ終了後にすべてのブレイクポイントを解除する場合に便利です。

プログラムの実行中に、ブレイクポイントを設定することもできます。プログラムは読み込み専用モードでオープンされるためプログラム自体の変更はできませんが、ブレイクポイントの設定／解除は可能です。

さらに、以下に示すように、ブレイクポイントペインでブレイクポイントの処理を修正することができます。

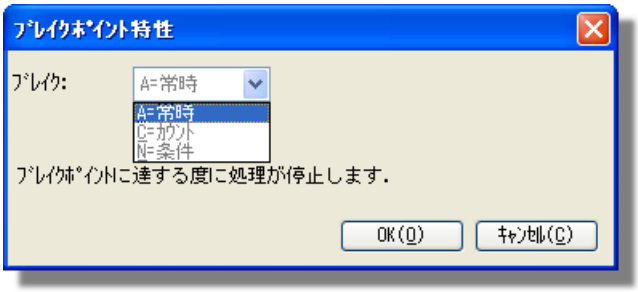
ブレイクポイントを切り替える



設定されたブレイクポイントは、ブレイクポイントペインに表示されます。ここでは、色々な方法で設定内容を修正することができます。

- 有効のチェックボックスをチェック／アンチェックすることで、ブレイクポイントの有効／無効が切り替わります。 ですべて有効になり、 ですべて無効になります。
- コンテキストメニュー（または *F3* の押下、または アイコンのクリック）から削除を選択することでブレイクポイントを削除できます。 をクリックするとすべて削除されます。

ブレイクポイント特性を設定する



さらに、ブレイクポイント特性を使用して、ブレイク処理を微調整することができます。

1. 修正したいブレイクポイント上にカーソルを置きます。
2. コンテキストメニューから特性を選択します（または のクリックか *Alt+Enter* の押下）。ブレイクポイント特性が表示されます。
3. ブレイク特性で、*A= 常時*、*C= カウント*、または *N= 条件*を選択します。
   - *A= 常時*を選択した場合、その他は何もする必要はありません。このポイントを通過した場合は、必ず停止します。
   - *C= カウント*を選択した場合、停止させたい繰返し数を入力してください。例えば、1,000 回ループする処理がある場合、カウントを 300 に設定することで、最初の 300 回の繰返し後に処理が停止します。
   - *N= 条件*を選択した場合、条件特性でズームしてブレイクを有効にさせる条件を設定することができます。条件が True として評価されると、ブレイクポイントで処理が停止します。

ソースに移動する

ブレイクポイントペインでコンテキストメニューからソースに移動を選択すると、ブレイクしている場所に位置付けることができます。

データビューエディタでの設定

データビューエディタ上に定義された項目にブレイクポイントを設定した場合、その項目の値が変更されると、デバッガは処理を停止させます。

ブレイクポイントを設定する

ブレイクポイントを設定するには、以下のようにします。

1. データビューエディタを開き、停止させたいデータ項目の定義行に移動します。
2. *F9*（デバッグ→ブレイクポイント）を押下します。

データ項目の更新によるブレイクポイントは、処理コマンドの場合と同じようにブレイクポイントリストに表示されます。

項目にウォッチに設定する

これは、データ項目の内容を監視するための簡易リストです。ウォッチリストには項目を追加することができます。データ項目が更新されブレイクが発生すると、項目の内容を参照することができます。ウォッチリストに項目を追加するには以下のようにします。

1. データビューエディタを開き、追加したい項目（カラムか変数／パラメータ）にカーソルを置きます。
2. デバッグ→ウォッチに追加（または、コンテキストメニューからウォッチに追加）を選択します。

処理コマンドに定義した時のようにブレイクポイントを設定しても、項目の隣に赤いドットは表示されませんが、ウォッチペインに項目リストが表示されます。ウォッチペインに表示される項目は、項目ペインにも表示されますが、指定された項目だけが表示されるので項目ペインより便利な場合があります。

ウォッチペインでは以下の処理が可能です。

- 項目名を削除することでウォッチを解除できます（または  をクリックするか、*F3* をクリックするか、コンテキストメニューから行削除を選択します）。
- 項目の値を別の値に変更できます（コンテキストメニューからデータの設定を選択するか、  をクリックします）。
- 設定可能であれば、NULL 値を設定できます（コンテキストメニューからデータに Null を設定を選択するか、  をクリックします）。
- ソースコードのデータ定義行に移動できますコンテキストメニューからソースに移動を選択するか、  をクリックします）。

ブレイクの使用

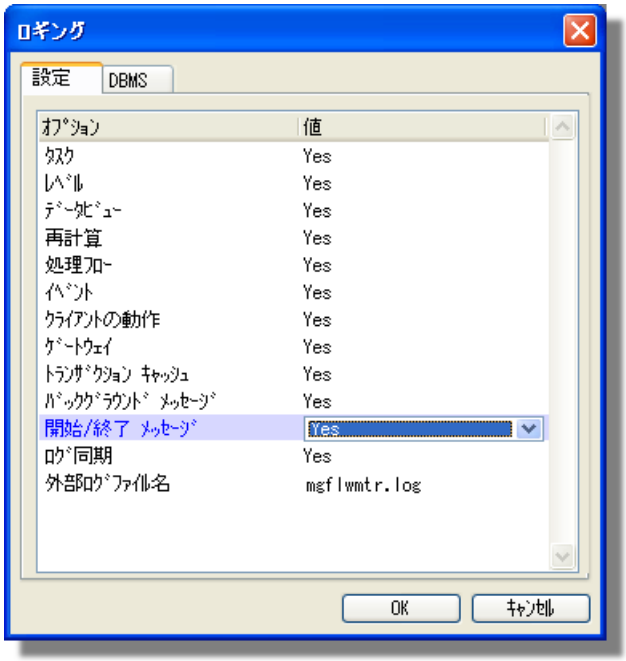
ブレイクポイントやウォッチが設定されていない場合でも、項目の内容やコールスタックを参照するためにブレイクを使用することができます。（特にオンラインの）プログラムの開発時、それをテストしていて予期しない結果が返るような場合、この機能は便利です。項目やコンテキスト、およびコールスタックはブレイクしている場合のみ参照できるため、この機能を利用することで内容を確認することができます。

ブレイクを使用するには以下のようにします。

1. プログラムの実行中に開発ウィンドウを表示します。
2. *Alt+Ctrl+F7* を押下します（デバッグ→ブレイクを選択するか、ツールバーの  をクリックします）。

デバッガは、ブレイクポイントやウォッチが設定されている場合と同じように停止します。

デバッガによって記録される情報を制御するには



ロギングテーブル（オプション→設定→ロギング）の設定を変更することで、デバッガによって記録される情報を制御することができます。

これらの設定は、デバッガだけでなく出力されるログファイルの内容にも影響します。

Logging() 関数を使用する

Logging() 関数を使用してログ出力の有効／無効を切り替えることもできます。この関数は、Magic.ini のロギングの設定を変更しません。関数の構文は以下の通りです。

Logging(*start/stop*, *Filter***)**

パラメータ：

- *start/stop* …… ***'TRUE'LOG*** を指定するとログの出力が開始されます。***'FALSE'LOG*** を指定すると停止します。
- *Filter* …… 対象となるログ内容を指定するキーワード。

指定例は以下の通りです。

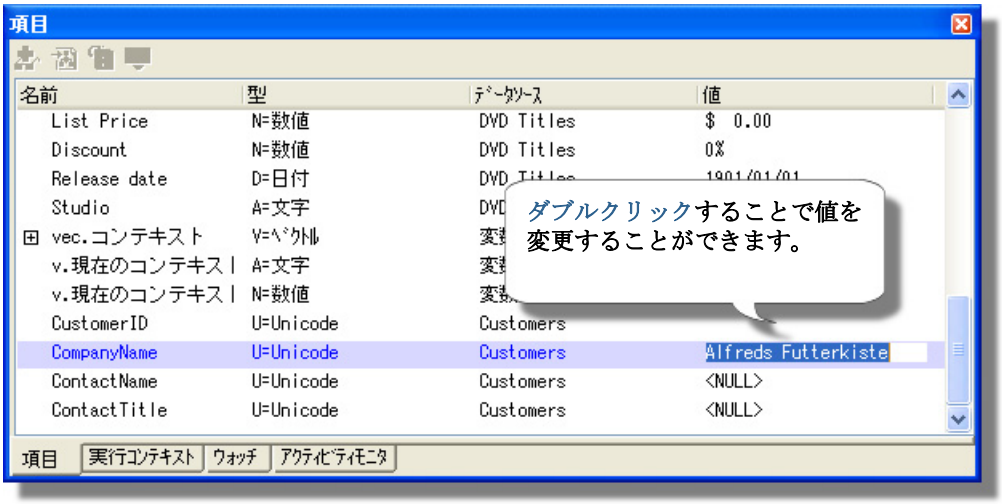
式の例	結果
Logging(*'TRUE'LOG, 'Levels'***)**	処理レベルに対するログ出力が開始されます。
Logging(*'FALSE'LOG, 'Recompute'***)**	再計算に対するログ出力が停止します。
Logging(*'FALSE'LOG, 'ALL'***)**	すべてのログが停止します。
Logging(*'TRUE'LOG, 'RESET'***)**	Magic.ini に格納された値に戻します。
Logging(*'TRUE'LOG, 'Oracle=D'***)**	Oracle のログを開発者レベルに設定します。

構文についての詳細情報は、『リファレンスヘルプ』を参照してください。

パフォーマンスの問題

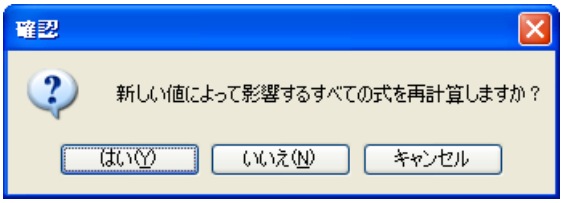
Logging() 関数を使用した場合、キャビネットファイルの作成前に無効になるように設定されているかどうかを確認してください。ログ出力機能が有効な場合、プログラム実行のパフォーマンスが低下します。

デバッグ中にデータを操作するには



デバッガが動作している間は、データ項目の内容を変更することができます。項目またはウォッチペインのどちらかで可能です。

1. 変更したい項目が参照できる処理で停止できるようにブレイクポイントを設定します。
2. 変更したい項目を見つけるために、項目ペインかウォッチペインに移動します。
3. 項目カラム上でダブルクリックするか、コンテキストメニューのデータを設定を選択するか、  をクリックします。
4. 変更された項目が再計算対象の場合、「新しい値によって影響するすべての式を再計算しますか?」という確認メッセージが表示されます。再計算を行う場合は、はいをクリックします。



コンポーネントをデバッグするには

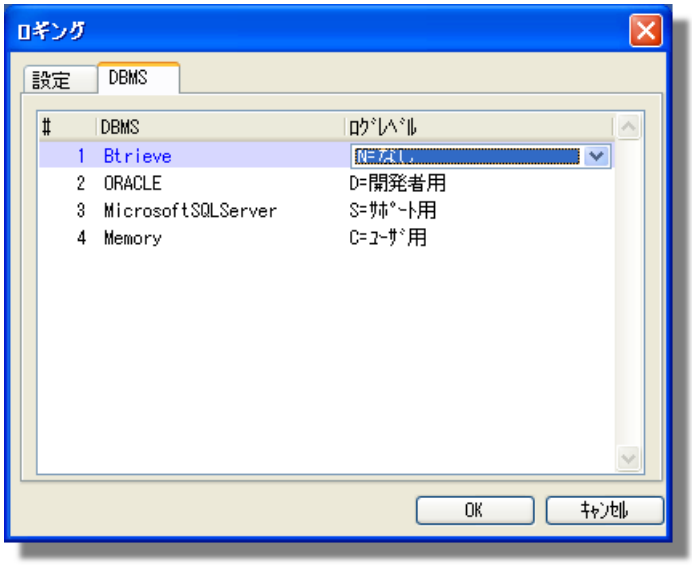
コンポーネントが現在のプログラムのモジュールとして定義されている場合、現在のプロジェクトと同じようにコンポーネントプログラムをデバッグすることができます。

モジュールを現在のプロジェクトに追加するには以下のようにします。

1. **プロジェクト→モジュール追加**を選択します。**プロジェクトを選択**ダイアログが表示されます。
2. 追加したいプロジェクトファイル（.edp ファイル）を選択します。

ナビゲータペインのモジュールツリーにコンポーネントが追加されたら、モジュール名を**クリック**してプロジェクトをオープンします。処理を停止させたい場所に移動してブレイクポイントを設定します。ホストプロジェクトと同じように**ブレイクポイントリスト**に**ブレイクポイント**が表示されます。デバッグ中にコンポーネントに定義された**ブレイクポイント**に達すると、現在のタスクが閉じコンポーネントが開きます。そしてコンポーネント内での実行内容が確認できます。

データベースの動作を記録するには



使用する DBMS へのアクセスを記録するログのオプションもあります。このログを有効にするには以下のようにします。

1. ロギングテーブル（オプション→設定→ロギング）を開きます。
2. DBMS タブをクリックします。
3. 記録したい DBMS にカーソルを置きます。
4. ログレベルを *N= なし* 以外に選択します。ログのレベルは *C= ユーザ用*、*S= サポート用*、*D= 開発者用* の 3 種類あります。*C= ユーザ用* は最も短く最も簡単なログレベルで、*D= 開発者用* は最も詳細なログレベルです。

ログレベル

ログレベル	動作内容
N= なし	ログは出力されません。
C= ユーザ用	最も簡単な内容のログが出力されます。SQL コマンドのみが出力されます。
S= サポート用	中程度の内容のログが出力されます。
D= 開発者用	詳細な内容のログが出力されます。

DBMS のロギング

Magic のロギング機能にに加えて、使用している DBMS に組み込まれているロギング機能を利用することも有効です。機能の内容は DBMS によって変わりますが、何らかのロギングツールが用意されています。

パフォーマンスの問題

ロギング機能を使用している場合、キャビネットファイルの作成前に無効になるように設定されているかどうかを確認してください。ログ出力機能が有効な場合、プログラム実行のパフォーマンスが低下します。

リモートアプリケーションをデバッグするには

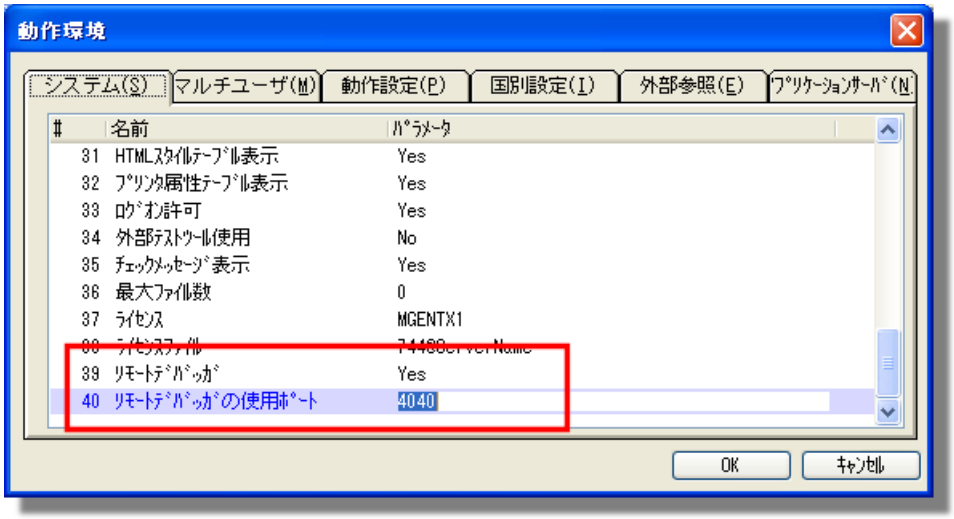
リモートデバッガを利用することで、リモート PC 上で実行しているアプリケーションのデバッグを行うことができます。この場合、リモート側とローカル側は以下のような構成にする必要があります。

- リモート PC：**Magic Client** または、**Magic Enterprise Server** でアプリケーションを実行
- クライアント PC：**Magic Studio** で、リモート側で実行している同じアプリケーションのプロジェクトファイルをオープンする。

リモートデバッガを有効にする

最初にリモート PC 側でリモートデバッガを有効にする必要があります。

1. サーバ PC 側の Magic で、**動作環境**ダイアログ（**オプション→設定→動作環境**）を開き、**システム**タブの**リモートデバッガ**を ***Yes*** に設定し、**リモートデバッガの使用ポート**に TCP のポート番号を指定します。ポート番号は、他に使用されていない番号を指定してください（Windows の netstat コマンドなどで確認できます）。

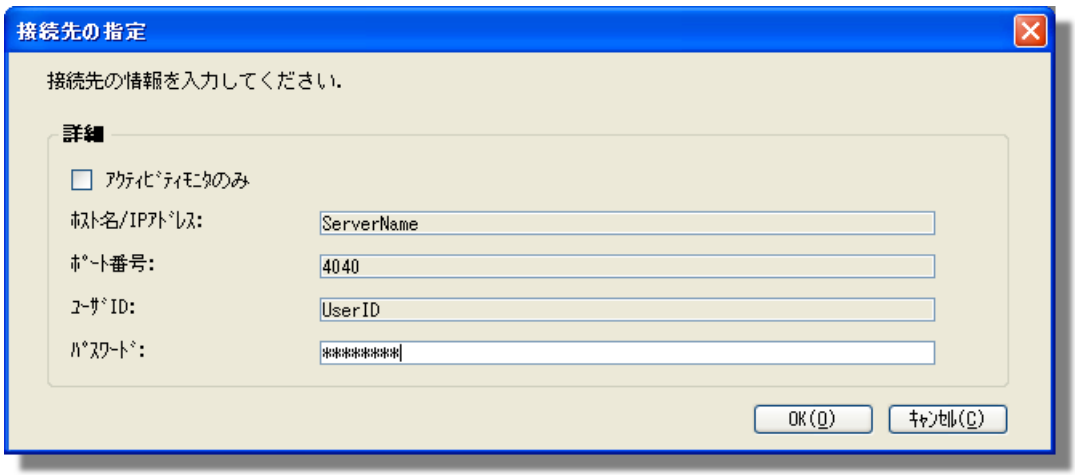


2. Magic エンジンを再起動し、アプリケーション（ecf）をオープンします。

リモートアプリケーションに接続する

次に、クライアント PC 側でリモートアプリケーションに接続します。

1. クライアント PC 側で **Magic Studio** を起動し、リモート側で実行しているアプリケーションのもとになるプロジェクトをオープンします。
2. **デバッグ→リモートエンジンに接続**を選択し、**接続先の指定**ダイアログを開きます。



- **ホスト名 /IP アドレス …… リモート PC のホスト名か IP アドレスを指定します。**
- **ポート番号 …… リモート PC 側の動作環境ダイアログで設定したポート番号**

- ユーザ ID …… リモート PC 側の Magic 環境で登録されているユーザ ID を指定します。この ID は、アプリケーションをアクセスする権利やリモートデバッガにアクセスする権利も持っている必要があります。
- パスワード …… ユーザ ID に対応したパスワードを指定します。
- アクティビティモニタのみ …… 通常は、リモート側のアプリケーションファイルとローカル側のプロジェクトファイルは同期がとれている必要がありますが、ここをチェックすることでアクティビティモニタ表示のみサポートすることでプロジェクトが異なる状態でも実行できるように鳴ります。

全てのパラメータを設置し、OK をクリックすると通常のデバッガと同じようにモニタウィンドウにデバッグ情報が表示されるようになります。

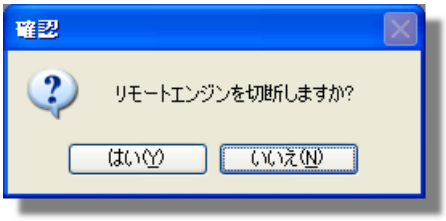
リモートアプリケーションを切断する

リモートアプリケーションのモニタを終了させる方法は、２あります。

- リモート側でアプリケーションを終了させる。
- クライアント側で切断処理を行う。

クライアント側で切断処理を行う

1. クライアント PC 側で**デバッグ→リモートエンジンを切断**を選択します。**確認**ダイアログを開きます。はいをクリックするとリモート側との接続が終了します。



［このページは意図的に空白にしています。］

第 30 章 : イベントとハンドラ

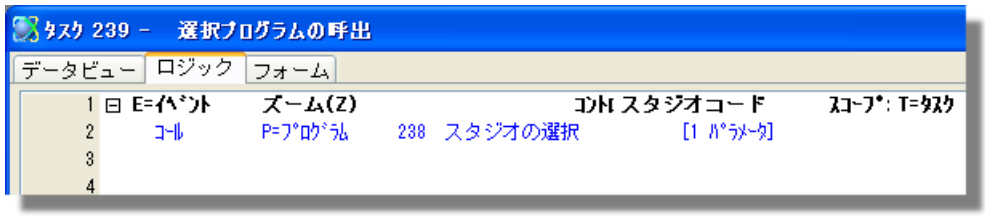
注 : V9Plus 以前における「イベントハンドラ」は、Magic V10 では「ロジックユニット」という名称になりました。ただし、ロジックユニットを実行する上での内部機構をこのドキュメントではイベントハンドラ（またはハンドラ）という表記で説明しています。また、処理レベル（レコード前／後、タスク前／後、等）と区別するためにも、この章ではイベントハンドラという表記を使用しています。

イベントによって起動された選択プログラムの戻り値でデータ項目を再表示するには

ハンドラ（ロジックユニット）で処理される場合、ハンドラを抜けるまで項目の値は更新されません。これは、フォームに配置されているデータ項目がハンドラ内で更新された場合、ユーザが表示項目を抜けるまで更新された値が画面上に表示されないことを意味しています。

これは、選択リストを使用する場合に特に問題になります。一般的に、選択リストは表示項目からズームすることでアクセスされます。ここでは、このようなズーム処理を実現する場合の定義方法について説明します。

問題点 : 項目が再表示されない



最初に、正しく動作しないズーム処理の例を示します。ユーザがスタジオコードコントロールでパークしている時にズームすると、スタジオの選択プログラムが起動され、スタジオコードを選択できるようになっています。しかし、カーソルが次のコントロールに移動するまで項目は空白の状態で表示されるはずです。

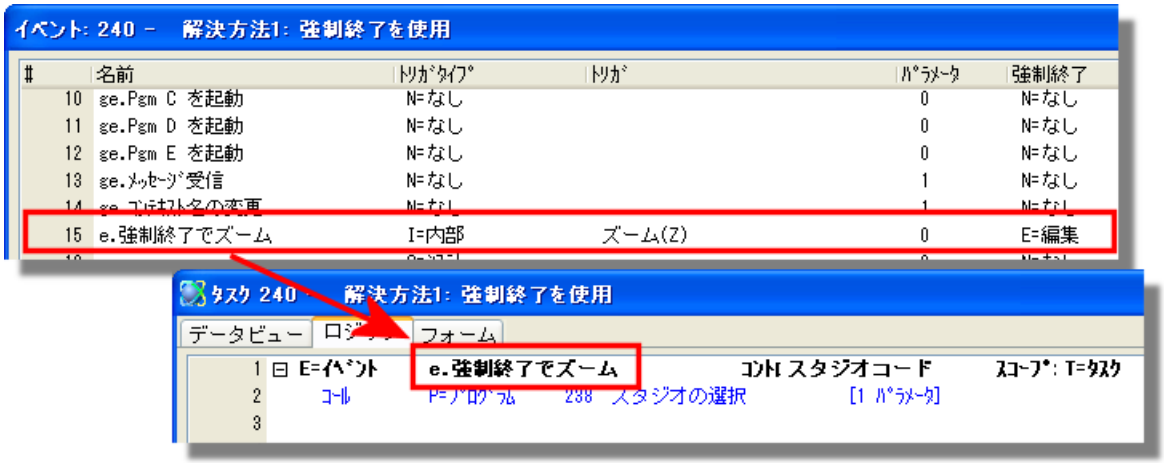
解決方法 :

更新された項目が再表示されない問題点の解決方法には、以下の 3 つの方法が考えられます。

- イベントの**強制終了**特性を使用する
- **コントロール特性**の**選択プログラム**特性を使用する
- **コンボボックス**を使用する

以下、これらの詳細について説明します。

解決方法 1: イベントの強制終了特性を使用する



項目を更新する場合に発生する問題点の解決方法は以下の通りです。

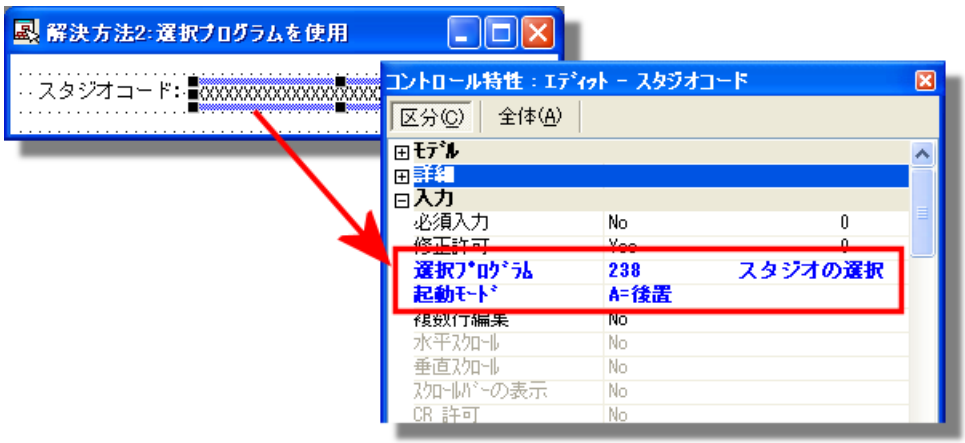
1. 新しいユーザイベントを作成します（この例では、e. 強制終了でズームという名前で**強制終了**特性を設定しています)。このイベントは、**ズーム**の内部イベントが割り当てられているため、**ズーム**と同じよう動作します。**強制終了**特性には、図で示されているように、***E= 編集***に設定されています。これは、表示項目を再表示させるように定義するものです。
2. 新しいユーザイベントを内部イベントの**ズーム**の代わりに使用します。

メインプログラム内でこの新しいイベントを作成した場合、アプリケーションのグローバルイベントとして使用できます。

注： 通常、ズーム項目にパークしている場合、ステータスバーに「ズーム」という文字が表示されます。上記の方法でロジックを定義した場合、このイベントはズームをトリガとして定義されているため、同じように「ズーム」が表示されます。

ヒント：*後置の ズーム処理を定義したい場合、次項目 の内部イベントを発行するイベント実行処理コマンドを追加してください。*

解決方法 2: 選択プログラム特性を使用する



別の方法として、イベントを使用しないで選択リストを起動させることができます。この場合、表示項目に**選択プログラム**特性を定義するだけで対応できます。

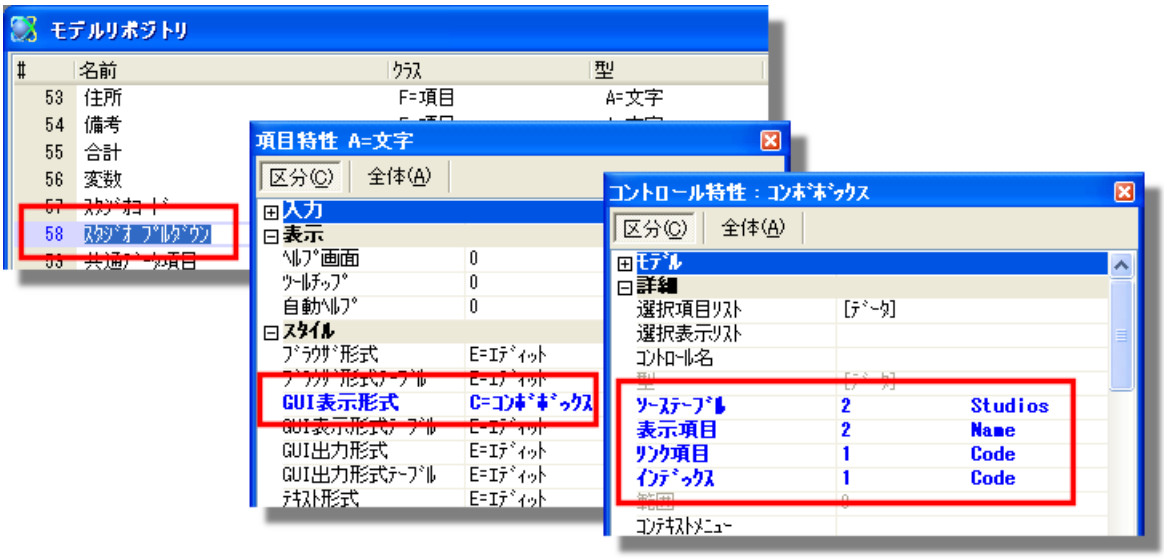
選択プログラム特性は、モデルやフォーム上のコントロールに対して定義できます。この特性が定義されると**ズーム**が有効になり、指定された選択プログラムを起動することができるようになります。

この方法は、タスク毎にロジックを定義する必要がなくなる点で便利です。この例では、**項目モデル**に対して**選択プログラム**特性で**スタジオの選択**プログラムが定義されています。プログラム側でこのモデルとの継承関係を解除しない限り、常にこのプログラムが起動されるようになります。

この方法の不便な点は、1 つの値しか返らないということです。この例では、**スタジオコード**のみ選択していますが、**スタジオコード**と**スタジオタイプ**の両方を取得する必要がある場合、イベントを使用することになります。

解決方法 3: コンボボックスを使用する

Magic では、複数の項目から選択するための方法として**データコントロール**（**コンボボックス**、**リストボックス**、**タブ**）を使用することもできます。この場合も、モデルで定義することができるので、プログラム毎に定義する必要はありません。以下の例では、**スタジオプルダウン**という名前のモデルを作成しています（これは、フォームにはコンボボックスとして配置され、スタジオテーブルと自動的に連動するようになっています）。


モデルリポジトリ
名前 クラス 型
53 住所 F=項目 A=文字
54 備考
55 合計
56 変数
58 スタジオ プルダウン
項目特性 A=文字
区分(C) 全体(A)
入力
表示
ヘルプ画面 0
ツールチップ 0
自動ヘルプ 0
スタイル
ブラウザ形式 E=エディット
GUI表示形式 C=コンボボックス
GUI出力形式 E=エディット
GUI出力形式テーブル E=エディット
テキスト形式 E=エディット
コントロール特性：コンボボックス
区分(C) 全体(A)
モデル
詳細
選択項目リスト [データ]
選択表示リスト
コントロール名
ソーステーブル 2 Studios
表示項目 2 Name
リンク項目 1 Code
インデックス 1 Code
コンテキストメニュー

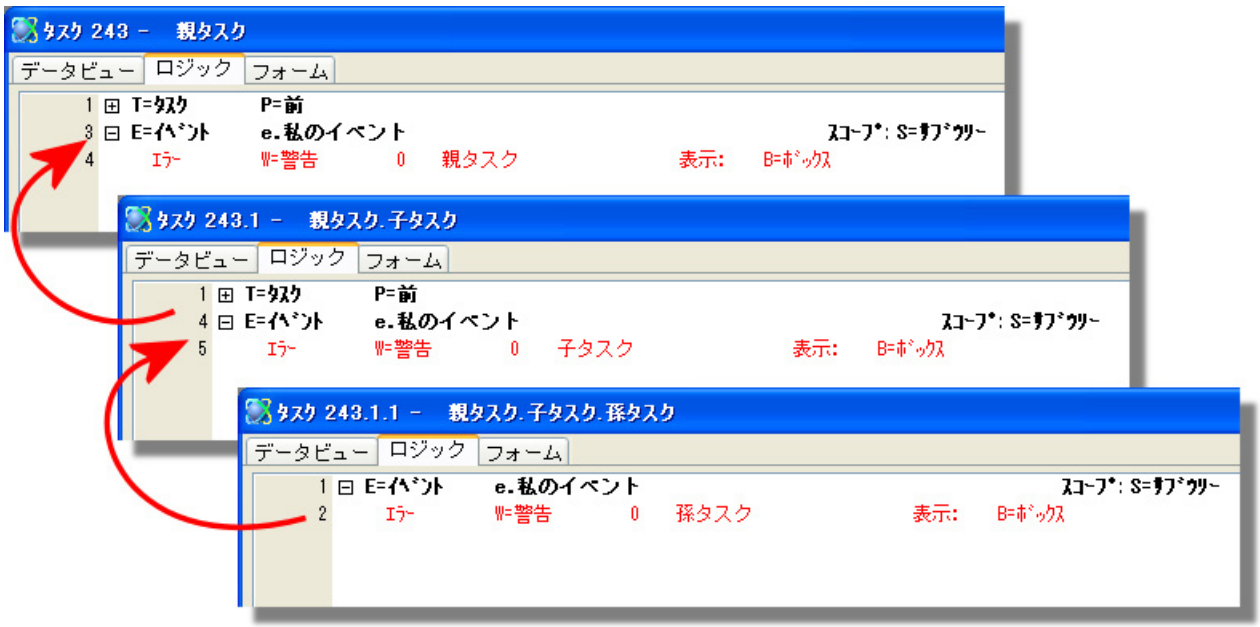
確定されていない更新値で強制的に更新するには

ハンドラで処理される場合、ハンドラを抜けるまで項目の値は更新されません。これは、フォームに配置されているデータ項目がハンドラ内で更新された場合、ユーザが表示項目を抜けるまで更新された値が画面上に表示されないことを意味しています。

別のケースとして、ハンドラ内の処理が完了する前にレコードを更新したい場合もあります。例えば、まだ編集中の受注データを印刷するためにプログラムを起動する例を考えてみます。この場合、プログラム起動時にレコードへの書き込みが行われなければなりません。

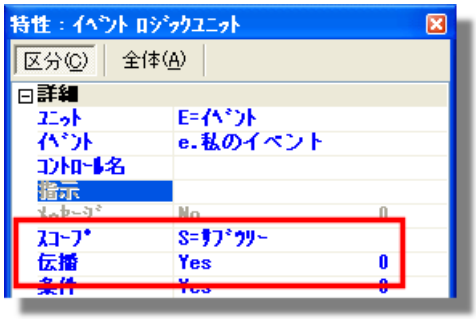
このような処理は、**イベント**テーブルの**強制終了**カラムを使用することで対応できます。このパラメータの詳細は、『リファレンスヘルプ』を参照してください。また動作の違いを理解するために、パラメータの設定を変えて、デバッガを使用して確認してみてください。**強制終了**を ***E= 編集***に設定することで、項目の更新処理の問題の解決に役立ちます。

同一イベントで複数のハンドラを実行させるには



タスク内で発行されるイベントは、上位に伝播させることができます。イベントが子タスクで発行された場合、メインプログラムを含めたすべての上位タスクで処理させることができます。

イベントの伝播を許可する



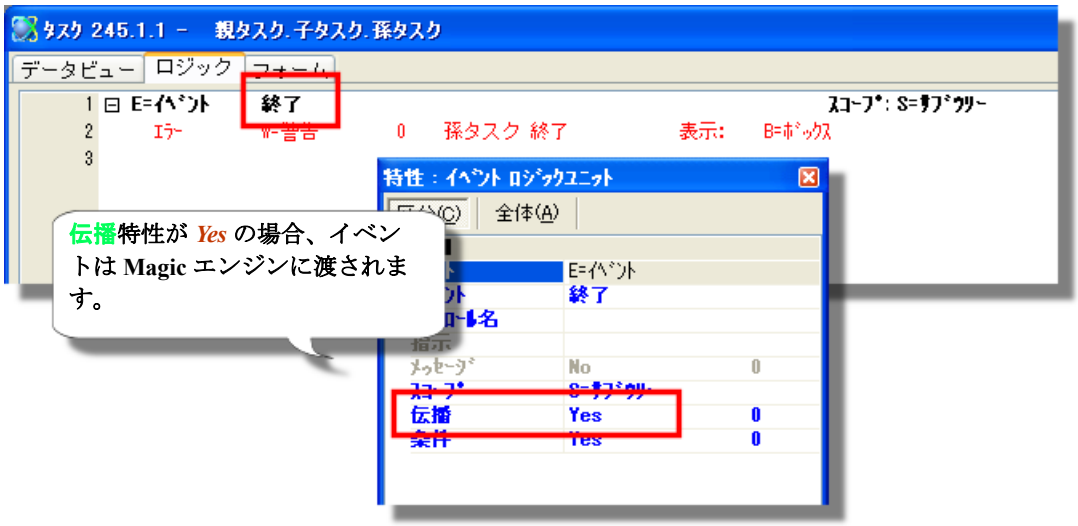
イベントツリー内のすべてのロジックユニット特性を設定することで、発行されたイベントの処理を伝播させるかどうかを制御できます。

1. 伝播特性を *Yes* に設定します（'TRUE'LOG と評価される論理式でも指定できます）。
2. 最下位レベルのタスクでなければ、スコープ特性を *S= サブツリー* に設定します。

これでイベントは、1 つのタスクレベルで処理された後、次のレベル（上位タスク）に伝播されます。

これはすべてのユーザイベントで有効です。しかし、内部イベントの場合はイベントを再度発行する必要があるかもしれません。例えば、終了イベントが孫タスクで実行された場合、孫タスクが終了した時点で完了することになります。従って、次のタスク（子タスク）に対しても終了イベントを処理させたい場合、孫タスク内で別の終了イベントを発行させる必要があります。

ユーザハンドラによって実行された内部イベントに Magic の処理を定義するには

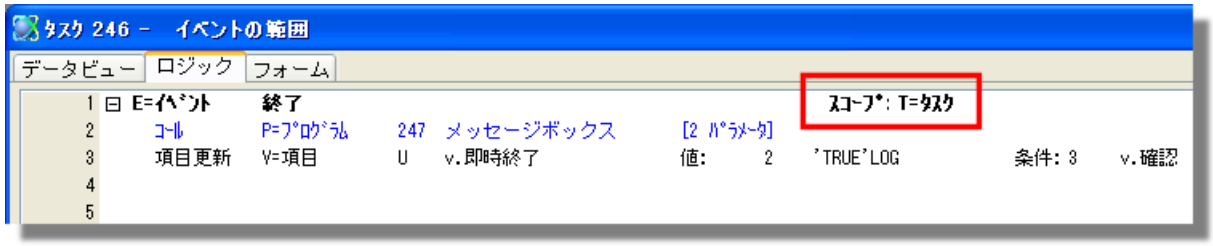


内部イベントによって実行される**ロジックユニット**を作成する場合、そのイベントを伝播させるかどうかを指定することができます。

この例では、*Esc* キーを押下するか、 をクリックしてタスクを終了させた場合、**終了**イベントが実行されます。

- **伝播**特性が *Yes* に設定されている場合、警告メッセージが表示され、**終了**イベントは Magic に渡されタスクが終了します。
- **伝播**特性が *No* に設定された場合、メッセージは表示されますが、イベントは Magic に渡されず、タスクはオープンされたままの状態になります。

イベントが定義されたタスクでのみ有効にするには

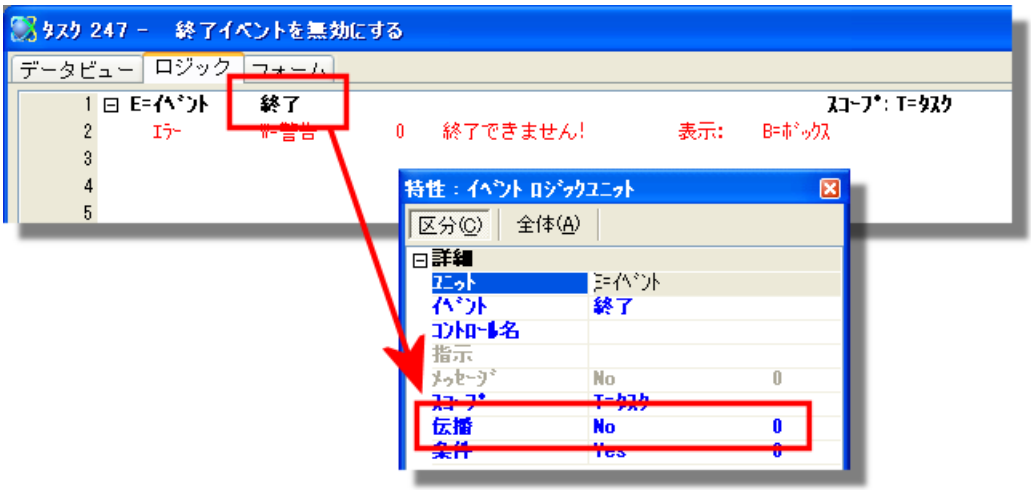


イベントを特定のタスク内でのみ有効にしたい場合があります。例えば、ユーザがある重要なタスクを終了する際に、警告メッセージを表示させる処理を考えてみます。しかし、そのタスクにはそれ程重要でない子タスクが定義されているかもしれません。この子タスクに対しては警告メッセージを表示させないようにします。

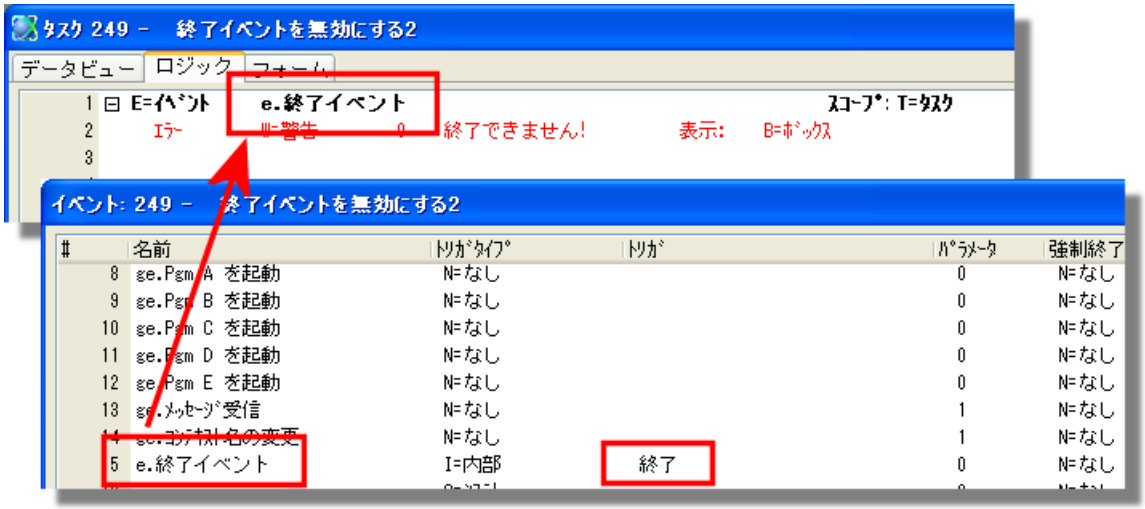
この例において、ユーザが ***Esc*** を押下した場合、「確認メッセージ」が表示されます。ユーザがはいを**クリック**した場合、項目 v. 即時有効がタスクを終了させるために更新されます。オリジナルの**終了**イベントは伝播されないため、タスクは終了しません。

イベントがこのタスク内でのみ処理されるようにするため、ハンドラの**スコープ**特性を ***T= タスク***に設定します。

内部イベントを処理しないようにするには



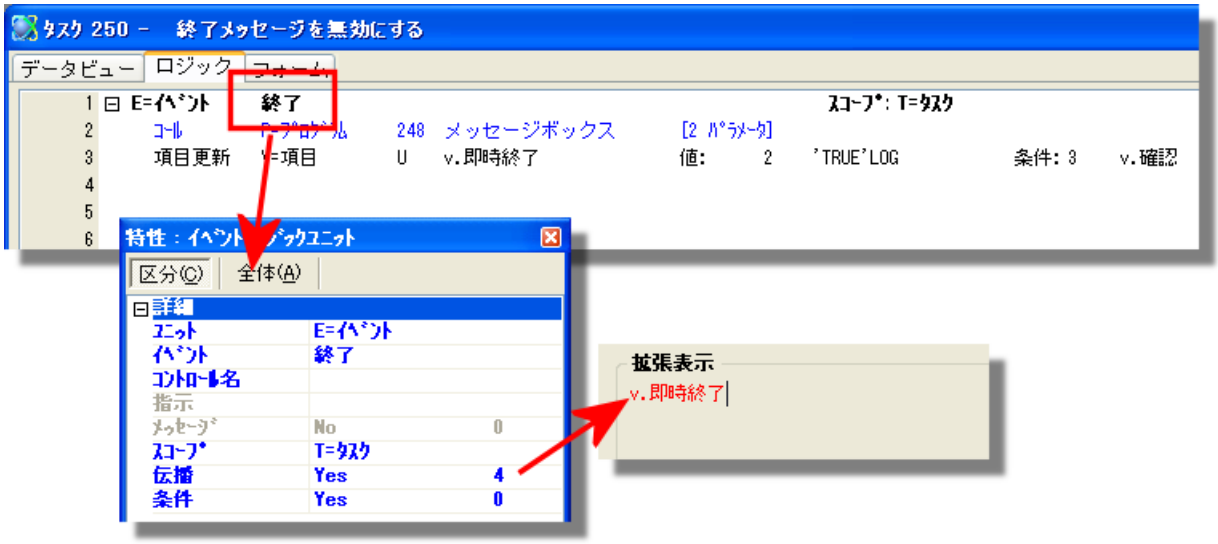
伝播特性を *No* に設定することによって内部イベントを処理しないようにすることができます。この例では、終了イベントがブロックされているため、ユーザが *Esc* を押下したり  をクリックしてもタスクは終了しません。(開発環境でテストする場合は、  をクリックするかデバッグ→停止を選択することで終了させることができます)。



内部イベントがユーザイベントのトリガとして使用されている場合もブロックされる場合があります。これによって全体的に終了処理がブロックされ、別のタスクのように（何もなかったように）動作します。

イベントの処理を特定の条件でのみ機能するようにしたい場合は、伝播特性を式で指定します。詳細は次の例で明します。

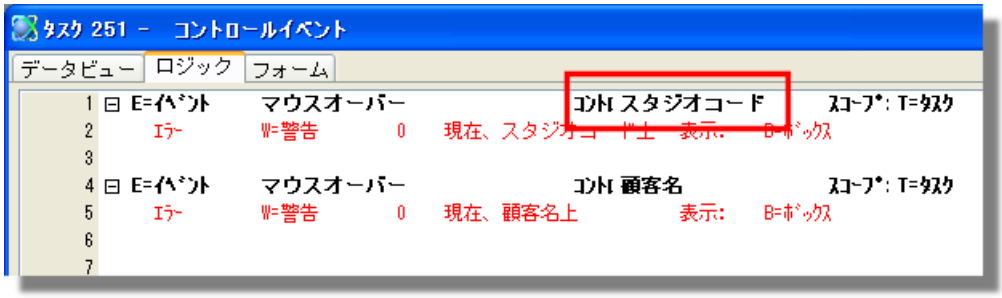
条件付きの伝播特性を使用する



伝播特性を式で指定することもできます。

この例では、終了イベントが発生したら「確認メッセージ」を表示させるようにしています。ユーザがはいをクリックしたら、変数項目 v. 即時終了を 'TRUE'LOG に更新します。この変数は、伝播特性に使用されています。このため、はいがクリックされると、終了イベントが Magic に渡されタスクが終了するようになります。

イベントの処理を特定のコントロールに限定するには



特定のコントロールでイベントが発生した場合にのみ処理したい場合があります。このような場合、コントロールを選択するには以下のようにします。

1. イベントロジックユニットのコントロールカラムに移動します。
2. ここからズームしてコントロール一覧を開きます。
3. 使用したいコントロール名を選択します。

コントロール名を直接入力して指定することもできます。イベントハンドラが親タスクかメインプログラム内に定義されている場合は、一覧から選択できません。このような場合は直接入力します。またこのようにすることで、コントロールがまだ定義されていなくても指定できます。

ヒント:*複数のコントロールに対して動作する汎用的なイベントを作成したい場合、ロジックユニットにコントロール名を割り当てることで対応できます。*

例えば、「注文」（通常は注文番号が格納されています）と名付けられたコントロールに対する複数のイベントロジックユニットを定義することができます。注文コントロールで F1 を押下すると入力された注文情報に対するヘルプ画面を表示させ、F2 を押下するとその注文の現在の状態を表示させ、Ctrl+P を押下すると注文内容が印刷させるようにできます。しかし、コントロール名として「顧客」と定義されている別のコントロールに対しては、同じショートカットキーで異なるプログラムを起動させることができます。

コンポーネント内に定義されたハンドラを使用して、ホストアプリケーションでイベントを処理するには

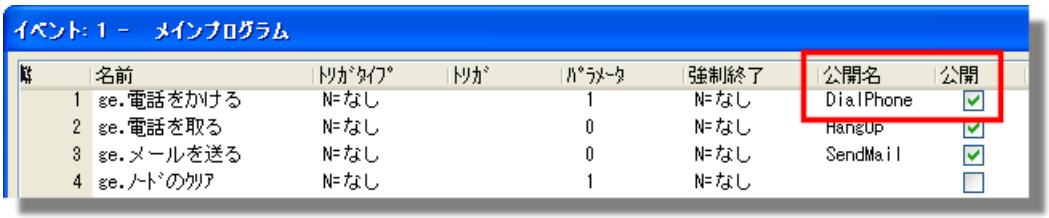
コンポーネント内で定義されたイベントを、ホストアプリケーションで実行させることができます。例えば、指定された電話番号で電話するイベントをコンポーネントで定義し、ユーザが電話番号フィールドでズームした場合、ホストアプリケーションでそのイベントを実行させるように定義できます。

このような処理を実現するには、以下の 3 つの手順が必要です。

1. コンポーネント内でイベントを定義します。
2. コンポーネント内でこのイベントに対するロジックユニットを定義します。
3. ホストプロジェクトでこのイベントを実行するように定義します。

これらの 3 つの手順の詳細を以下で説明します。

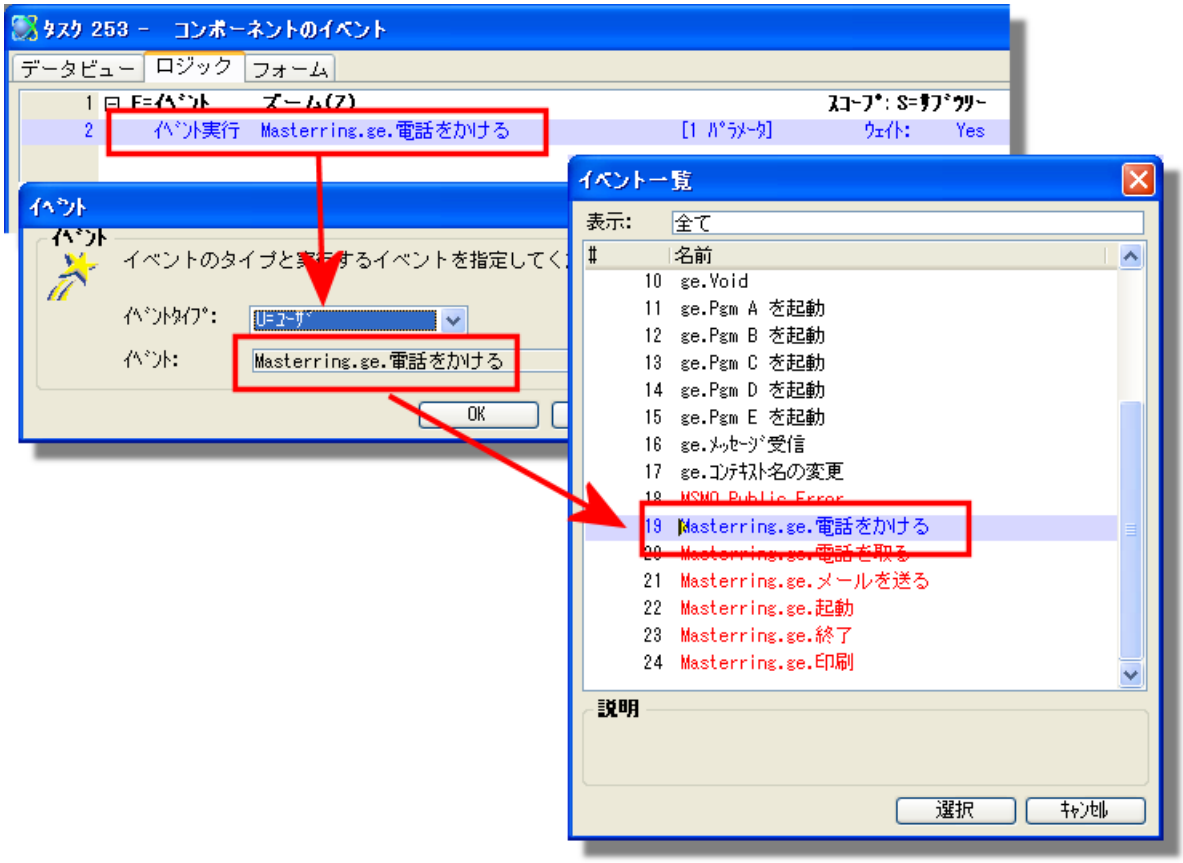
コンポーネント内でイベントを定義する



1. コンポーネントのメインプログラムでイベントを定義します。公開カラムがチェックされて、公開名カラムに名前が定義されていることを確認します。
2. インタフェースファイル（*.eci*）を作成する際に、ここで定義されたイベントを選択します。

これでコンポーネントを使用した場合、このイベントを選択することができます。

ホストアプリケーションでイベントを発行する



1. ホストアプリケーションのプログラムを開き、イベントを発行させたい場所に移動します。
2. 発行させるユーザイベントを選択します。コンポーネントイベントは、ホストアプリケーションで定義されたユーザイベントの下に赤色で表示されます。

コンポーネント内でイベントを処理する

1. コンポーネントのプロジェクトを開きます。
2. メインプログラムに移動します。
3. 定義されたイベントに対するイベントロジックユニットを作成します。スコープをグローバルに設定します。

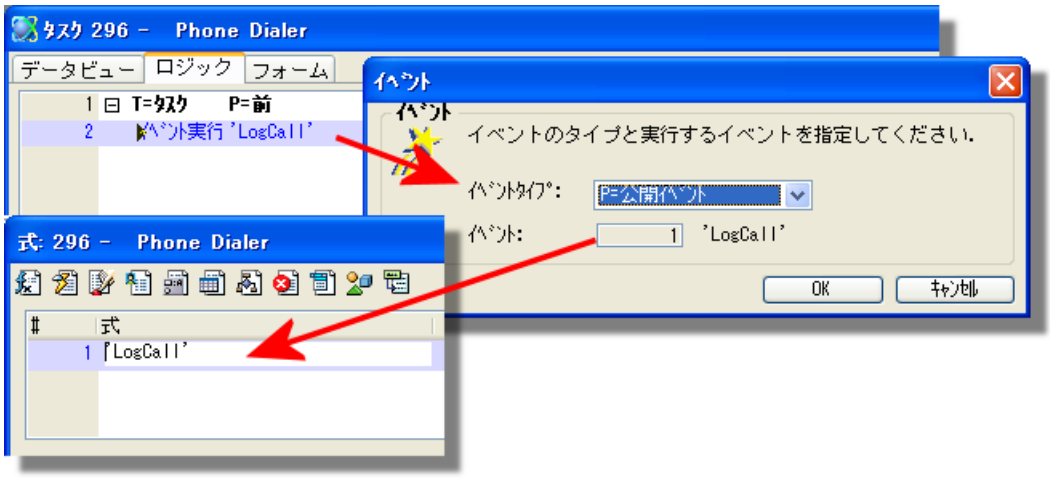
これで、ホストプログラムがイベントを発行すると、このロジックユニットが実行されます。

コンポーネントからホストアプリケーションで定義されたイベント実行するには

ホストアプリケーションで処理が可能なコンポーネント内でイベントを発行することができれば便利な場合があります。例えば、エラーが発生したら、パッケージ化されたモジュールによってイベントが発行されるようにすることが一般的に考えられます。

この例では、LogCall という名前のイベントをコンポーネント内で発行しています。このコンポーネントは、呼び出す際にパラメータを渡し、必要であればホストアプリケーションの情報を格納できるようになっています。

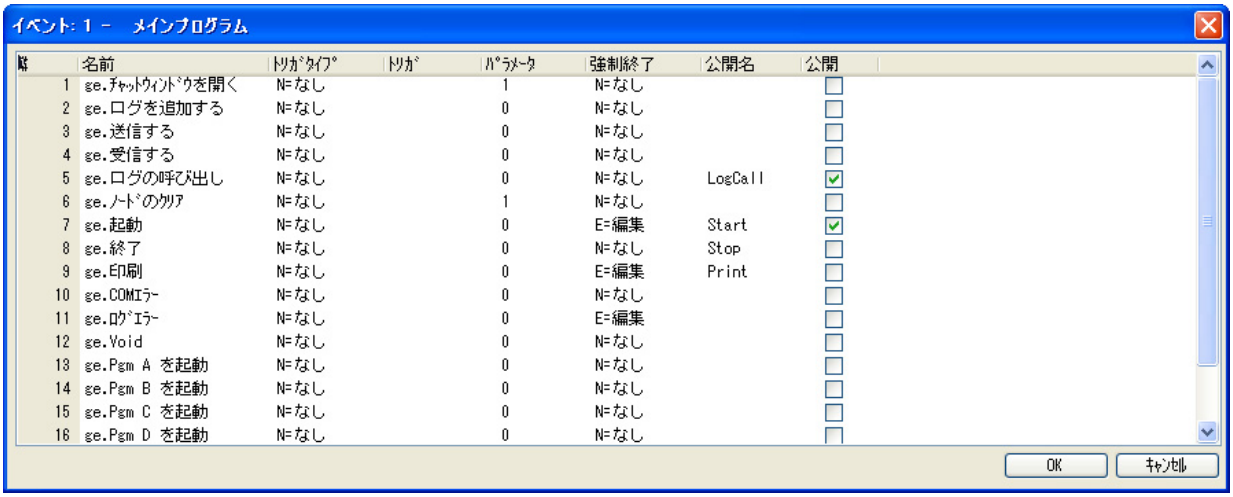
ホストイベントを発行する



1. 必要な場所にイベント実行処理コマンドを定義します。イベントダイアログが表示されます。
2. イベントタイプで *P= 公開イベント*を選択します。
3. イベントに移動します。ここからズームして式エディタを開きます。発行するイベント名を正確に指定します（イベント名は、シングルクォーテーションで囲みます）。

これにより、ホストアプリケーションによって処理できるイベントをコンポーネントが発行するようになります。次に、イベントを受け取るためのホストアプリケーションを設定する必要があります。

ホストアプリケーションでイベントを処理する



1. メインプログラム内でイベントを作成します。公開名カラムには、コンポーネントで定義されたイベント名と同じ名前を指定します。
2. 公開カラムのチェックボックスをチェックします。

これで、必要に応じてイベントを処理するロジックユニットを作成することができます。

イベントを即時に処理させるには

イベントを即時に処理するようにしたい場合、**ウェイト**特性を *Yes* に設定します。この場合、他のコマンドは処理されず直ちにイベントが処理されます。これは処理コマンドの定義順に処理されるため、同期処理と呼ばれます。

例えば、上記のように 4 つのイベント処理コマンドが定義されている場合、以下のように動作します。

1. イベント A は、**ウェイト** =*No* で発行されます。この場合、イベント A はキューに入りますが、即時には実行されません。次の処理コマンドが処理されます。
2. イベント B は、**ウェイト** =*No* で発行されます。この場合、イベント B はキューに入りますが、即時には実行されません。次の処理コマンドが処理されます。
3. イベント C は、**ウェイト** =*Yes* で発行されます。この場合、次のコマンドが処理されず、イベント C が直ちに実行されます。
4. イベント D は、**ウェイト** =*No* で発行されます。この場合、イベント D はキューに入りますが、即時には実行されません。次の処理コマンドが処理されます。
5. 次のコマンドは、E を表示するエラー処理コマンドです。これは即実行されます。
6. キューに入ったイベント A、B および D が実行されます。

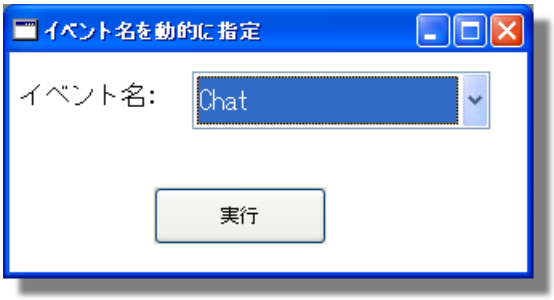
これにより、メッセージボックスが以下の順番で表示されます。C、E、A、B、D

イベントの処理を延期させるには

イベントを後で処理させたい場合、**ウェイト**特性を *No* に設定します。この場合、他のコマンドが処理された後でイベントが実行されます。イベントは割り込みを許すため、処理コマンドの定義順に実行されない場合があります。これを非同期処理と呼びます。

ウェイト特性の動作に関する詳細は、「イベントを即時に処理させるには」（635 ページ）」を参照してください。

イベント名を動的に指定してイベントを発行するには



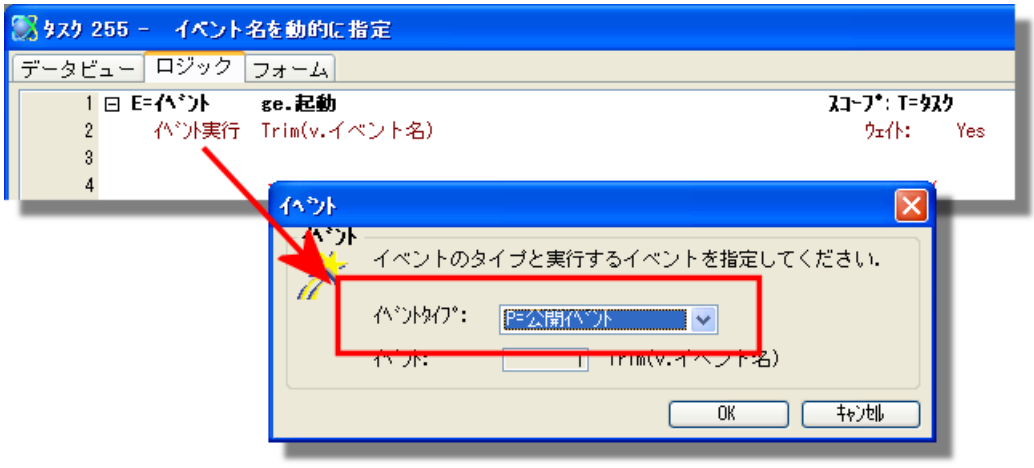
イベント名を式で動的に指定することもできます。例えば、コンポーネントからホストアプリケーションに定義されているイベントを発行したい場合、イベント名が直接参照できないため選択することができません。また、ユーザが入力したり、テーブルに格納されたイベント名を選択することでそのイベント名に応じて発行するイベントを切り替えたりする方法も必要です。

この例では、コンボボックスからイベントを選択することで、選択されたイベントが発行されるようにしています。

メインプログラム内でイベントを設定する

最初にメインプログラム内でイベントを設定し、それに公開名を定義する必要があります。この例では、以下のイベントが公開定義されています。Chat, Send, Receive, LogCall.

イベントを公開名で発行する



次に、イベントを発行する処理を定義します。

1. イベントを発行させたい場所に移動します。
2. *R* を入力し、イベント実行処理コマンドを作成します。イベントダイアログが表示されます。
3. イベントタイプで *P= 公開イベント* を選択します。
4. イベントに移動します。ここからズームして式エディタを開きます。発行するイベント名が返る式を定義します。(イベント名は、大文字小文字を区別しません。また空白は入れないでください。)

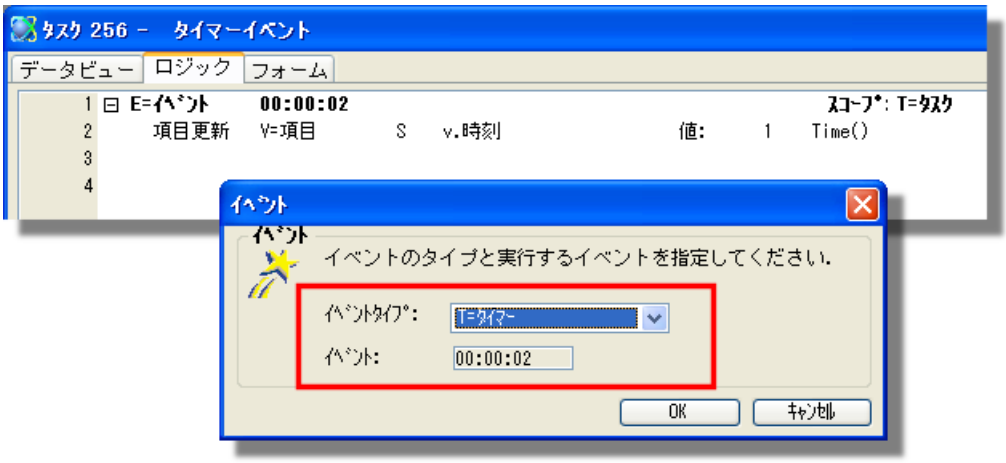
この例では、コンボボックスからイベント名を選択しています。イベント名は空白が削除されて発行されます。

一定時間の経過後に処理を実行させるは

一定の時間が経過後に処理が実行されるようにすることができます。例えば、数分ごとにメールのキューを確認するような処理などが考えられます。

Magic では、このような場合タイマーイベントを使用して実現できます。この例では、２秒ごとに画面上の時刻表示を更新するようにしています。

タイマーイベントを使用する



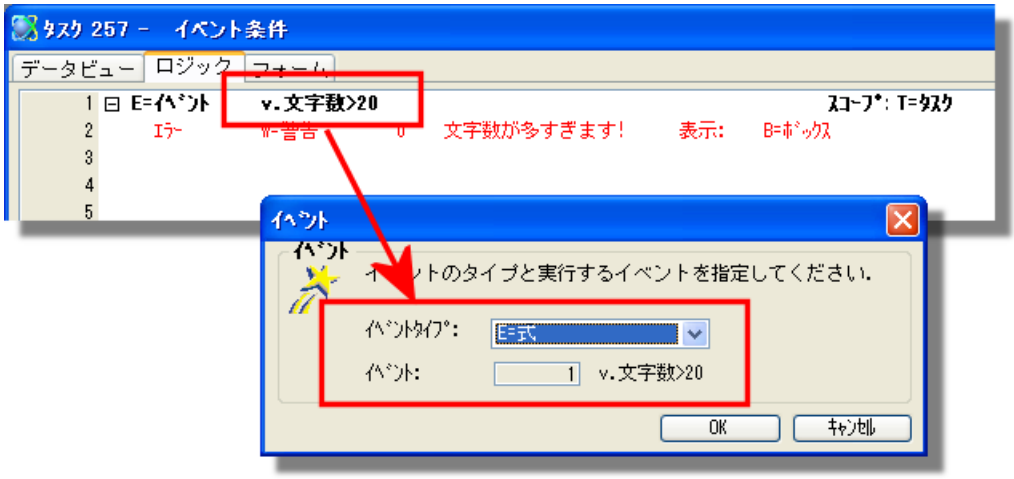
1. イベントを発行させたい場所に移動します。
2. *R* を入力し、イベント実行処理コマンドを作成します。イベントダイアログが表示されます。
3. イベントタイプで *T= タイマー*を選択します。
4. イベントに移動します。イベントを発行する時間間隔を入力します。書式は、HH：MM：SS です。この例では、00:00:02 と入力されているため、イベントは 2 秒ごとに発行されます。

これで、指定された時間間隔に従ってイベントが発行されます。

条件が満たされた場合に処理が実行されるようにするには

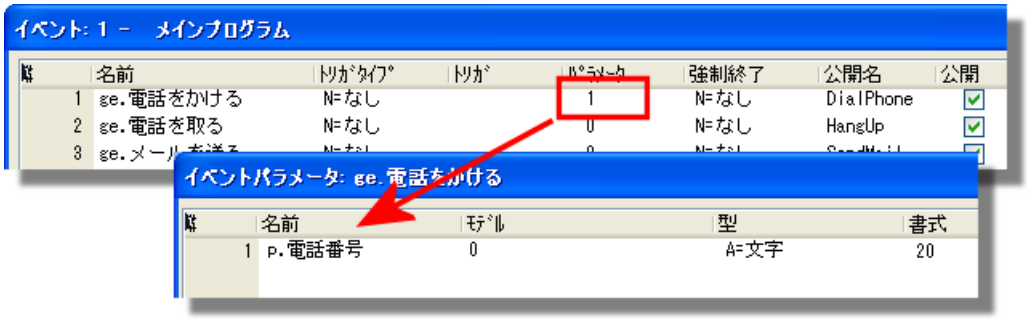
特定の条件にもとづいて処理が実行されるようにイベントを発行することができます。例えば、注文合計が大き過ぎたり電子メールが到着した場合にメッセージを表示させる処理が考えられます。これらのイベントは、式イベントとして定義できます。

式イベントを作成する



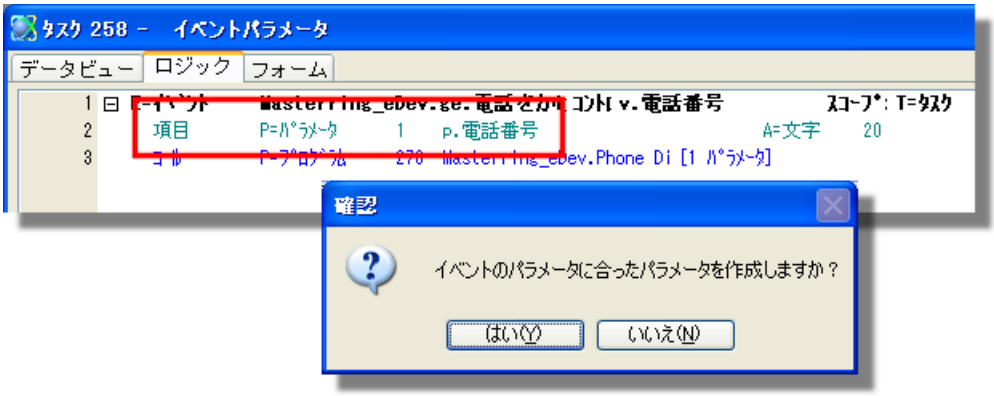
1. イベントを発行させたい場所に移動します。
2. *R* を入力し、イベント実行処理コマンドを作成します。イベントダイアログが表示されます。
3. イベントタイプ欄で*式*を選択します。
4. イベント欄に移動します。ここからズームして式エディタを開きます。イベントを発行する条件を定義します。式が 'TRUE'LOG と評価された場合にイベントは発行されます。この例では、ユーザが 20 桁を越える文字が入力された場合にイベントが発行されます。

パラメータ付きのイベントを作成する



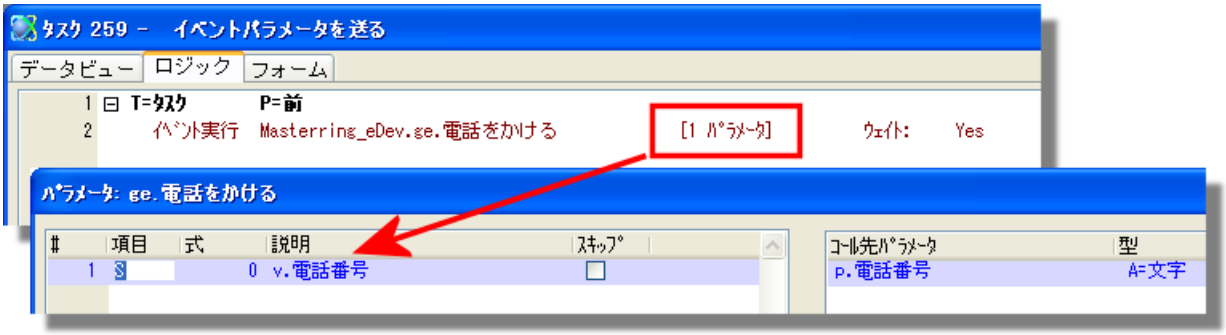
1. イベントテーブルでイベントを作成します。
2. パラメータカラムに移動します。ここからズームしてイベントパラメータテーブルを開きます。
3. イベントハンドラに渡したいパラメータを定義します。

イベントハンドラでパラメータを受け取るようにする



1. イベントを発行させたい場所に移動します。
2. *R* を入力し、イベント実行処理コマンドを作成します。イベントダイアログが表示されます。
3. イベントタイプで *U= ユーザ*を選択します。
4. イベントに移動します。ここからズームしてイベントを選択します。選択されたイベントにパラメータが定義されている場合、「イベントのパラメータに合ったパラメータを作成しますか？」というメッセージが表示されます。はいをクリックするとパラメータが自動的に追加されます。

イベントを発行する

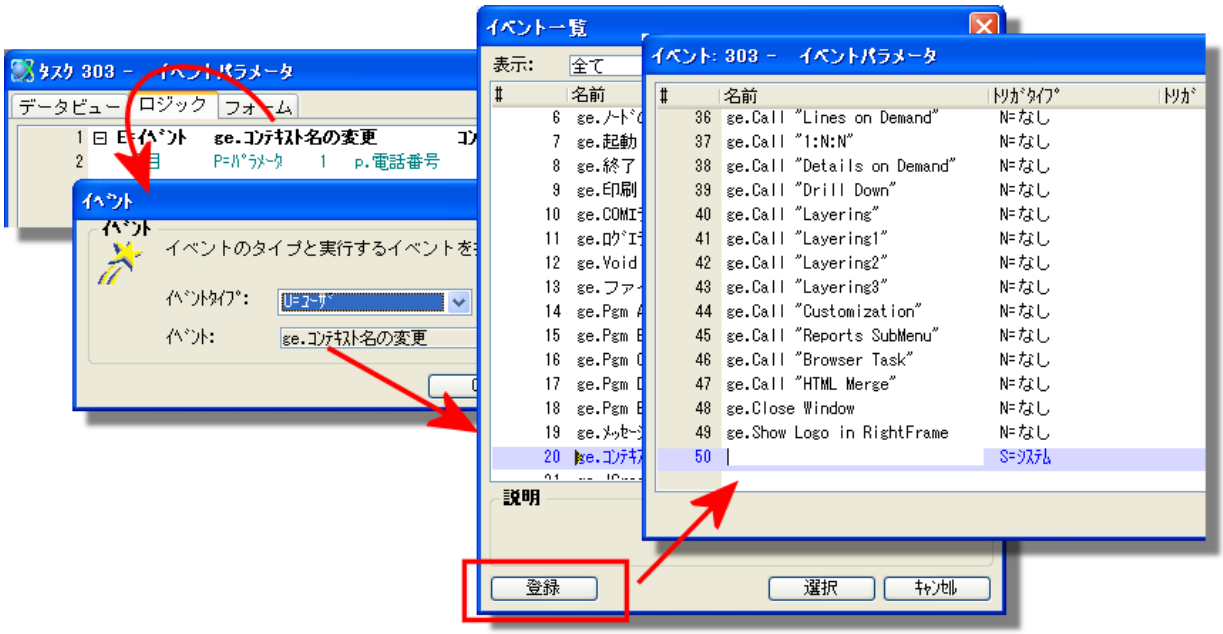


これで、イベント発行すると、コール処理コマンドと同じようにパラメータを渡すことができます。

ユーザイベント選択時にイベントを登録するには

ユーザイベントの選択時に該当するイベントが見つからず、登録する必要な場合があります。このような場合（SP4以降では)、イベント一覧からイベントテーブルを開くことで、イベントを追加することができます。

ユーザイベントを追加する

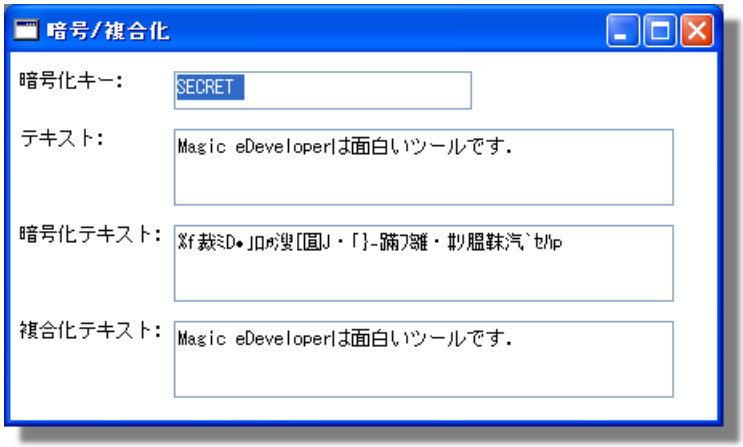


1. イベントロジックユニットのイベント選択カラムに移動します。
2. ズームしてイベントダイアログを開きます。
3. イベントタイプで *U= ユーザ*を選択します。
4. イベントからズームして、イベント一覧を開きます。
5. 登録ボタンをクリックするとイベントテーブルが表示されます。その際、１行追加された状態で開くためここに追加したいイベントを入力します。
6. OK をクリックするとイベント一覧に追加されたイベントが表示され、位置付けされます。
7. 選択をクリックすると追加されたイベントが設定されます。

［このページは意図的に空白にしています。］

第 31 章 : セキュリティ

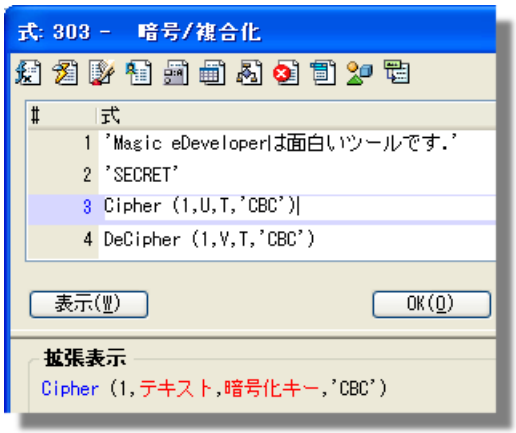
データの暗号化 / 復号化を行うには



Cipher() と **DeCipher()** 関数を使用することで、BLOB 型項目に格納されたデータの暗号化 / 復号化を行うことができます。これらの関数は、汎用的な暗号化アルゴリズムをサポートしているため、Magic 以外のアプリケーションで作成されたデータを解読することもできます。サポートされるアルゴリズムには、Blowfish のような対称型のものや RSA などの非対称なものが含まれています。

データの暗号化 / 復号化を行うために、対称型のアルゴリズムは同じキーを使用します。非対称型のアルゴリズムの場合は、対のキーと暗号化 / 復号化対象のデータが必要です。

Cipher() 関数を使用する



Cipher() 関数の構文は以下の通りです。

Cipher(*Cipher ID*, *Buffer*, *Key [, Mode, IV]***)**

パラメータ：

- *Cipher ID* …… 暗号化アルゴリズム ID を表す数値。上記の例の場合、Blowfish を表す 1 が指定されています（「サポートされる暗号化モード」（645 ページ）を参照してください）。
- *Buffer* …… 暗号化するデータを含む BLOB 型項目
- *Key* …… キーを含む BLOB 型項目。必要なキー長は、指定されたアルゴリズムに依存します。この例では、SECRET という文字列をキーとして指定しています。
- *Mode* …… どのモードを使用するかを指定するオプションパラメータです。指定可能なモードは暗号化アルゴリズムに依存します。
- *IV* …… 初期ベクトルを含む BLOB 型項目です。このパラメータはオプションです。

この関数は、暗号化されたデータを含む BLOB データを返します。

Decipher() 関数を使用する

Decipher() 関数の構文は、**Cipher()** と同じです。

Decipher(*Cipher ID*, *Buffer*, *Key [, Mode, IV]***)**

パラメータ：

- *Cipher ID* …… 暗号化アルゴリズム ID を表す数値。上記の例の場合、Blowfish を表す 1 が指定されています（「サポートされる暗号化モード」（645 ページ）を参照してください）。
- *Buffer* …… 復号化するデータを含む BLOB 型項目
- *Key* …… キーを含む BLOB 型項目。必要なキー長は、指定されたアルゴリズムに依存します。この例では、SECRET という文字列をキーとして指定しています。
- *Mode* …… どのモードを使用するかを指定するオプションパラメータです。指定可能なモードは暗号化アルゴリズムに依存します。
- *IV* …… 初期ベクトルを含む BLOB 型項目です。このパラメータはオプションです。

この関数は、復号化されたデータを含む BLOB データを返します。

サポートされる暗号化モード

アルゴリズム名	暗号化コード	サポートされるモードと IV の長さ	キーの数と長さ	対称／非対称
BLOWFISH	1	ECB - NA CBC - 8 CFB - 8 OFB - 8	最小値：1 最大値：56 推奨値：16	対称
CAST	2	ECB - NA CBC - 8 CFB - 8 OFB - 8	最小値：5 最大値：16 推奨値：8	対称
DES	3	ECB - NA CBC - 8 CFB - 8 OFB - 8	キーの数：1 サポート数：8 推奨値：8	対称
IDEA	4	ECB - NA CFB - 8 OFB - 8	最小値：1 最大値：16 推奨値：16	対称
RC2	5	ECB - NA CBC - 8 CFB - 8 OFB - 8	最小値：5 最大値：16 推奨値：8	対称
RC4	6	未対応	最小値：1 最大値：NR 推奨値：16	対称
RC5	7	ECB - NA CBC - 8 CFB - 8 OFB - 8	最小値：1 サポート数：255 推奨値：16	対称
DES3	8	ECB3 - NA CBC3 - 8	キーの数：2 最大値：16 or 24 推奨値：24	非対称
RSA	9	未対応	最小値：48 最大値：2048 推奨値：128	非対称

テーブルのデータを暗号化するには

パスワードを使用してアプリケーションへのアクセスを制限した場合でも、低レベルのツールを使用して直接ディスク内のデータを参照することが可能です。このような理由により、非常に重要な情報を含んだテーブルを暗号化する必要性があります。

Magic では、このようなことを容易に実現することができます。以下の手順で行います。

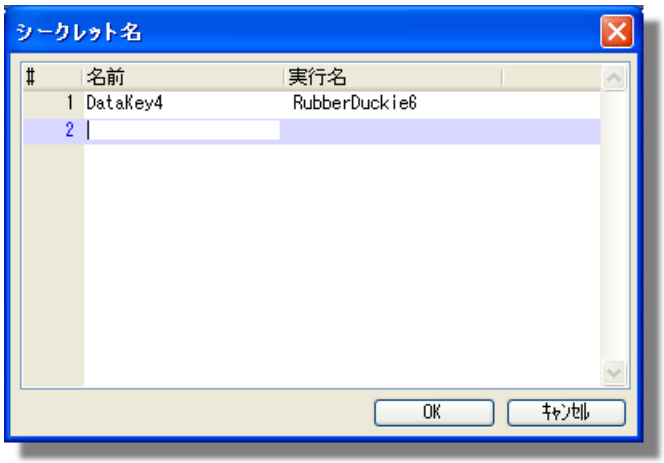
データベーステーブルを暗号化する

1. **データ**リポジトリを開き、暗号化したいデータソースに移動します。
2. **データソース特性**（*Alt+Enter* または、**編集→特性**）を開きます。
3. **アクセスキー**特性に任意の文字列を入力します。**シークレット名**を使用することを推奨します（「データベースのログイン情報を非表示にするには」（647 ページ）を参照してください）。
4. **テーブルの暗号化**特性を *Yes* に設定します。

これで、テーブル内にデータは暗号化されます。同じキーを持っているユーザのみがこのデータを参照することができます。

注： すべての DBMS が暗号化をサポートしているわけではありません。サポートされていない場合、**アクセスキー**特性や**テーブルの暗号化**特性が無効表示になります。

データベースのログイン情報を非表示にするには

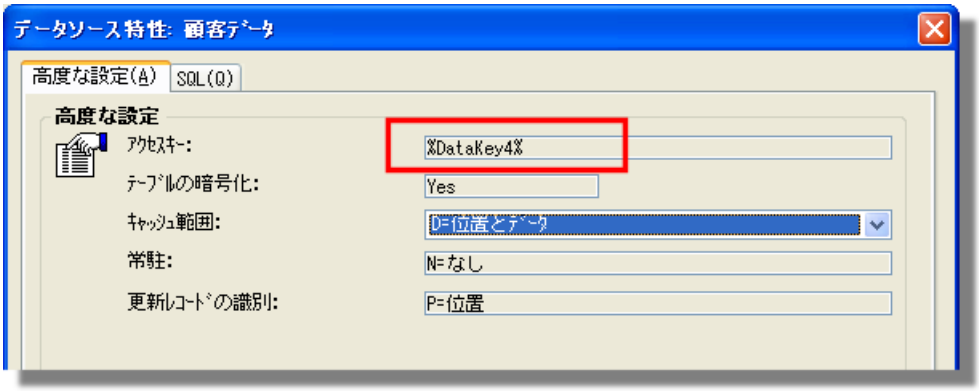


シークレット名を設定する

1. SUPERVISOR で Magic にログオンします。
2. シークレット名テーブル（オプション→設定→シークレット名）を開きます。
3. シークレット名を入力します。名前カラムの内容は、SUPERVISOR でしか参照できない点を除いて、論理名と同じように使用できます。

シークレット名は、セキュリティファイル内で暗号化された状態でユーザ ID とパスワード一緒に格納されます。

シークレット名を使用する



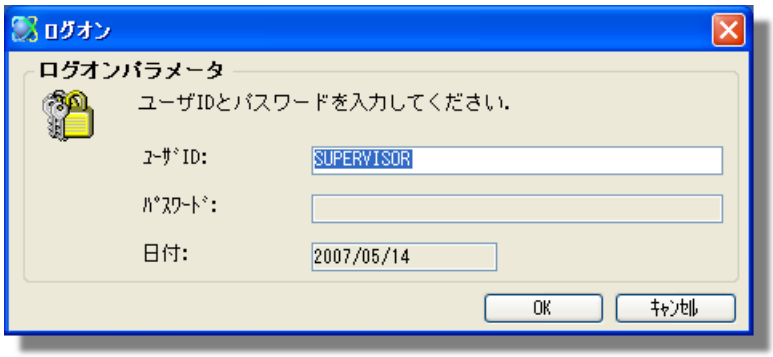
アクセスキーにシークレット名を使用する場合は、論理名と同じ方法で指定します。この名前は、実行時に実際の名前に変換されます。シークレット名は、プロジェクト特性のアプリケーションアクセスキー特性やサーバ／データベース特性のパスワード特性、およびデータソース特性のアクセスキー特性などの特定の特性値でしか使用できません。

シークレット名を使用することで、アプリケーション毎に独自のセキュリティ情報が定義できることになります。開発者は、論理名を使用して値をコード化できますが、実行時にどのような値に変換されるかは分かりません。同じように、例えば、運用時のデータベースのパスワードを開発者が知る必要がなくなります。

アプリケーションの管理者権利を定義するには

Magic アプリケーションを開発する場合、SUPERVISOR と呼ばれるユーザは特別な権利を持っています。SUPERVISOR としてログインすると、他のユーザ用にアカウントを作成したり、修正することができます。

最初に Magic アプリケーションを作成する場合、SUPERVISOR はすでにユーザリスト内に定義された状態になっています。SUPERVISOR としてログインするには、以下の操作が必要です。



1. プロジェクトがオープンされている場合、**ファイル→プロジェクトを閉じる**を選択します。
2. **オプション→ログオン**を選択します。
3. **ログオン**ダイアログが表示されます。
4. **ユーザ ID** に、***SUPERVISOR*** を入力します。
5. **パスワード**を空白のままにします。
6. **OK** を押下します。

これで、SUPERVISOR としてログインされ、ユーザ ID の保守が可能になります。SUPERVISOR ユーザにパスワードをすることを推奨します。

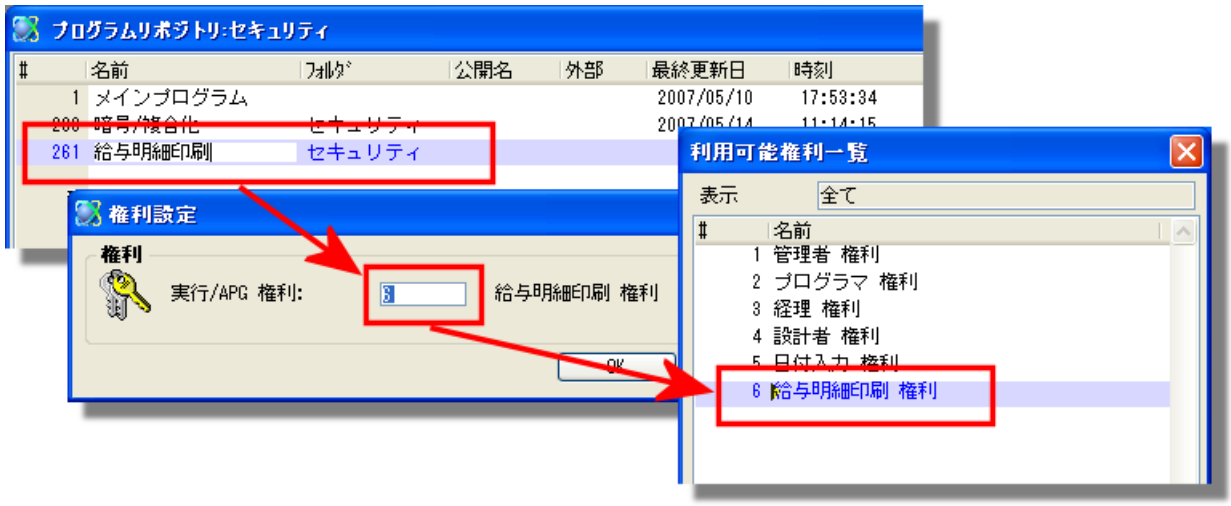
セキュリティファイルを削除する

セキュリティファイルが削除されたり、指定された場所に存在しない場合は、パスワードを空白にした状態で SUPERVISOR としてログインすることができます。SUPERVISOR としてログインすることでアプリケーションに対するユーザ ID やグループや権利などへのアクセスが可能になります。

このようにデフォルトの SUPERVISOR でアクセスできないようにするには、以下のオプションを使用してアプリケーション毎のセキュリティ設定を行う必要があります。

- 権利の非公開
- **アプリケーション特性**の**スーパー権利キー**
- **シークレット名**

特定プログラムの実行を制限するには



権利のないユーザによって実行されないようにプログラムに権利を設定することができます。

例えば、給与明細を印刷するプログラムを考えてみます。ほとんどのユーザはこのプログラムを実行することはできません。メニューに権利を設定することで表示を制限することができますが、開発者レベルでの実行を制限できるようにしたい場合があります。このような場合は、以下のようにプログラムに実行権を設定することで対応できます。

1. 実行を制限させたいプログラムに移動します。
2. **オプション→権利**を選択します。**権利設定**ダイアログが表示されます。
3. **実行／APG 権利**で**ズーム**して、設定したい権利名を選択します。

これで、この権利を持っていないユーザはプログラムを実行することができなくなります。

この方法でプログラムの実行を制御できますが、メニューに対しても同じように権利を設定しておくことで、より確実に制限することができます。メニューに対する権利設定については、「ログオンユーザにもとづいてメニューをカスタマイズするには」（650 ページ）を参照してください。

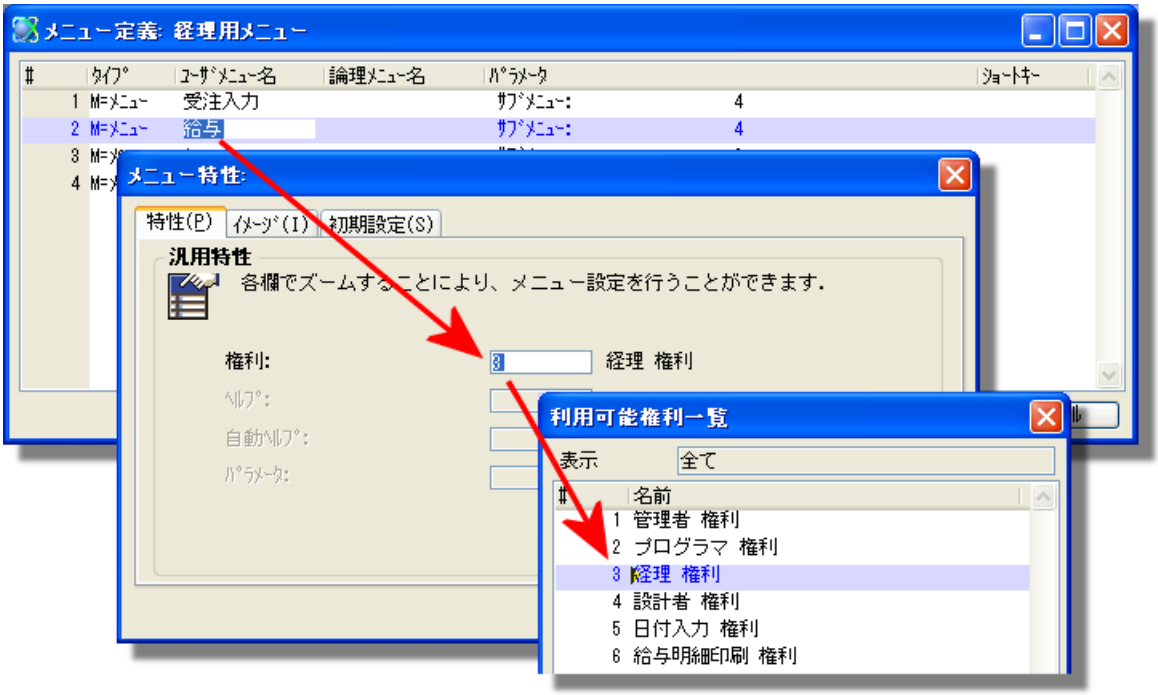
ヒント : *開発者が、このプログラムをテストするために実行する必要がある場合は、評価用と開発用の異なる２つのセキュリティファイルを用意してください。その際開発者は、評価用のデータで実行するようにしてください。*

ログオンユーザにもとづいてメニューをカスタマイズするには

ユーザが使用するメニューのみを表示するようにした方が、使いやすく見た目もすっきりしたものにります。また、セキュリティの面から考えて、すべての機能を表示させないようにした方が安全です。

Magic では、メニュー表示を任意にカスタマイズすることができます。詳細は、第 27 章：「メニューの表示／非表示を行うには」（595 ページ）を参照してください。しかし、プログラムでメニュー表示を制御する方法より、セキュリティ機能を利用した方が簡単な場合があります。

メニュー項目に権利を設定する



1. メニューリポジトリ（*Shift+F6*）を開きます。
2. 権利を設定したいメニュー項目に移動します。一般的に、職務権限にもとづいて利用可能な最も高い権利レベルをメニューに割り当てるようにしますが、どのメニュー項目に対しても権利を割り当てることができます。
3. *Alt+Enter* を押下して、メニュー特性を開きます。
4. カーソルを特性タブの、権利特性に置きます。ここからズームして権利を選択します。

これで、権利を持たないユーザがアプリケーションを実行した場合、メニューが表示されなくなります。この例では、経理の権限を持たないユーザは給与支払台帳のメニューが表示されなくなります。

注： 上記の操作を行うには、予め開発者が権利にアクセスできるようにしておく必要があります。

ログオンユーザにもとづいて機能を制限するには

Rights() 関数を使用することで、ユーザの権利レベルに応じて実行できる機能をプログラム内で変更することができます。ここでは、関数の定義方法や使用例を説明します。

Rights() 関数を使用する

Rights() 関数の構文は以下の通りです。

Rights(*right*)

パラメータ：

- *right* ……権利リポジトリに定義されている権利名

プログラムを実行しているユーザが指定された権利を持っている場合、'TRUE'LOG が返ります。例えば、

Rights(*'Accounting Right'RIGHT*)

は、'Accounting Right' という権利を持っているユーザが実行すると 'TRUE'LOG が返ります。

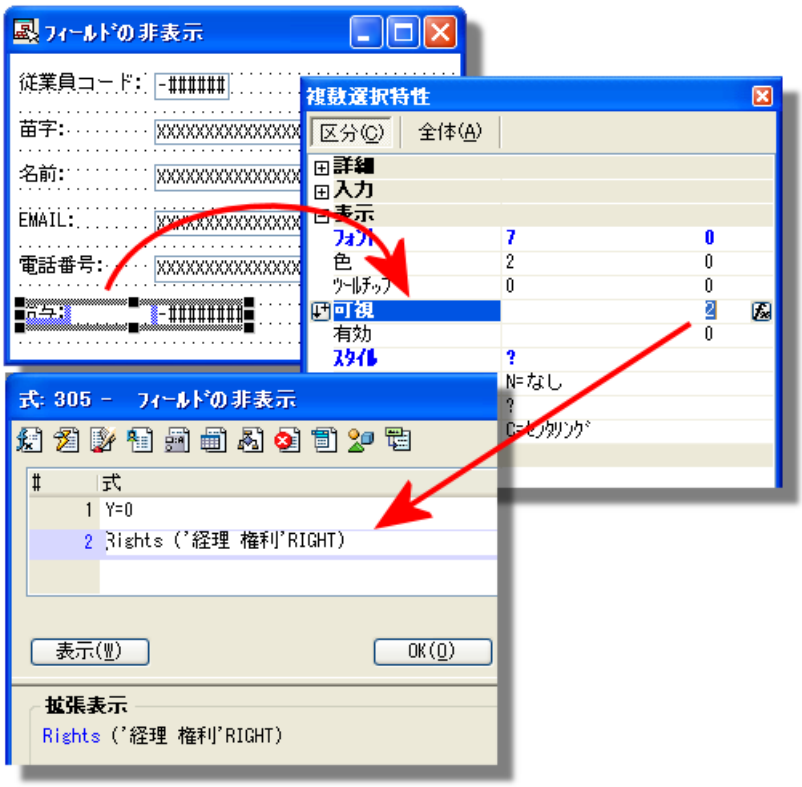
以下のような操作を行うことで簡単に入力することができます。

1. 式テーブル（*Ctrl+E*）を開きます。
2. *F4* を押下して１行追加します。
3. ri を入力して、*Ctrl+Space* を押下します。オートコンプリート機能が働いて関数リストが表示されます。ここから **Right** を選択します。
4. これにより式には以下のように入力されます。
   ```
   Rights(
   ```
5. コンテキストメニューから権利を選択します。権利一覧が表示され、アクセス可能な権利のリストが表示されます。設定したい権利（この例では、Accounting Right）を選択します。これによって以下のように権利名が入力されます。
   ```
   Rights ('Accounting Right'RIGHT
   ```
6. 次に、') 'を入力して括弧を閉じるだけで式の入力が完了します。

複数の権利を指定したり、他の論理型の条件式に追加することもできます。

次に指定された式によって、どのように動作するかを紹介します。

権利を持たないユーザに対してコントロールを非表示にする



セキュリティレベルにもとづいて実現させたい内容の１つとして、画面にデータを表示させないようにすることです。この例では、給与の表示を制御しようとしています。この方法は、プッシュボタン等を含めたすべてのコントロールで有効です。

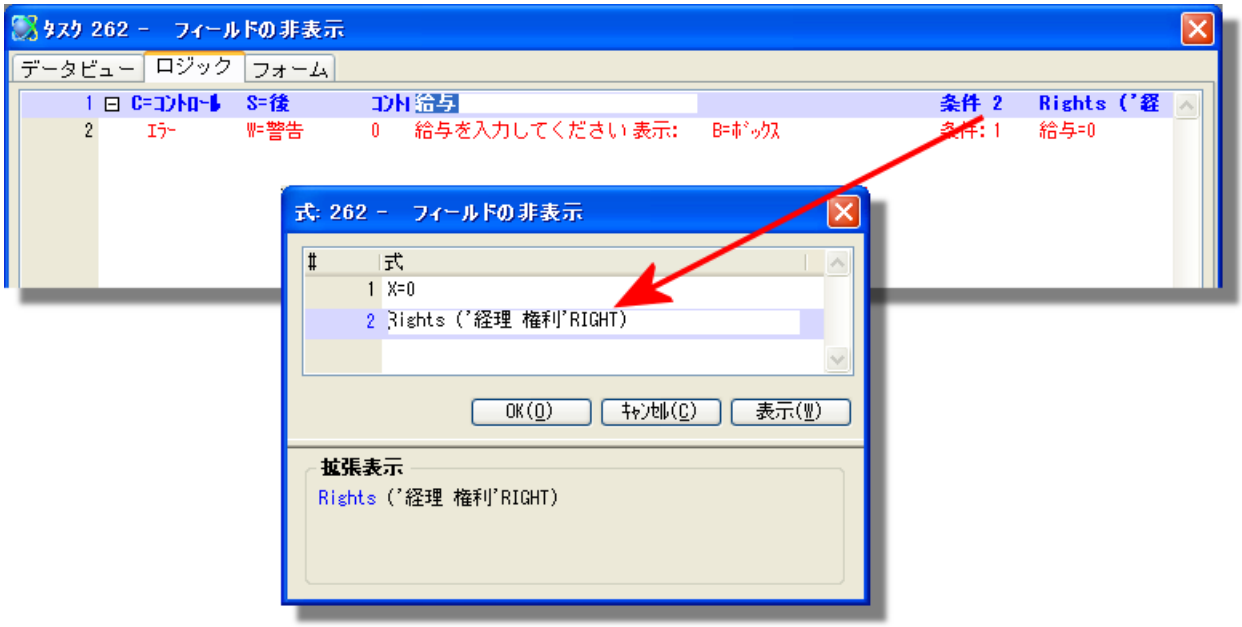
1. 制御したいコントロールを選択します。複数のコントロールに対して同じように制御させたい場合は、*Ctrl* キーを押下しながら対称となるコントロールを１つずつクリックします。この例では給与に関するフィールド（テキストとエディット）を対象にしています。
2. *Alt+Enter* を押下してコントロール特性を開きます。
3. 可視特性に移動し、右側の式特性に移動します。
4. ここからズームして式エディタで **Rights()** 関数を使用した式を定義します。

これで、権利を持っていないユーザがこのプログラムを実行すると設定されたフィールドは表示されなくなります。

表示はさせるけれど、アクセスできないようにしたい場合は、有効特性に式を定義します。

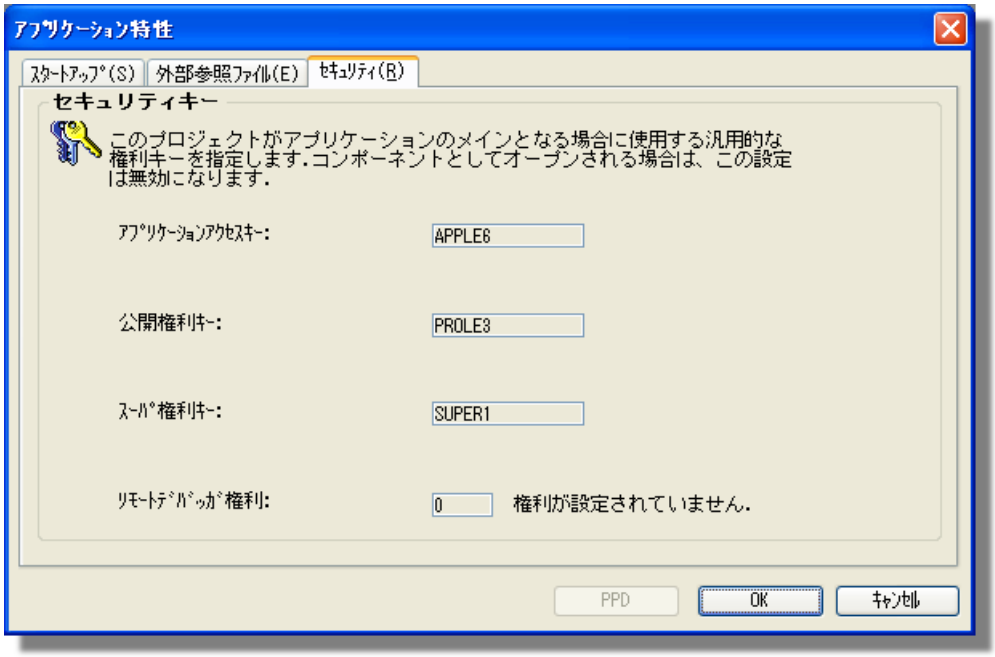
注： フィールドが非表示になった場合、そのフィールドに対する検証ロジックが実行されないことを確認する必要があります。例えば、ほとんどのユーザに対して表示されない給与フィールドがフォームに定義されているものとします。また、「給与を入力してください！」と言う確認を表示するエラー処理コマンドが定義されています。このような場合、ユーザがフィールドにデータを入力できない場合、必ずエラーメッセージが表示されることになります。従って、エラーを表示するロジックに、表示を制御させるために定義された条件式を同じように設定する必要があります。しかし、コントロールが無効の場合、このコントロールに対するコントロール検証ロジックユニットも無効になるため、問題にはなりません。

権利を持たないユーザに対してロジックの実行を防止する



Rights() 関数を使用してロジックの実行を制御するには、**条件**特性に式を設定します。この例では、**コントロール後**に条件を設定しています。このように、**処理コマンド**や**ロジックユニット**に条件式を設定することで無効にすることができます。

アプリケーション全体のアクセスを制限するには



アプリケーション特性（*Ctrl+Shift+P*）の**セキュリティ**タブにある特性値を使用することでアプリケーション全体へのアクセスを制限することができます。ここには、セキュリティに関係する４つの特性があります。各特性の詳細はMagic の『リファレンスヘルプ』を参照してください。

特性	機能	効果
アプリケーションアクセスキー	アプリケーションへのアクセスを許可する	ユーザがこのキーを持っていない場合、アプリケーションのオープン時に以下のメッセージが表示されます。 「アクセスが拒否されました：実行エンジンは、アプリケーションのオープンに失敗しました」
公開権利キー	権利キーを持つユーザに対して管理者権利を許可する	このキーが定義されていない場合、SUPERVISOR は権利を選択することができます。しかし、公開権利キーが設定されていると SUPERVISOR であってもこのキーを持っていなければ権利へのアクセスができなくなります。
スーパー権利キー	アプリケーションに対するすべての権利を許可します	これはすべての権利より優先されます。この権利を持つことですべてのプログラムを実行することができるので、テストを行う場合は便利です。
リモートデバッガ権利	リモートデバッガの使用を許可します。	リモートデバッガ起動時に指定されたユーザに、ここで指定されたアクセスキーがない場合は接続できません。

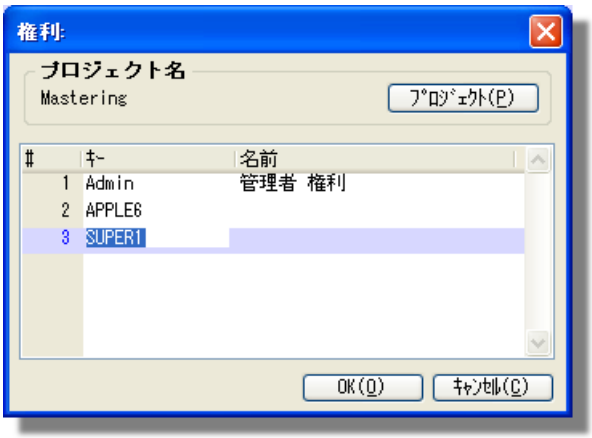
リモートデバッガ権利は、**権利**リポジトリに**ズーム**して設定します。他のキーは権利キーを直接入力します。権利キーが入力されると、そのキーを持たないユーザからは参照することができなくなります。このため、入力されたキーは正しく管理しておいてください。

セキュリティキーを使用する

ユーザに対してセキュリティキーを割り当てるには、以下の手順で実行します。

1. プロジェクトが開いている場合は一旦閉じます。
2. SUPERVISOR でログオンします。
3. **ユーザ ID** テーブルを開き、権利を割り当てたいユーザにカーソルを置きます。
4. **権利**カラムからズームします。**権利一覧**が表示されます。
5. **F4** を押下して、１行追加します。
6. **アプリケーション特性**に入力されキー名と同じ名前を、**キー**カラムに入力します。

これで、ユーザはキーを持つことになります。



権利グループを定義するには

Magic でセキュリティシステムを利用する場合、通常はユーザの職務権限にもとづいて権利を定義します。この職務権限を Magic では**グループ**として扱っています。例えば、経理グループに所属するユーザは、開発グループのメンバーとは異なるメニューや画面を持つことになります。しかし、給料明細を印刷したり勤務表を確定するような権利は、グループではなく特定の個人に割り当てることになります。

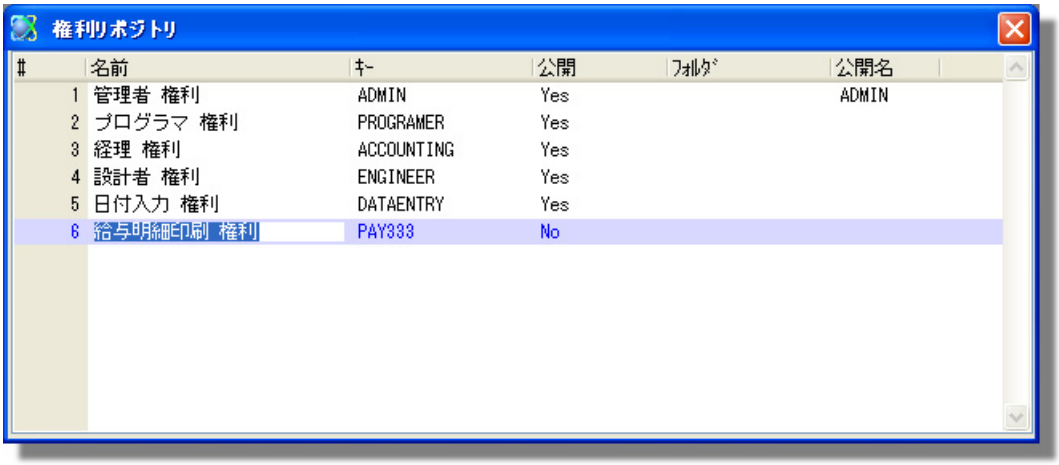
多くの権利を詳細に分割することはシステムを運用する上で扱いにくいものになりますが、抽象的すぎるのも柔軟性を失う可能性があります。システムの設計時には、このような点を考慮する必要があります。

ユーザに権利グループを設定するための手順は次の通りです。

1. **権利**を設定します。
2. **グループ**を設定します。
3. **ユーザ**を設定します。

以下、これらの手順の詳細について説明します。

1. 権利を設定する



権利は、各プロジェクトの**権利**リポジトリで設定されます。権利に**公開名**を設定することで、コンポーネントとして公開することができます。

権利を入力するには以下のようにします。

1. ***F4*** を押下して１行追加します。
2. **名前**カラムを入力します。任意の文字が入力できます。式や権利特性で権利を選択する場合に表示される**権利一覧**には、この名前が表示されます。
3. **キー**カラムを入力します。
4. 必要であれば**公開**カラムを ***No*** に設定します（この内容については、以下で説明します。）
5. 必要であれば、**公開名**カラムを入力します。

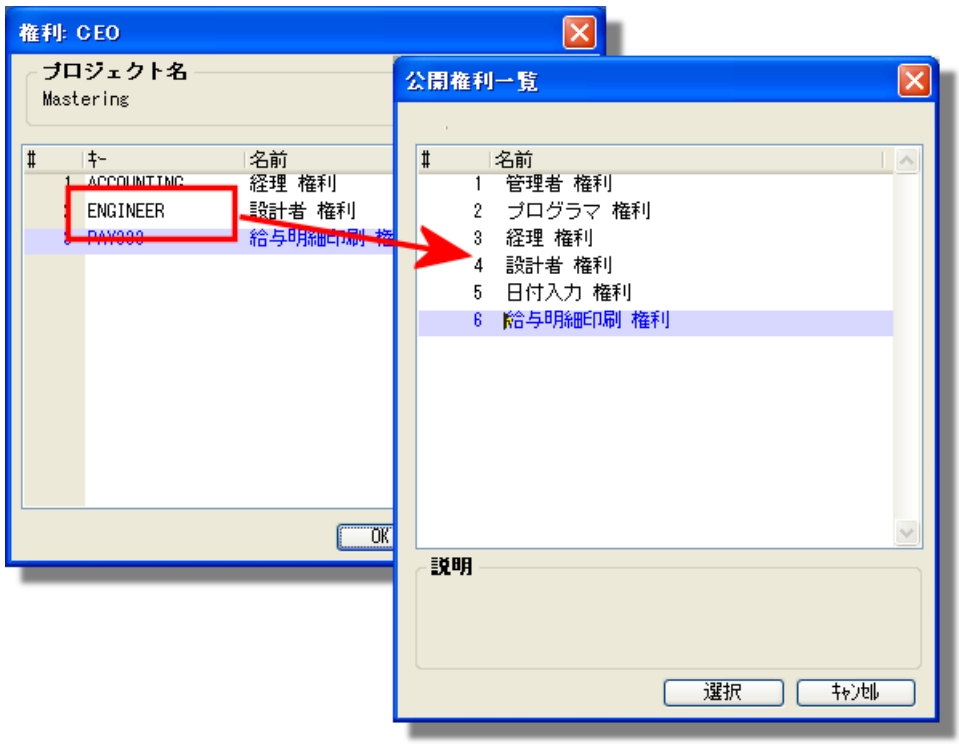
公開権利

公開カラムを ***No*** に設定した場合、この権利を持たないユーザがキー名を参照できなくなることになります。この例では、給料明細印刷の権利が非公開になっています。SUPERVISOR がキー名 PAY333 を知っている場合は、この権利を他のユーザに割り当てることができます。

セキュリティファイルが存在していなかったり、何も定義されていない場合、Magic ではデフォルトの SUPERVISOR としてログインし、権利設定を行うことができるため、権利を公開するか否かの指定が必要になります。
SUPERVISOR に対して公開したくない項目が存在する場合、公開カラムを No に設定してください。

しかし、ログイン時のユーザ ID とパスワードおよび、非公開のキー名を忘れた場合、誰もアクセスできなくなることになるので、注意してこの機能を使用してください。

2. グループを設定する



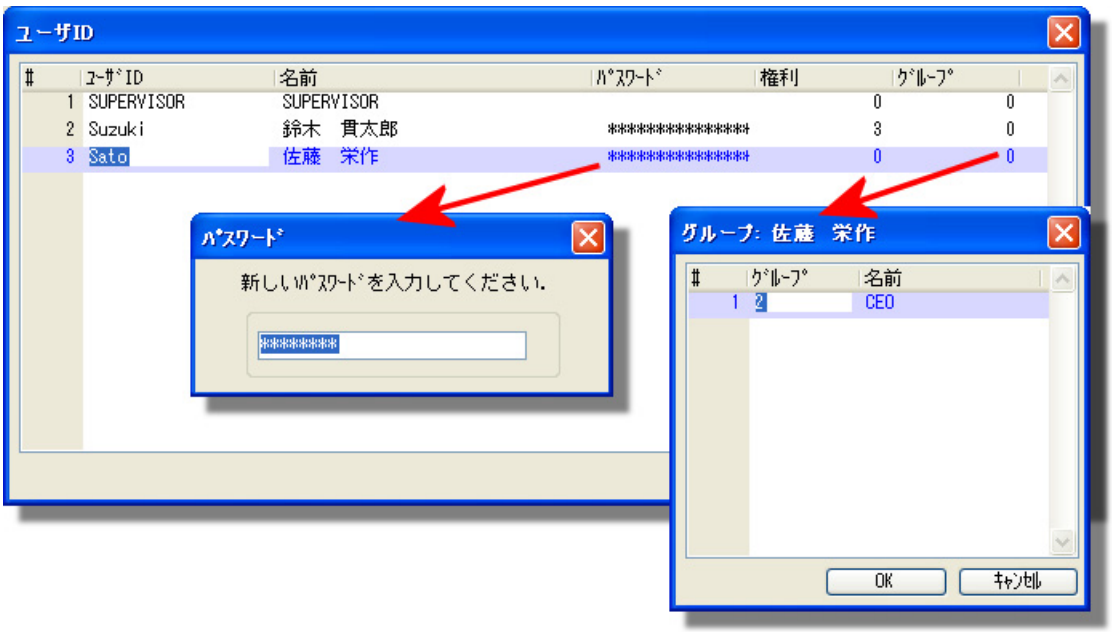
グループ情報は、プロジェクト内には定義されません。これらはセキュリティファイルに格納されます。このファイルは、複数の Magic アプリケーション間で共有することができます。**動作環境**ダイアログ（**オプション→設定→動作環境**）の**セキュリティファイル**でファイル名やパス名を確認することはできますが、暗号化されているため直接ファイルを編集することはできません。従って、グループを設定するには **Magic Studio** が必要です。

必要条件： 最初に、SUPERVISOR としてログインする必要があります。詳細は、「アプリケーションの管理者権利を定義するには」（648 ページ）を参照してください。

1. プロジェクトが開いている場合は一旦閉じます。Magic のスタートアップ画面が表示されます。
2. **オプション→設定→ユーザグループ**を選択します。
3. **ユーザグループ**テーブルが表示されます。デフォルト状態は、SUPERVISOR GROUP のみが表示されています。
4. ***F4***（**編集→行作成**）を押下して 1 行追加します。
5. 任意のグループ名を入力します。この例では、CEO と入力されています。
6. **権利**カラムに ***Tab*** 移動して**ズーム**します。このグループにはまだ権利が割り当てられていないため、空のリストが表示されます。ここに任意の権利を追加します。
   - ***F4***（**編集→行作成**）を押下して 1 行追加します。
   - **キー**を入力するか、**ズーム**して一覧から選択します。
   - 権利が非公開の場合、一覧からは選択できません。それを直接入力する必要があります。この例では、非公開権利として、PAY333 が入力されています。

同様に必要なグループを作成します。

3. ユーザを定義する



まだ、アプリケーションがクローズされている状態のはずです。この状態のまま、次にユーザを設定します。

1. **ユーザ ID** テーブル（**オプション→設定→ユーザ ID**）を表示します。を選択します。デフォルトでは、SUPERVISOR のみが表示されています。
2. ***F4***（**編集→行作成**）を押下して１行追加します。
3. **ユーザ ID** カラムで、ユーザのログイン ID を入力します。ActiveDirectory 認証や LDAP 認証を行う場合、ユーザ ID は OS から渡すことができるため、ネットワークログインを使用することができます。
4. **名前**カラムには、ユーザ名を入力します。この名前は Magic では使用されませんが、ログインしたユーザの名前を表示させるために使用することができます。
5. **パスワード**カラムでズームすることでログイン時のパスワードを指定することができます。ネットワーク経由でログインする場合は、パスワードが不要になる場合があります。
6. ユーザにグループを割り当てる場合は、**グループ**カラムから**ズーム**します。
7. 各グループに対して、このユーザを追加するには以下のようにします。
   - ***F4***（**編集→行作成**）を押下して１行追加します。
   - ここから**ズーム**して**グループ**を選択します。

権利キーを設定したグループにユーザを割り当てることで、ユーザに権利を割り当てる必要はありません。しかし、グループを使用しない場合は、ユーザ毎に**権利**カラムから**ズーム**して権利を設定する必要が必要があります。

ActiveDirectory サーバを使用して認証させるには

ActiveDirectory サーバに登録したユーザ情報をもとに、Magic にログインすることができます。これによりユーザ管理を Windows サーバ側で一本化させることができます。ActiveDirectory サーバを使用した認証方法には以下の２通りあります。

- ActiveDirectory 認証を使用してドメインにログインしているクライアント PC の情報をもとに自動的に Magic をログインする。
- ActiveDirectory サーバに対し LDAP 認証でログインする。

ここでは、これらの方法について説明します。

ActiveDirectory 認証を使用する

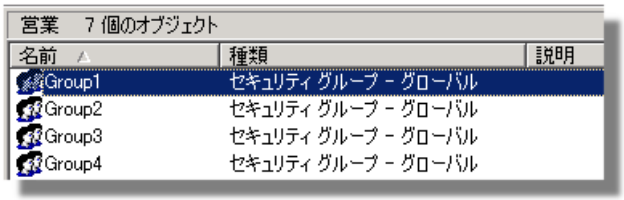
ActiveDirectory 認証を使用する場合は、クライアント PC がドメインに参加している必要があります。既に参加済みであることを前提に説明します。ドメインへの参加方法については、Windows の解説書などを参考にしてください。

環境設定を行う

1. 各クライアンの Magic を SUPERVISOR でログインし、シークレットテーブル（オプション→設定→シークレット名）を開きます。
2. シークレットテーブルに以下の２行を追加します。

名前	実行名
Directory_Binding	WinNT:// ドメイン名 /
Domain_Name	ドメイン名

3. ユーザグループテーブル（オプション→設定→ユーザグループ）を開いてグループを登録します。ここには、Active Directory サーバ側で*セキュリティグループ*として登録されているグループ名と同じ名前を登録します。各グループには、権利キーを割り当てます。



Active Directory サーバ側

Magic のグループテーブル

4. 動作環境ダイアログ（オプション→設定→動作環境）のシステムログオンを *D=Active Directory* に設定します。

クライアント PC をドメインにログオンする

1. 各クライアンをドメインに対してログオンさせます。

2. この後、Magic を起動すると自動的に Windows のログインユーザでログオンされた状態になります。

Windows のログオンダイアログ

Magic のステータスバー

ヒント :Magic を Active Directory 認証に設定すると、ログオンダイアログを表示させることができなくなります。これにより、Windouws のログオンユーザ以外で使用することを防止させることができます。

LDAP 認証を使用する

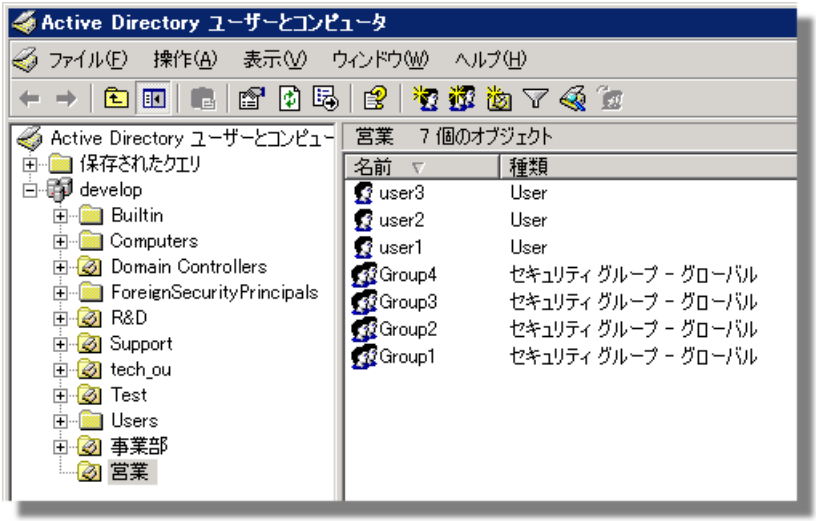
ActiveDirectory サーバに対しても LDAP 認証を使用することができます。LDAP 認証を使用する場合は、クライアント PC をドメインに参加させる必要はありません。ここでは、Active Directory サーバ上には上記と同じユーザが登録されていることを前提とします。

環境設定を行う

1. 各クライアンの Magic を SUPERVISOR でログインし、Active Directory 認証の場合と同じようにユーザグループテーブル（オプション→設定→ユーザグループ）を開いてグループを登録します。
2. 動作環境ダイアログ（オプション→設定→動作環境）の外部参照タブを開き、LDAP サーバへの接続情報を設定します。

パラメータ	設定内容
LDAP アドレス：ポート番号	Active Directory サーバのホスト名：ポート番号（デフォルトは、389）
LDAP 接続文字	CN=$USER$,OU= ユーザが定義されている OU 名 ,DC= ドメイン名
LDAP ドメインコンテキスト	DC= ドメイン名

例えば、Active Directory サーバ上に以下のようにユーザやグループが登録されていた場合。



以下のように定義します。

- LDAP アドレス：ポート番号 …… ServerName:389
- LDAP 接続文字 …… CN=$USER$,OU= 営業 ,DC=develop
- LDAP ドメインコンテキスト…… DC=develop

この設定により、OU（組織単位）が営業内に登録されているユーザに対してのみログオンできます。n

3. 動作環境ダイアログのシステムログオンを *L=LDAP* に設定します。

Magic でログオンする

1. 各クライアンで Magic を起動します。
2. ログオンダイアログ（オプション→ログオン）を開き、ActiveDirectory サーバに登録されているユーザ ID とパスワードでログオンします。
3. ログオン情報に誤りがある場合は、Invalid credentials というエラーメッセージがステータスバーに表示されます。

自動ログオンを設定する

LDAP 認証の場合も、Active Directory 認証と同じように自動ログオンを行うことができます。この場合、クライアント PC は、ドメインに参加する必要はありませんが、Active Directory サーバに登録されているユーザ ID と同じ ID でログオンされている必要があります。

1. 各クライアンの Magic を SUPERVISOR でログインし、シークレットテーブル（オプション→設定→シークレット名）を開きます。
2. シークレットテーブルに以下の２行を追加します。

名前	実行名
LDAP_USER	Active Directory サーバに登録されているユーザ ID
LDAP_PASS	パスワード

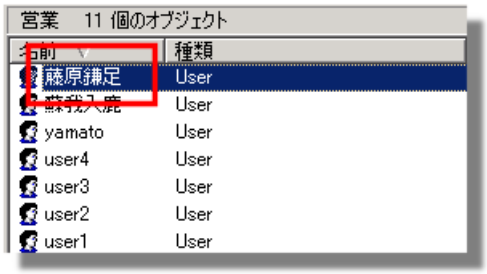
これで Magic を起動すると、Windows にログオンしたユーザ ID で Magic はログオンされます。Active Directory サーバに登録されていないユーザ ID で Windows にログオンした場合、Magic 起動時に以下のエラーが表示されます。

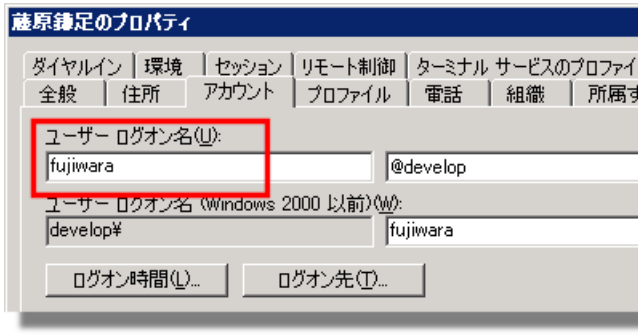
The user does not exist in the LDAP server or the credentials are invalid.

このような場合も、ログオンダイアログを開いてログオンし直すことができます。

ユーザログオン名を使用する

LDAP で認証する場合、通常は DN（識別名）を使用しますが、Windows が Active Directory サーバにログインする場合は、ユーザログオン名が使用されます。Magic では、このどちらを使用しても認証させることができます。





dn: CN= 藤原鎌足 ,OU= 営業 ,DC=develop

DN（識別名 ）

userPrincipalName: fujiwara@develop

ユーザログオン名

LDAP 接続文字列の設定

ユーザログオン名を使用して認証させる場合は、**動作環境**ダイアログ（**オプション→設定→動作環境**）で **LDAP 接続文字**を、***CN=$USER$@ ドメイン名***と設定します。

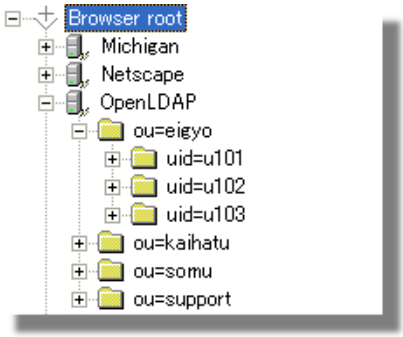
注：　この設定にすると、DN でログオンできなくなります。

OpenLDAP サーバを使用して認証させるには

OpenLDAP は、フリーの LDAP サーバです。ここでは、OpenLDAP を使用した認証方法について説明します。
OpenLDAP 自体の操作については、関連する書籍や Web サイトなどを参照してください。

ディレクトリ構造例

ここでは、以下のようなディレクトリ構造が定義されていることを前提としています。



ldif ファイル上は以下のようになります。

```
#TOP ツリー用
dn: o=example
objectClass: organization
o: example
# 部署ツリー用
dn: ou=somu,o=example
objectClass: organizationalUnit
ou: somu

dn: ou=eigyo,o=example
objectClass: organizationalUnit
ou: eigyo

# 社員ツリー用
dn: uid=u001,ou=somu,o=example
objectClass:inetOrgPerson
cn: nobunaga
sn: oda
uid: u001
userPassword: p-oda
homePhone:123-456-7890
mail:oda@magic.co.jp
homePostalAddress:Inazawa City Aich-ken

dn: uid=u101,ou=eigyo,o=example
objectClass:inetOrgPerson
cn: ieyasu
sn: tokugawa
uid: u101
userPassword: p-tokugawa
homePhone:098-765-4321
mail:tokugawa@magic.co.jp
homePostalAddress:okazaki City Aich-ken
```

環境設定を行う

1. 各クライアンの Magic を SUPERVISOR でログインしユーザグループテーブル（オプション→設定→ユーザグループ）を開いてグループを登録します。

2. **動作環境**ダイアログ（**オプション→設定→動作環境**）の**外部参照**タブを開き、LDAP サーバへの接続情報を設定します。

パラメータ	設定内容
LDAP アドレス：ポート番号	LDAP サーバのホスト名：ポート番号（デフォルトは、389）
LDAP 接続文字	uid=$USER$,OU= ユーザが定義されている OU 名 ,o= ツリーの組織名
LDAP ドメインコンテキスト	o= ツリーの組織名

3. **動作環境**ダイアログの**システムログオン**を ***L=LDAP*** に設定します。

Magic でログオンする

1. 各クライアンで Magic を起動します。
2. **ログオン**ダイアログ（**オプション→ログオン**）を開き、LDAP サーバに登録されているユーザ ID（uid）とパスワード（userPassword）でログオンします。
3. ログオン情報に誤りがある場合は、Invalid credentials というエラーメッセージがステータスバーに表示されます。

ディレクトリのデータ検索を行う

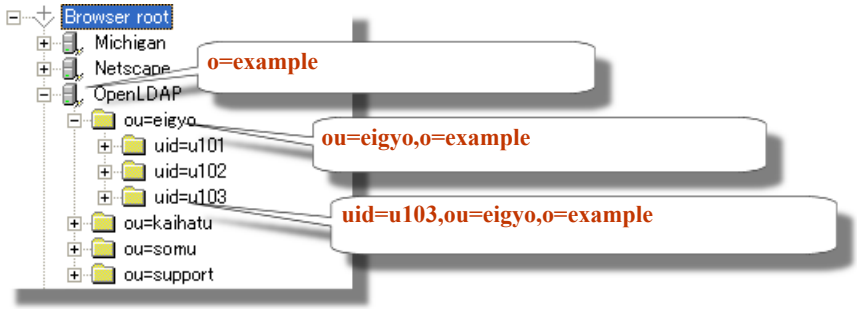
LDAP サーバを使用してログオンした場合、**LDAPGet** 関数を使用してディレクトリサーバで管理だれているデータベースを検索することができます。関数の構文は以下の通りです。

LDAPGet(*SearchBase, SearchLevel, SearchFilter, Attribute, Delimiter*)

パラメータ：

- ***SearchBase*** ：検索の開始点を指定した文字列。空白の場合は、ドメインコンテキストの値が使用されます。

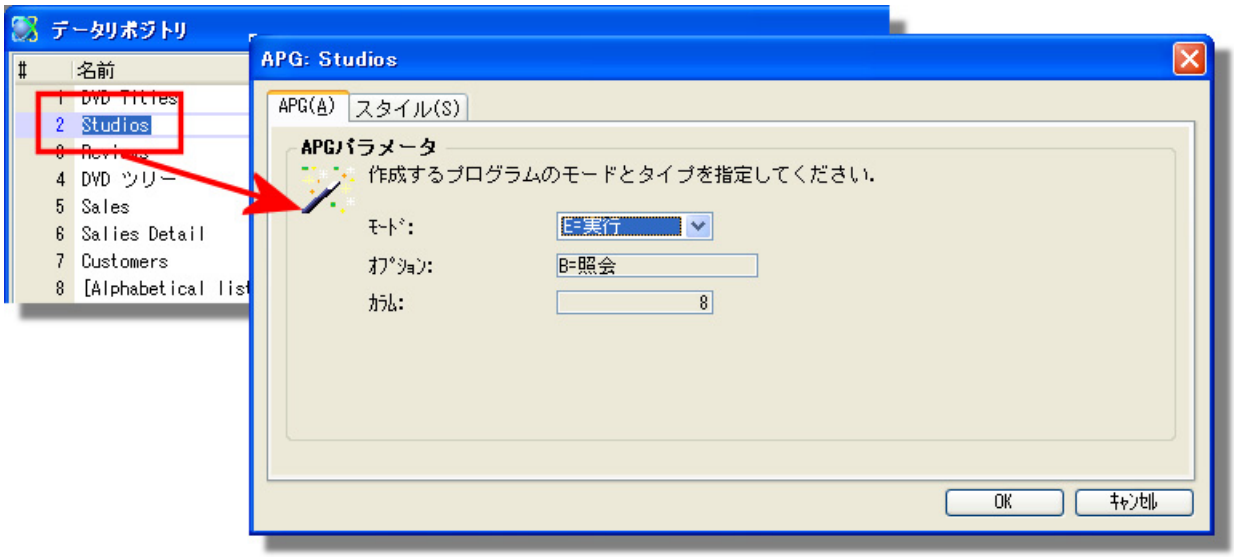
例えば、***SearchBase*** は以下のようになります。



- ***SearchLevel*** ：検索レベルを指定します。
  SearchBase が **o=example** として指定された場合、検索可能な範囲は以下のようになります。
  • B…**基本検索 → *SearchBase*** で指定されたルートのみ。
  • T…**サブツリー検索 → ツリー内の全てのノードに対して検索することができます。**
  • O…**１階層検索 → 各 ou レベルまで検索されます**（uid 以下は検索されません）。
- ***SearchFilter*** ：LDAP の検索フィルタを含む文字列です。
  例えば、homePhone=1* と指定した場合、homePhone の先頭が 1 になっているユーザ情報が対象となります。
- ***Attribute*** ：必要とされる情報タイプを定義する文字列です。
  例えば、uid と指定した場合、uid の内容が戻り値として返ります。
- ***Delimiter*** ：戻り値の値が複数の場合、値と値の間を区切る文字列を指定します。

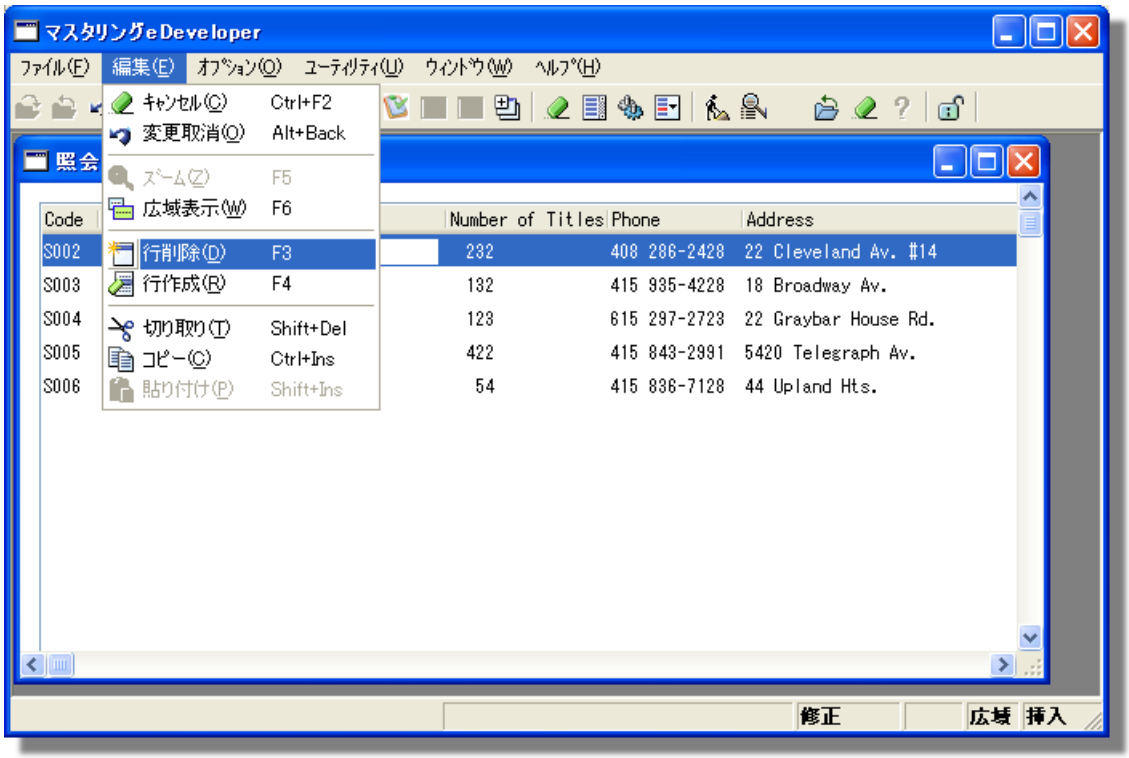
第 32 章： ユーティリティ

データソースを参照するには



アプリケーションの開発中に、使用しているデータソースの内容を簡単に参照できれば便利です。Magic では、以下のようにすることで参照することができます。

1. データソースリポジトリ（*Shift+F2*）を開きます。
2. 参照したいデータソースにカーソルを置きます。
3. *Ctrl+G*（オプション→ APG）を押下します。**APG** ダイアログが表示されます。
4. OK をクリックします。データソースのすべてのレコードが表示される参照プログラムが起動されます。



表示されるまでには、2 ～ 3 秒かかる場合もあります。表示結果をもとに、特定のレコードを検索するために位置付けや範囲指定の機能を使用することができます。レコードの追加／削除／修正を行ったり、プルダウンメニューに表示されるオプションを利用することができます。

APG には、表示をより便利に行うためのさまざまなオプションがあります。詳細は、「データソースを参照する簡単なプログラムを作成するには」（667 ページ）を参照してください。

ヒント : 参照プログラムが実行される前に、メインプログラムのタスク前が実行されます。作業時間管理表などの参照プログラムを実行しようとする間、タスク前で時間のかかる処理が定義されている場合、RunMode() 関数を使用してそれを使用不可にすることもできます。詳細は、第 19 章 :「初期設定のプログラムを省略するには」（425 ページ）を参照してください。

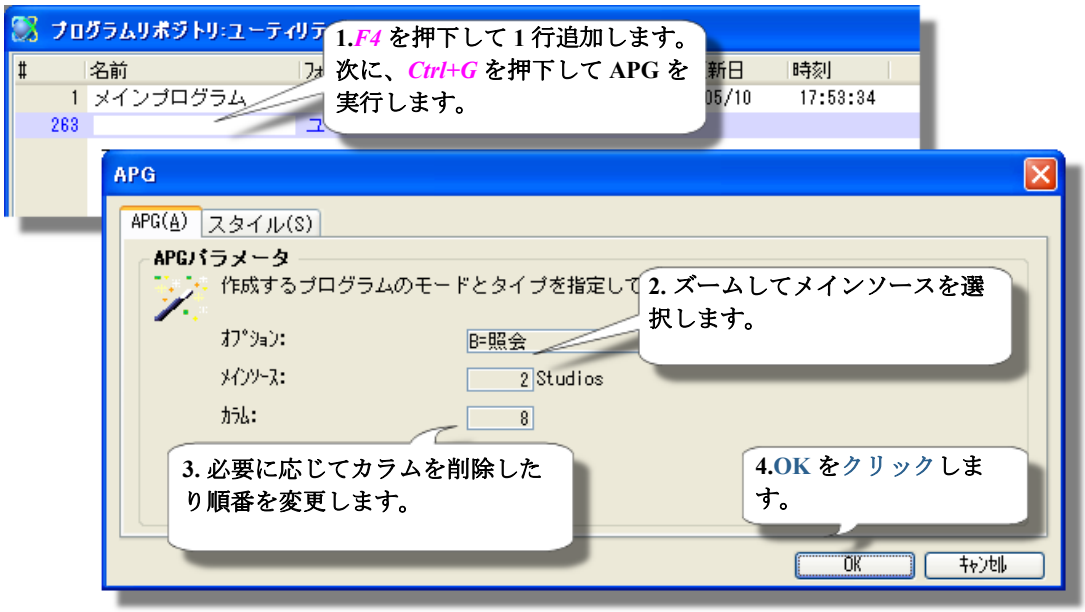
注 : データソースを参照するプログラムをプログラムリポジトリ上に作成したい場合は、似たような操作を行います。「データソースを参照する簡単なプログラムを作成するには」（667 ページ）を参照してください。

データソースを参照する簡単なプログラムを作成するには

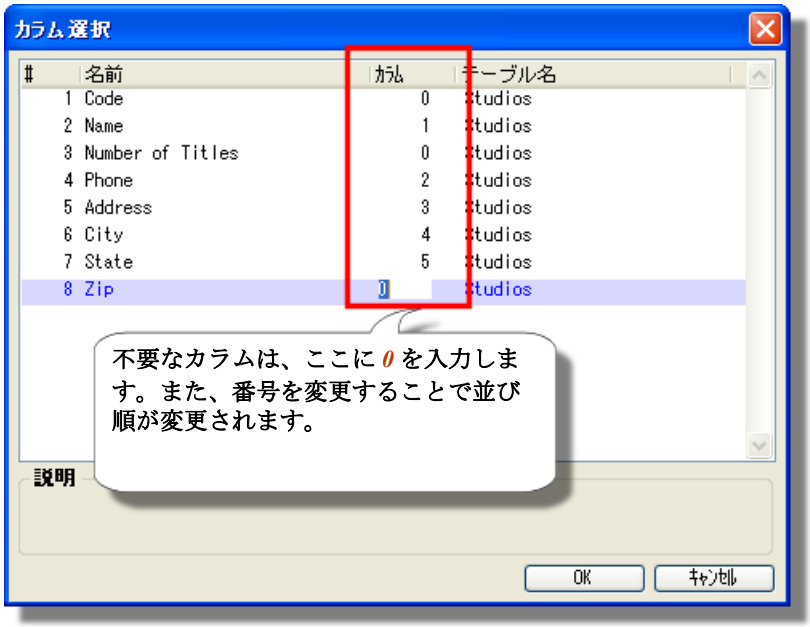
データソースの簡単な参照プログラムを容易に作成することができます。これらの簡単な参照プログラムは、デバッグ時に利用でき、レコードの追加、削除や、修正を行うことができます。Magic のデータ出力ウィザードを実行したり、データを XML などの他のフォーマットで出力するための基本プログラムとして使用することもできます。また、より複雑なプログラムを作成するためのスケルトンプログラムとして使用することができます。

どのように作成するかを以下で説明します。

参照プログラムを作成する

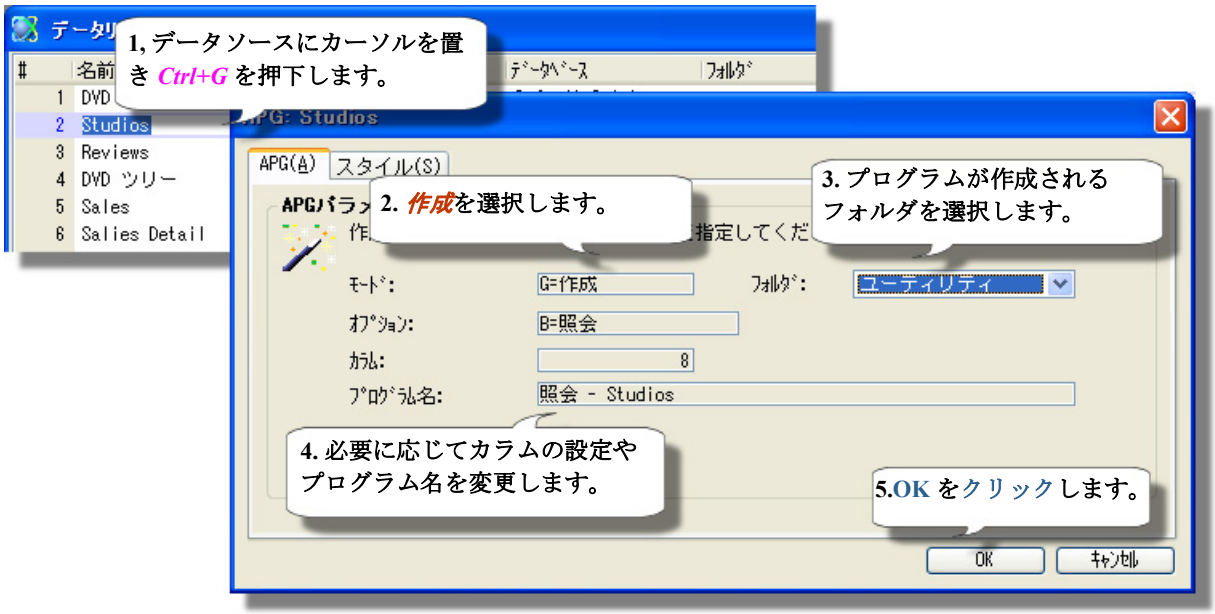


1. *F4*（編集→行作成）を押下してプログラムリポジトリに 1 行追加します。*Ctrl+G*（オプション→ APG）を押下して APG を起動します。
2. メインソースからズーム（*F5* または、ダブルクリック）して一覧から使用するのデータソースを選択します。



3. カラムカラムからズームすることで不要なカラムを削除したり、カラムの順番を変更したりできます。デフォルトでは、すべてのカラムがデータソースに定義されている順番に並びます。
4. OK をクリックすると参照プログラムが作成されます。

データソースリポジトリ上でカーソルがパークしているデータを参照するプログラムを作成することもできます。APG ダイアログでデータソースを選択する必要がない以外は、上記の場合と同じような操作で作成できます。



スタイルタブでは、データをテーブル形式で表示させるか、１レコード１画面の形式で表示させるかとか、立体表示させるか平面表示させるか、ウィンドウのサイズの指定、モデルを使用するかどうかが指定できます。

OK をクリックすると、プログラムが作成されます。*F7* を押下してプログラムを実行したり、ズームして編集することができます。

プログラムの構文チェックを行うには

どのような開発言語であってもプログラムを実行する前に、構文が正しいかどうかのチェックを行う必要があります。Magic では、プログラムの実行前にコンパイルを行わないため、構文チェックを行うか否かは任意ですが、プログラムにエラーがある場合は正しく動作しません。

構文チェックで Magic プログラムの確認を行うと（プログラム量によりますが）数秒かかります。構文チェックが完了すると、エラー箇所が存在すればエラーリストが表示されます。エラーをクリックすると、該当するオブジェクトに移動し、エラー箇所を修正することができます。

すべてのプログラムの構文チェックを一度に行うことができます。これは、アプリケーションを製品化する前にどこにもエラーが存在しないことを確認する上で必要な作業です。

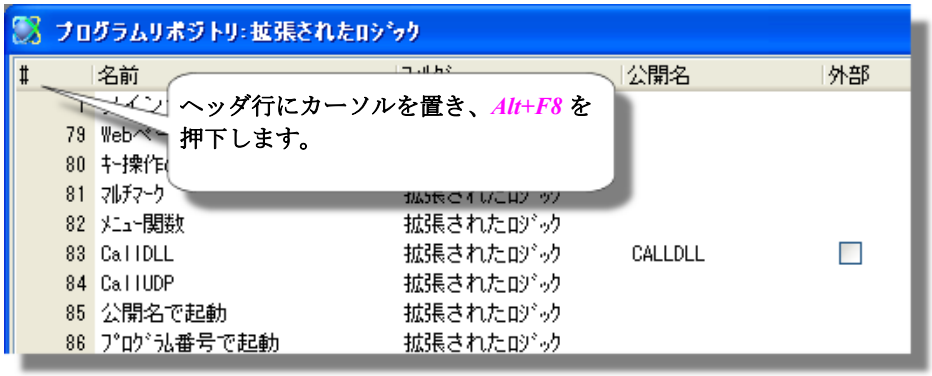
1 つのプログラムの構文をチェックする

1. チェックしたいプログラム上にカーソルを置きます。
2. *F8*（オプション→構文チェック）を押下します。プログラムのチェック中にいくつかのウィンドウが表示されます。また、以下の２つのどちらかの状態になります。
   - ステータス行に、「プログラムは正常です。」が表示されます。
   - ステータス行に「構文チェックが終了しました - チェック結果を確認してください 。」が表示され、チェック結果ペインにいくつかのエラーメッセージが表示されます。チェック結果ペインが表示されない場合は、表示→チェック結果（*Alt+F3*）を選択してください。

プログラムにエラーがある場合、チェック結果ペインに表示されるエラーのリストを確認しながらプログラムを修正します。詳細は、「チェック結果を使用するには」（674 ページ）を参照してください。

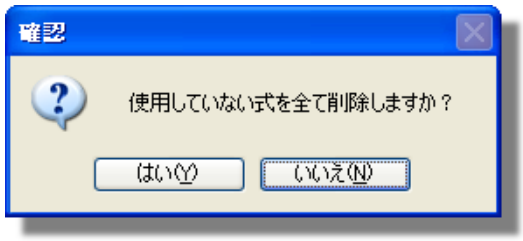
どのエラーをどのように表示させるかといったカスタマイズを行うこともできます。第 17 章：「表示されたチェックメッセージを制御するには」（390 ページ）を参照してください。

複数のプログラムを一度にチェックする



1 度に複数のプログラムをチェックすることができます。エラーメッセージは、**チェック結果**ペインでグループ化されて表示されます。1 つのプログラムをチェックした場合と同じように表示されるエラーリストをもとにプログラムを修正します。

1. チェックする一番先頭のプログラムか、**プログラム**リポジトリのヘッダ行にカーソルを置きます。特定のフォルダのみを表示させている場合は、そのフォルダ内のプログラムのみチェックされます。
2. *Alt+F8*（**オプション→カーソル以降チェック**）を押下します。

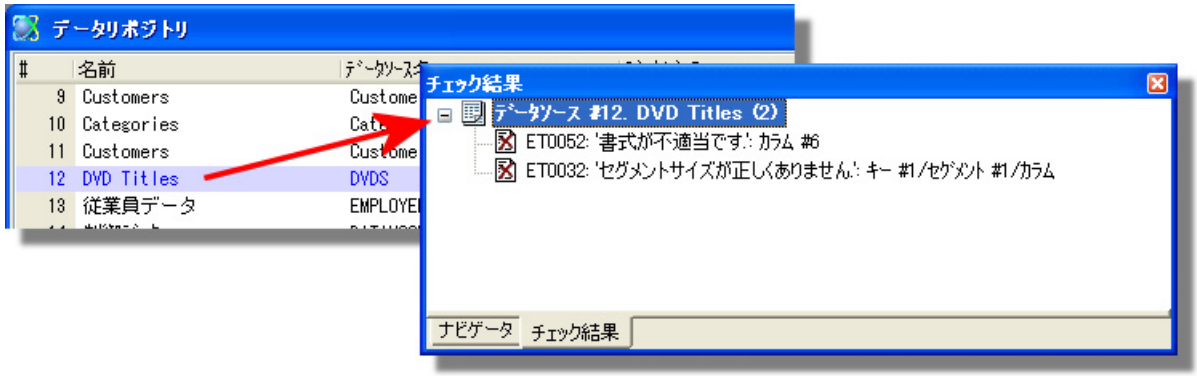


3. 「使用していない式をすべて削除しますか？」という**確認**ダイアログが表示されます。はいを**クリック**すると、どこからも参照されていない式が存在した場合、**確認**ダイアログを表示しないで自動的に削除されます。
4. 構文チェッカーは**プログラム**リポジトリ内のすべてのプログラムをチェックし、エラーが存在した場合、**チェック結果**ペインにリスト表示されます。

データソースの構造を検証するには

新しいデータソースを作成する場合、それを使用する前に構造に問題がないかどうかをチェックすることは、バグを減らす１つの手段です。この処理には、（データソースのサイズによりますが）数秒かかります。

１つのデータソースを構文チェックする

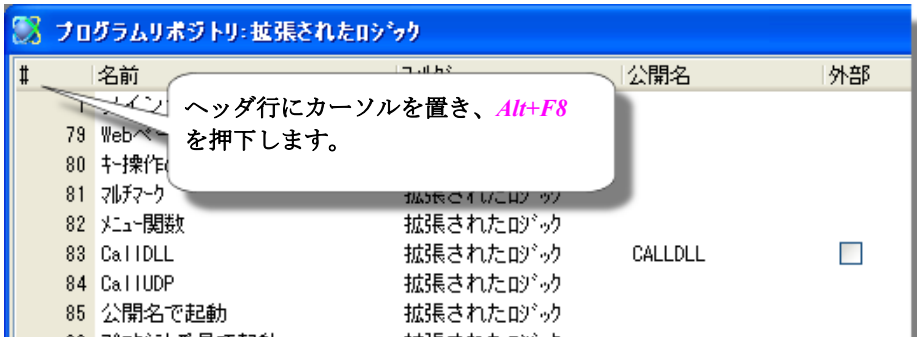


1. チェックしたいデータソース上にカーソルを置きます。
2. *F8*（オプション→構文チェック）を押下します。データソースのチェック中にいくつかのウィンドウが表示されます。また、以下の２つのどちらかの状態になります。
   - ステータス行に、「データソースは正常です .」が表示されます。
   - ステータス行に「構文チェックが終了しました - チェック結果を確認してください .」が表示され、チェック結果ペインにいくつかのエラーメッセージが表示されます。チェック結果ペインが表示されない場合は、表示→チェック結果（*Alt+F3*）を選択してください。

データソースにエラーがある場合、チェック結果ペインに表示されるエラーのリストを確認しながらデータソースを修正します。詳細は、「チェック結果を使用するには」（674 ページ）を参照してください。

どのエラーをどのように表示させるかといったカスタマイズを行うこともできます。第 17 章 :「表示されたチェックメッセージを制御するには」（390 ページ）を参照してください。

複数のデータソースを一度にチェックする



1 度に複数のデータソースをチェックすることができます。エラーメッセージは、チェック結果ペインでグループ化されて表示されます。1 つのデータソースをチェックした場合と同じように表示されるエラーリストをもとに修正します。

1. チェックする一番先頭のデータソースか、データリポジトリのヘッダ行にカーソルを置きます。特定のフォルダのみを表示させている場合は、そのフォルダ内のデータソースのみチェックされます。
2. *Alt+F8*（オプション→カーソル以降チェック）を押下します。
3. 構文チェッカーはデータリポジトリ内のすべてのプログラムをチェックし、エラーが存在した場合、チェック結果ペインにリスト表示されます。

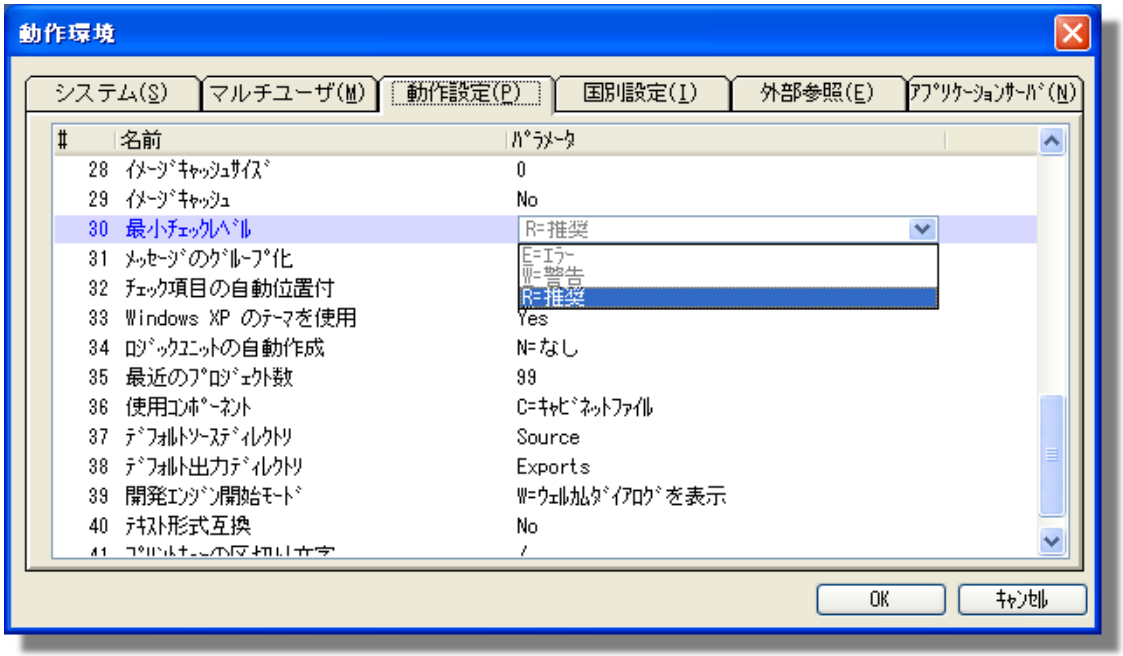
構文チェックの表示メッセージを絞るには

構文チェックユーティリティを使用する際に、どのメッセージを表示させるかを指定することができます。これは２つの方法で指定できます。

- チェック結果ペインに表示されるエラーの最小のレベルを設定する
- 各メッセージレベルを設定する

以下、これらの設定方法について説明します。

構文チェックの最小レベルを設定する

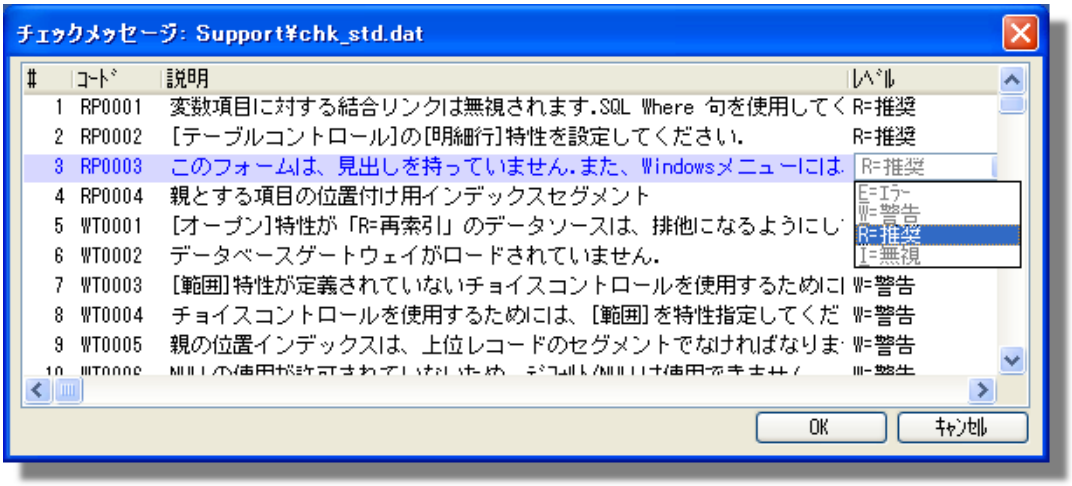


1. オプション→設定→動作環境→動作設定の最小チェックレベルに移動します。
2. 以下のように設定できます。

設定	表示されるメッセージ
E= エラー	エラー
W= 警告	エラー 警告
R= 推奨	エラー 警告 推奨

すべてのメッセージを表示させたい場合は、***推奨レベル***を使用します。

メッセージのレベルを設定する



1. チェックメッセージテーブル（オプション→設定→チェックメッセージ）を開きます。
2. 各メッセージ毎にレベルを *R= 推奨*、*W= 警告*、*E= エラー*、または *N= 無視*のどれかに設定します。

エラーリストをカスタマイズし最小レベルに変更すると、チェッカー内では表示させたいメッセージのみが表示されます。

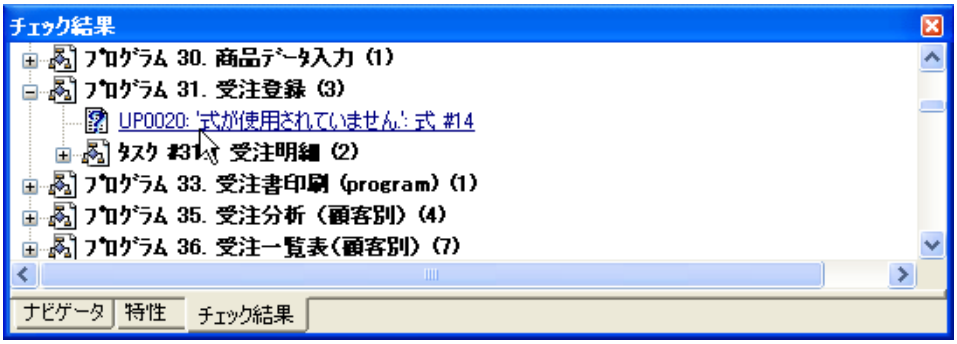
チェック結果を使用するには

構文チェックが終了すると、チェック結果ペインにチェック結果がリスト表示されます。次に、これらのメッセージに対処する必要があります。これを行うには、２つの基本的な方法があります。

- チェック結果ペインの各メッセージをクリックします。
- *Ctrl+F8* を押下してメッセージからメッセージに移動します。

それぞれの方法について説明します。

チェック結果のリストを使用する



1. 修正したいエラー上にカーソルを置きます。
2. ダブルクリックします。
3. エラーの対象となるオブジェクトに移動します。
4. すべてのエラーを修正するまで、上記の操作を続けます。

各タスクまたはデータソースがツリー上に独自のノードを持っている場合、上の図で示されているように、チェック結果はオブジェクトをもとにグループ毎に表示されます。それらが、どのようにグループ分けされるかは、動作環境で設定できます（第 17 章：「表示されたチェックメッセージを制御するには」（390 ページ）を参照してください）。

キーボード操作で移動する

Ctrl+F8（オプション→次のチェックメッセージ）を押下して移動することもできます。

必要条件： 設定→オプション→動作環境→動作設定の**チェック項目の自動位置付**を *Yes* に設定する必要があります。

1. 構文チェックを実行すると、メッセージ内で参照されたタスクが自動的にオープンされ、問題のある箇所にカーソルが移動します。
2. 次のエラーに移動する場合は、*Ctrl+F8* を押下します。

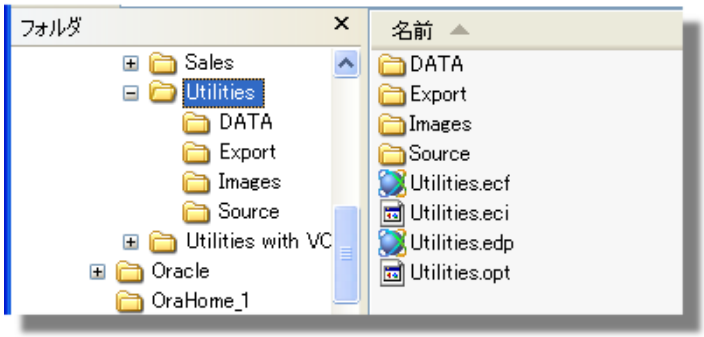
プロジェクトをバックアップするには

作業内容を常にバックアップすることは重要なことです。自動的にバックアップ処理を行うことのできるソース管理用のソフトを使用しているのであれば問題ありませんが、使用していない場合のために、バックアップ用の簡単なオプション機能があります。

- OS 上のファイルとしてバックアップする
- Magic のリポジトリ出力ユーティリティを使用して 1 つの XML ファイルに出力する
- プログラムをコピーする

以下、これらの操作方法について説明します。

OS 上のファイルとしてバックアップする



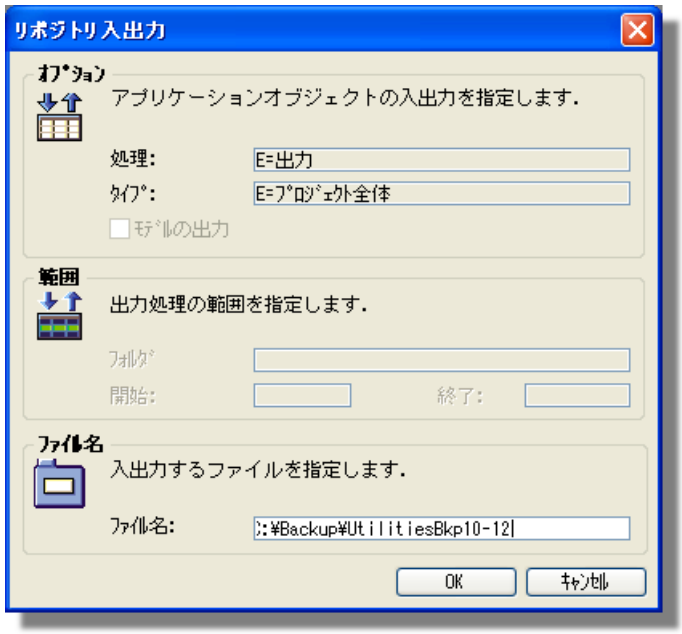
Magic が新しいプロジェクトを作成した場合、プロジェクト構造はルート上にプロジェクト（***.edp***）ファイルといくつかのサブディレクトリが作成されます。実際のプログラムソースは Source サブディレクトリ内に一連の XML ファイルとして格納されます。通常、イメージや HTML テンプレートなどの他のサポートファイルは、別のサブディレクトリに置かれることになります。

従って、プロジェクト全体をバックアップする最も簡単な方法は、ルートレベルのフォルダ（この例では、Utilities）から圧縮するだけです。必要であれば、バージョンごとに圧縮ファイルを作成し、保存することもできます。この方法は、現在のプロジェクトに対するすべてのスナップショットを保管することになります。

ハードウェアが故障した場合に備えて、他のメディア（CD や外部 HDD など）上にこのディレクトリをコピーすることもできます。

ヒント : *Magic.ini ファイルが同じディレクトリ内に存在しているか否かは、インストール方法によります。Magic.ini ファイルもいろいろな変更が行われる可能性があるため、バックアップしておいた方が無難です。*

リポジトリ出力ファイルを作成する



Magic プロジェクトからオブジェクトを出力したり、プロジェクト全体を出力することができます。この方法は、プロジェクトをバージョンごとに 1 つのファイルで管理する場合に利用できます。

1. ファイル→リポジトリ入出力（*Ctrl+Shift+E*）を選択し、リポジトリ入出力ダイアログを開きます。
2. デフォルトでは、処理は *E= 出力*になっています。このままにしておきます。
3. タイプは、デフォルトでは *E= プロジェクト全体*になっています。プロジェクト全体をバックアップする場合は、このままにしておきます。特定のオブジェクトのみをバックアップする場合は、対象となるリポジトリを指定します。また、出力対象のフォルダを選択したり、一定の連続した出力範囲を指定することもできます。
4. ファイル名を指定します。ここには、出力するファイルのパスと名前を指定します。ここからズームして保存先を選択することもできます。ファイルが作成されると「.xml」という拡張子が自動的に付加されます。
5. OK をクリックします。

出力処理が終了すると、ファイルが作成されます。

参照 : 第 2 章 :「別のプロジェクトにオブジェクトを転送するには」（31 ページ）

プログラムをコピーする

プログラムの修正作業を行う前に、編集→登録→複写登録（*Ctrl+Shift+R*）を選択してバックアップコピーを作成することができます。この方法は、プロジェクト全体をバックアップするほど全体的なものではありませんが、プログラムを修正前に戻すことができる簡単な方法です。実験的に修正処理を行いたい場合は、有効な方法です。

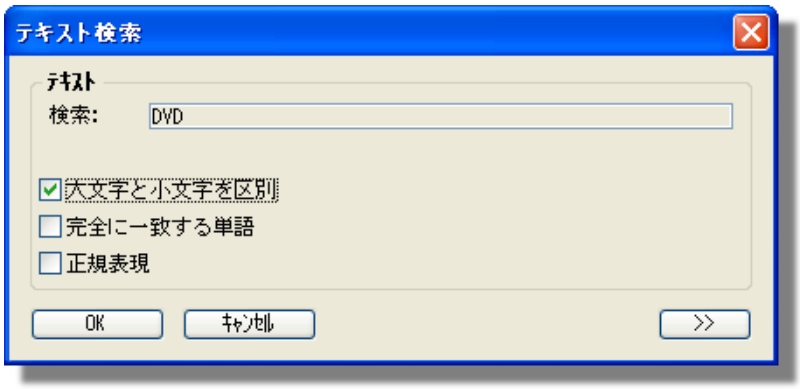
作業手順についての詳細は、第 1 章 :「入力行を複写するには」（10 ページ）か、第 1 章 :「入力行を他の行の内容に置き換えるには」（12 ページ）を参照してください。

オブジェクト内のテキストを検索 / 置換するには

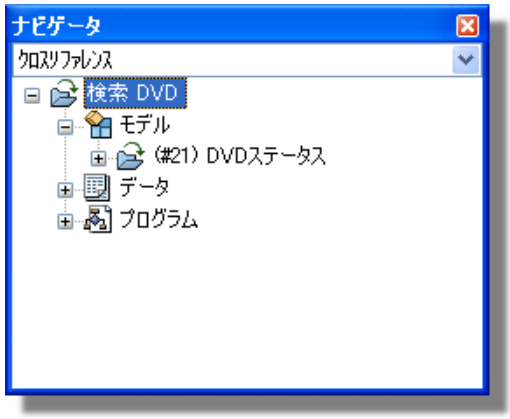
Magic は、プロジェクト内のテキストを検索し、その内容を変更することができます。例えば、帳票処理の出力結果や画面をハードコピーしたものをもっていても、どのタスクに関係するものかが分からない場合があります。また、あるデータの名前を変更する必要が発生した場合を想定してもかまいません。例えばセールスマンの代わりに営業員という名前に変更したり、ハードコートされている会社名を変更することなどが考えられます。

以下では、Magic に備えられているテキストの検索機能や置換機能についての説明を行います。

テキストを検索する



1. **編集→検索と置換→テキスト検索**（*Ctrl+Shift+F*）を選択します。
2. **テキスト検索**ダイアログが表示されます。
3. 検索したいテキストを入力します。この例では DVD と入力しています。
4. 大文字と小文字を区別して検索する場合は、**大文字と小文字を区別**をチェックします。
5. 文字列全体を検索対象にする場合は、**完全に一致する単語**をチェックします。
6. マスク文字を使用する場合は、**正規表現**をチェックします。これらのオプションの説明は、Magic の『リファレンスヘルプ』を参照してください。
7. 下に表示されている >> ボタンを**クリック**すると、検索対象のオブジェクトを指定できる詳細ウィンドウが表示されます。
8. OK を**クリック**します。

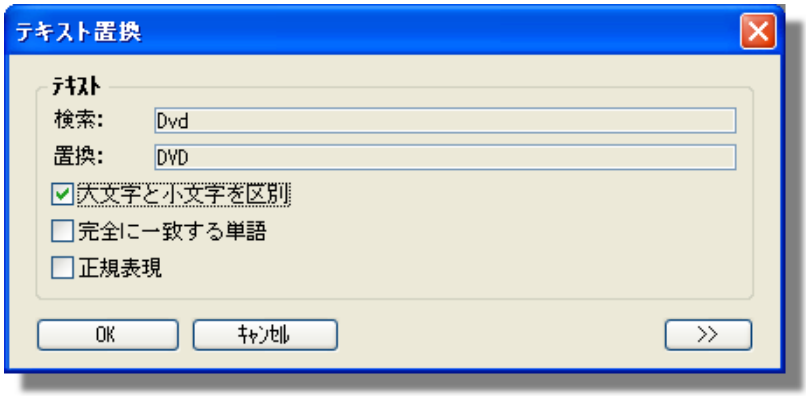


検索処理が終了すると、**ナビゲーション**ペインに検索されたテキストに対するすべての参照リストが表示されます。この検索結果を**クリック**すると該当するテキストを持つオブジェクトに移動します。

さて、検索テキストのリストが表示されたら、このリストを保存したり印刷したりすることができます（「検索結果を保存／印刷するには」（679 ページ）を参照してください）。

ヒント : *検索リストを使用して作業する場合、必要であれば F3（または編集→行削除）を押下してリスト上の項目を削除することができます。リストが大量に表示される場合は、テキストを修正後にリストを削除することで、作業が効率化する場合もあります。*

テキストを置換する



1. 編集→検索と置換→テキスト置換を選択します。
2. 検索には、検索対象のテキストを入力します。この例では、Dvd が入力されています。また大文字小文字を区別するように指定されています。
3. 置換には、検索されたテキストを置き換えるテキストを入力します。
4. 前の例で説明されているようにテキストを検索したい場合は、他のオプションを設定します。
5. OK をクリックします。

6. テキストが見つかると、検索された最初のテキストに位置付き、置換の確認ダイアログが表示されます。ここには以下のオプションがあります。
   - 置換：位置付けられたテキストを置換して次の置換対象に移動します。
   - 全て置換：すべてのテキストを一度に置換します。
   - スキップ：このテキストは置換せず、次の置換対象に移動します。
   - キャンセル：検索処理を中断します。

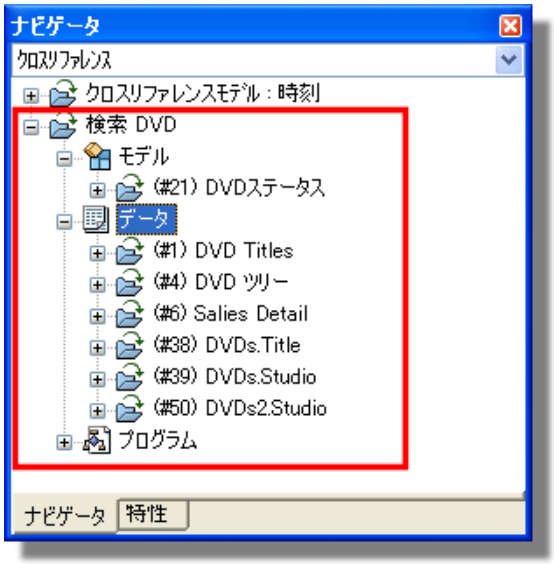
7. 置換が終了すると、ナビゲータペインに置換されたすべての項目リストが表示されます。これらの項目をクリックすると対応するオブジェクトに移動し、置換結果を確認することができます。

検索結果を保存／印刷するには

テキストの検索や置換、またはクロスリファレンスを行うと、実行結果はナビゲータペインに表示されます。これらの結果をファイルに保存したり印刷することで便利な場合があるかもしれません。例えば、修正処理にどれだけの工数がかかるかを概算する場合に利用できます。

検索結果を保存する

1. 保存したいセクション内のツリー項目にカーソルをパークします。検索 DVD のようなノードの先頭か、ツリー内の項目にパークできます。この例では、データノードにパークしているため、四角でマークされた検索 DVD セクション全体が保存されます。
2. 編集→検索と置換→検索結果の保存を選択します。
3. Windows の名前をつけて保存ダイアログが表示されます。ファイルの保存場所とファイル名を指定します。
4. 保存をクリックします。
5. 選択されたセクションがテキストファイルに保存されます。



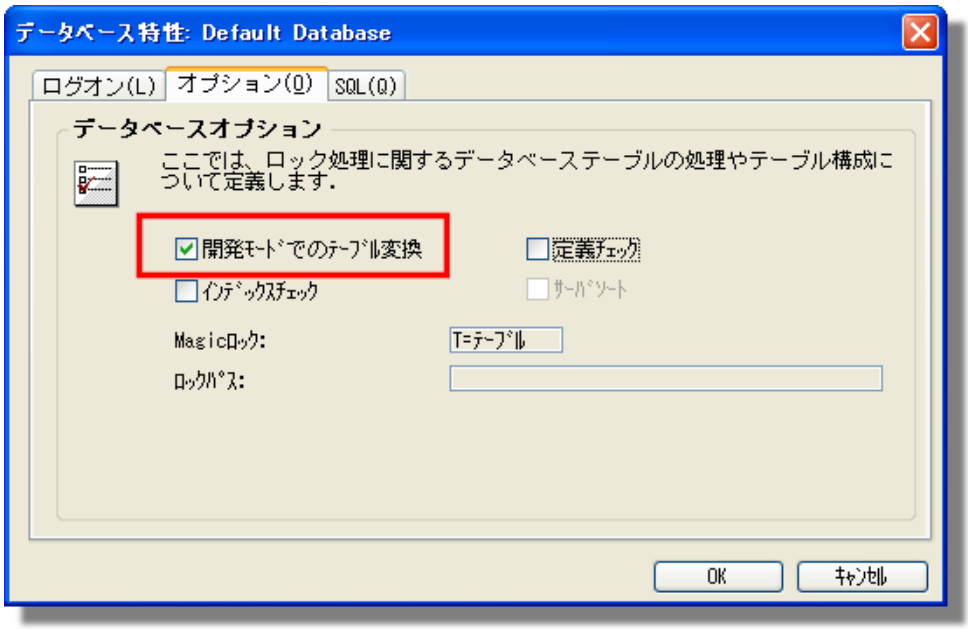
検索結果を印刷する

1. 印刷したい項目にカーソルをパークします。この例では、検索 DVD のようなノードの先頭か、ツリー内の項目にパークできます。選択されたセクション全体が印刷されます。
2. 編集→検索と置換→検索結果の印刷を選択します。
3. Windows の印刷ダイアログが表示されます。出力先のプリンタを選択し、印刷をクリックします。
4. 選択されたセクションが印刷されます。

Magic とデータベース間でのテーブル構造の不整合に対応するには

ISAM テーブルを使用している場合、実際のテーブル構造と Magic 側での定義内容とが合っていない可能性があります。このような状態は、Magic 内で変更したために発生する場合と、DBMS 側で変更された場合、２つの異なる Magic プロジェクトで同期を取って変更しなかった場合に発生する可能性があります。

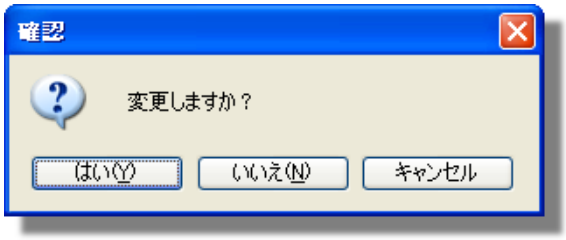
Magic で自動的にテーブルを変換できるようにする



1. プロジェクトが開いている場合、一旦閉じます。
2. **データベース**テーブル（**オプション→設定→データベース**）を開きます。
3. 使用するデータベースを選択し、***Alt+Enter*** を押下して**データベース特性**を開きます。
4. **オプション**タブを**クリック**します。
5. **開発モードでのテーブル変換**特性がチェックされていることを確認します。

開発モードでのテーブル変換特性がチェックされている場合、**データソース**リポジトリでテーブル定義を変更すると、Magic は自動的に実際のファイルの変更処理を実行します。

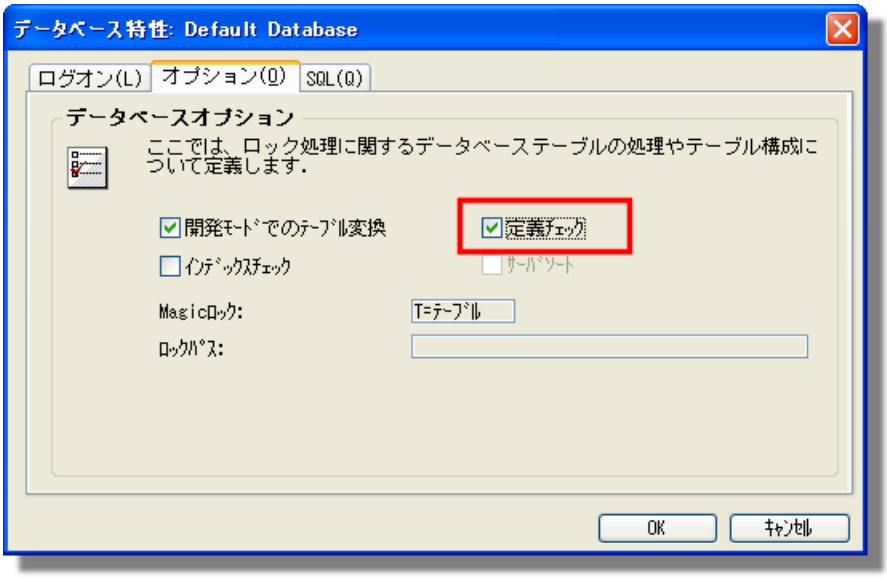
例えば、カラムの型を数値型から文字型に変更した場合、そのファイルが存在していれば以下のようなメッセージダイアログが表示されます。



はいを**クリック**すると、数値型項目は文字型に変更されます。また、項目の位置やインデックスが変更された場合も同様に変更処理が実行されます。

テーブルを変更した場合、毎回データを変更するようにする必要があります。**確認**ダイアログが表示されたときに、**No** を**クリック**するとテーブル定義はデータと同期されなくなります。

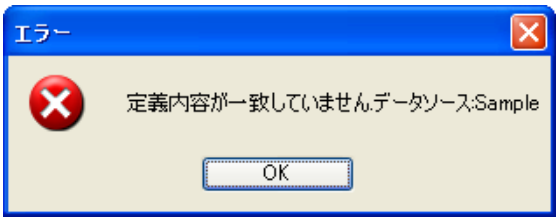
互換性をチェックする



テスト中に、ISAM テーブルに対して同期が取られているかどうかを確認することができます。これを行うには、以下のように設定します。

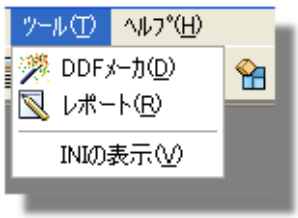
1. プロジェクトが開いている場合、一旦閉じます。
2. **データベース**テーブル（**オプション→設定→データベース**）を開きます。
3. 使用するデータベースを選択し、***Alt+Enter*** を押下して**データベース特性**を開きます。
4. **オプション**タブを**クリック**します。
5. **定義チェック**特性がチェックされていることを確認します。

これで、同期していないテーブルにアクセスしようとすると、以下の**メッセージ**ダイアログが表示されます。



また、エラーイベントが発生するため、カスタマイズされたメッセージを表示したり、ログを出力したりすることができます。

外部ツールを Studio に追加するには



Magic には**ユーザ定義開発メニュー**と呼ばれる機能があります。

ユーザ定義開発メニューを使用して、アプリケーション用のツールメニューをカスタマイズすることができます。Magic で使用されるウィザードのいくつかは Magic アプリケーションです。

ツールメニューは、Magic.ini の［TOOLS_MENU］セクションに定義することで Magic Studio のウィンドウに表示されるようになります。以下の例で説明します。

```
[TOOLS_MENU]
Menu1 = A,DDF メーカ (&D),,Add_On¥DDFMaker¥DDFMaker.ecf,,,Add_On¥DDFMaker¥DDF+
Maker_suf.opr,ImageFor = B ToolNumber = 60 ToolGroup 1
Menu2 = A, レポート (&R),,Add_On¥ReportGenerator¥ReportGenerator.ecf,,,+
,ImageFor = B ToolNumber = 23 ToolGroup 1
Menu3 = S,,,,,,
Menu4 = O, INI の表示 (&V),,notepad.exe %WorkingDir%magic.ini,,,,
```

設定できるメニューとして以下の 4 種類があります。

- A（アプリケーション）：Magic の .ecf ファイルを呼び出します。
- O（OS コマンド）：OS のコマンド（バッチファイルや EX ファイル E）を実行します。
- M（サブメニュー）：サブメニュー郡のヘッダ
- L（ライン）：メニューの区切り線

後者の 3 つは、一目瞭然です。しかし、最初のタイプ（***.ecf*** ファイルの呼び出し）は、便利なツールアプリケーションを定義することで開発時に便利に利用することができます。***.ecf*** ファイルのオープン前やクローズ後に実行するマクロ処理を実行するスクリプトを定義することができます。

パラメータの構文はメニュータイプによって異なります。タイプによっては不要なパラメータもあります。

メニュー	構文
アプリケーション （読みやすくするため改行しています。）	< メニュー名 > = A, < タイトル >, < 親のメニュー >, <ECF ファイル >, < アクセスキー >, < 事前処理ファイル >, < 事後処理ファイル >, < アイコン >
OS コマンド	< メニュー名 > = O,< タイトル >,< 親のメニュー >,< コマンド >, < アクセスキー >,,,< アイコン >
サブメニュー	< メニュー名 > = M,< タイトル >,< 親のメニュー >,,,,,
区切り	< メニュー名 > = S,,< 親のメニュー >,,,,,

以下はメニュー項目の概要です。詳細は、『リファレンスヘルプ』を参照してください。

メニューパラメータ		
メニュー名	メニュー定義の名前	ここで指定される名前は、サブメニューの定義の際に親メニューとして指定することができます。実際に表示される名前でありません。
メニュータイプ	• A = アプリケーション • O = OS コマンド • M = サブメニュー • S = 区切り	
タイトル	実際に表示されるメニューの名前	これは、メニュー上に表示されるものです。 ショートカットキーを指定するために '&' を含めることができます。
親のメニュー	親メニューのメニュー名を指定します。	先頭のメニューは空白を指定します。 親メニューのメニュータイプは、M にしてください。.
ECF ファイル／コマンド	• ECF:Magicのキャビネットファイルを指定します。 • OS コマンド：有効な OS のコマンド	メニュータイプにもとづいて、***.ecf*** ファイルか OS のコマンドを指定します。

メニューパラメータ		
アクセスキー	*Ctrl+1* などのショートカットキー組み合わせ。	
事前処理ファイル	ECF または OS コマンドが処理される前に実行されるコマンドが定義されたスクリプトファイル	コマンドファイルは、いろいろなコマンドを定義できます。詳細は、『リファレンスヘルプ』を参照してください。
事後処理ファイル	ECF または OS コマンドの処理後に実行されるコマンドが定義されたスクリプトファイル	
アイコン	ImageFor = B ToolNumber = 60 ToolGroup 1	メニュー上に表示されるアイコン。 構文で指定される内容は、Magic のメニューリポジトリで使用される内容と同じです。

外部ツールを追加する

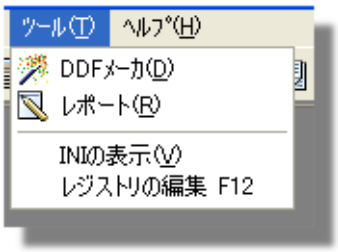
次に、外部ツールを現在のメニューに追加する手順を説明します。ここでは、レジストリエディタを追加してみます。

1. 現在のプロジェクトをクローズします。
2. テキストエディタを使用して Magic.ini ファイルを開きます。
3. ［TOOLS_MENU］セクションに移動します。
4. 上記で説明されている構文に従って OS コマンドとして Regedit を追加します。

```
[TOOLS_MENU]
Menu1 = A,DDF メーカ (&D),,Add_On¥DDFMaker¥DDFMaker.ecf,,,Add_On¥DDFMaker¥DDF+
Maker_suf.opr,ImageFor = B ToolNumber = 60 ToolGroup 1
Menu2 = A, レポート (&R),,Add_On¥ReportGenerator¥ReportGenerator.ecf,,,+
,ImageFor = B ToolNumber = 23 ToolGroup 1
Menu3 = S,,,,,,
Menu4 = O, INI の表示 (&V),,notepad.exe %WorkingDir%magic.ini,,,,
Menu5 = O, レジストリの編集 ,,regedit.exe,F12,,,
```

この例では、*F12* をショートカットキーとして定義しています。

5. **Magic.ini** ファイルを保存し、**Magic Studio** を再起動します。これで、追加されたメニューが表示されます。このメニューを選択すると Regedit が起動されます。



自動的に外部プロセスを実行するには

Magic を起動する時に、自動的にコマンドを実行させることができます。これらのコマンドは、Magic の内部でマクロコマンドとして実行されます。このコマンドでは、リポジトリの入出力やテキスト入力などを行うことができます。

外部プロセスを実行するには、２つの処理が必要です。

- 外部コマンドファイルを作成する
- Magic の起動時にコマンドファイルが実行されるように設定する

コマンドファイルを作成する

コマンドファイルには独自の構文でコマンドが設定されています。このコマンド構文は、**ユーザ定義開発メニュー**で使用される**事前／事後処理ファイル**（「外部ツールを Studio に追加するには」（682 ページ）を参照）と同じです。

構文の詳細は『リファレンスヘルプ』を参照してください。Magic 内で同じ処理を繰り返し実行させたい場合、基本的にはコマンドをコピーすることで対応できます。

以下は、コマンドの概要です。

- Export：プロジェクト全体やその中の一部のオブジェクトを出力します。例えば真夜にすべてのプロジェクトをバックアップしたい場合、この機能が利用できます。
- Import：プロジェクト全体やその中の一部のオブジェクトを入力します。
- ECF：プロジェクトからキャビネットファイルを作成します。すべてのプロジェクトのキャビネットファイルを 1 つのバッチ処理で作成したい場合は、この機能が利用できます。
- Getdef：SQL テーブルの定義取得を行います。
- Project：プロジェクトを開きます。
- Simulate：キーボード処理をシミュレートします。

また、プロジェクト内のプログラムやモデルがいくつ定義されているとかといった、プロジェクトに関する情報を参照するためのグローバル変数を使用することもできます。

自動的に実行するコマンドファイルを設定する

コマンドファイルが作成されたら、自動的にそれを実行するための環境設定が必要です。コマンドファイルによって、オープンするプロジェクトを（リポジトリ入力で）作成することもできるため、プロジェクトの外に定義することになります。つまり、Magic.ini ファイルで設定されます。

以下の行を［MAGIC_ENV］セクションに追加します。

```
AutomaticProcessingSequenceFile=%Path%Filename.txt
```

コマンドファイルの指定には、論理名を使用することができます。

バッチ内に自動処理を設定する

アプリケーションをオープンすることなく、バックグラウンドで処理を実行させることもできます。例えば、アプリケーションのバックアップのためにバックグラウンドモードでリポジトリ出力を行うことができます。この場合に、Magic.ini の [MAGIC_ENV] セクションに以下のように設定します。

```
AutomaticProcessingMode= B
```

これにより Magic 起動時に自動的に処理が実行され、自動的に処理が終了します。

Magic ライセンスの使用状況を確認するには

現在のライセンスの利用状況を確認するには、製品毎に２つの方法があります。

Magic Studio/Magic Client の場合

Support サブフォルダ内のある MGStaions ユーティリティを使用することで、ライセンスの利用可能数やライセンスが使用数を確認することができます。また利用しているユーザー一覧を取得することができます。

構文：

```
MGStations <license> <license file>
```

例：

```
"C:¥Program Files¥Magic¥Client V10¥mgstations" MGCSRTX "C:¥Program Files¥Magic¥Client
V10¥license.dat"
```

指定されたライセンスが２つの PC で使用されている場合、以下のような結果が返ります。

```
 MGCSRTX  (MGCSRT, E171C161A4601B030AC5, 20 users)      ... please wait ...

1 :     Server1              : 1 users
2 :     Server2              : 1 users

MGCSRT, E171C161A4601B030AC5:   2 users consumed
```

Magic EnterpriseServer の場合

C:¥FlexLM フォルダ内のある Lmutil ユーティリティを使用することで、ライセンスの利用可能数やライセンスが使用数を確認することができます。

構文：

```
Lmutil kmstat Oc 744@ServerName -f <LicenseName>
```

例：

```
c:¥FlexlmLmutil lmstat -c 744@ServerName -f MGENTX1
```

MGENTX1（Magic EnterpriseSeverV10）のライセンスを２つの PC で使用している場合、以下のような結果が返ります。

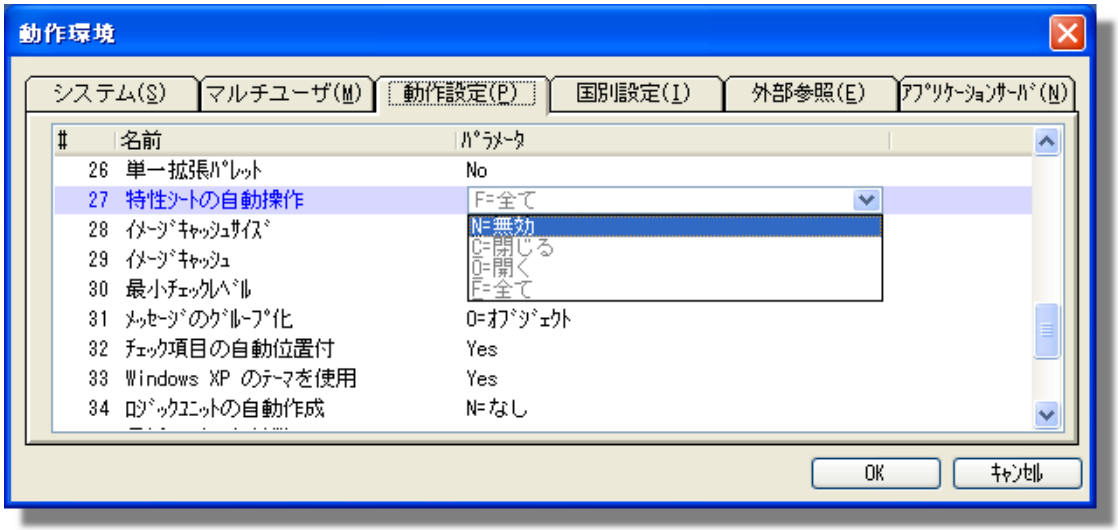
```
lmutil - Copyright (C) 1989-1999 Globetrotter Software, Inc.
Flexible License Manager status on Tue 4/15/2008 13:53

[Detecting lmgrd processes...]
Users of MGENTX1:  (Total of 35 licenses available)

  "MGENTX1" v10.000, vendor: MAGIC
  floating license

    user_id host_name host_name (v10.000) (ServerName/744 644), start Tue 4/15 9:01, 5
licenses
```

host_name という PC 上で user_id でログオンされている状態で Magic を使用し、5 スレッドのライセンスを使用していることが確認できます。

［このページは意図的に空白にしています。］

第 33 章： 開発環境

特性シートやナビゲータペインが自動的に表示されないようにするには



特性シートの表示方法を変更するには、オプション→設定→動作環境→動作設定の特性シートの自動操作で行います。
ここには、以下の４つのオプションがあります。

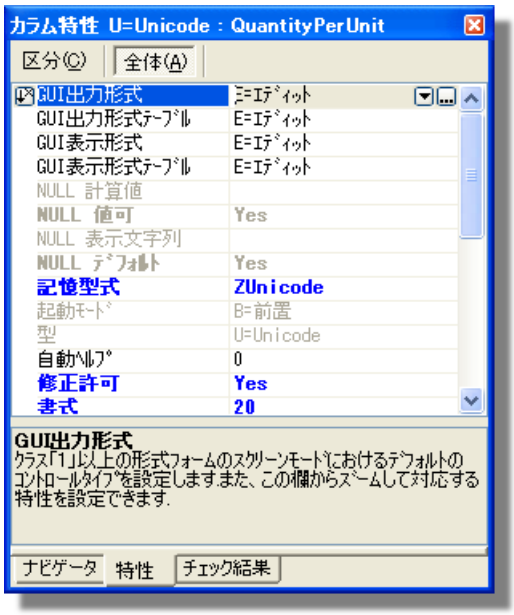
- *N= 無効*……特性シートは自動的に開いたり閉じたりしません。
- *C= 閉じる*……関連しない特性シートが開いている場合、自動的に閉じます。ただし、自動的には開きません。
- *O= 開く*……関連する特性シートが自動的に開きます。ただし、自動的には閉じません。
- *F= 全て*……関連する特性シートが自動的に開き、関連しない場合は自動的に閉じます。

特性シートが自動的に開かないようにし、更に自動的に閉じるようにしたい場合は、*C= 閉じる*を選択します。

また、特性シートの手動で操作したい場合は、*N= 無効*を選択します。

複数のペインを一つに統合するには

Magic の各ペインは、作業のやり方に応じてサイズを変更したり、移動したり、MDI の端にくっつけたりと柔軟に表示させることができます。スペースを節約するために、複数のペインを 1 つのウィンドウに結合させることもできます。あるペインを別のペインに被せるように移動することで結合させることができます。

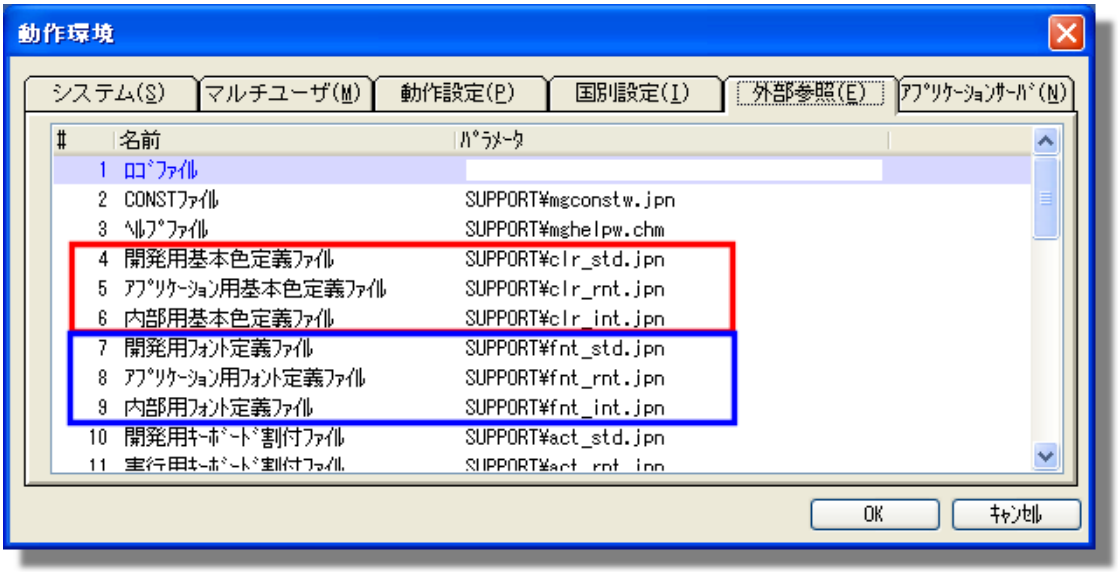


結合されたペインを分離する

1. ペインにフォーカスを移動します。
2. *Ctrl* を押下しながら、ペインのタイトルバーを任意の方向にドラッグします。
3. *Ctrl* を離します。
4. ペインが３つ以上の結合されている場合は、上記の操作を繰り返します。

ペインが分離されます。

使用するフォントや色を変更するには



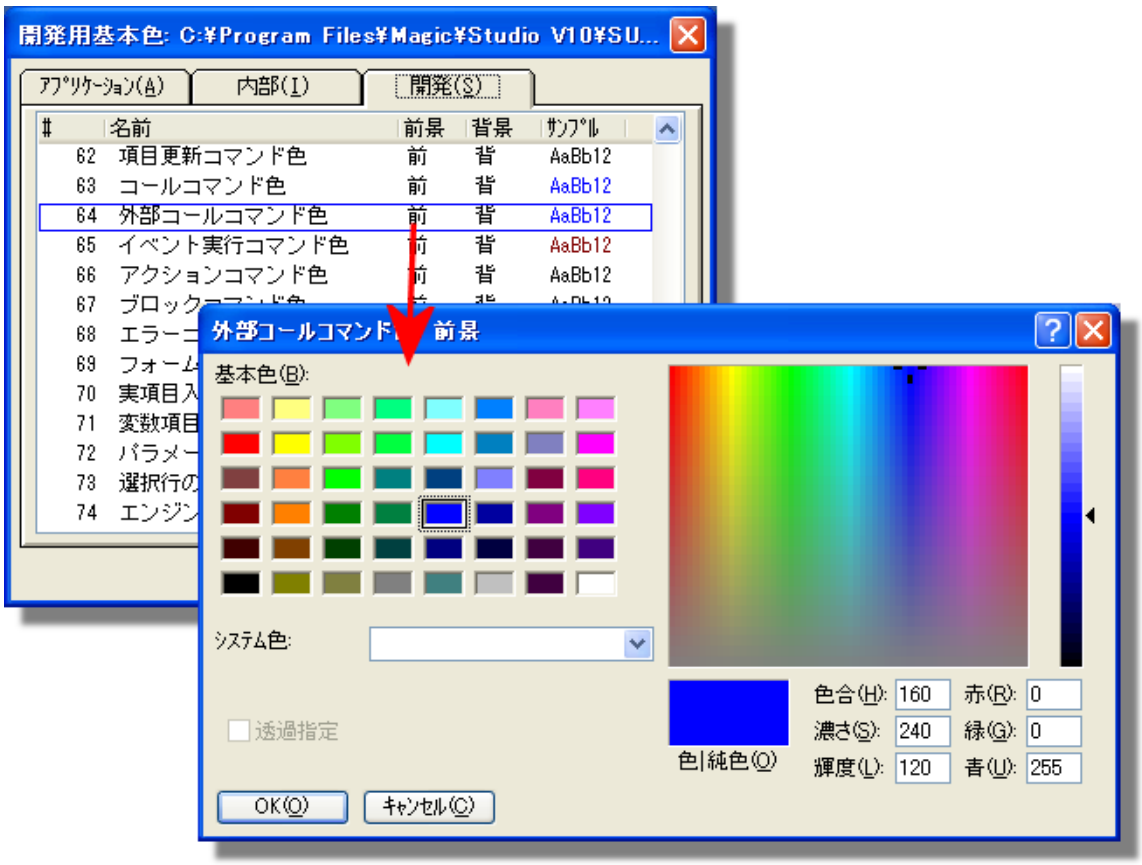
Magic では、基本色定義とフォント定義は３つの別個のファイルに分けて定義されます。

- 開発用：開発用ウィンドウで有効な色とフォント
- アプリケーション用：アプリケーションの作成で使用される色とフォント
- 内部用：実行時に Magic が使用する色とフォント（ポップアップのダイアログやメニューなど Magic の基本構造の一部となっている項目が対象）

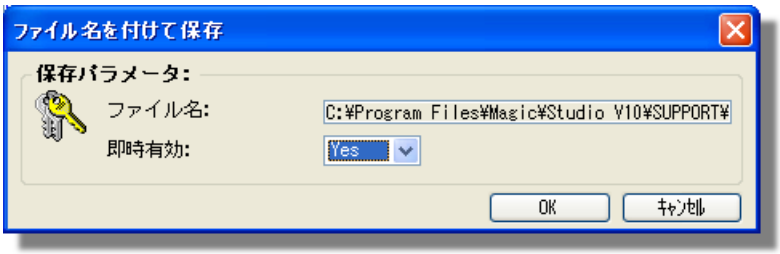
これらの定義ファイルのファイル名やパス名は、Magic.ini に指定されます。上の図に示されるように、これらの設定は、**オプション→設定→動作環境→外部参照タブ**で行うことができます。

作業内容や、ウィンドウの解像度やサイズに合わせて開発用の基本色やフォントをカスタマイズすることができます。例えば、使用する PC がノート型かデスクトップ型のどちらかによって色とフォントを切り替えることで作業し易くなる場合があります。

開発用基本色を変更する

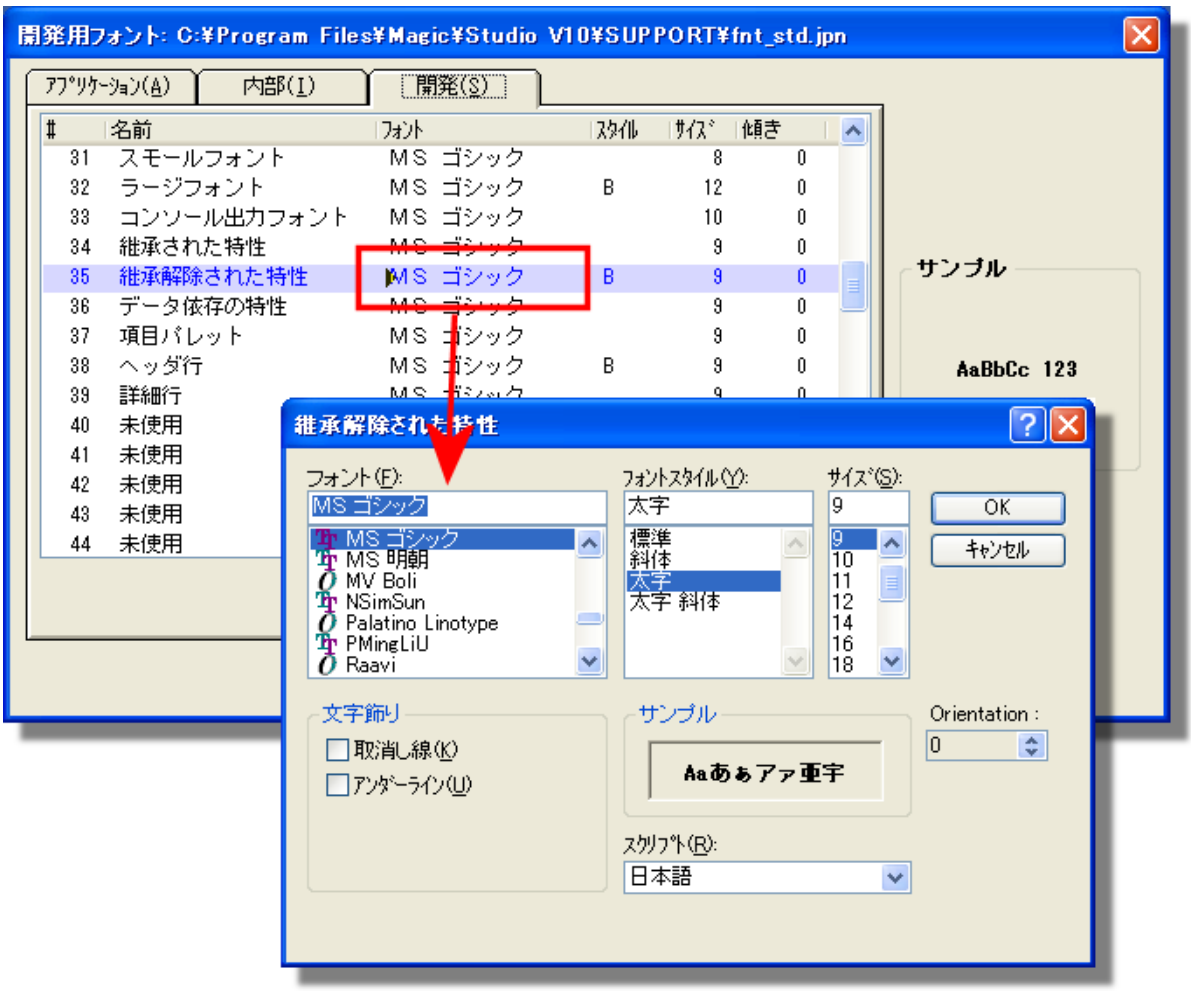


1. オプション→設定→基本色を選択し、基本色テーブルを開きます。
2. 開発タブをクリックします。
3. 変更したい値にカーソルを置きます。例えば、処理コマンド色を変更することでプログラムを読みやすくすることができます。
4. 前景色（前景カラム、文字の色を変更）や背景色（背景カラム、文字の周りの色を変更）からズームして、Windows のカラーパレットを開きます。
5. システム色を選択した場合、OS から引き継がれた色が設定されます。
6. システム色を空白にした場合、基本色から選択したり、カスタマイズした色を設定することができます。
7. 色を設定したら OK をクリックします。
8. 基本色テーブルを終了すると、保存確認ダイアログが表示されます。即時有効を *Yes* に設定し、OK をクリックします。
9. これで、設定された色が有効になります。

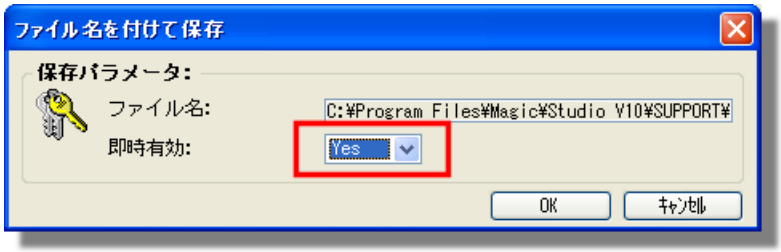


設定された色の情報は、Magic.ini で指定された定義ファイル内に保存され、直ちに有効になります。

開発用フォントを変更する

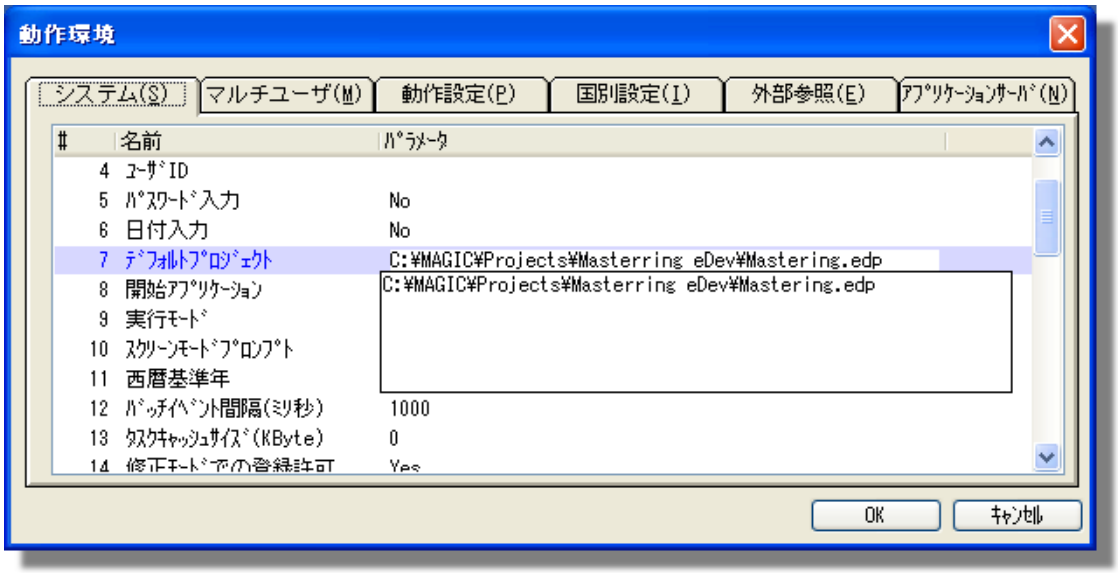


1. **オプション→設定→フォント**を選択し、**フォント**テーブルを開きます。
2. **開発**タブを**クリック**します。
3. 変更したい値にカーソルを置きます。
4. **フォント**カラムから**ズーム**して、**フォント**ダイアログを開きます。ここからフォントやスタイル、サイズを選択することができます。選択された内容にもとづいてサンプル欄に例が表示されます。
5. フォントを設定したら OK を**クリック**します。
6. **フォント**テーブルを終了すると、**保存確認**ダイアログが表示されます。**即時有効**を *Yes* に設定し、OK を**クリック**します。
7. これで、設定されたフォントが有効になります。



設定されたフォントの情報は、**Magic.ini** で指定された定義ファイル内に保存され、直ちに有効になります。

Magic Studio 起動時に、自動的にプロジェクトを読み込ませるには



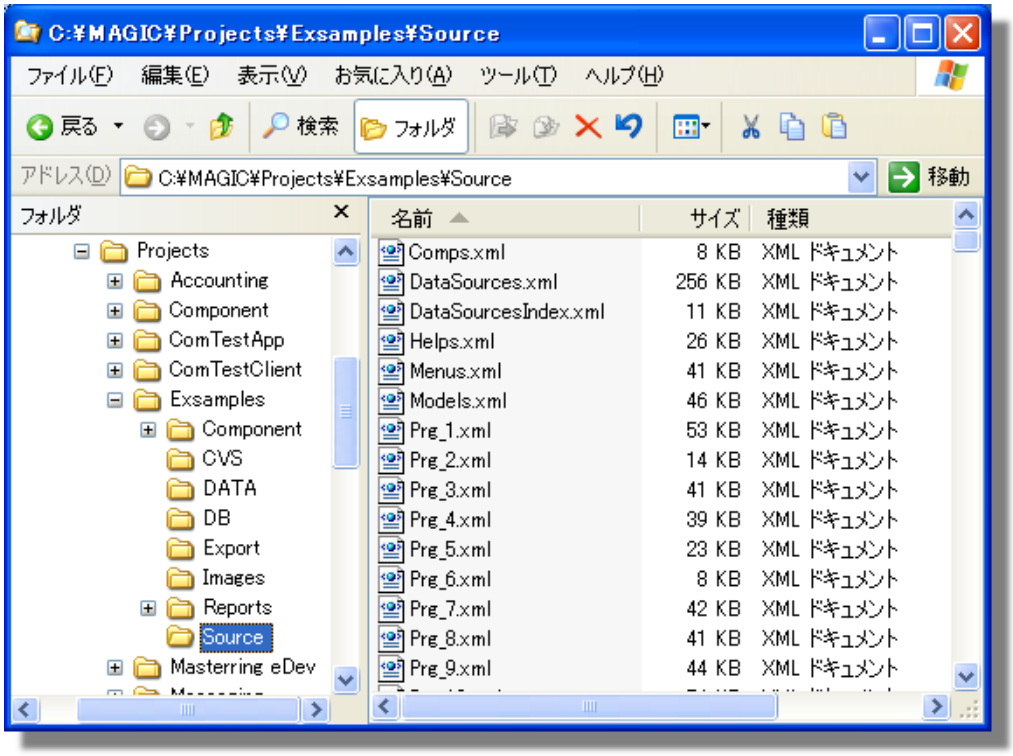
１つのプロジェクトのみを開発している場合、Magic 起動時に自動的にそのプロジェクトのみが読み込まれるようにすることができます。

Magic.ini に、デフォルトプロジェクトを設定することで可能になります。設定方法は以下の通りです。

1. **オプション→設定→動作環境→システムタブ**を選択し、**デフォルトプロジェクト**に移動します。
2. ここから**ズーム**してデフォルトプロジェクトとして使用する ***..edp*** ファイルを選択します。
3. OK を**クリック**します。

これで **Magic Studio** を起動すると、指定されたプロジェクトが自動的にオープンされます。

プロジェクトのソースファイルの格納場所を変更するには



Magic プロジェクトのソースコードは一連の XML ファイルとして保持されます。デフォルトでは、ここに示されているように、**Source** と呼ばれるサブディレクトリ内に格納されます。

ソースファイルの格納ディレクトリは、常に ***.edp*** ファイルに関連した場所に定義されます。この例では、***.edp*** ファイルは C:¥Magic10_Projects¥Exsample にあるため、ソースディレクトリは、C:¥Magic10_Projects¥Exsample¥Source になります。

既存の .edp のソースディレクトリ定義を変更する

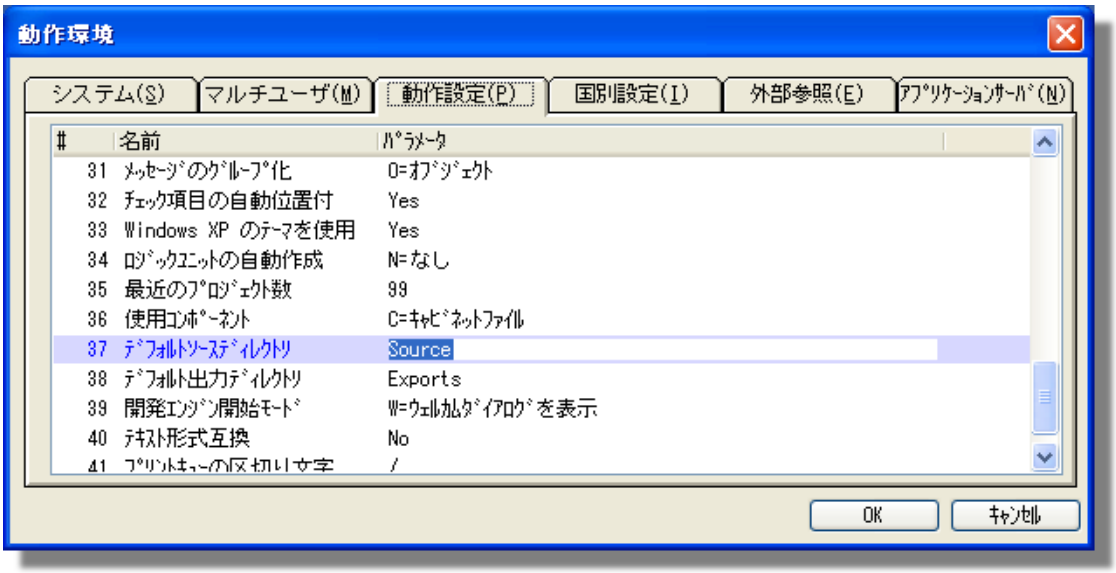
.edp ファイルが作成されると、ソースファイルの位置はその中にコード化されます。この ***.edp*** ファイルは XML ファイルで、編集することができます。この例では、以下の用に設定されています。

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<Application>

  <Project>
    <ProjectName val="Examples"/>
    <VCActive val="N"/>
    <WorkOffline val="N"/>
    <SourceDirectory val="Source"/>
    <ExportsDirectory val="Exports"/>
    <GUID val="{A0275EFF-1939-49CC-970A-98BA38F752A1}"/>
    <ReferencedProjects>
      <RefProject FileName="Examples¥MyMessagingComponent¥MyMessagingComponent.Edp"
ProjectName="MyMessagingComponent"/>
    </ReferencedProjects>
  </Project>

</Application>
```

この設定を変更することで、ソースファイルの格納先を変えることができます。

デフォルトのソースディレクトリを変更する

今後作成するプロジェクトに対するデフォルトのソースディレクトリを変更することができます。この設定は、**Magic.ini** ファイルの内の DefaultSourceDir の設定で行います。



この内容は、**設定→オプション→動作環境→動作設定タブ**の**デフォルトソースディレクトリ**でも変更することができます。変更された設定内容は、**Magic Studio** が再度起動された時点で有効になります。

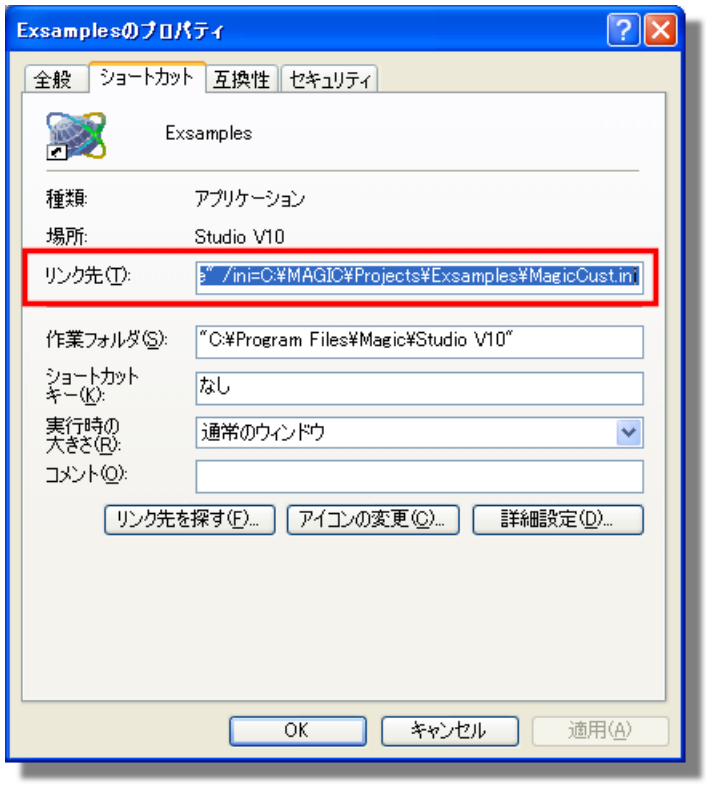
Magic.ini を指定して起動するには

デフォルトでは、***.edp*** ファイルをクリックして Magic を起動した場合、***.edp*** ファイルと同じディレクトリに置かれた *Magic.ini* と呼ばれるファイルを使用します。このファイルが存在しない場合、Magic のインストールディレクトリ内に置かれた **Magic.ini** ファイルが使用されます（Magic Studio の起動メニューをクリックした場合は、Magic のインストールディレクトリ内のファイルが使用されます）。

しかし、任意の場所に格納されている **Magic.ini** を使用して起動させることもできます。この方法は、デフォルトのログイン環境と異なるユーザ（評価担当者や開発者など）用に個別のログイン環境を提供する場合などに使用されます。

この場合、以下のように設定します。

異なる Magic.ini ファイルを使用する



1. 起動用のアイコンを作成またはコピーします。
2. リンク先には、使用したい Magic.ini を読み込むように指定します。この場合、以下の構文で指定できます。

   <eDeveloper .exe file> /INI=<ini file name>

例えば以下のように設定できます。

```
"C:¥Program Files¥MSE¥eDeveloper 10.1¥eDevStudio.exe" /
INI=C:¥eDeveloper10_Projects¥MagicCust.ini
```

バッチファイルやその他のスクリプトを使用して指定することもできます。

Magc.ini の環境情報を追加指定するには

デフォルトでは、***.edp*** ファイルをクリックして Magic を起動した場合、***.edp*** ファイルと同じディレクトリに置かれた *Magic.ini* と呼ばれるファイルを使用します。このファイルが存在しない場合、Magic のインストールディレクトリ内に置かれた **Magic.ini** ファイルが使用されます。

しかし、Magic.ini の特定の項目だけを変更したり追加して Magic を起動することができます。例えば特定の基本色定義ファイルを指定したり、評価のために別のファイルを使用する場合にこの方法が利用できます。

この場合、以下のように設定します。

Magic.ini の設定をオーバライドする

最初に、変更したい情報だけを格納したファイルを作成します。各項目の先頭には、/（スラッシュ）を設定する必要があります。

例えば、以下のように設定します。

```
/[MAGIC_ENV]InputPassword=N
/[MAGIC_ENV]User = Supervisor
/[MAGIC_ENV]Password =
/[MAGIC_ENV]StartApplication=1
/[MAGIC_ENV]ColorDefinitionFile=usrclr2.jpn
/[MAGIC_LOGICAL_NAMES]TEMP=C:¥temp¥
/[MAGIC_LOGICAL_NAMES]Testing=Y
```

この例では、ユーザ ID やパスワードなどのデフォルト設定のいくつかが無効にしています。プロジェクトの開発時には、これによって時間の節約になります。

スラッシュが付加された各アイテムは、[]（ブラケット）内に Magic.ini のセクション名を指定します。

オーバライドを使用する

次に、Magic 起動時にこの設定ファイルを読み込ませる指定が必要です。以下の構文で指定します。

```
<eDeveloper .exe file> @<Override File Name>
```

例えば以下のように設定できます。

```
"C:¥Program Files¥MSE¥eDeveloper 10.1¥eDevStudio.exe" @C:¥eDeveloper10_Projects¥Testing.ini
```

これで、Testing.ini で設定された値によって、Magic.ini に設定されている値がオーバライドされます。

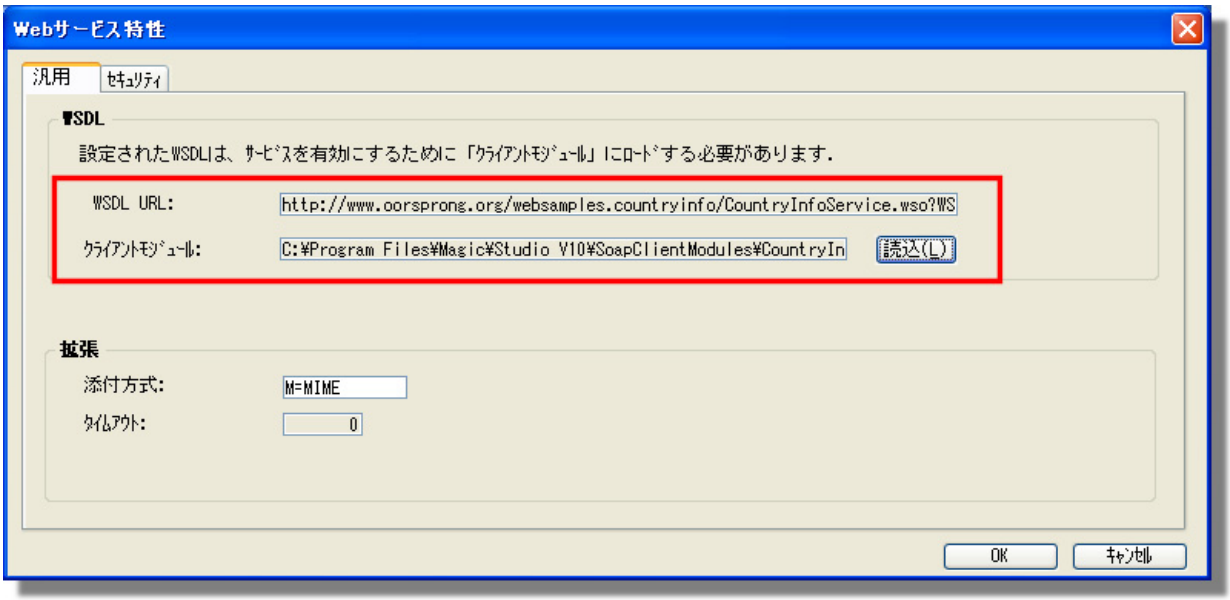
第 34 章： Web サービス（コンシューマ）

Web サービスにアクセスするには

Web サービスの接続情報は、WSDL（Web Service Description Language）と呼ばれる特別な XML ファイルで提供されます。WSDL には、入出力の構造を含むサービス仕様が記述されています。Web サービスにアクセスするためには、最初に WSDL を見つける必要があります。一般的には、Web 上の URL を指定することになります。Magic から Web サービスにアクセスするには、サービステーブルに SOAP 入力を定義し、WSDL をロードする必要があります。WSDL がロードされると、クライアントモジュール（JAR ファイル）が作成され、XML スキーマがオプションで作成されます。

注： Magic eDeveloper V10 で Web サービスにアクセスするには、Systinet Server for Java が必要です。

Web サービスクライアントを作成する

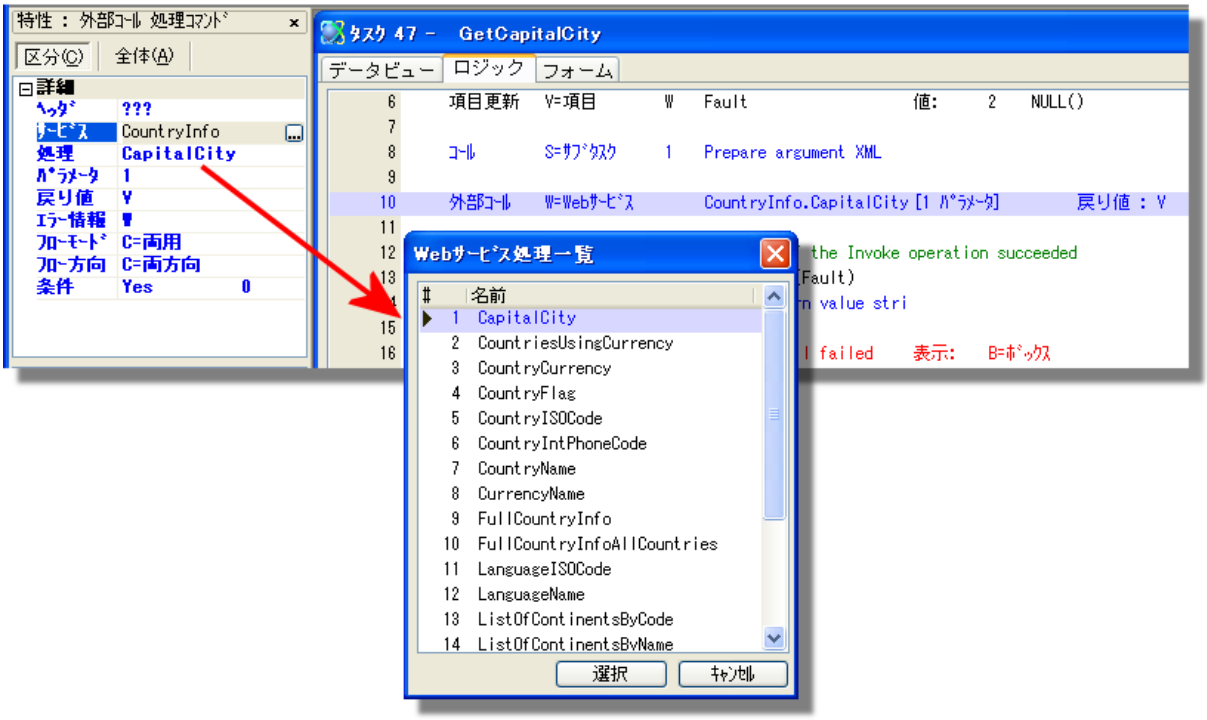


1. サービステーブルを開きます（オプション→設定→サービス）。
2. *F4*（編集→行作成）を押下して 1 行追加します。
3. Web サービス名を入力します。この例では、CountryInfo と入力されています。
4. サーバカラムで、ズームして ***SOAP*** を選択します。
5. *Alt+Enter* を押下してサーバ特性を開きます。
6. WSDL URL 特性にアクセスする WSDL の URL を入力します。
7. 指定された WSDL の内容が有効なものであれば、クライアントモジュール特性に、モジュールファイル名が自動的に設定されます。
8. 読込ボタンをクリックします。クライアントモジュールと XML スキーマファイルの作成処理が実行します。

またサービス特性では、添付ファイルのエンコードタイプやセキュリティの設定を行うことができます。

注： Jar ファイルの作成には、Java の SDK がインストールされている必要があります。また、Magic の実行版では、Jar ファイルの作成はできません。開発環境で作成された Jar ファイルを実行環境にコピーし、**Web サービス特性**の**クライアントモジュール特性**でファイルを指定する必要があります。

Web サービスにアクセスする



Web サービスが定義されると、Magic から簡単にアクセスすることができます。

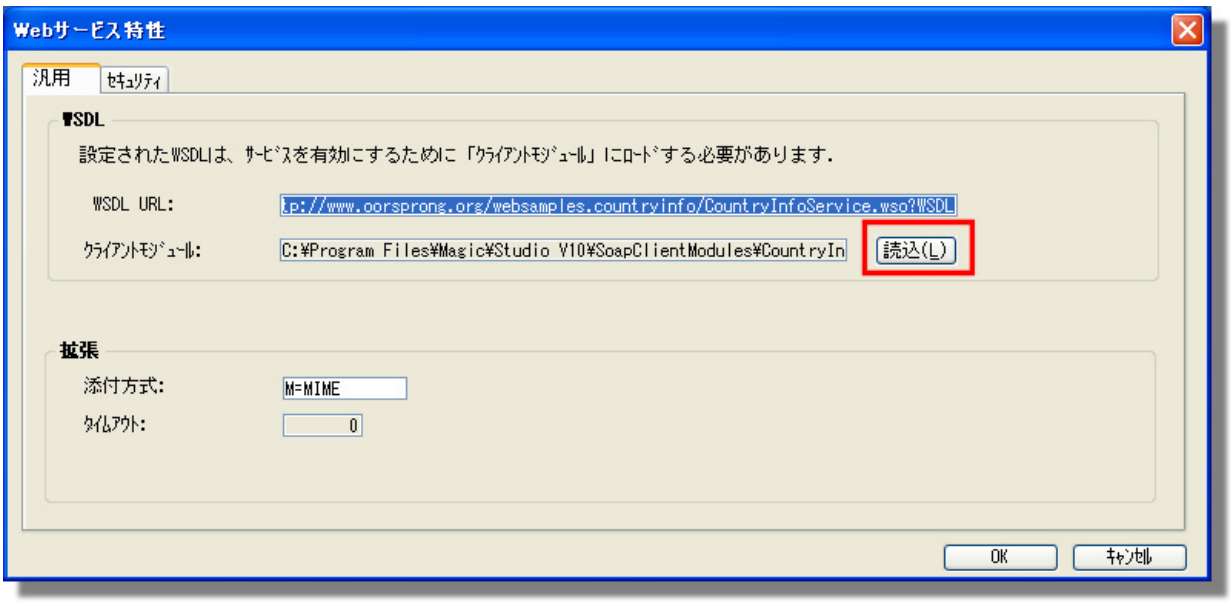
1. *F4*（**編集→行作成**）を押下して 1 行追加します。
2. *I* を入力して**外部コール**処理コマンドを選択します。カーソルが次のフィールドに移動します。
3. *W=Web サービス*がデフォルトで選択された状態になります。次のフィールドに *Tab* 移動します。
4. *Alt+Enter* を押下して、**特性**シートを開きます。
5. **サービス**特性から**ズーム**してアクセスしたいサービスを選択します。この例では、*CountryInfo* が選択されています。
6. **処理**特性から**ズーム**してアクセスしたい Web サービスの処理を選択します。
7. **エラー**特性から**ズーム**して（BLOB 型か文字型の）データ項目を選択します。ここには、Web サービス実行後のエラーコードが格納されます（処理が正常に終わった場合は、空白が返ります）。
8. **パラメータ**特性から**ズーム**して Web サービスへのパラメータを設定します。パラメータの内容は、アクセスする Web サービスによって異なりますが、**パラメータ**テーブルには設定すべきパラメータの内容が表示されます。パラメータの中には、XML ドキュメントで定義された複合型のものがありますが、そのような場合、XML スキーマファイルのパスが表示されます。
9. **戻り値**特性から**ズーム**して、Web サービスからの戻り値を格納する項目を指定します。

これで設定は終了しました。複合型のパラメータを使用して Web サービスにアクセスする場合は、XML データを BLOB 化して渡す処理が必要になります。

Web サービス定義の再読込を行うには

Web サービスの内容が変更された場合、変更内容を反映させたり、新しいサービスを利用するようにアプリケーションを変更する必要が発生します。ただし、このサービスをすでに使用している場合、サービスを削除したり再作成しないでください。サービスを再読込するだけで対応できます。

既存のサービスを再読込する



1. サービステーブルを開きます（オプション→設定→サービス）。
2. 再読み込みしたいサービス名にカーソルを置きます。
3. *Alt+Enter* を押下してサーバ特性を開きます。
4. 読込ボタンをクリックします。

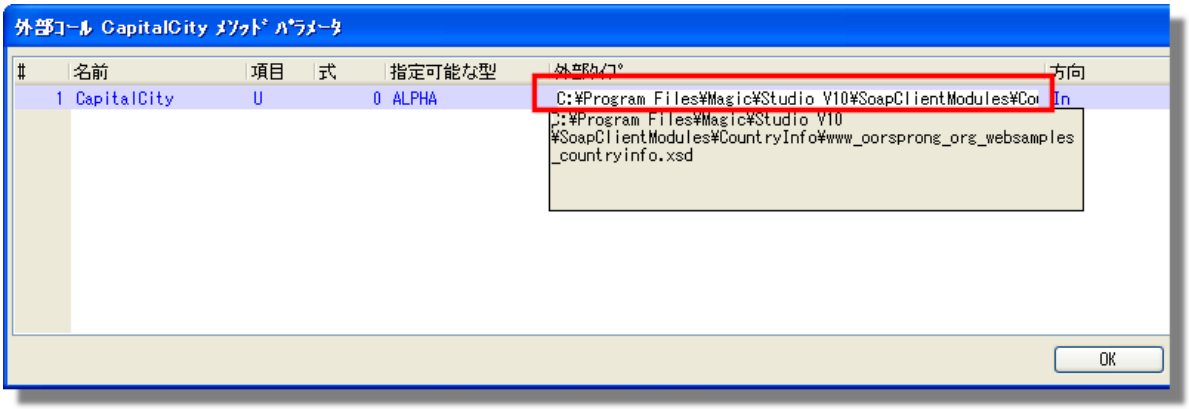
XML 定義も変更された場合、XML スキーマから XML データソース定義を再読込する必要があります。

複雑なパラメータの送受信を行うには

SOAP サービスの中には、Magic のデータ項目を定義するだけで（Magic 内で変換することで）簡単にパラメータを渡すことができるものもありますが、ほとんどの SOAP サービスは複雑なパラメータ構造を持っており、XML ドキュメント形式で渡すために XML データソースを使用する必要があります。

WSDL が複雑なパラメータを使用しているかどうかを確認する

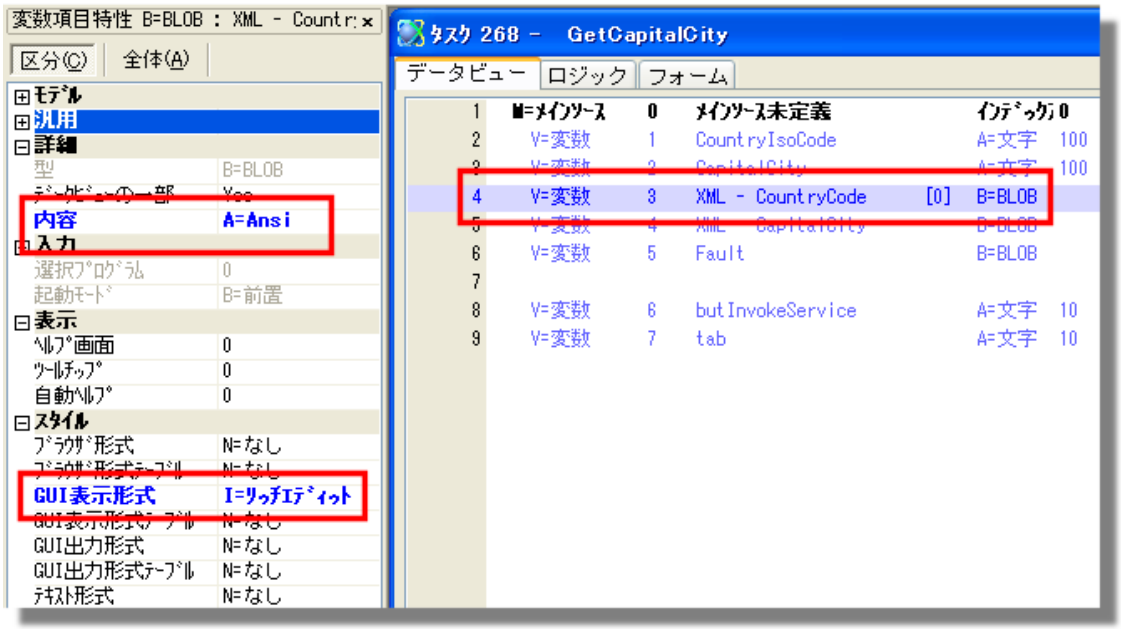
WSDL が複雑なパラメータを使用している場合、データの送受信のために XML ファイルを設定する必要があります。



外部タイプカラムがが XML スキーマ（XSD）ファイルを示すパスを含んでいる場合、XML データが必要であると判断できます。

Web サービスを定義した時点で、XML スキーマファイルが PC に作成されます。この XML スキーマを使用して、XML データソースを作成する方法については、第 14 章：「XML ビューを作成するには」（298 ページ）を参照してください。

BLOB 項目を定義する



最初に、XML ドキュメントを格納する項目を定義する必要があります。この場合、BLOB 型の項目のみ定義できます。テキストが含まれる場合、**内容**特性は ***A=Ansi*** か ***U=Unicode*** に設定してください。

GUI 表示の**スタイル**特性が ***I= リッチエディット***に設定されている場合、この内容をウィンドウに表示させることができます。

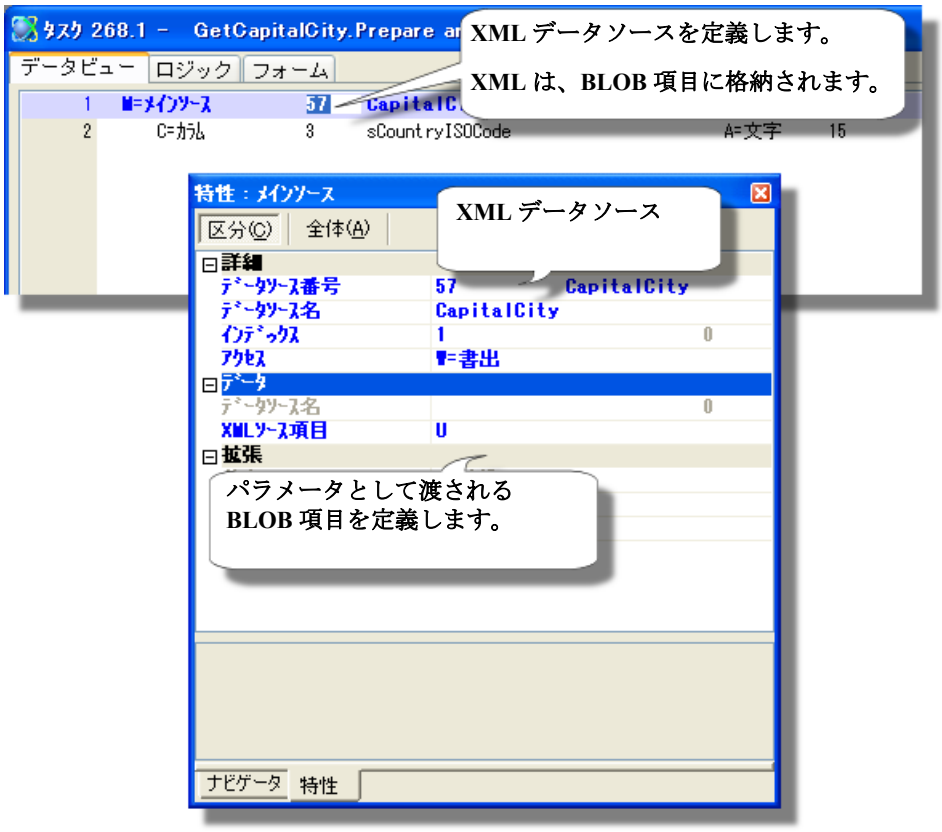
XML BLOB の作成 / 読込を行う

XML BLOB を作成したり、読み込んだりする場合、最初に XML スキーマにもとづいて、XML データソースを設定する必要があります。XML データソースの作成方法は、第 14 章：「XML ビューを作成するには」（298 ページ）を参照してください。

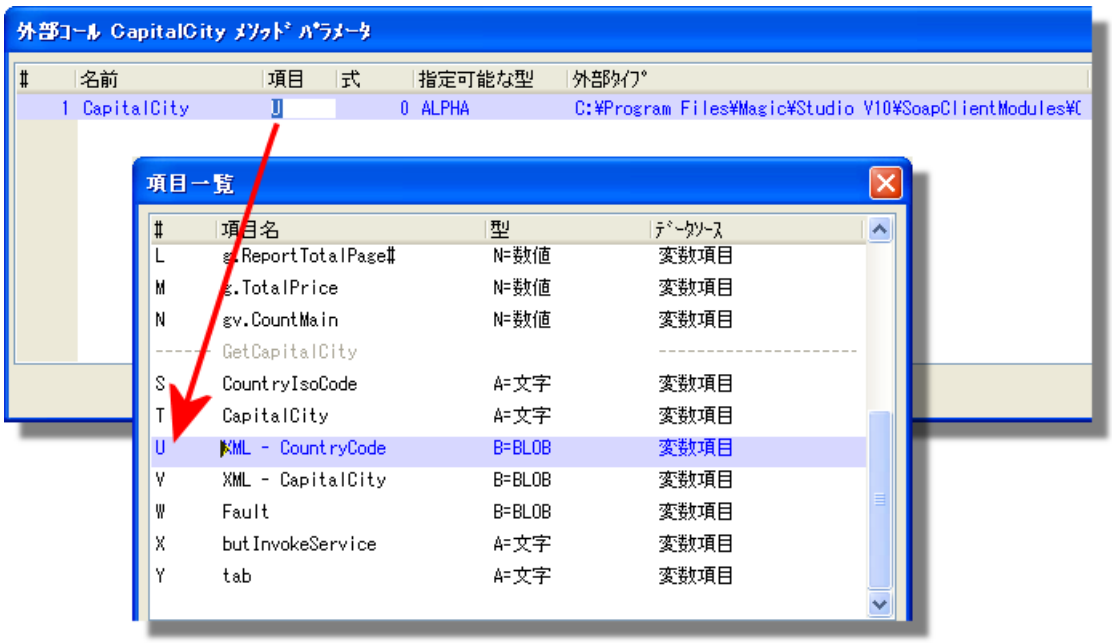
XML データソースが定義されたら、この XML ファイルにデータを格納することができるようになります。

1. XML ファイルを作成するためのサブタスクを呼び出す前に、XML データの内容を初期化するために BLOB 項目を **NULL()** 関数で更新します。
2. データを BLOB 項目に格納するために、サブタスクを呼び出します。このサブタスクのメインソースは、XML データソースになります。**XML ソース項目**特性には、データを書き込む XML BLOB を定義します。
3. このデータソースは他のデータソースと同じように扱うことができます。1 行分のデータを送るだけであれば、サブタスクでは 1 レコード分のみを作成するようにします。この例では、更新されるデータ項目は 1 つ（country code）だけなので、この項目に国コードを設定します。

   簡単なデータ要素が含まれている XML ドキュメントの処理は、関連するデータ要素を更新するだけで対応できます。
4. XML BLOB の作成処理は自動的に行われます。XML データソース定義にもとづいて XML 形式でデータを作成し、BLOB 項目に格納します。



パラメータと BLOB 項目を使用する



次に、BLOB 項目を Web サービスパラメータに定義します。

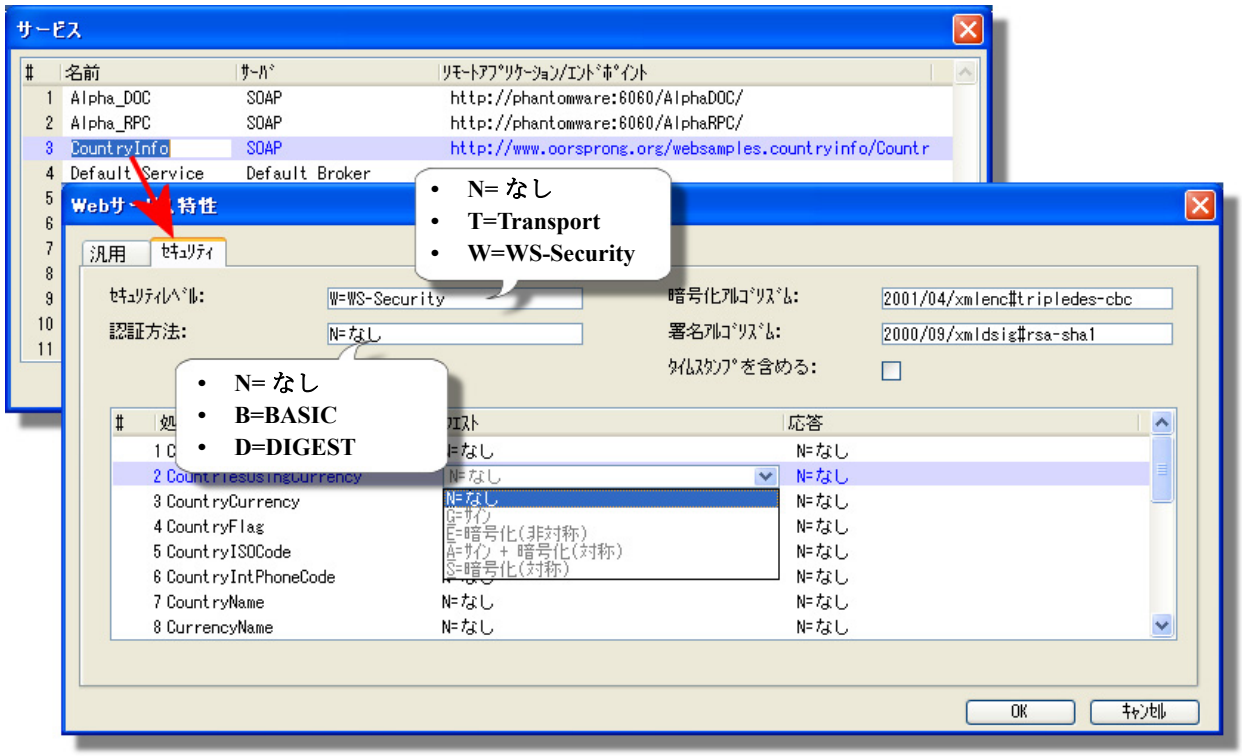
Web サービスに安全にアクセスするには

Web サービスにアクセスする場合、必要なセキュリティレベル（Web サービスプロバイダによって定義されるような）を Magic で設定する必要があります。

セキュリティの設定には２つのレベルがあります。最初に、サービス全体に対するセキュリティレベルを設定します。そしてプログラムで Web サービスを使用する場合、必要であればユーザ ID とパスワードを指定することができます。

注： セキュリティの設定が行われているプロバイダ側のサービスに対してのみ有効です。プロバイダ側の設定内容と同じセキュリティを設定する必要があります。プロバイダ側の設定については、第 35 章 :「デプロイされたサービスに対するセキュリティを設定するには」（725 ページ）を参照してください。

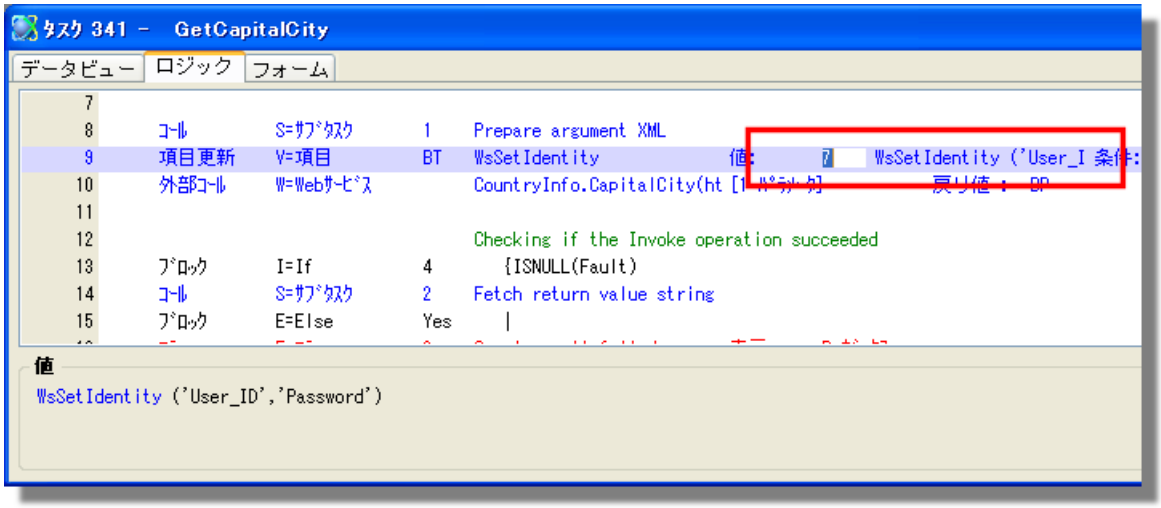
サービスにセキュリティを設定する



1. **サービス**テーブルを開きます（**オプション→設定→サービス**）。
2. セキュリティレベルを設定したいサービス名にカーソルを置きます。***Alt+Enter*** を押下して**サーバ特性**を開きます。
3. 必要なセキュリティレベルを設定します。以下のテーブルにオプションが示されています。セキュリティレベルが ***Transport*** と設定された場合、通信チャンネルが保護されます。また、***WS-Security*** が選択された場合、メッセージは、暗号化または暗号化とデジタル認証の両方が使用されて保護されます。（WSDL で定義された）サービスへのアクセスポイントが（https の URL で）保護される場合、通信は更に保護されます。

セキュリティレベル	認証方式	暗号化アルゴリズム	処理
N= なし	無効	無効	
T=Transport	N= なし B=BASIC D=DIGEST S=SSL K=Kerberos	無効	無効
W=WS-Security	N= なし B=BASIC D=DIGEST	暗号化と認証方式にもとづいて、いくつかの選択肢があります。	N= なし S= サイン E= 暗号化（非対称） A= サイン＋暗号化（対称） S= 暗号化（対称）

タスクでセキュリティを設定する



認証を必要とする Web サービスにアクセスする場合、プログラム内で **WsSetIdentity()** 関数を使用することで Web サービスプロバイダに自身を識別させることができます。**セキュリティレベル**が ***Transport*** や ***WS-Security*** に設定されているサービスにアクセスする場合、**WsSetIdentity()** 関数を使用してプロバイダ側のユーザ ID とパスワードが設定されると、同じ関数が再度実行されるまで指定された同じユーザ ID とパスワードが**コール Web サービス**処理コマンドで使用されます。

注： アプリケーション内でハードコード化されたユーザ証明を行わないように、ユーザ ID やパスワードを固定値ではなくデータ項目に値を格納して使用することを推奨します。

Web サービスアタッチメントを使用するには

Web サービスに WSDL で定義されたアタッチメント（添付ファイル）を持っている場合、Web サービスのパラメータリストに表示され、BLOB 項目を割り当てることができます。

Web サービスが、WSDL で定義されていないアタッチメントの送受信を行う場合は、以下の関数を使用してください。

添付ファイルを Web サービスプロバイダに送信する

添付ファイルを Web サービスプロバイダに送信するには、**WsConsumerAttachmentAdd()** 関数を使用します。関数の構文は以下の通りです。

WsConsumerAttachmentAdd(*Attachment*)

パラメータ：

- ***Attachment***：添付ファイルが格納されている BLOB 型項目

この関数は、UUID を表す文字列が返ります。

Web サービスプロバイダからの受信データから添付ファイルを取り出す

Web サービスプロバイダから受信されたリクエストから添付ファイルを取り出すには、**WsConsumerAttachmentAdd()** 関数を使用します。関数の構文は以下の通りです。

WsConsumerAttachmentGet(*ID*)

パラメータ：

- ***ID***：プロバイダ側で添付した際に発行された UUID か、1 から始まるインデックス値

この関数は、添付ファイルが格納された BLOB 値が返ります。

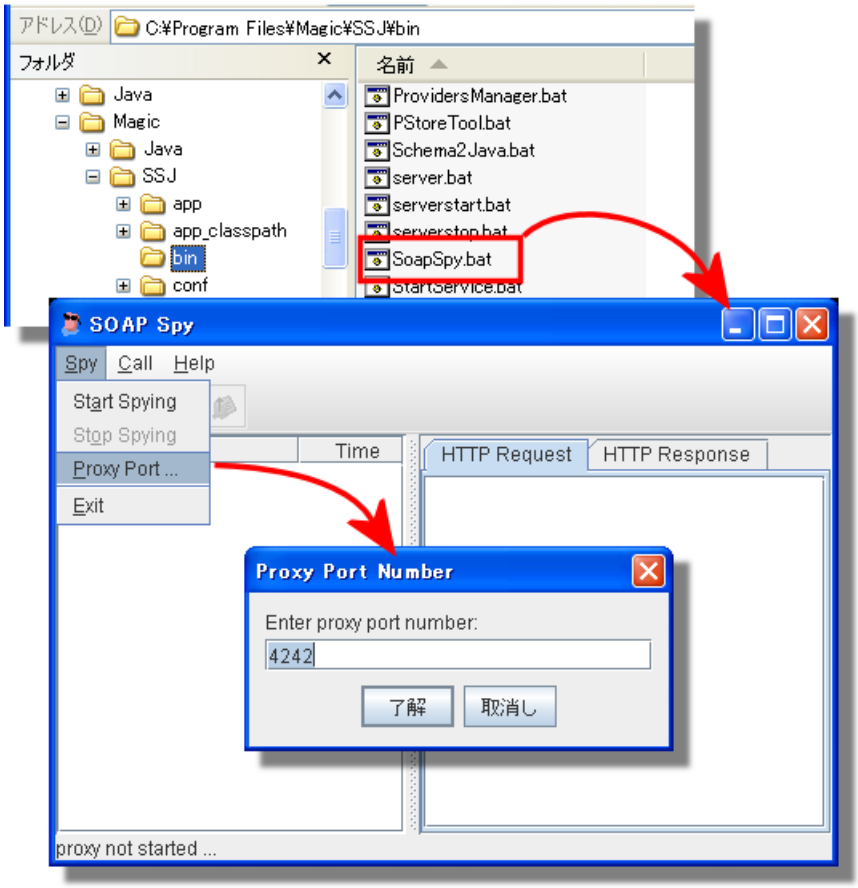
参照： これらの関数の詳細は、『リファレンスヘルプ』を参照してください。

Web サービスへの呼び出しをトレースするには

Magic が Web サービス用のフレームワークとして使用している Systinet Server for Java には、プロバイダとのアクセス内容をトレースする SOAP Spy というユーティリティがあります。Web サービスの呼び出し機能のテストやトラブルシューティングにおいて、この機能を利用することができます。

SOAP Spy を起動する

Systinet Server for Java は、Magic と一緒にインストールした場合、Magic のインストールフォルダと並列レベルの **SSJ** フォルダにインストールされます。SOAP Spy は、SSJ¥bin サブフォルダ内の SoapSpy.bat をクリックすることで起動することができます。



SOAP Spy のトレースを開始する

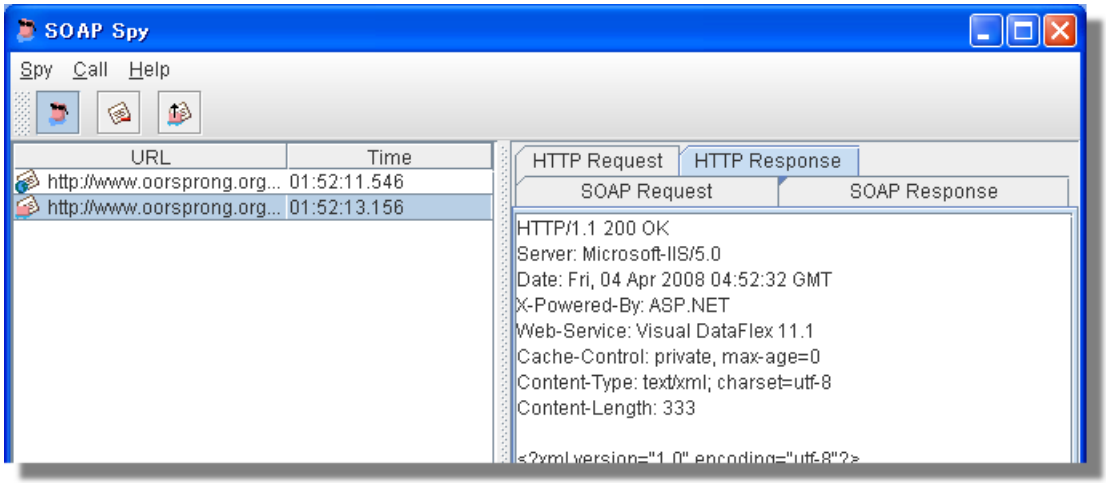
1. **Proxy Port** ダイアログ（**Spy → Proxy Port**）を開き、トレース用のポートを設定します。デフォルト値は、***4242*** になっています。
2. 了解をクリックします。
3. **Spy → Start Spying** を選択するとトレースが開始されます。

Magic 側で SOAP Spy を有効にする

動作環境ダイアログ（**設定→オプション→動作環境→動作設定タブ**）を開き、**SOAP Spy アドレス**に Proxy Port の番号を設定します。

注： この設定を行った場合、SOAP Spy のトレース機能が開始されていない状態でコール Web サービス処理コマンドを実行するとエラーになります。

これでプログラムを実行し、コール Web サービス処理コマンドが起動されると、SOAP Spy ウィンドウにパケットの内容が表示されます。



［このページは意図的に空白にしています。］

第 35 章： Web サービス（プロバイダ）

Magic で Web サービスを提供するには

Web サービスは、異なるプラットフォームやフレームワークで動作するアプリケーション間で相互運用するための標準的な方法です。Web サービスを利用することで、Magic アプリケーションを外部に公開することができます。Web サービスを使用することで、世界のあらゆる場所にある様々な言語で作成された、色々なタイプのアプリケーションとアクセスすることができます。しかし、このサービスを手動で作成するには非常に手間がかかる場合があります。Magic を使用することで、このようなサービスの作成が自動化され作業が容易になります。

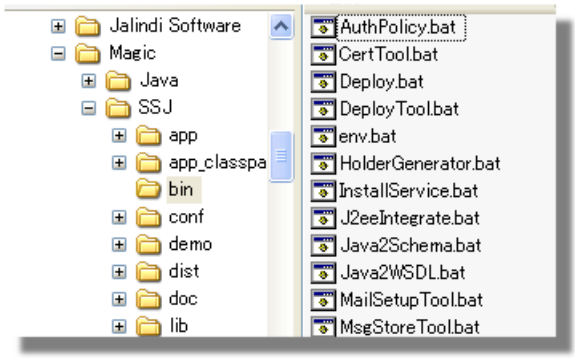
Magic プロジェクトを Web サービスとして提供することができます。Web サービス用のプログラムは、バッチタスクにパラメータと戻り値を定義することで作成することができます。これらのプログラムが Web サービスで使用できるように、Magic はインタフェースを作成します。Web サービスを提供するための手順を以下で説明します。

Magic のセットアップ

Web サービスを作成する前に、Magi が適切にセットアップされていることを確認してください。Magic をインストールする際に、Web サービスのフレームワークをコンポーネントとして選択する必要があります。

Magic Studio のインストール処理では、Java の SDK もインストールされます。これは SOAP サービスで必要な Java コンポーネントをコンパイルするため使用されます。Magic Client や Magic Enterprise Server の場合は、JRE がインストールされます。

この後、Systinet Server for Java がインストールされます。これは、実際に SOAP サービスをデプロイするために使用されます。

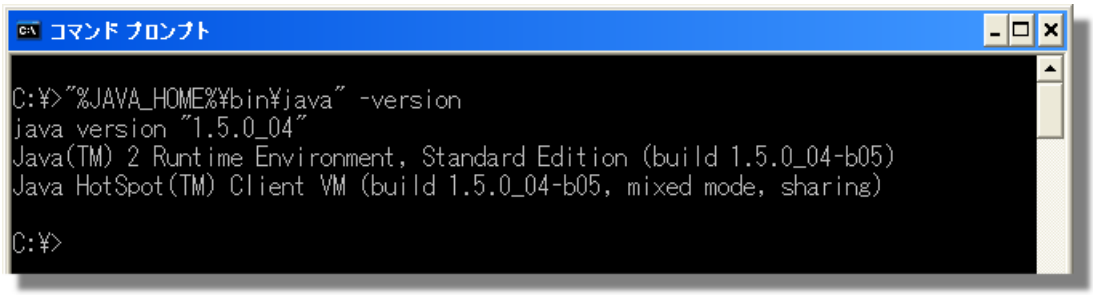


注： JRE のみがインストールされた環境では、Magic 側でデプロイさせることはできません。

Java の確認

これらの製品がすでに PC にインストールされている場合は、Magic はすでにインストールされたバージョンを使用します。インストールされていない場合は、、Magic ディレクトリ内にサブディレクトリを作成してインストールされま

す。このように、Web サービスをデプロイするには、上記の 2 つの Java 関連の製品がインストールされ実行可能な状態になっている必要があります。



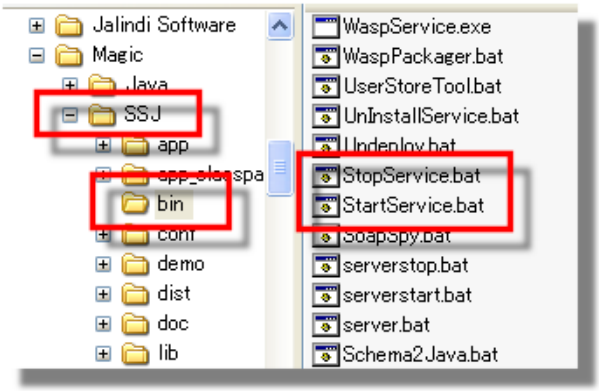
Java のインストール状況を確認するには、以下のコマンドを実行します。

```
"%JAVA_HOME%¥bin¥java" -version
```

JAVA_HOME 環境変数が正しく設定されており、Java SDK がインストールされる場合は、そのバージョンが表示されます。

Systinet の確認

Systinet がインストールされると、Windows の**スタート**メニューにサービスの起動や停止、Systinet コンソールの表示用のメニューが作成されます。



SSJ ディレクトリ内にバッチファイル（StartService.bat と StopService .bat）があります。サーバを起動するには、以下のようにコマンドラインを実行します。

```
%WASP_HOME%¥bin¥serverstart.bati
```

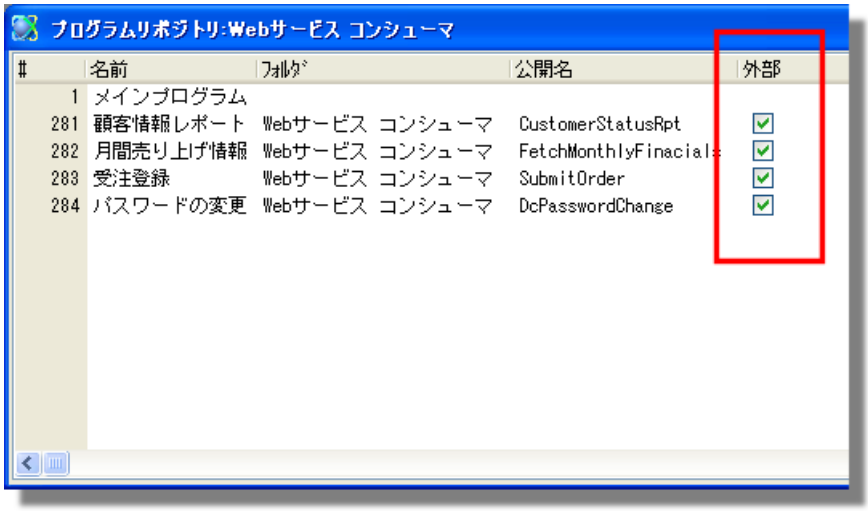
サービスが起動されると、Web ブラウザで以下の URL を入力することで Systinet コンソールを表示させることができます。

```
http://localhost:6060/admin/console
```

コンソールの操作方法については、この後説明します。ここでは、Systinet for Java がインストールされていることのみを確認します。コンソールが正しく表示されれば、正しくインストールされ Web サービスをデプロイする用意ができたことになります。

Magic アプリケーションのデプロイ

1. バッチプログラムを作成する



最初に、提供したいプログラムにユニークな**公開名**が定義され、**外部**カラムがチェックされていることを確認する必要があります。これらは、バッチプログラムでなければなりません。プログラムは、**データビューエディタ**に定義されたパラメータ項目を使用して値の受け渡しを行ったり、**戻り値**特性（**タスク特性→汎用タブ**）を使用して値を１つ返すことができます。これらの値は簡単なデータ項目であったり、複雑な XML データを含んでいる BLOB 項目の場合もあります。

2.Systinet server を起動する

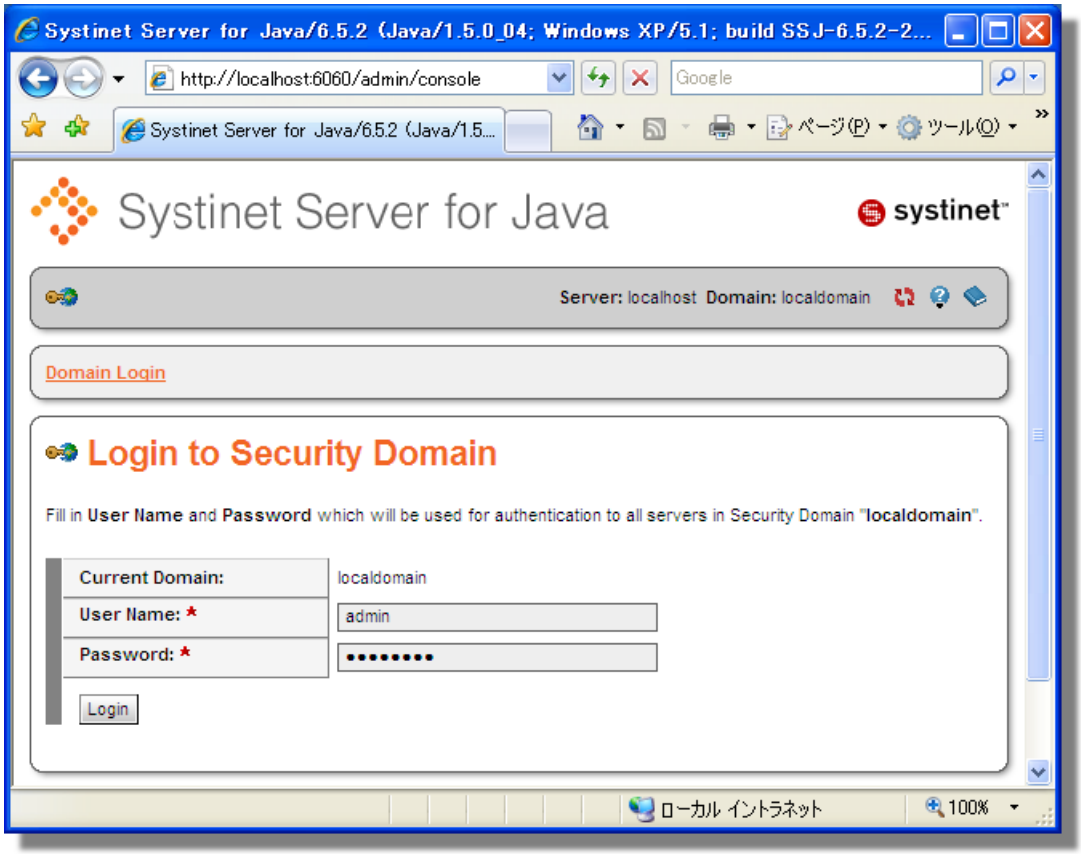
Systinet Server を起動します。Systinet Server for Java という Windows のサービス名を起動するか、スタートメニューの **Start Systinet Server** アイコンをクリックするか、以下のバッチプログラムを使用することで起動できます。

```
%WASP_HOME%¥bin¥serverstart.bat
```

*Systinet コンソール*を起動することで Systinet Server が実行しているかどうかを確認できます。**Systinet Server Adminstratoin Console** のアイコンをクリックするか、Web ブラウザから以下の URL を入力します。

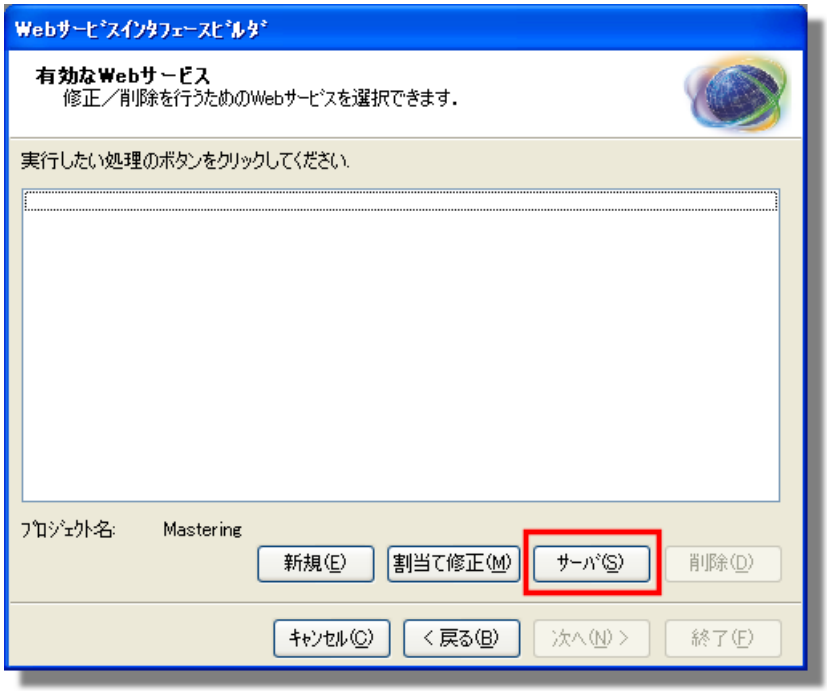
```
http://localhost:6060/admin/console
```

Systinet Server が起動されていない状態で Magic からサービスをデプロイした場合、エラーが発生します。



デフォルトのユーザ ID は ***admin***、パスワードは ***changeit*** です。

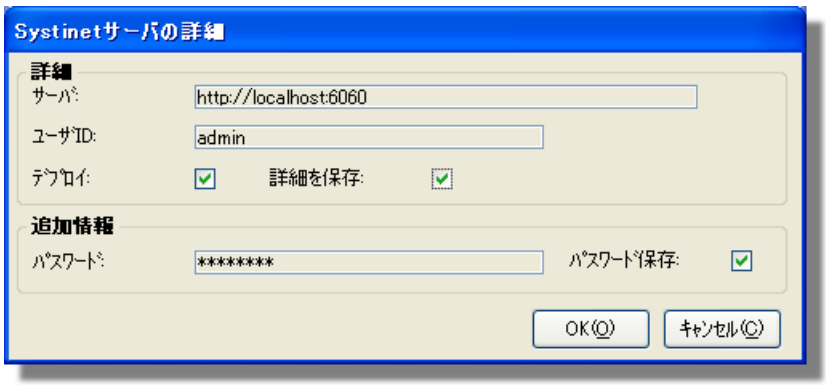
3.Web サービスインタフェースビルダを起動する



プロジェクトが開いている状態で、**オプション→インタフェースビルダ→ Web サービス**を選択します。**ウェルカム**ダイアログが表示されます。次へを**クリック**します。

有効な Web サービスダイアログには、現在定義されている Web サービスの一覧が表示されます。まだ何も作成していない場合は、一覧には何も表示されません。サーバボタンをクリックします。

4. サーバの詳細ダイアログ



Systinet サーバの詳細ダイアログでは、サーバの位置とログイン情報を入力する必要があります。デフォルトの設定は以下の通りです。

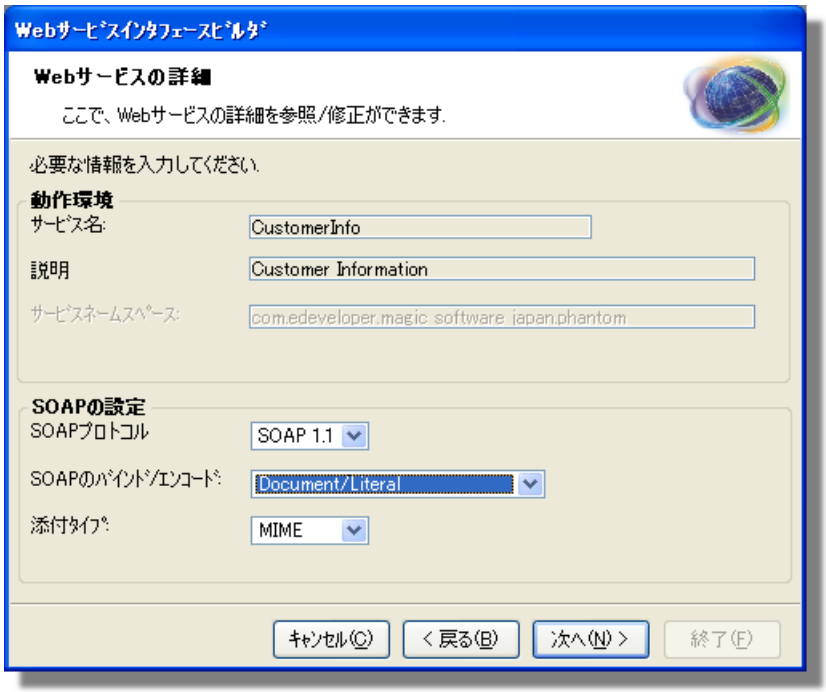
- サーバ：*http://localhost:6060*
- ユーザ ID：*admin*
- パスワード：*changeit*

注： サービスを定義するたびにパスワードの入力要求が発生することを避けたい場合は、パスワード保存のチェックボックスをチェックしてください。

デプロイをチェックすると、サービスが作成された後に自動的にデプロイされます。

次に、Web サービスリストから新規をクリックします。Web サービスの詳細ダイアログが表示されます。

5.Web サービスの詳細ダイアログ



ここで、サービス名や説明、およびネームスペースを設定します。ここには、ユーザに対して意味のあるテキストを指定してください。

- SOAP プロトコルは、*SOAP1.1* が最も互換性の高い設定です。

- **SOAP のバインディング / エンコード**では、*Document/Literal*、*RPC/literal*、*RPC/Encoded* のどれかを選択します。
- **添付タイプ**は、*MIME* か *DIME* を選択します。MIME が最も互換性の高い設定です。

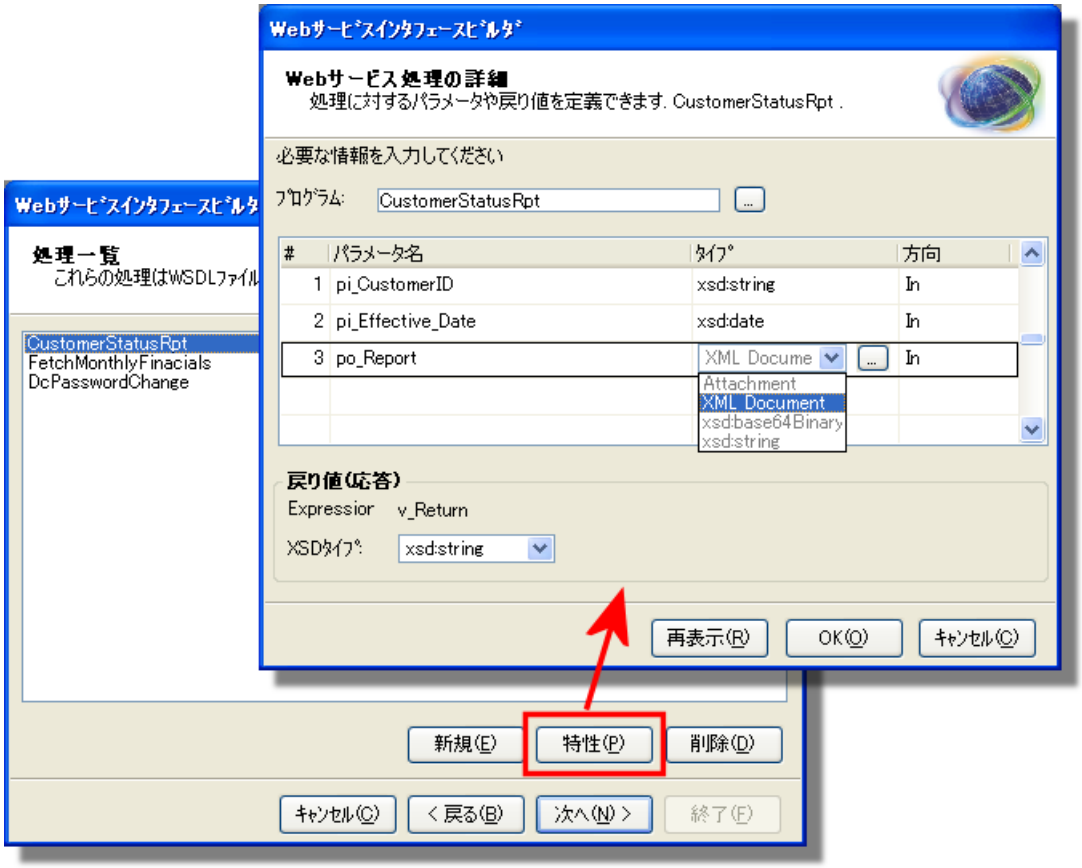
設定が終了したら、次へを**クリック**します。

注： **サービス名**や**説明**、**ネームスペース**を日本語（全角文字や半角カタカナ）を使用すると処理に失敗します。

6. プログラムの選択ダイアログ

プログラム一覧ダイアログには、公開名が定義され、外部カラムのボックスがチェックされているすべてのバッチプログラムが**有効値**に表示されます。中間に表示されているボタンを使用して、Web サービスとして公開したいプログラムを**選択値**に移動します。そして、次へを**クリック**します。選択されたプログラム毎にリストが表示されます。

7. 処理一覧ダイアログ

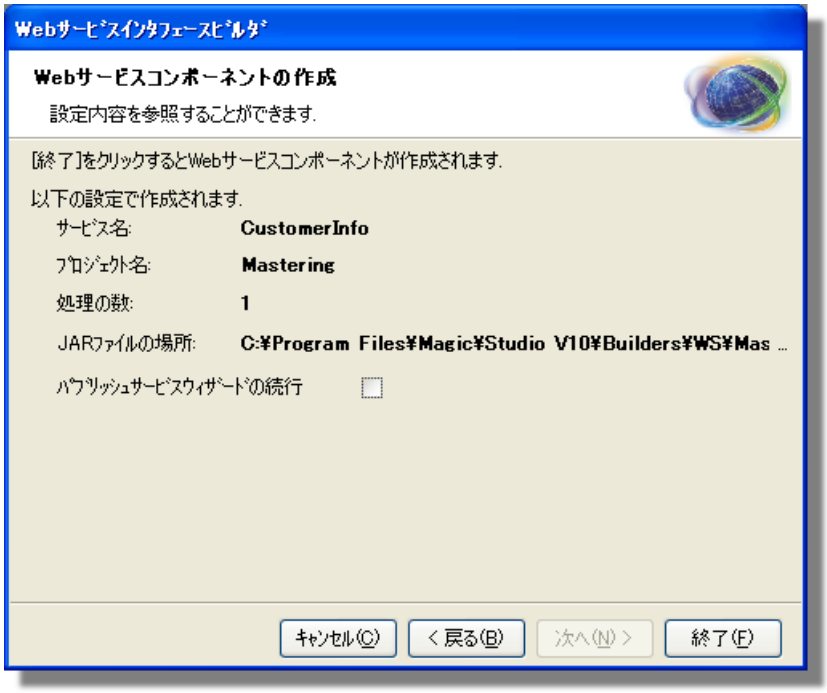


選択されたプログラム毎に**処理一覧**ダイアログが表示されます。各ダイアログでは、特性や処理の詳細を参照したり修正することができます。設定内容は、Web サービスがどのように表示されるかに影響します。

パラメータが文字型、Unicode 型、または BLOB 型で複合タイプを表す場合、ここで ***XML Document*** を選択すると、XML 形式で記述された XSD を選択するための長円ボタンを**クリック**することができます。

これで終了する場合、次へを**クリック**します。

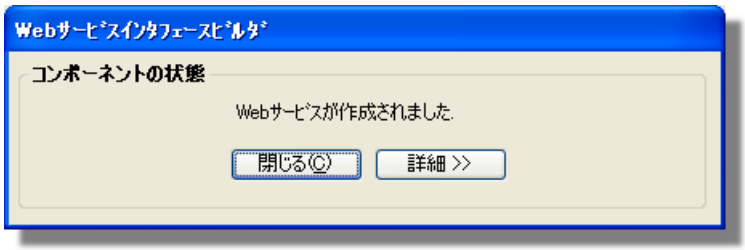
8. 確認ダイアログ



確認ダイアログが表示されます。終了をクリックします。

デプロイされた後に UDDI レジストリへのサービス登録を行わない限りは、**パブリッシュサービスウィザードの続行**のチェックボックスをチェックしないでください。

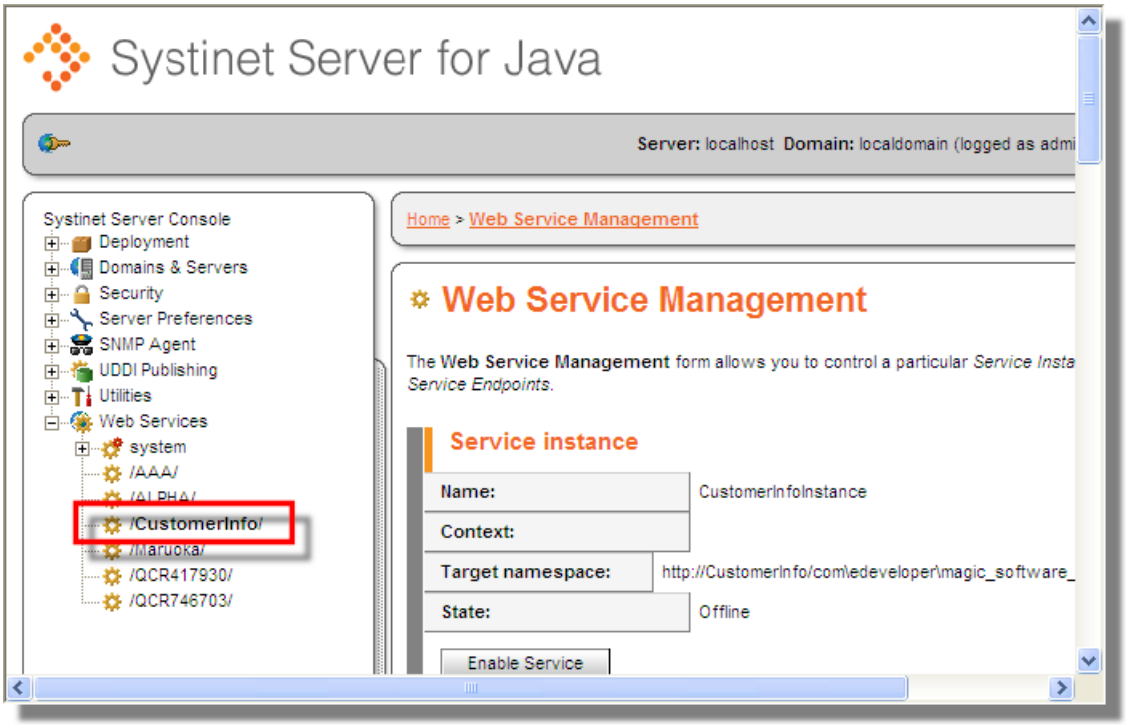
9. 終了



処理が完了すると、ステータスウィンドウが表示されます。エラーが発生した場合、エラーメッセージが表示されます。その際、詳細ボタンをクリックするとことでエラーの詳細を参照することができます。

エラーが発生しなかった場合、詳細ボタンをクリックすることで作成された JAR ファイルの位置が表示されます。手動でサービスをデプロイしたい場合は、この JAR ファイルの位置をコピーし記録しておいてください。

10. デプロイ結果を確認する



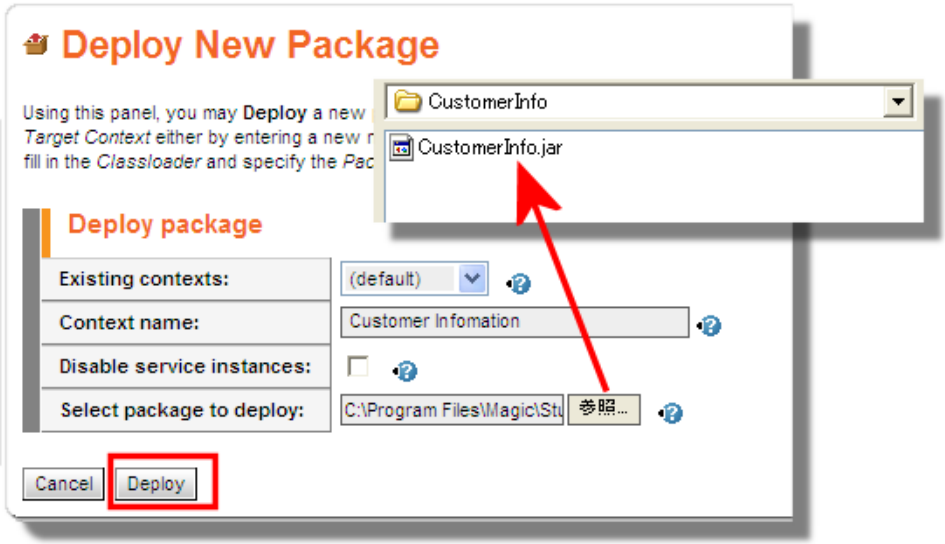
Web サービスインタフェースビルダの **Systinet サーバの詳細**ダイアログで**デプロイ**のチェックボックスをチェックした場合、Magic はサービスモジュール（JAR ファイル）を作成し、それをデプロイします（「4. サーバの詳細ダイアログ」（713 ページ）を参照）。Systinet コンソールを使用して確認することができます（720 ページを参照）。

サービスを手動でデプロイすることもできます（720 ページを参照）。

Web サービスのモジュールをデプロイするには

Magic で Web サービスを作成する際、**Systinet サーバの詳細**ダイアログで**デプロイ**のチェックボックスをチェックしておくことで、自動的に Systinet にデプロイされます。詳細は、「Web サービス（プロバイダ）」（709 ページ）を参照してください。

これとは別に、以下の手順で Systinet のコンソールから手動でモジュールをデプロイすることができます。この方法は、開発環境で Web サービス用の Jar ファイルを作成し、実行環境でデプロイする場合に利用できます。



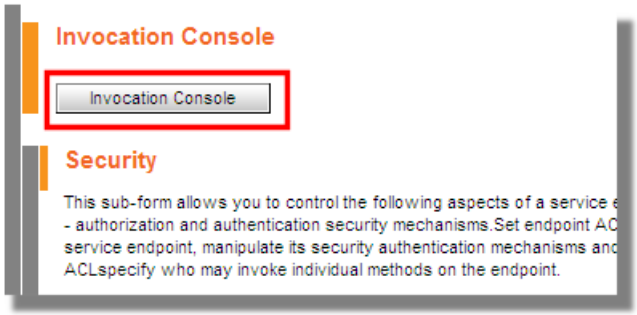
1. Systinet コンソールを開きます。
2. **Deployment → Deploy New Package** をクリックします。
3. **Context name** を入力します。
4. **Select package to deploy** でデプロイするパッケージを選択します。ここには Magic によって作成された .jar ファイルを指定します。
5. Deploy ボタンをクリックします。

これで、Web サービスが利用可能になります。

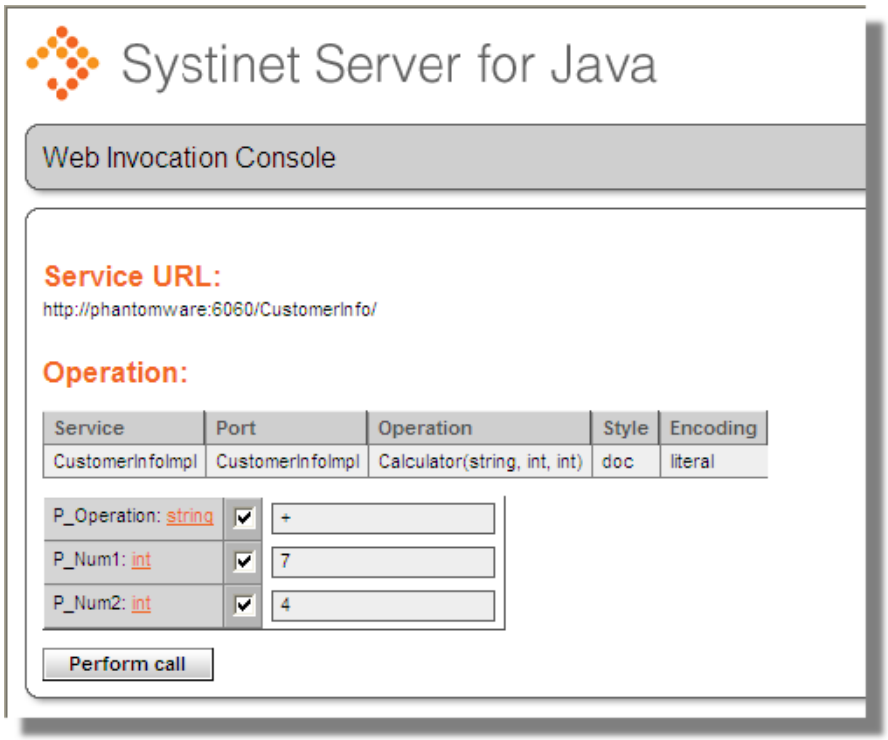
Web サービスをテストするには

サービスを呼び出すクライアントプログラムを作成することで Web サービスをテストすることができますが、ドキュメント形式の Web サービスの場合 Systinet Server の管理コンソールから直接テストすることもできます。

1. Systinet コンソールを開きます。
2. 左側のツリー上の **Web Services** をクリックして Web サービスノードを拡張します。テストしたいサービス名を選択します。



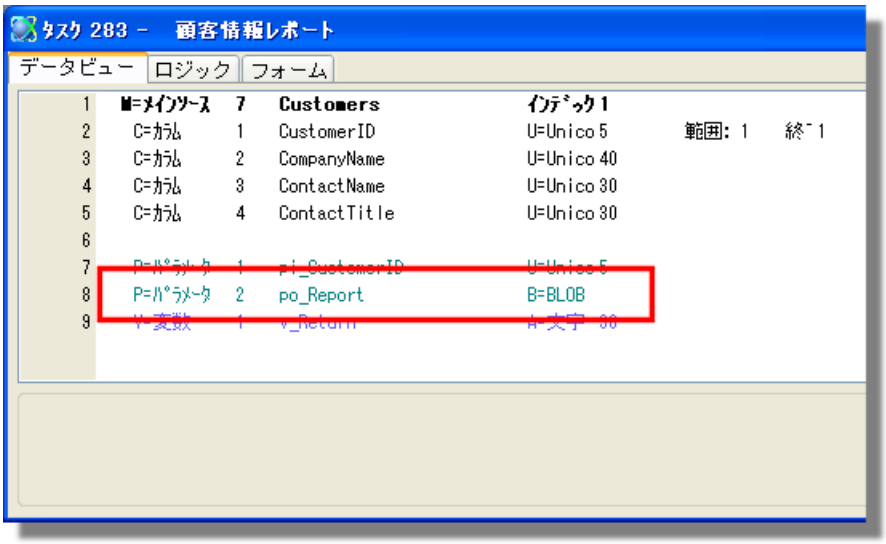
3. 右側のウィンドウに様々な選択項目が表示されます。少し下にスクロールし、Invocation Console ボタンをクリックします。



4. 新しいウィンドウが表示されます。ここには、サービス用のテスト値を入力するフィールドが表示されます。Magic の実行エンジンが動作することを確認し、Perform call ボタンをクリックします。

5. 入出力された値が表示されます。上記の例では、３つの入力値（+ と 7 と 4）が上の方に、出力値（11）が下の方に表示されます。

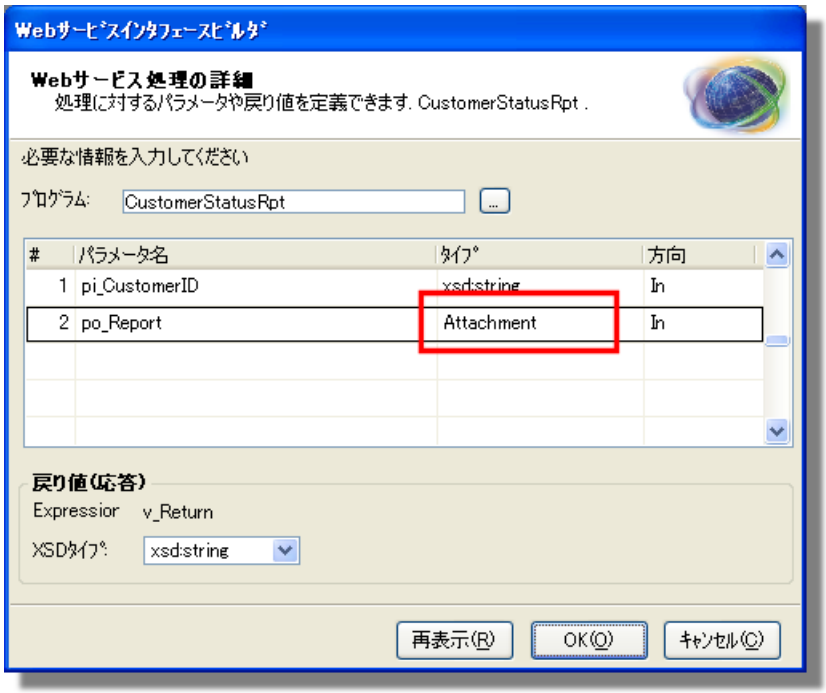
アタッチメント付きの Web サービスを提供するには



パラメータとして添付ファイルを受け取る

BLOB 型のパラメータにカプセル化することで、Web サービスとの間でファイルを送受信することができます。この処理には、**Blb2File()** 関数を使用することができます。

Blb2File (*Blob*,*File Name*)



Web サービスを定義する際に、パラメータのタイプを ***Attachment*** と設定します。

これで、ファイルは添付ファイルとして処理されます。

関数で添付ファイルを受け取る

添付ファイルがパラメータで定義されていない場合でも、コンシューマ側で **WsConsumerAttachmentAdd()** 関数を実行することで添付ファイルが含まれたリクエストが送信されます。このリクエストから添付ファイルを取り出すには、**WSProviderAttachmentGeteb()** 関数を使用します。関数の構文は以下の通りです。

WSProviderAttachmentGeteb(*ID*)

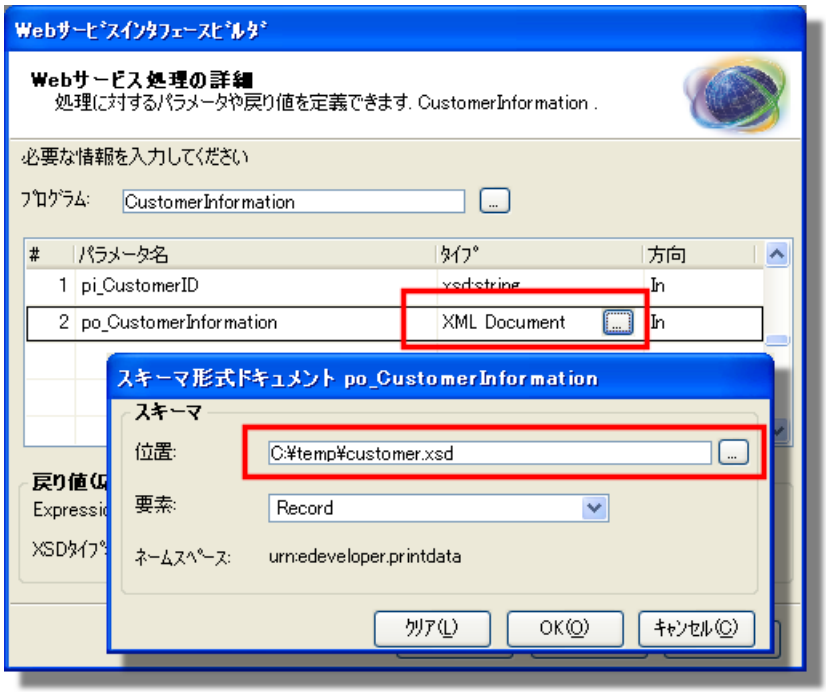
パラメータ：

- *ID*：コンシューマ側で添付した際に発行された UUID か、1 から始まるインデックス値

この関数は、添付ファイルが格納された BLOB 値が返ります。

参照： これらの関数の詳細は、『リファレンスヘルプ』を参照してください。

プロバイダとしての Web サービスから複雑なパラメータの送受信を行うには



複雑なパラメータを Web サービスへ送るには、XML ファイルを使用する必要があります。以下の手順で設定します。

1. XML ファイルの記述に使用される XML スキーマ（XSD）を作成します。
   XML スキーマには、ネームスペース定義（targetNamespace）を含める必要があります。相互運用性（インターオペラビリテ）の問題を回避するために、異なるパラメータ／戻り値のスキーマには異なるネームスペースを指定する必要があります
   （つまり、同じ処理のすべてのパラメータに対して同じスキーマを使用しないでください）。
2. XML スキーマを使用して XML データソースを作成します。この XML データソースによって、送信されるデータをフォーマットします（詳細は、第 14 章：「最初から XML ドキュメントを作成するには」（295 ページ）を参照してください）。
3. Web サービスを作成する場合、BLOB 型のパラメータを 1 つ定義します。このパラメータは XML データを保持するために使用されます。
4. Web サービスを作成する場合、このパラメータ用のタイプとして ***XML Document*** を選択します。 ... をクリックしてスキーマファイルを選択するファイルを開くダイアログを開きます。手順 **#2** で定義された XML スキーマファイルを選択します。
5. OK をクリックします。

これで、作成された WSDL ファイルに必要な XML スキーマが含まれるようになります。

Systinet Web コンソール（http://localhost：6060/admin/console）を使用することで、サービスの WSDL にアクセスすることができます。

Web Service Runtime View の画面（コンソールウィンドウの左側に表示されるツリーの **Web Services** をクリックすることでオープンされます）でサービス名を指定します。

List of Service Instance's Endpoints セクションに表示されているテーブルの **Url** カラムのハイパーリンクをクリックすると WSDL が表示されます。WSDL の URL は、以下のような書式になっています。

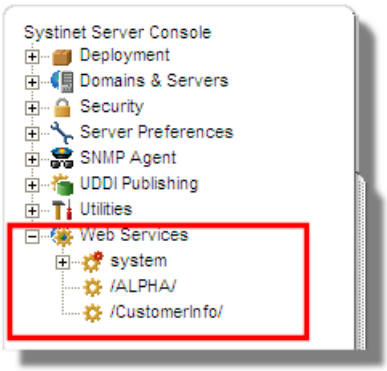
```
http://servername:6060/ServiceName
```

SOAP リクエストを追跡するには

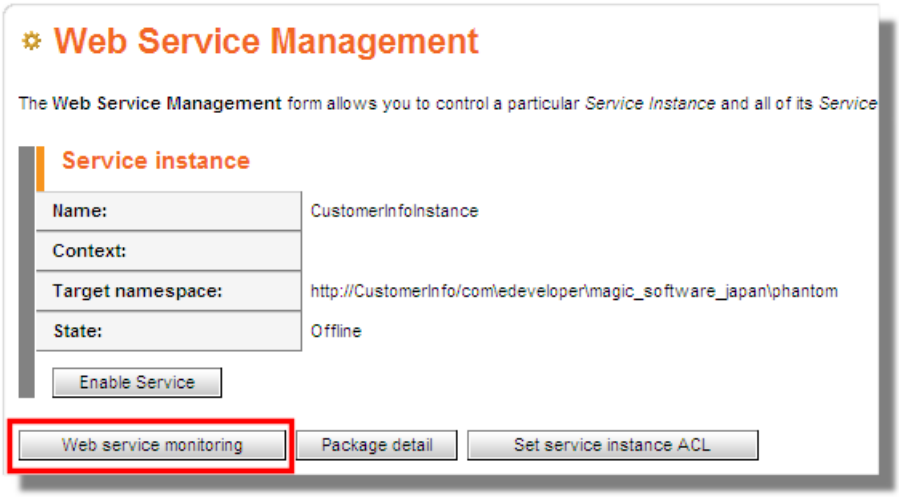
Web サービスに入る SOAP リクエストを追跡することができます。Systinet のコンソールでこのような処理が可能です。

1. ショートカットか以下の URL を使用して、Systinet コンソールを起動します。
   http://localhost:6060/admin/console

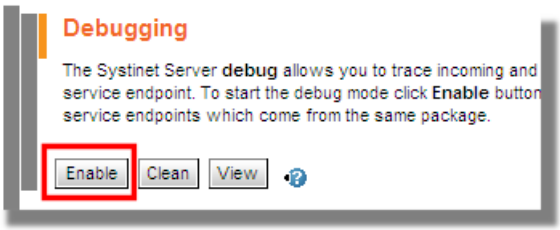
2. 左側のツリー上の **Web Services** をクリックして Web サービスノードを拡張します。



3. 追跡したいサービス名を選択します。



4. Web service monitoring ボタンをクリックします。



5. デバッグセクションにスクロールします。Enable をクリックします。これで、Web サービスはウォッチの配下に置かれます。
6. クライアントからの呼び出し処理が実行されたら、View ボタンをクリックすることで HTTP や SOAP の動作内容が表示されます。このページにアクセスするには、このサービスにアクセス可能なユーザ ID とパスワードが必要です。

デプロイされたサービスに対するセキュリティを設定するには

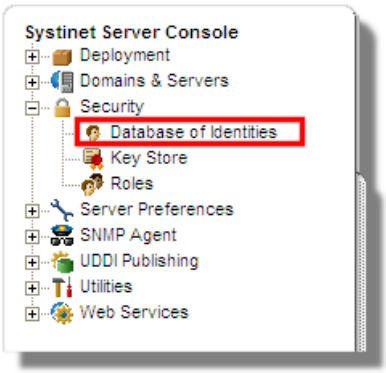
デプロイされた Web サービスに対して、特定のユーザのみがアクセス可能になるように設定することができます。セキュリティの設定は、以下の３つの手順で行います。

- ユーザを定義します。
- どのユーザが Web サービスにアクセスできるかを定義します。
- サービスにアクセスするための認証方式を定義します。

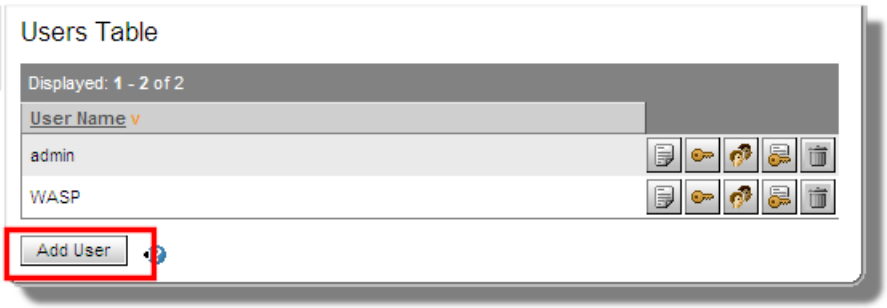
これらの手順について以下で説明します。

ユーザとパスワードを定義する

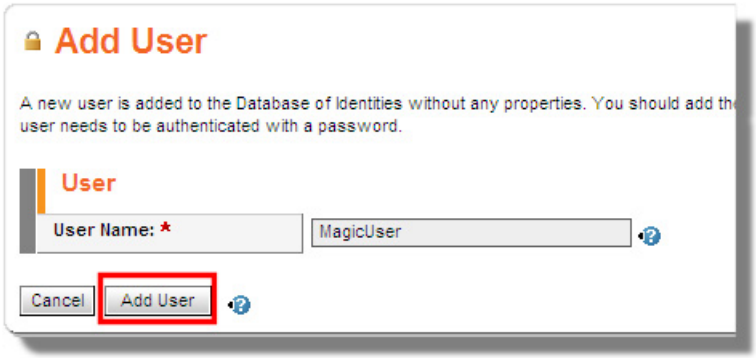
1. Systinet コンソールを開きます。



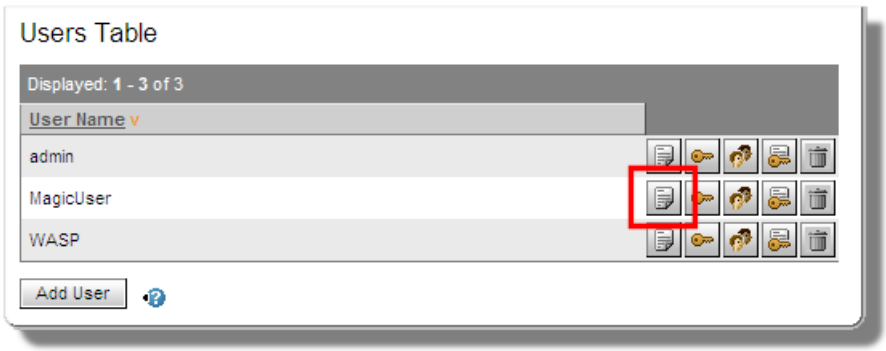
2. 左側のツリーから **Security → Database of Identities** を選択します。



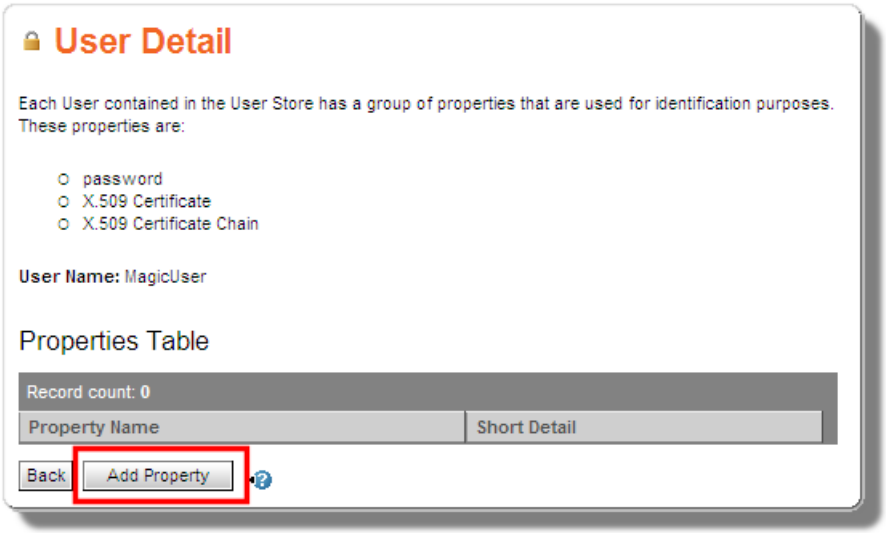
3. 右側に既存のユーザリスト（Users Table）が表示されます。リストの下に表示されている Add User ボタンをクリックします。



4. ユーザ ID を入力するボックスが表示されます。ユーザ ID を入力し、Add User ボタンをクリックします。
5. ユーザリストに追加されたユーザ ID が表示されます。各ユーザの右側に表示されるボタンをクリックすることでユーザの管理を行うことができます。

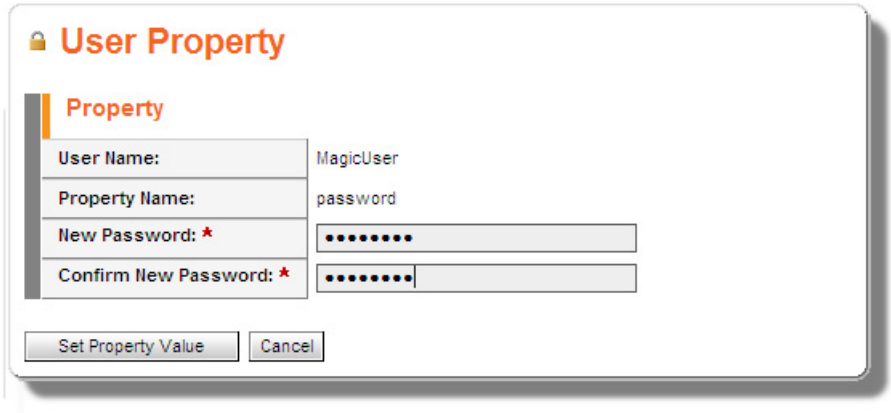


6. 次に  ボタンをクリックしてパスワードを設定します。



7. このユーザの特性リスト（Properties Table）が表示されます。Add property ボタンをクリックします。

8. 追加可能な特性のリストが表示されます。その中に Password があります。Password の右側の  をクリックします。

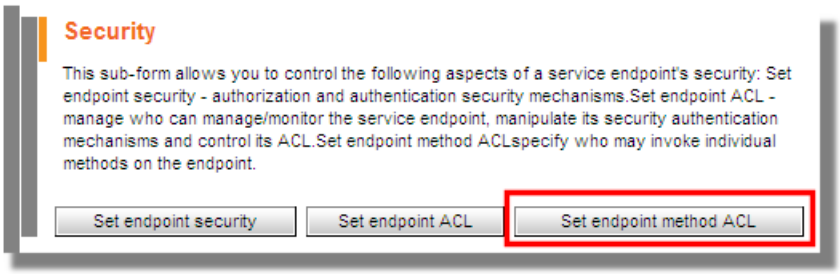


9. プロパティを入力するためのウィンドウ（User Property）が表示されます。ここでパスワードを入力し Set Property Value ボタンをクリックします。

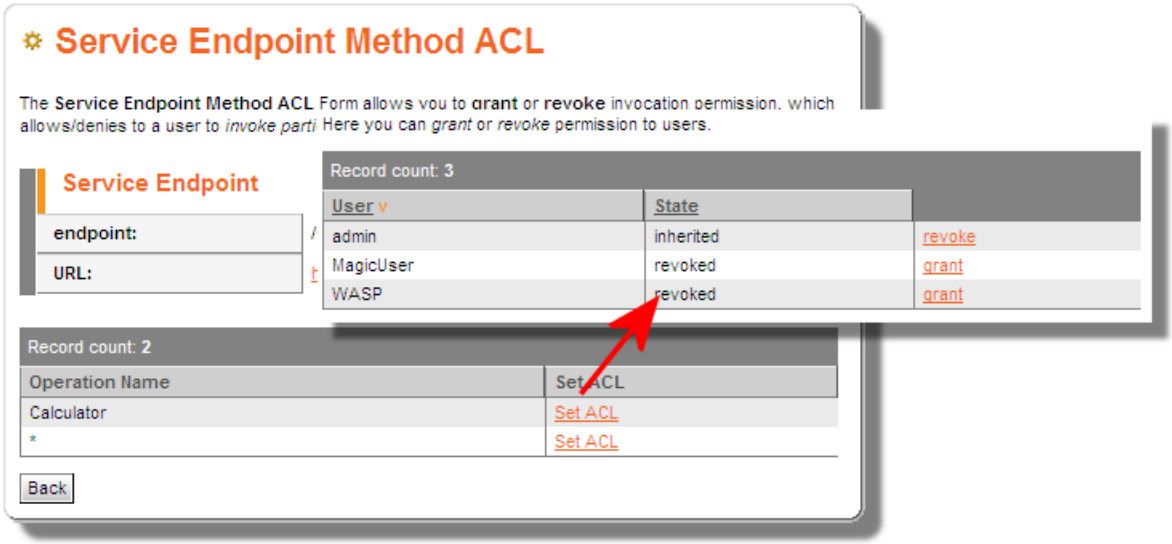
サービスへのアクセスを許可する

次に、認証されるようにサービスを変更し、利用できるようにユーザを定義する必要があります。

1. Systinet コンソールを開きます。
2. 左側のツリー上の **Web Services** をクリックして Web サービスノードを拡張します。テストしたいサービス名を選択します。



3. 右側のウィンドウを下にスクロールし Security セクションに表示される Set Endpoint Method ACL ボタンをクリックします。

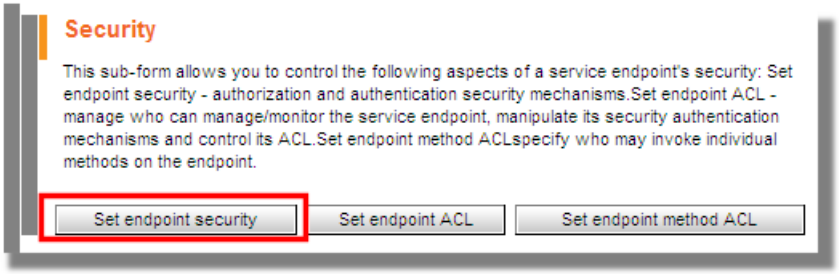


4. テーブル上に処理名が表示されます。Set ACL をクリックして各処理に対する認証設定を行います。
   Set ACL をクリックすると、各ユーザが処理に対してどのような権利を持っているかがリスト表示されます。**grant** をクリックすると権利を設定することができます（ステータスが granted に変更します）。**revoke** をクリックすると権利は削除されます（ステータスが revoked に変更します）。

サービスの認証方法を定義する

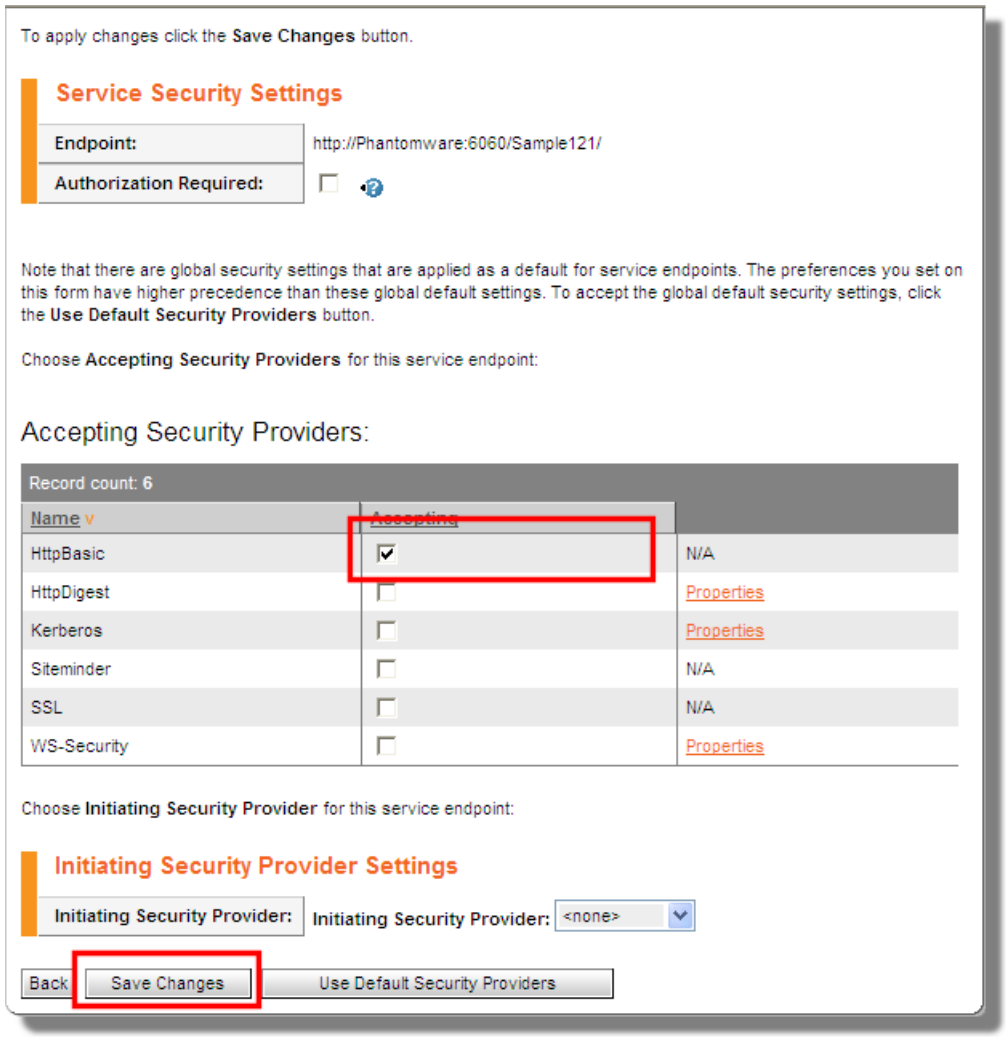
最後に、このサービスをが認証されたものとして定義する必要があります。

1. 左側のツリーから Web サービスを選択します。
2. 利用するサービスを選択します。



3. Security セクションの Set endpoint security ボタンをクリックします。

4. **Authenication and authorizatoin** のページが表示されます。ここから Custom Security Providers ボタンをクリックします。



5. 利用できるセキュリティ方式がリスト表示されます。利用したいセキュリティプロバイダをチェックします。Save Changes ボタンをクリックして保存します。

   セキュリティプロバイダが設定されたサービスを Magic のアプリケーションから呼び出すには、サービステーブルで対応するセキュリティの設定を行う必要があります。

プロバイダ側	Magic 側（サービステーブル）	
	セキュリティレベル	認証方式
HttpBasic	Transport	BASIC
HttpDigest	Transport	DIGEST
Kerberos	Transport	Kerberos
Siteminder	未サポート	
SSL	Transport	SSL
WS-Security	WS-Security	

以上がプロバイダ側で行うすべての操作です。この Web サービスにアクセスする場合、ここで指定された認証情報を指定する必要があります。

注： Kerberos や SSL、WS-Security の認証方法を使用する場合、Systinet Server やクライアント側で詳細な環境設定が必要になります。設定方法については、Systient Server for Java のマニュアルを参照してください。

参照： 第 34 章：「Web サービスに安全にアクセスするには」（703 ページ）

［このページは意図的に空白にしています。］

第 36 章： データベース

データベースとの接続を定義するには

Magic は、Oracle、MS-SQL Server、DB2 UDB、ODBC、および Pervasive.SQL を含む複数のデータベースと接続することができます。また、同時に複数のサーバ上の複数のデータベースにリンクすることもできます。

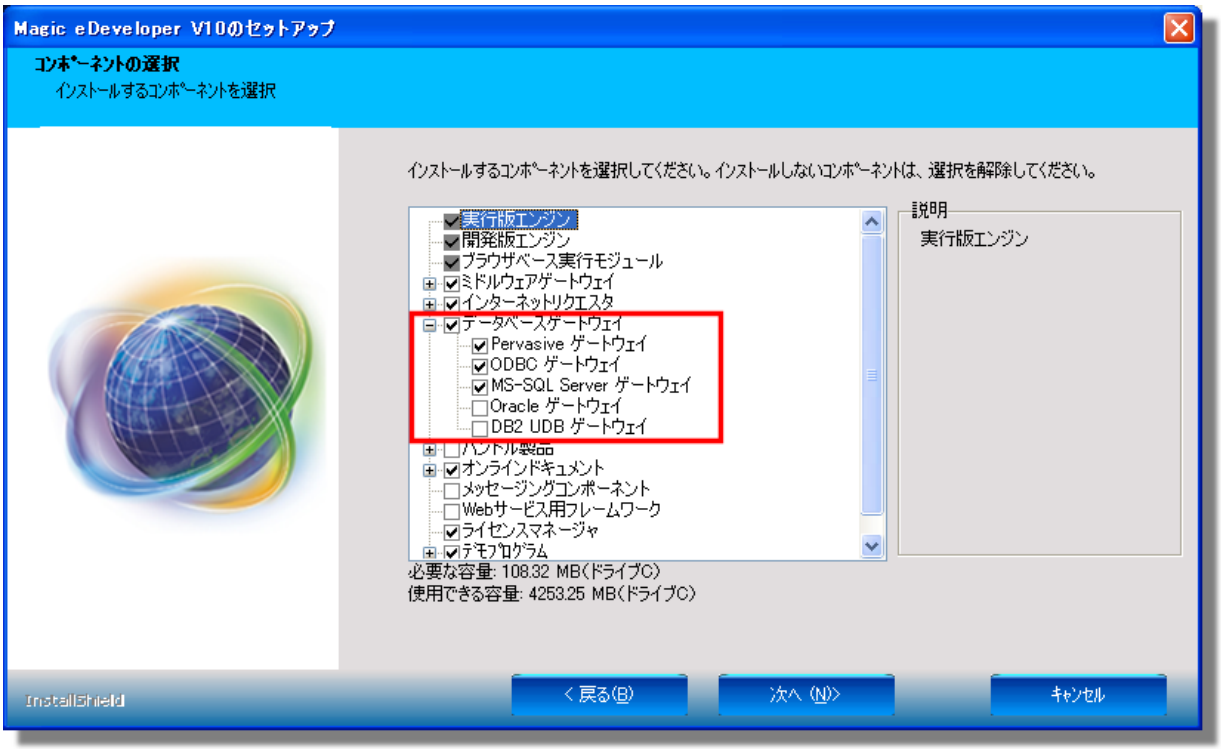
これらのデータベースを使用するには、事前に Magic でこれらの RDBMS に接続するための環境設定を行う必要があります。

環境設定は、以下の手順で行います。

- データベースゲートウェイをロードできるように設定します。
- データベースを定義します。
- 接続を確認します。

各手順の詳細を以下で説明します。接続環境の内容は、使用するデータベースに依存するため、RDBMS 毎に説明します。

1. データベースゲートウェイの設定を行う



1. Magic のインストール時に、使用するデータベースゲートウェイを選択することで、対応する DLL ファイルがインストールされます。Magic のインストール時にゲートウェイのインストール選択を行わなかった場合、メンテナンス

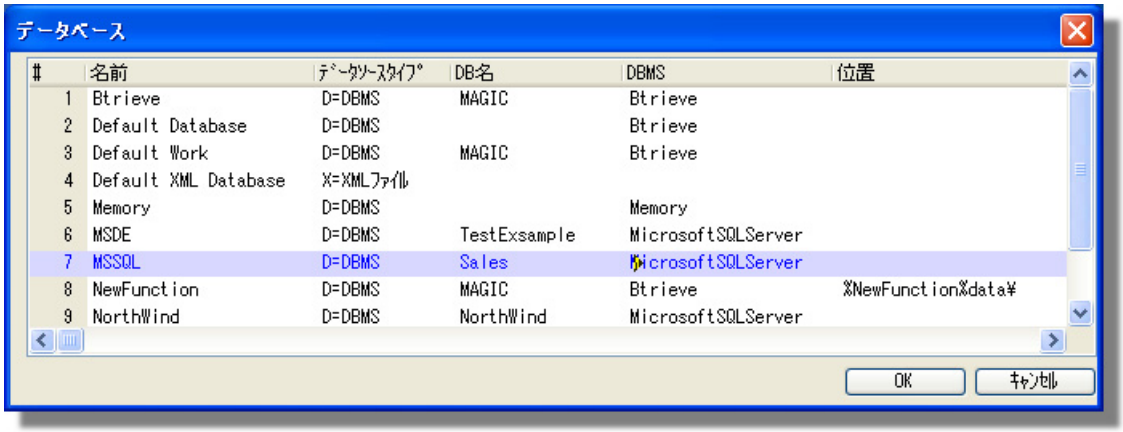
モードで Magic のインストーラを起動する（**スタート→コントロールパネル→プログラムの追加と削除**を選択し、Magic の製品名を選択した後、変更と削除を**クリック**する）ことで追加することができます。

```
308
309 [MAGIC_GATEWAYS]
310 ;MGCOMM01=mgwsock.dll
311 MGDB00=Gateways¥MGbtrieve.dll
312 ;MGDB06=Gateways¥MGdb2400.DLL
313 MGDB13=Gateways¥MGoracle.dll
314 ;MGDB16=Gateways¥MGeac32.dll
315 ;MGDB18=Gateways¥MGdb2.DLL
316 ;MGDB19=Gateways¥MGobc.dll
317 MGDB20=Gateways¥MGmssql.dll
318 MGDB21=Gateways¥MGmemory.dll
```

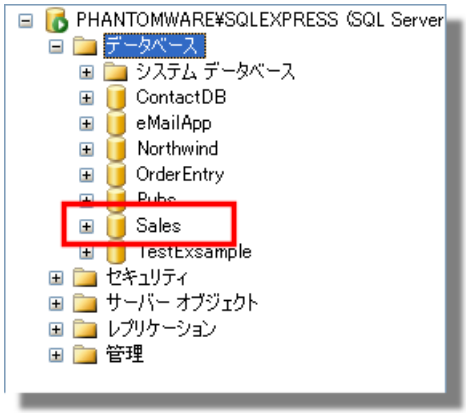
2. 次に、使用する Magic.ini ファイルの［MAGIC_GATEWAYS］セクションでデータベースゲートウェイの設定行がコメントになっていないことや、DLL ファイルのパスが正しいことを確認します（通常は、ゲートウェイをインストールすることで Magic.ini の設定も自動的に行われます）。
3. また、Magic.ini ファイルの［MAGIC_DATABASES］セクションで使用するデータベースの設定行がコメントになっていないことを確認します。
4. Magic.ini を編集したら、Magic を起動します。

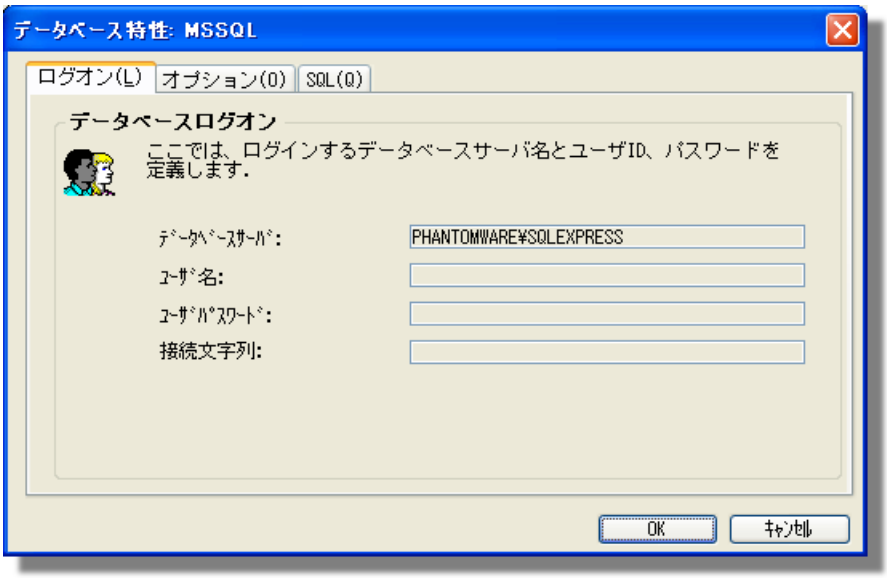
これで、データベースを定義する用意ができました。

2. データベースを定義する

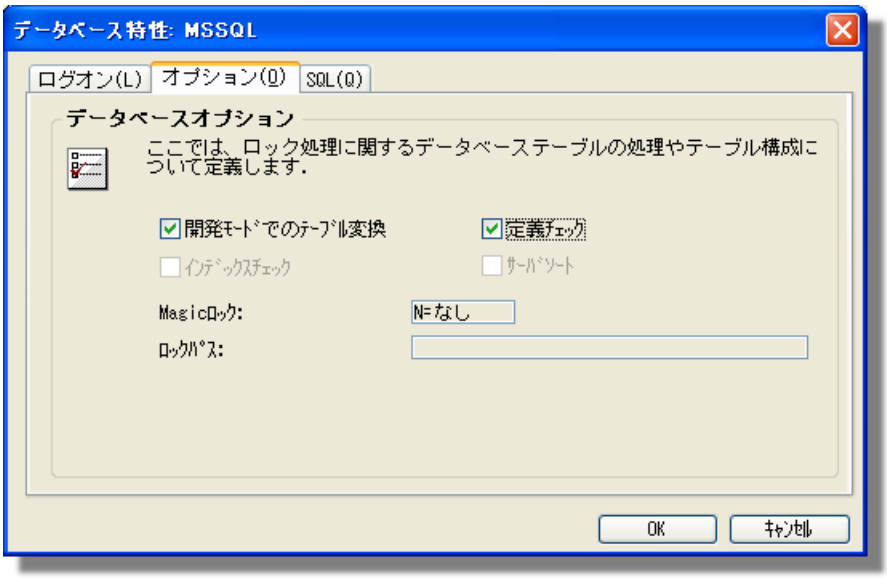


1. プロジェクト（またはアプリケーション）が開いている場合は、一旦閉じます。
2. **データベース**テーブル（**オプション→設定→データベース**）を開きます。
3. ***F4*** を押下して１行追加し、**名前**カラムに任意のデータベース名を入力します。この名前は、データベースを選択する際の表示用としてのみ使用されます。
4. **データソースタイプ**カラムで ***D=DBMS*** を選択します。
5. **データベース名**カラムでは、アクセスするデータベースの名前を入力します。
   データベース名はデータベースマネージャを使用して事前に定義しておく必要があります。この例では、MS-SQL2005 Express に定義されている ***Sales*** というデータベースを使用するように定義されています。
6. **DBMS** カラムから**ズーム**します。Magic.ini に定義されたデータベースの名前が一覧表示されます。この例では、***Microsoft SQLServer*** が選択されています。次に、***Alt+Enter*** を押下して **DBMS 特性**を開きます。

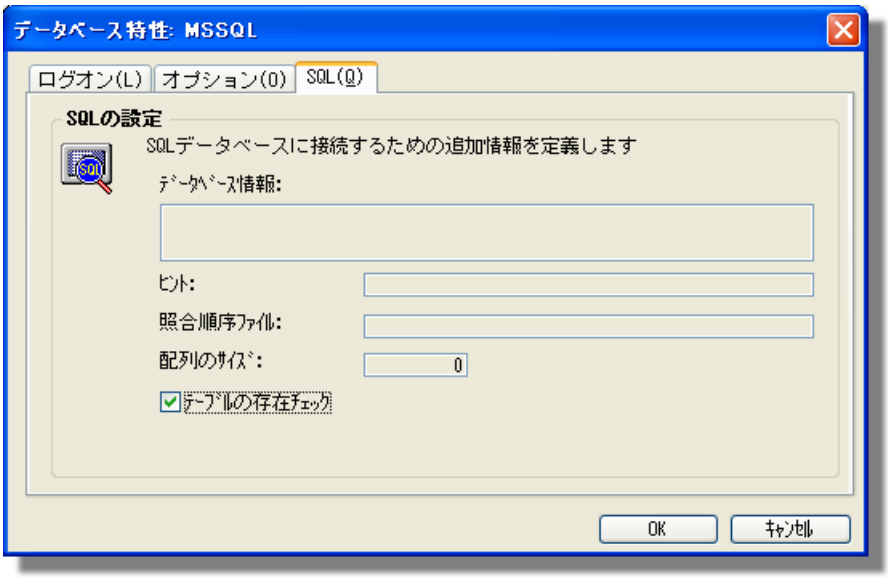




7. ログオンタブをクリックし、データベースサーバ名を設定します。必要であればユーザ ID やパスワードも指定します。この例では、Windows 認証を使用しているため、ユーザ ID やパスワードは必要ありません。



8. オプションタブをクリックします。Magic 内でテーブル定義の変更を行いたい場合は、**開発モードでのテーブル変換**特性をチェックします。このテーブルが別のアプリケーションによって作成されたもので、Magic 側で変更しないようにする場合は、チェックを外します。



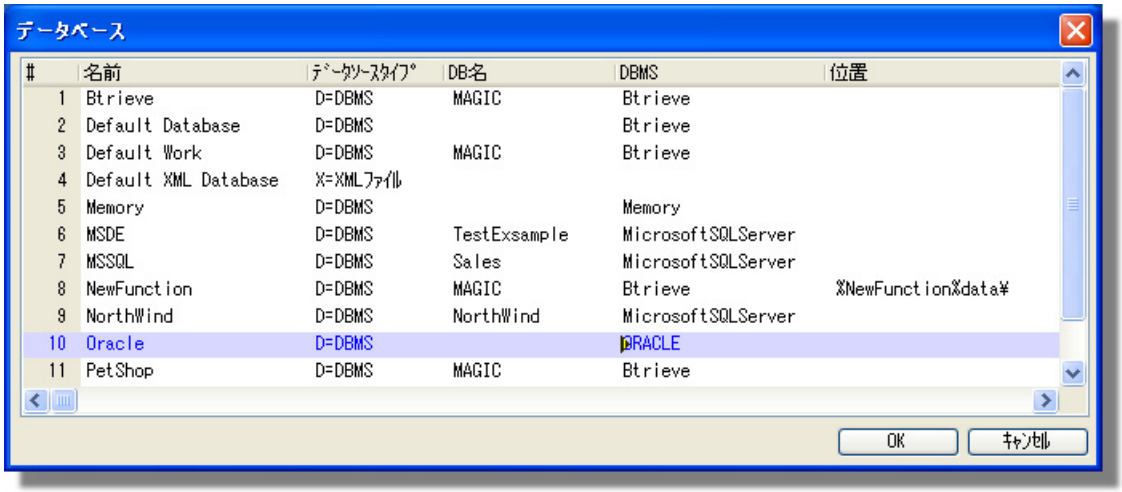
9. SQL タブをクリックします。**データベース情報**特性には、テーブルと接続するために使用する SQL コマンドを追加することができます。Magic 側でテーブルを作成できるようにする場合は、**テーブルの存在チェック**特性をチェックします。

3. 接続を確認する

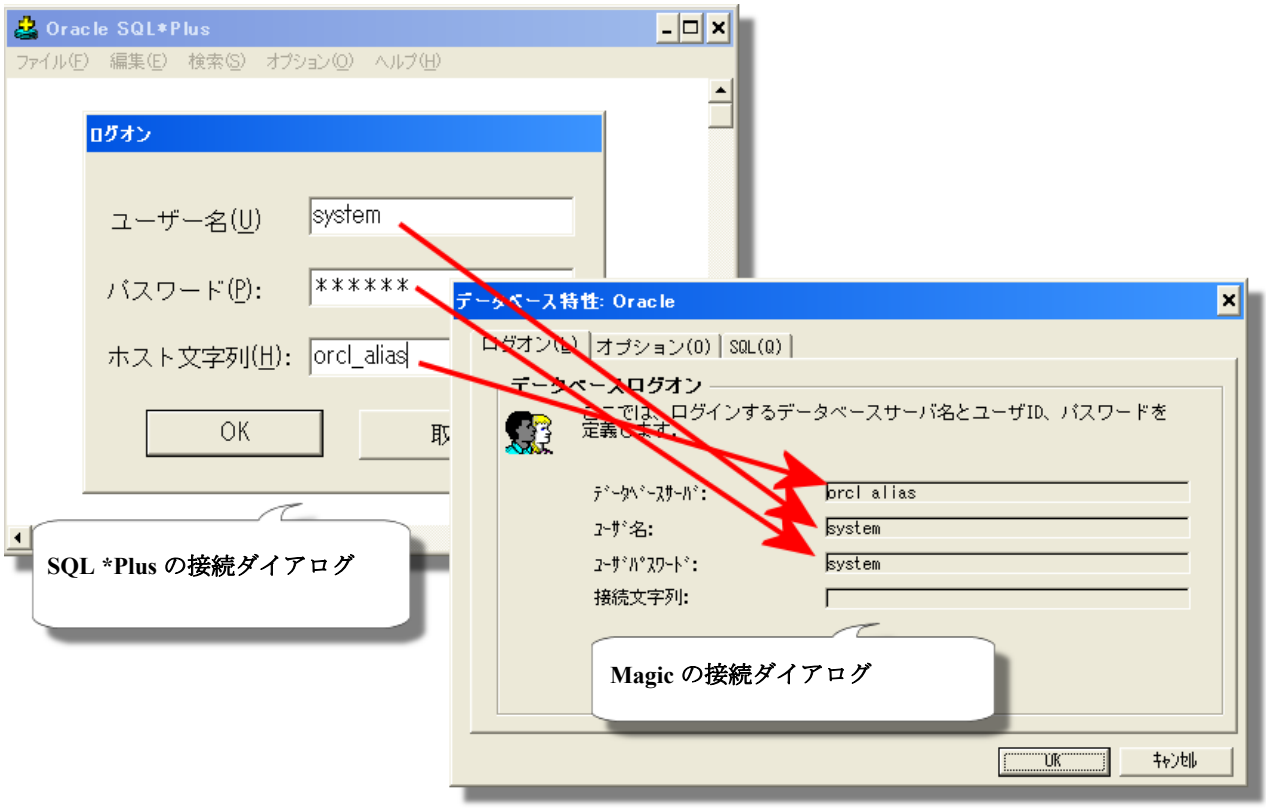
データリポジトリ内で**定義取得**を実行することで、データベースが正しく設定されたことを確認することができます。**定義取得**の方法については、第 18 章 :「既存のデータベーステーブルにアクセスするには」（402 ページ）を参照してください。

DBMS 別の設定方法

Oracle



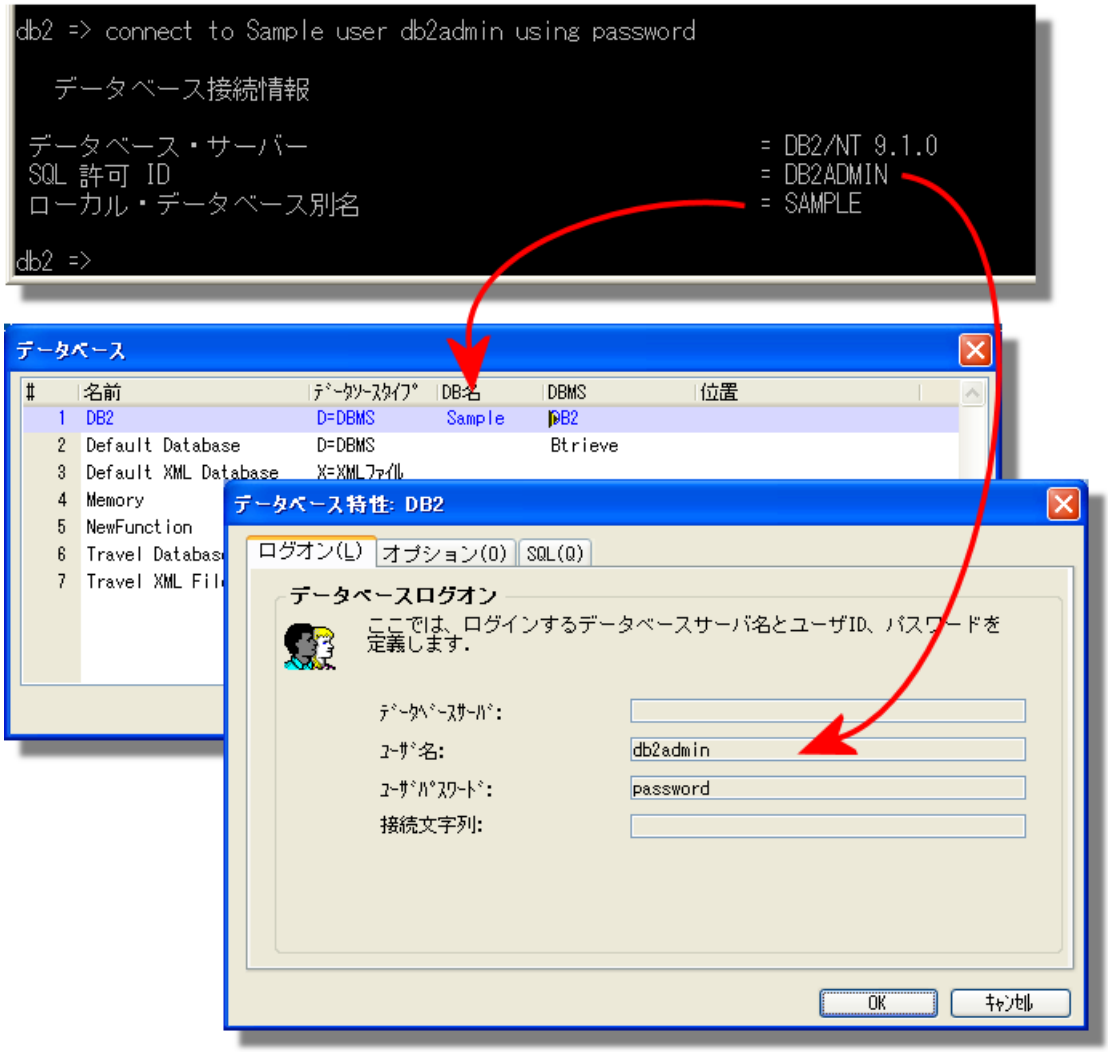
Oracle データベースを定義するには、**DBMS** カラムで ***Oracle*** を設定します。Oracle のエイリアスはすでにデータベースを示しているため、**DB 名**カラムには何も指定する必要がありません。



データベース特性で**ユーザ ID** と**パスワード**が同じ場合、**データベースサーバ**には**ホスト文字列**を指定する必要があります。

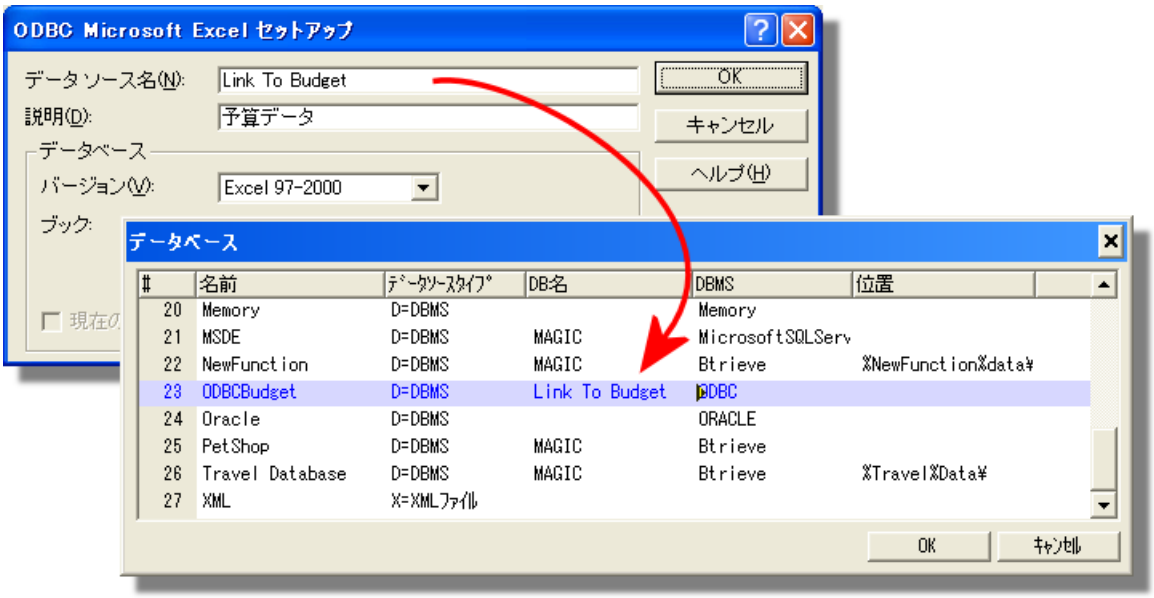
DB2 UDB

DB2 UDB データベースを定義するには、**DBMS** カラムで ***DB2*** を設定します。**DB 名**カラムに DB2 のエイリアスを入力しなければなりません。

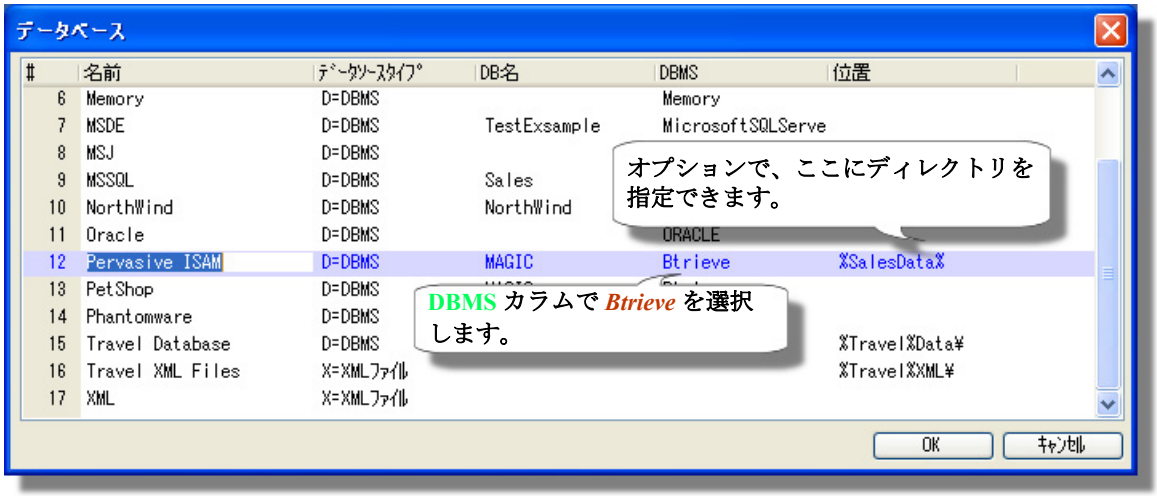


ODBC

ODBC データベースを定義するには、**DB 名**カラムで ODBC のデータソース名を設定します。**データベース特性**には何も設定する必要がありません。



Pervasive ISAM

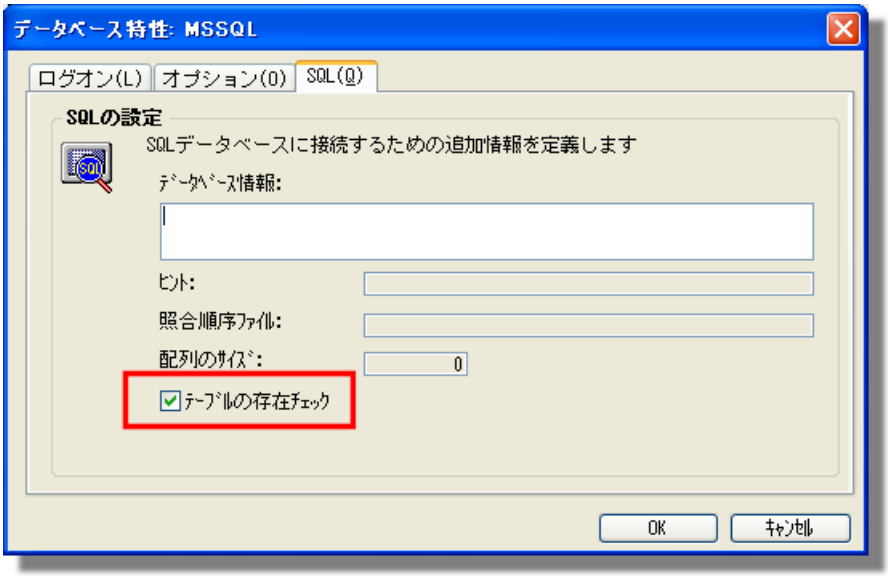


Pervasive SQL（ISAM）データベースを定義するには、データベーステーブルを設定する必要がありません。ISAM ファイルが通常の OS ファイルのように作成されます。

DBMS カラムで ***Btrieve*** を設定します。必要であれば、**位置**カラムにパスを指定することができます。**データ**リポジトリの**データソース名**カラムでパスを指定することもできます。

Magic からデータベーステーブルを作成するには

第 18 章：「Magic でデータベーステーブルを作成するには」（395 ページ）で説明されているように、Magic でデータベースを作成することで、Magic によって DBMS に自動的にテーブルを作成したり、構成を変更したりすることができます。



Magic のこのような機能を有効にさせるには、**テーブルの存在チェック**特性をチェックする必要があります。この特性は、以下のようにして設定できます。

1. プロジェクト（またはアプリケーション）が開いている場合は、一旦閉じます。
2. **データベース**テーブル（**オプション→設定→データベース**）を開きます。
3. 設定を変更したいデータベースを選択します。
4. ***Alt+Enter*** を**クリック**して**データベース特性**を開きます。
5. **SQL** タブを**クリック**します。
6. **テーブルの存在チェック**特性を**チェック**します。

これで、データソース内で新しいテーブルを作成したり、既存のデータソースを変更した場合、**データベース**テーブルが DBMS 内で変更されるようになります。

既存のデータベーステーブルまたはビューにアクセスするには

データベーステーブルがすでに DBMS の中に存在している場合、Magic で**定義取得**を行うことで自動的にデータソースを作成することができます。**定義取得**については、第 18 章：「既存のデータベーステーブルにアクセスするには」（402 ページ）を参照しています。

データベースに送信される SQL ステートメントを参照するには

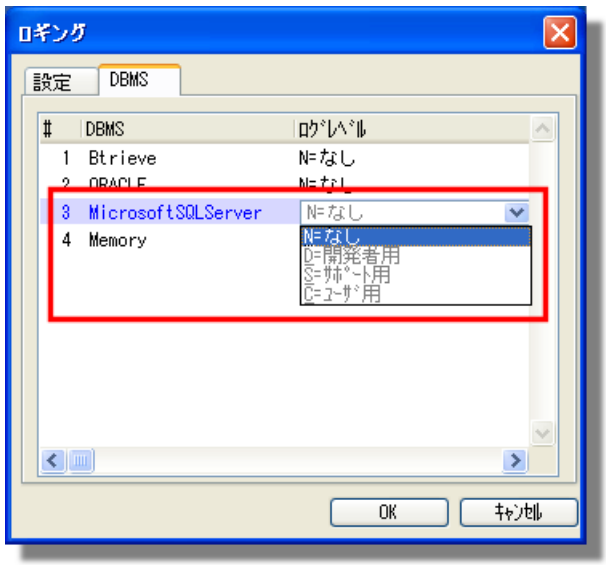
Magic は背後で SQL ステートメントの作成／送信を行っていますが、実際に送信される SQL ステートメントを参照することができます。

- Magic のデバッグ機能を有効にする
- 使用する DBMS のログ採取機能を使用する

２番目の方法は、DBMS に依存するためここでは説明しません。Magic で SQL ステートメントを参照することは、以下で説明するように簡単に行うことができます。

Magic 内での SQL ステートメントのロギング機能を有効にする

1. 最初に、ロギング機能を有効にする必要があります。ロギングテーブル（オプション→設定→ロギング）でこの設定を行います。
   利用する DBMS に対して必要なロギングレベルを設定します。
   - ***N= なし*** ：ロギングを行いません。
   - ***C= ユーザ*** ：発行される SQL コマンドのみを出力します。
   - ***D= 開発者*** ：MSE/MSJ の開発向けの最大レベルの情報を出力します。
   - ***S= サポート*** ：アプリケーション開発者用の情報を出力します。



   Magic で送信される SQL ステートメントを参照するだけであれば***ユーザ***レベルで十分ですが、開発者またはサポートレベルを指定することで更に多くの情報を確認することができます。以下の出力例は、***サポート***レベルで出力されたものです。

```
,09484  session - 0, in use - 1, write - 1, reusable - 1, results pending - 0,not only for cursors - 1
,09500  ms7_trans(): OPEN_READ
,09500  ms7_trans(): <<<<< ctxID = -1.000000, retcode = 0
ﾄﾗﾝｻﾞｸｼｮﾝ読込 0
,09500  ms7_trans(): >>>>> ctxID = -1.000000, transmode = 8, db = 20
,09500  Hold thru ms7_trans(): >>>>> ctxID = -1.000000, serverID = 1, dbconn_hdl = 0
,09500  Hold thru ms7_trans(): <<<<< ctxID = -1.000000, errcode = 0, RC = RC_OK
,09500  session - 0, in use - 1, write - 1, reusable - 1, results pending - 0,not only for cursors - 1
,09500  ms7_trans(): <<<<< ctxID = -1.000000, retcode = 0
ﾄﾗﾝｻﾞｸｼｮﾝのｺﾐｯﾄ 0
```

2. 必要であれば、デバッグモード（デバッグ→デバッグモード）を有効にします。
3. アクティビティモニタ（表示→アクティビティモニタ）を開きます。

これで、デバッガを起動しながらプログラムを実行することで、アクティビティモニタに SQL ステートメントが表示されます。

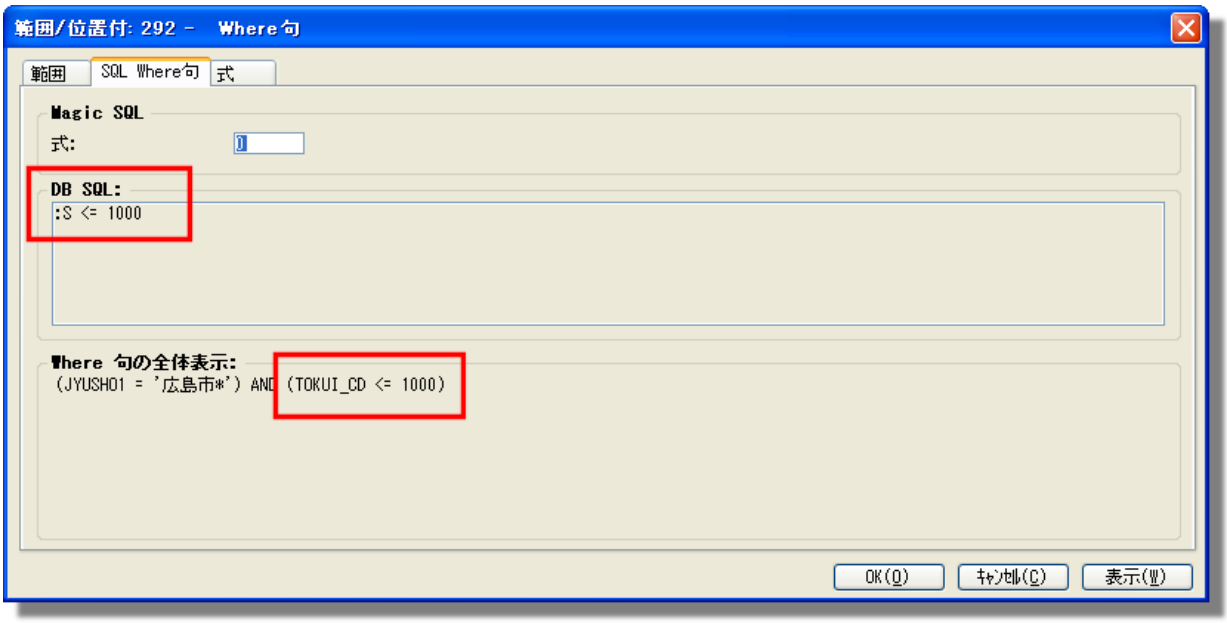
参照： 第 29 章：「デバッガを使用してデバッグするには」（607 ページ）を参照してください。

独自の SQL ステートメントをデータベースに送るには

SQL データベースを利用して Magic を使用する場合、Magic は必要な SQL コードを生成します。しかし、デフォルトのコード生成機能を無効にすることもできます。例えば、ストアドプロシジャを呼び出したり、レコードグループの取得方法をより詳細に制御したい場合に利用できます。

このようなことは、Magic プログラム内で簡単に行うことができます。SQL ステートメントの送信方法には、以下に示すようなものがあります。

WHERE 句を手動で入力する

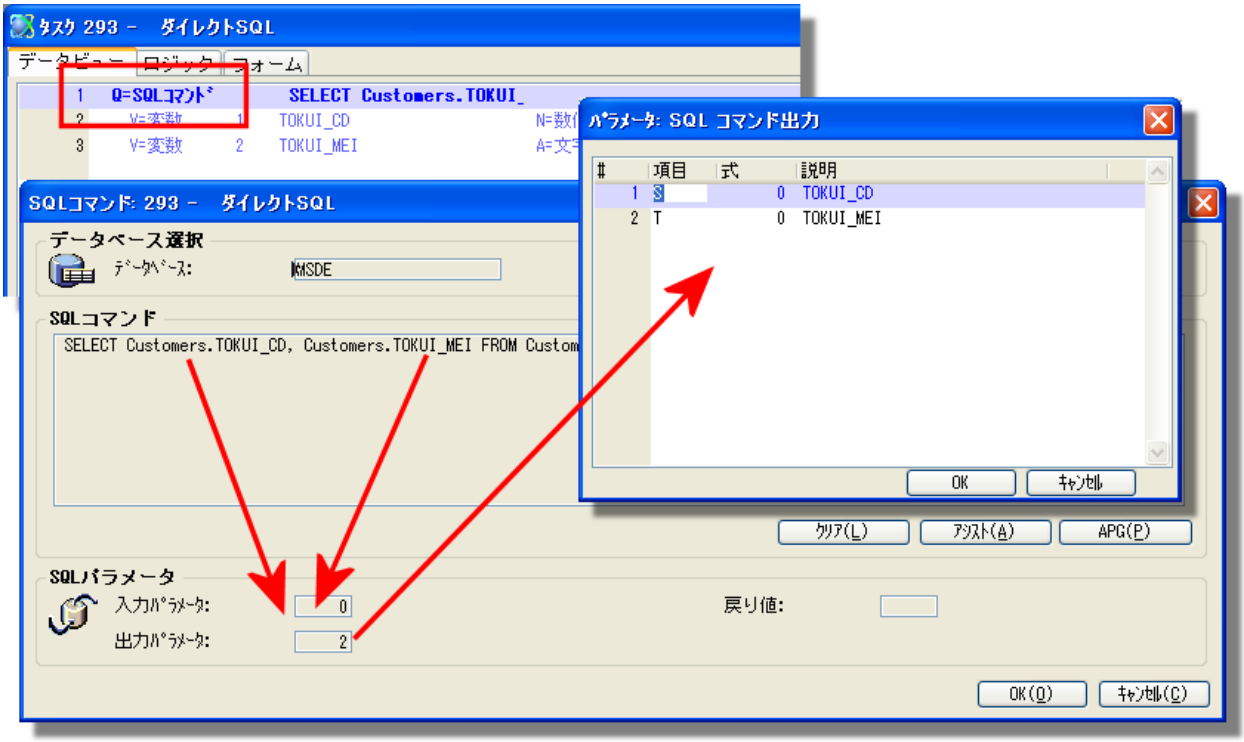


WHERE 句を送信したいだけの場合、**範囲 / 位置付**ダイアログを使用することで実現できます。**DB SQL** で Where 句ステートメントを入力することで、**Where 句の全体表示**に Magic で生成される Where 句が表示されます。

1. **範囲 / 位置付**ダイアログ（**タスク→範囲 / 位置付**）を開きます。
2. **SQL Where** タブを**クリック**します。
3. **DB SQL** に直接入力したり、**式**から**ズーム**して**式エディタ**に必要な式を入力することで、任意の Where 句を入力します（**範囲**タブで範囲式を定義した場合も有効です）。
4. **Where 句の全体表示**に送信される Where 句の内容が表示されます。

注：　物理トランザクション（タスク特性のトランザクションモード**特性を *P= 物理*に設定）を使用してい場合のみ、このオプションは有効になります。**

他の SQL ステートメントを手動で入力する



WHERE 句ステートメントより複雑なものを入力したい場合、ダイレクト SQL ステートメントを使用することで実現できます。

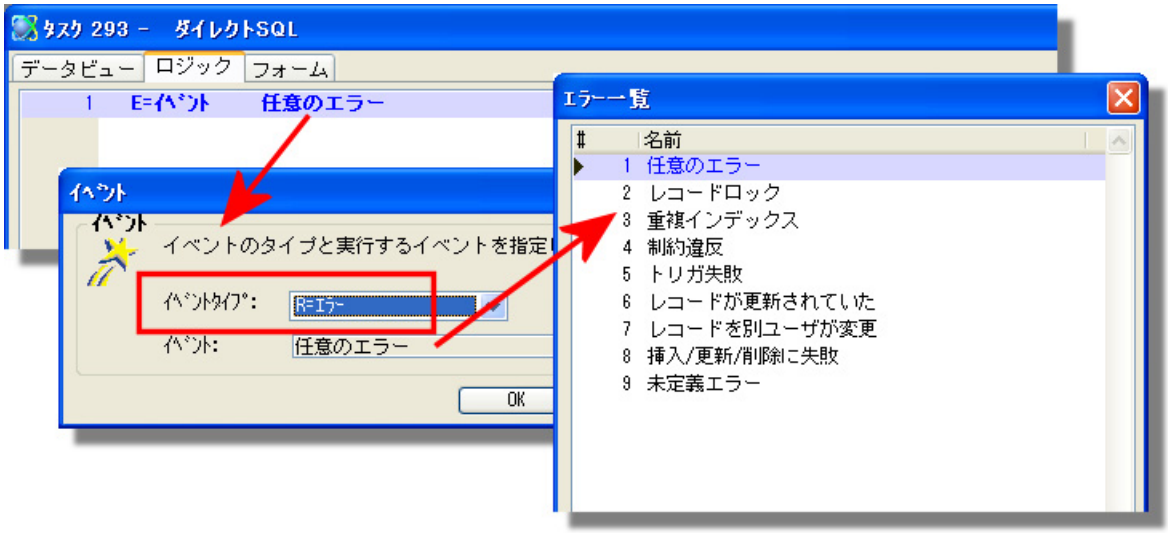
1. データビューエディタを開きます。
2. １行目のヘッダ行で ***Q=SQL コマンド***を選択し、***Tab*** 移動します。
3. **SQL コマンド**ダイアログが表示されます。ここで SQL コマンドを入力することができます。アシストボタンをクリックすることで SQL コマンドを入力を支援するための**アシスト**ダイアログが表示されます。
4. **出力パラメータ**から**ズーム**して、タスクに定義された変数項目と SQL コマンドで定義されたフィールドを割り当てます。

メインソースのカラム項目と同じように変数項目をプログラム内で使用することができます。

データベースエラーまたは例外を処理するには

プログラムがデータベースで発生するエラーを処理する場合、Magic のデフォルト機能を使用しないで、プログラム内で独自のエラー処理を行うようにすることができます。**イベントタイプ**特性が ***R= エラー***と定義されているイベントハンドラを使用することで実現できます。

エラーハンドラを作成する



1. **ロジックエディタ**を開きます。
2. ***Ctrl+H*** を押下してヘッダ行を作成します。
3. ***E*** を入力して***イベント***を選択します。カーソルが次のカラムに移動します。
4. **ズーム**して**イベント**ダイアログを開きます。
5. **イベントタイプ**で ***R= エラー***を選択します。
6. **イベント**から**ズーム**して取得したいエラーを選択します。
7. **指示**特性を設定し、Magic エンジンに対してどのようなエラー対応処理を行うかを指定します。エラー対応とは、エラーハンドラが実行された後にエンジンが実行する処理を意味します。***S= タスク特性に依存***を選択すると、**タスク特性**の**エラー発生時**特性（***Ctrl+P*** →**データ**タブ）の設定内容にもとづいて動作します。***I= 無視する***や ***B= ロールバックして再起動***などの他のオプションを指定することで処理を明示的に指定できます。各エラーに対するサポートされたエンジンの動作。異なるエラー処理については、『リファレンスヘルプ』を参照してください。
8. DBMS の実際のメッセージを Magic で自動的に表示させるには、**メッセージ**特性を ***Yes*** に設定する必要があります。これにより、**動作環境**（**オプション→設定→動作環境→動作設定**）の**メッセージの詳細表示**の設定内容を上書きすることができます。

いくつかの異なるエラーを処理することができまが、どのようなデータベースエラーが発生した場合でも処理させたい場合は、***任意のエラー***を選択します。

これで、選択されたエラーに対応するハンドラが定義できました。次に、このエラーを処理するための処理コマンドを追加することができます。

処理コマンドが何も定義されず、**指示**特性が ***I= 無視する***に設定されている場合、エラーが発生しても何も行われません。レコードの重複エラーを無視させたい場合など、重要でないエラーと認識した上であればこのような設定を行うこともできます。

ハンドラ内で行うことができる事の 1 つは、ユーザにエラーメッセージを表示したり、エラーメッセージをログに書き込んだりすることです。Magic には、エラーに関する情報を表示したり保存するために使用できる **ErrDatabaseName()**、**ErrDbmsCode()**、**ErrTableName()**、および **ErrDbmsMessage()** などの一連の関数があります。

しかし、**メッセージ**特性を ***Yes*** に設定した場合、メッセージが表示された後でメッセージバッファはクリアされるため **DbERR()** と **ErrDbmsMessage()** 関数は空白を返します。従って、プログラム内で DBMS エラー情報を使用し、メッセージを表示させたい場合は、**メッセージ**特性を ***No*** に設定して**エラー**処理コマンドを使用して手動でメッセージを表示してください。

エンドユーザのアクセスとデータ操作を制限するには

APG を使用してプログラムを作成した場合、デフォルトでは、ユーザはすべてのカラムにアクセスし、すべてのレコードを変更することができます。しかし、タスクの様々な特性を変更することで Magic によるテーブルデータのアクセスを制限させることができます。ここでは、テーブルへのアクセスを制限するためのオプションの設定方法について説明します。

- **初期モード**特性（**タスク特性→汎用**タブ）：プログラムの初期モードを設定します。照会と設定された場合、カラム内のデータを修正することができません。
- **オプション**（**タスク特性**）：ここには、一連の特性があり、*Yes*、*No*、または*式*が指定できます。例えば、**修正**が *No* に設定された場合、ユーザは、タスクを修正モードに変更することができなくなります。**削除**が *No* に設定された場合、レコードを削除することができなくなります。
- **アクセス**特性（**メインソース特性**または**リンク特性**）：この特性は、テーブルに対するアクセスモードを設定します。読込に設定された場合、テーブルは読込モードでオープンされ、タスク環境がどのように設定されていてもテーブルのデータは保存されません。他の特性によって修正が許可されていてレコードが不注意に更新しようとした場合、エラーメッセージが表示され、レコードは更新されません。
- 上記の設定を全く行わない場合は、タスク内に必要なカラムのみを定義するようにします。
- **照会モード時の更新許可**（**動作環境→システム**タブ）を *No* に設定します。タスクが照会モードの場合、データが入力できなくなります。この設定を、*Yes* にすると**項目更新**処理コマンドによるデータ修正ができるようになります。

データベースのレコードの取得順を指定するには

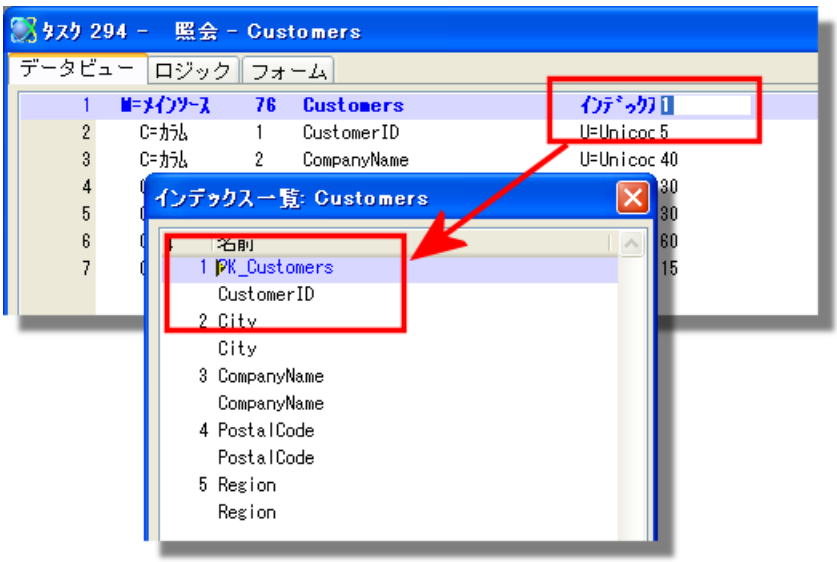
レコードをテーブル形式で表示させる場合、レコードが検索される順序が重要になってきます。

エンドユーザは、最も理解し易い方法レコードを参照することを望むため、レコードの並び順はユーザインタフェースを考える上で重要な要因です。また、レコードをフィルタリングする上で最も効率的な方法で検索する必要があるという点からも、レコードの並び順は機能面を考える上で重要な要因になります。

Magic では、レコードの並び順を制御する上でいくつかの方法があります。そのうち以下の主要な２つの方法について説明します。

- テーブルのインデックス
- タスクソートの使用

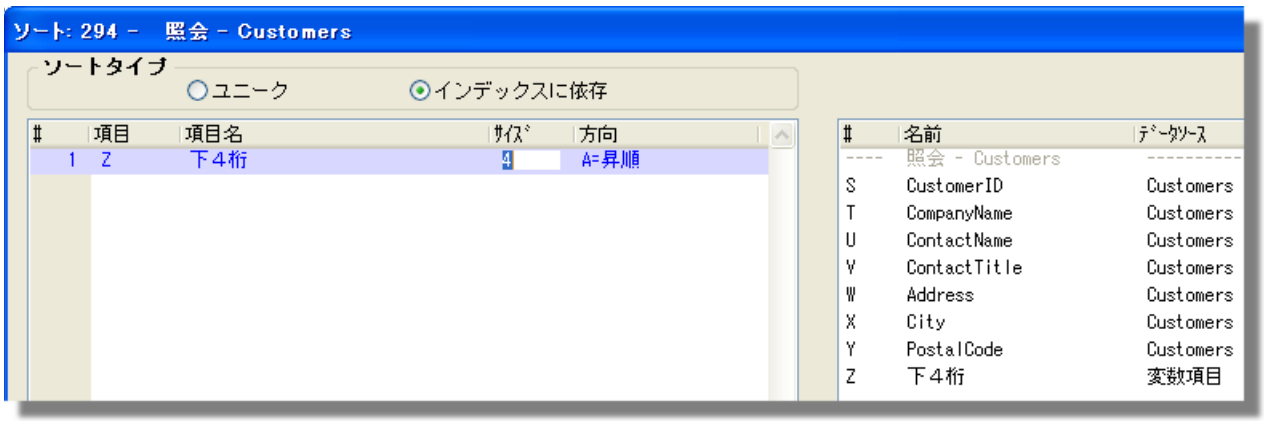
テーブルインデックスを設定する



メインソースやリンクテーブルをタスクで定義する場合、**インデックス特性**にて使用するインデックスを指定します。この例では、Customers というデータソースの PK_Customers インデックスを指定しています。

メインソースに対しては、使用する範囲条件に最も有効なインデックスを使用することで効率的に動作します。リンクされたソースに対しては、リンクで使用される位置付条件に最も有効なインデックスを使用することで効率的に動作します。この例では、Customer_ID が A001 から A003 までの範囲で検索する必要があるので、PK_Customers をインデックスとして使用することが最も効率的となります。

タスクソートを使用する



タスクソートは、最初にレコードが表示された後にその並び順を変更することを可能にする機能です。タスクソートを使用することでとても柔軟に表示させることができます。例えば、何らかの値で初期設定された変数項目を定義し、ソート処理時にこの変数項目を使用することができます。

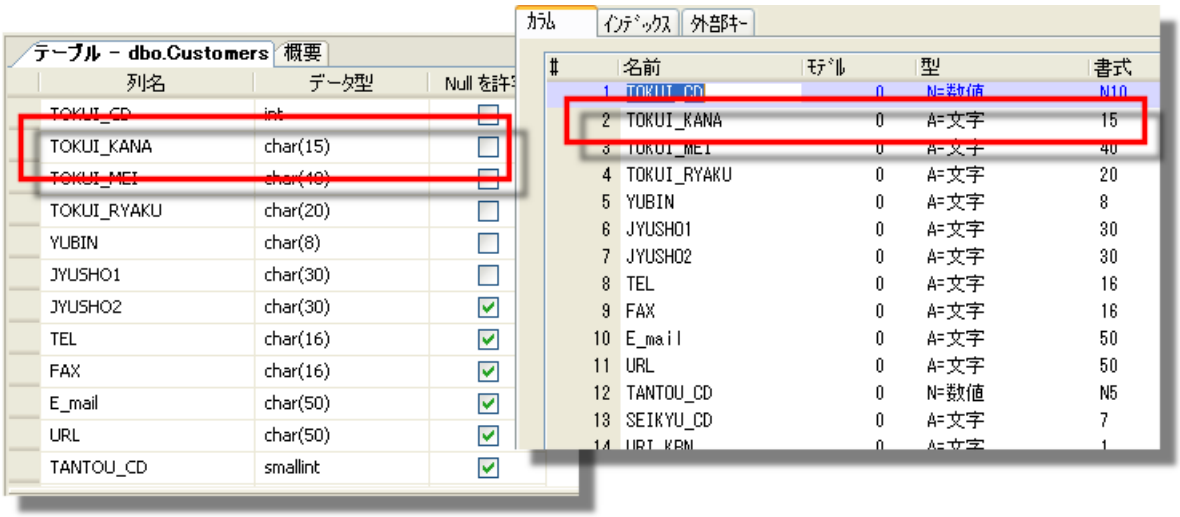
タスクソートを設定するには以下のようにします。

1. ソートテーブル（タスク→ソートまたは、*Ctrl+T*）を開きます。
2. 左側のカラムで *F4* を押下して 1 行追加します。
3. 右側のリストからソート処理で使用する項目を選択します。
4. 項目が長い場合、サイズカラムの値を変更することでソートに使用する文字の長さを指定することができます。
5. ソートの順序を逆にしたい場合、方向カラムで降順を選択します。
6. 同様な方法で、必要な項目を選択します。

これで、タスクが実行されるとウィンドウに表示される前にレコードはソートされます。この例では、PostalCode の下 4 桁の文字によってソートされています。

参照： 第 5 章：「レコードの表示順を動的に変更するには」（77 ページ）
第 18 章：「決められた順番でデータベーステーブルからレコードを取得するには」（412 ページ）を参照してください。

特定の SQL タイプにアクセスするには



Magic の各カラムは、データベース内の対応するデータを表しています。すべてのデータベースサーバは、独自のデータ型を持っています。しかし、Magic はデータベースゲートウェイを使用して、それらのタイプを Magic で処理できるように変換しています。

データベースサーバと Magic とのデータのタイプの対応については、Magic の『リファレンスヘルプ』(データ管理 >SQL に関する考慮事項 > 構成とパフォーマンス >SQL ゲートウェイでサポートされるパラメータ) を参照してください。

しかし、既存のテーブルで定義取得を行った場合や、Magic で作成されたテーブルの内容を参照することで、Magic の変換処理内容を確認することができます。

上記の例では、２つのテーブルが表示されています。左側は、MS-SQL Server の管理ツールにて Customers というテーブルのカラム内容を表示しています。右側では、同じテーブルを Magic で定義取得した場合のカラムテーブルを表示しています。

各カラムの特性を見てみると、NULL の扱いなど、各カラムがどのように定義されているかを確認することができます。オリジナルのデータベース定義タイプは、カラム特性の SQL セクションのタイプ特性に設定されています。

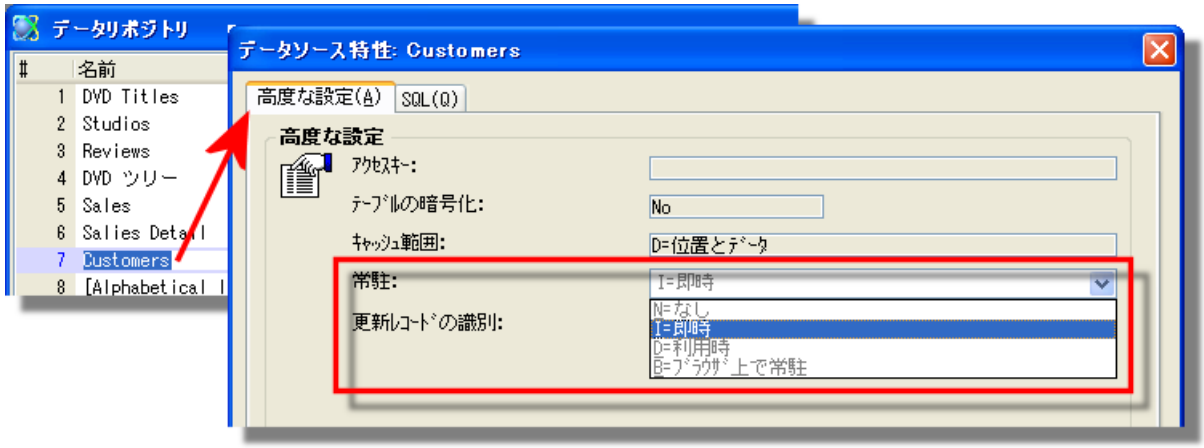
いくつかのカラムは特別な処理が行われます。DATETIME タイプのカラムは、Magic で 2 つの個別の項目として処理できるように日付型と時刻型の 2 つのカラムに分割されます。しかし、データベースに対しては DATETIME カラムとして格納されます。

読込専用データのアクセスを最小化するには

データにアクセスするためのリクエストをデータベースサーバに送る必要があるため、データベースアクセスはプログラムの処理速度を落とす傾向があります。アクセス方法によっては、より時間がかかるものがあります。ロックされたデータを読み込む処理は、読込専用のデータを処理する場合より時間がかかります。

しかし、(参照用テーブルのように) 頻繁にアクセスするが変更が少ないような場合、アプリケーションがオープンされた時点 (または、テーブルが最初にアクセスされた時点) で一回だけ読み込むようにして、実行中は読み込んだ内容を利用し続けるようにすることができます。

これは、テーブルを***常駐テーブル***として定義することで実現できます。



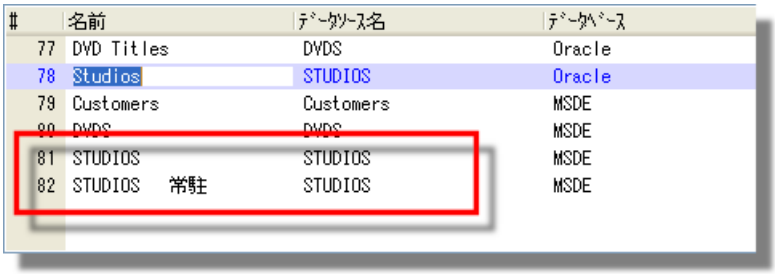
常駐テーブルの設定を変更する

1. データリポジトリで、変更したいデータソース上にカーソルを置きます。
2. *Alt+Enter* を押下して**データソース**特性にアクセスします。デフォルトとして**高度な設定**タブが表示されます。
3. **常駐**特性で設定したいオプションを選択します。

ここには以下のオプションがあります。

- ***N= なし*** : デフォルト設定です。タスクが起動されるたびにテーブルはオープンされます。
- ***I= 即時*** : アプリケーションの起動時に、テーブルは 1 回だけオープンされます。このテーブルを使用するタスクは、最初にオープンされた時点でのデータを利用することができます。
- ***O= 利用時*** : このテーブルを使用するタスクが最初に起動された時に、1 回だけオープンされます。この後、他のタスクはオープンされた時点でのデータを利用することができます。
- ***B= 即時でブラウザ上*** : これはブラウザクライアントタスクでのみ有効です。テーブルの内容がクライアント側のブラウザにダウンロードされます。

常駐テーブルの内容を更新する



データソースを常駐テーブルとして定義した場合、アプリケーション内のどのプログラムもそれを更新することができなくなります。

同じアプリケーション内でそのデータソースを更新したい場合は、**データ**リポジトリでデータソースのコピーを作成し、このコピーの**常駐テーブル**特性を ***No*** に設定する必要があります。この例では、STUDIOS テーブルの 2 つのコピーが定義されています。一方は常駐テーブルで、他方は常駐にはなっていません。

次に、データソース #81 を更新するプログラムを作成することで、このプログラムによる更新内容は STUDIO テーブルに格納されます。しかし、他のプログラムが使用している STUDIOS テーブルのコピー（データソース #82）は常駐テーブルとして設定されているため、この内容には反映されません。

この場合、**DBReload()** と呼ばれる Magic 関数を使用することで STUDIOS テーブルのコピーをリフレッシュする必要があります。**DBReload()** 関数は、データソース番号（とオプションでデータソース名）をパラメータとして指定します。**DBReload()** 関数が実行されると、メモリ上に読み込まれているデータソースの内容がリフレッシュされます。

データベースのアクセス頻度を減らすには

Magic は自動的にデータベースへのアクセス処理を行うため、レコードをいつフェッチするかを指定するステートメントを記述する必要がありません。しかし、アクセス頻度に影響する要素を制御する必要があります。アプリケーションに最適な方法で Magic がデータベースにアクセスする特性を設定することは、プログラムの処理速度に最も大きな影響の与える要因の 1 つになります。

ヒント : *テーブルのオープン中に、データベースへのアクセス回数を確認するには、デバッガやネイティブな SQL プロファイラーツールを使用することができます。プログラムがリソースを効率的に使用しているかどうかを確認するために、どのように処理されているかを確認することを推奨します。*

読込専用テーブル

アプリケーションが使用するテーブルの中には、主にコードの参照用として使用されるものがあり、これらは頻繁に更新されるようなことはありません。このようなテーブルに対しては、**常駐**特性のような特殊なアクセス特性を使用することができます。この特性については、「読込専用データのアクセスを最小化するには」（748 ページ）を参照してください。

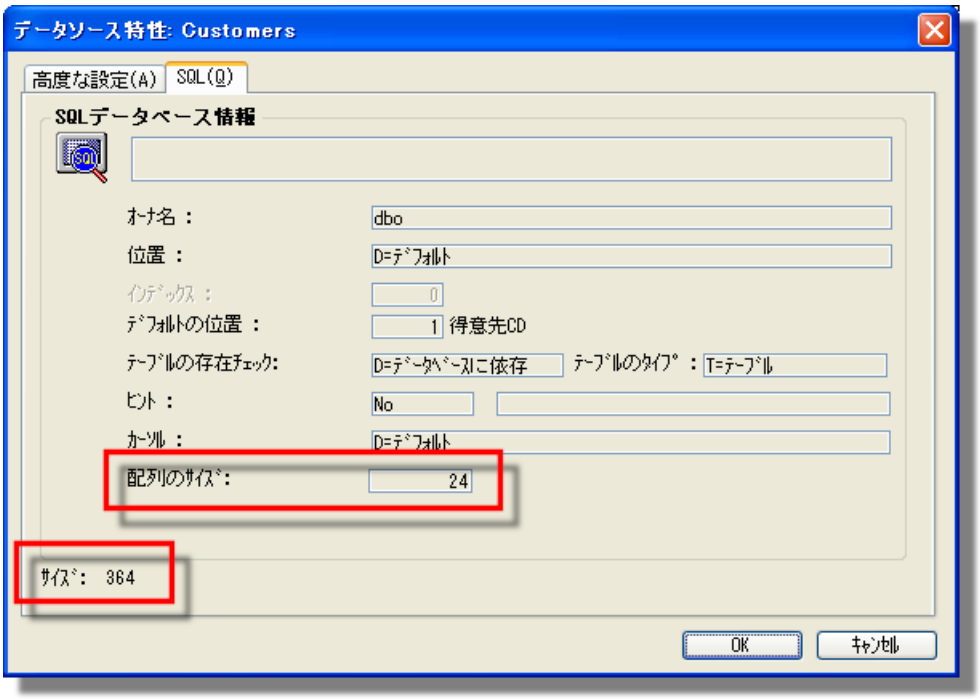
レコードのフェッチの制御

レコードからいつ、どれだけのレコードをフェッチするかを指定するための設定もあります。以下の設定内容について説明します。

- 配列サイズ
- キャッシュ
- ビューの事前読込
- 範囲やダイレクト SQL の使用

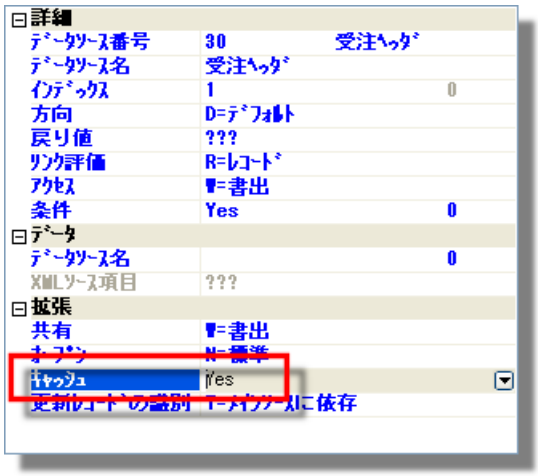
配列サイズ

Magic がレコードをフェッチする際、一度に全てのレコードをフェッチせず、ブロック毎にフェッチします。**データソース特性**の**配列サイズ**特性によって、一回にフェッチするレコード数を指定することができます。



配列サイズが *0* に設定された場合、**データベース特性**に設定された値がデフォルト値として使用されます。**データベース特性**の**配列サイズ**も *0* の場合、Magic のデフォルトが使用されます。

Magic のデフォルト値は、1200/ レコードです。これは、レコードが 200 バイト長の場合、6 つのレコードを一度にフェッチすることができます。SQL ファイルの場合、実際のレコード長はフェッチされるカラム数に応じて、実行時に変動します。



キャッシュ

データベースのアクセス頻度を減らすために**キャッシュ**特性を使用することができます。

キャッシュは、すべてのテーブルに対してオンラインタスクで使用することができます。

メインソースの**キャッシュ範囲**特性が設定（***位置とデータ***）されている場合、Magic はメインソースのアクセス時にデータベースからレコードをフェッチしません。リンクテーブルにキャッシュが設定されている場合、Magic は一度データベースからレコードを読み込み、同じタスク内で他のレコードをリンクする必要があった場合も、この読込内容を再利用します。タスクがテーブルをオープンしている間は、キャッシュ内容は保持されます。

ビュー事前読込

ビュー事前読込特性（**タスク特性→データ**）は、タスクのウィンドウがオープンされる前にフェッチされるレコード数に影響します。この特性が ***Yes*** に設定されている場合、ウィンドウがオープンされる前に全てのコードがフェッチされます。この設定によって、スクロールバーを正確に表示させることが可能になります。しかし、レコード数が非常に多くユーザが最後までスクロールする事がないような場合、必要以上のレコードをフェッチする結果になってしまいます。

範囲とダイレクト SQL の使用

フェッチされたレコード数を減らすためには、データベースサーバ側でできるだけフェッチするレコード数を絞るように指定するようにしてください。

例えば、期限切れの総額が 12,000 ドルを越える全ての顧客レコードを印刷したい場合、**フォーム出力**処理コマンドの条件を使用することで実現することができます。しかしこの場合は、条件を評価するためにすべての顧客レコードをフェッチする必要があります。場合によっては、数百レコードのフェッチ処理が必要になるかもしれません。このような場合は、フェッチするデータに範囲条件を設定することで、Magic は 1 つの SQL ステートメントを発行し、サーバは期限切れの総額が 12,000 ドルを越えるレコードのみを返すようにします。何人のユーザがお金を借りているかに依存しますが、場合によっては 1 回のフェッチ処理だけで済むことにもなります。

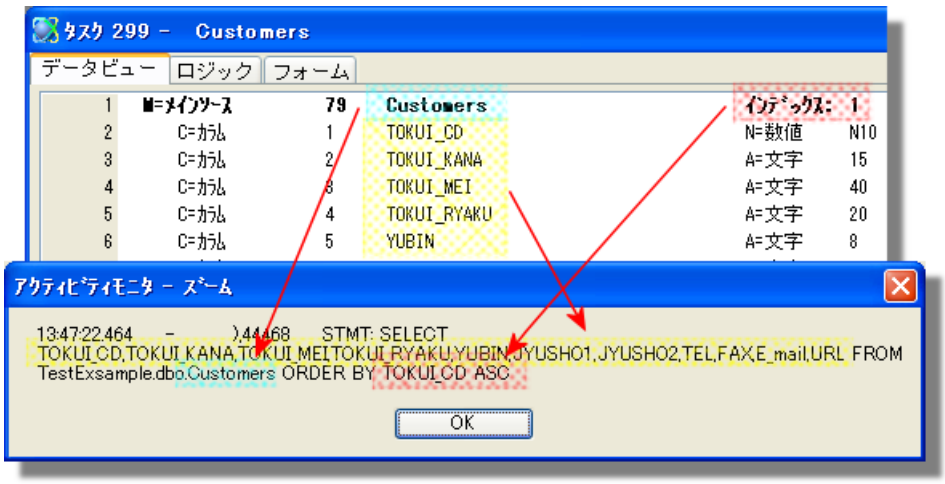
ダイレクト SQL ステートメントが、データを処理する方法として最も効率的に扱われるケースとして、インスタンスがあります。この詳細については、「独自の SQL ステートメントをデータベースに送るには」（741 ページ）」を参照してください。

データベースに送られる SELECT ステートメントを制御するには

タスクを作成するために Magic Studio を使用する場合、データベースへのクエリを迅速に記述するために Studio インタフェースを使用します。データベースゲートウェイが処理するため、使用する SQL、ISAM、XML、またはメモリテーブルに対して、実際どのようにアクセスするかを知る必要がありません。

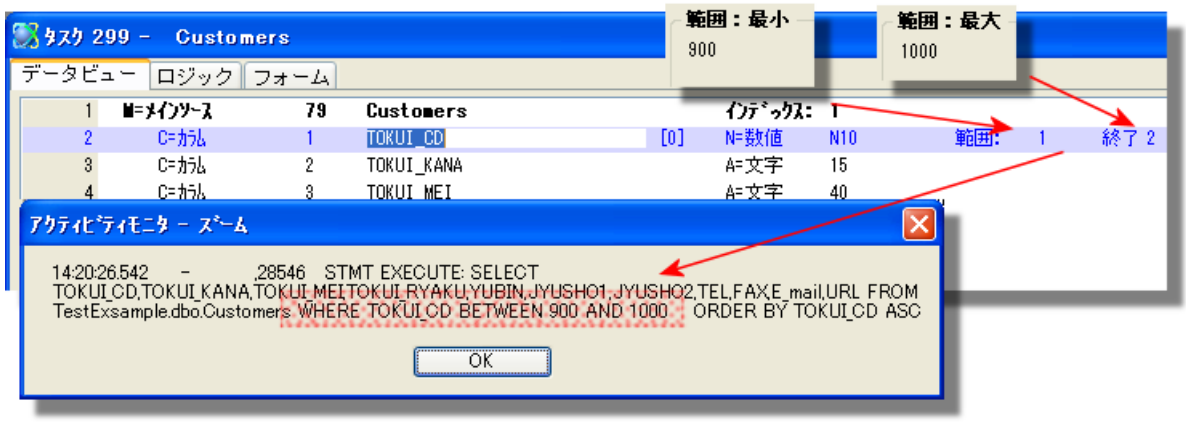
しかし、Magic プログラムの構成によって発行される SQL クエリが、どのように影響されるかは知っておく必要があります。

注： デバッガを使用することで、作成される SQL クエリを確認することができます。詳細は、「データベースに送信される SQL ステートメントを参照するには」（740 ページ）を参照してください。



SQL クエリに影響する要因を以下に列挙します。

- カラムの選択 …… 選択されたカラムはクエリに含まれ、データビューには選択された順番にカラムがフェッチされます。
- Magic 側でのインデックスの指定(ユニークであるかどうかにかかわらず) …… インデックスは ORDER BY 句に変換されます。



- タスク範囲 …… 範囲条件は、Where 句として含まれます。
- タスクソート …… 定義されたタスクソートは、ORDER BY 句に変換されます。

そして、ダイレクト SQL ステートメントをクエリに追加することもできます。詳細は、「独自の SQL ステートメントをデータベースに送るには」（741 ページ）を参照してください。

複数のユーザが同じレコードを修正した場合の Magic の動作を決定するには

Magic は指定されたトランザクションの内容にもとづいて、修正されたレコードに対して異なる動作を実行します。

遅延トランザクションが最初に Magic で処理され、次に個別の手順でデータベースにコミットする間、物理トランザクションが処理されます。

物理トランザクション使用時の修正行の識別

物理トランザクションを使用している場合、行がどのように識別されるかは使用する DBMS の設定に依存します。

遅延トランザクション使用時の修正行の識別

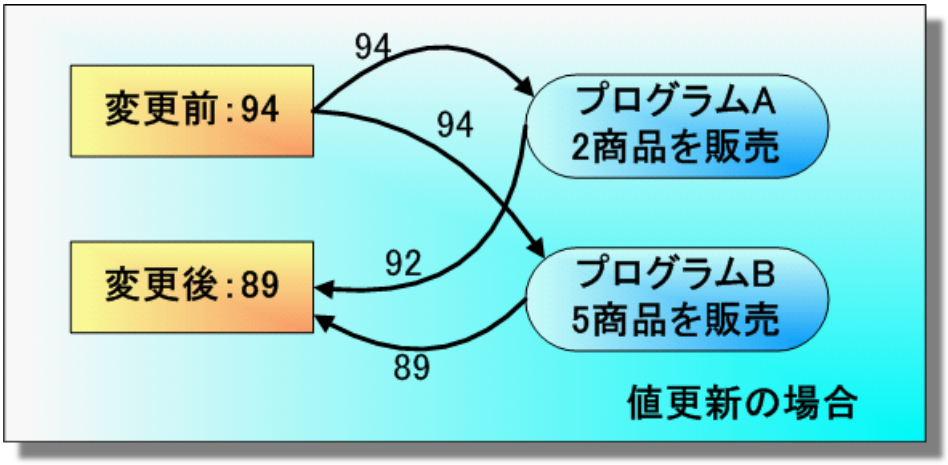
最初に、Magic は行が修正されたことを識別する必要があります。この処理は、**メインソース特性**の**更新レコードの識別**特性に依存します。この特性には、３つの異なるオプションがあり、設定内容によって以下のように動作が異なります。

2 つのユーザ（ユーザ A、およびユーザ B）が SQL テーブルで同じレコードを更新している場合を例にとって説明します。どちらのユーザも更新のために同じレコードをフェッチします。「ユーザ B」が更新する前に「ユーザ A」がレコードを更新します。

- ***位置*** : 修正されたレコードのユニークな ID のみを Magic が参照します。従って、ユーザ A によって書き込まれたレコードは、ユーザ B によるレコード更新によって上書きされます。ユーザ B による更新のみが保存され、エラーは発生しません。
- ***位置と更新項目*** ：修正されたレコードのユニークな ID と、どの項目が更新されたかを Magic が参照します。従って、ユーザ A とユーザ B が異なる項目を更新した場合、両方の更新内容が保存されエラーは発生しません。もし同じ項目が更新された場合、ユーザ B に対してエラーが発生し、変更内容は保存されません。
- ***位置と選択項目*** ：修正されたレコードのユニークな ID と、（更新されたかどうかに関わらず）タスクで定義された項目を Magic が参照します。従って、ユーザ A とユーザ B が実行しているタスクに同じ項目が定義されている場合、ユーザ B に対してエラーが発生し、変更内容は保存されません。
- ***メインソースに依存*** ：**データソース特性**の**更新レコードの識別**特性に依存します。

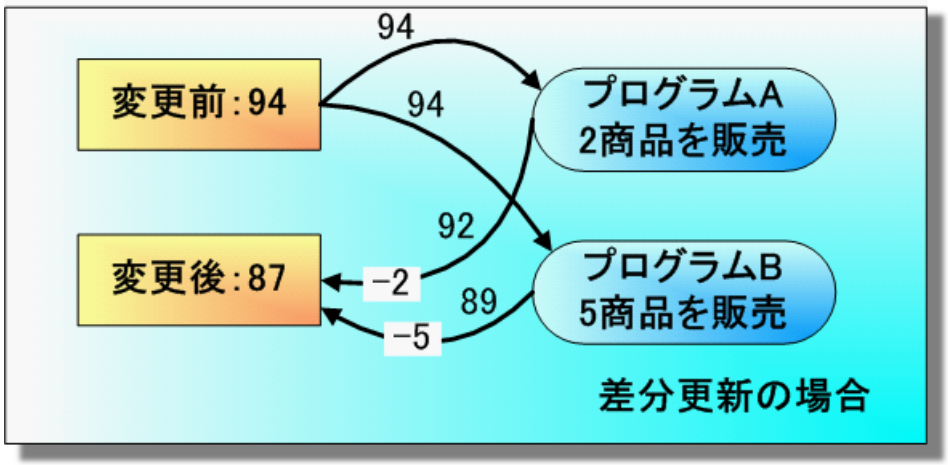
更新形式の効果

同じ数値データを更新する際に発生する問題を最小化するために**更新形式**特性を使用することもできます。



2 つのショッピングカート・プログラムを実行しているものとします。プログラム A は、在庫内に 94 の商品があることを確認しています。プログラム B も同じです。プログラム A は、2 つの商品を販売し在庫数を 92 に変更します。プログラム B は、5 つの商品を販売したことで在庫数を 89 に変更します。この場合、現在の在庫数が 89 になり、正確な値ではなくなります。

しかし、**データ**リポジトリの**カラム特性**で**更新形式**特性を差分に設定した場合、異なる更新結果になります。



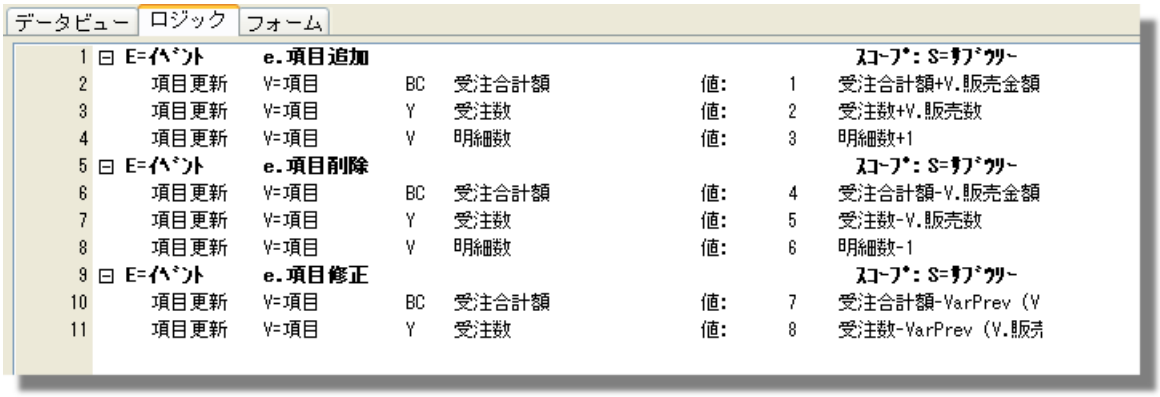
これは、前述のようなプログラムを実行させることで確認することができます。

しかし、項目の内容を 92 または 89 に変更する代わりに、項目の値は、読み込まれた値と書き込まれる値の差分をもとに算出されます。従ってプログラム A は 92 という値を書き込む代わりに、項目の値を 2 だけ減算します。また、プログラム B は 89 という値を書き込む代わりに、項目の値を 5 だけ減算します。

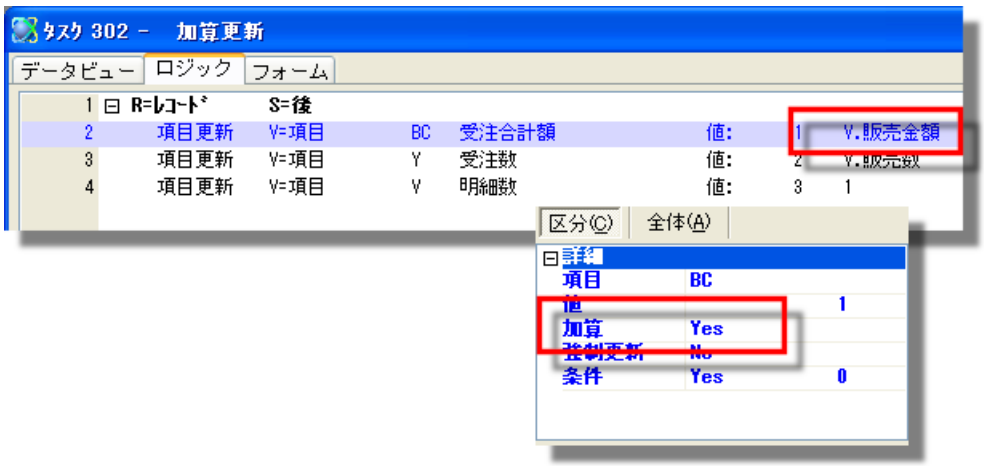
プログラム A またはプログラム B は、プログラムを全く変更していません。データソースのカラム特性のみを変更しています。

差分更新と加算更新の違い

差分更新と加算更新は、まったく異なることに注意してください。差分更新はデータソースのカラム特性に設定するのに対して、加算更新は項目更新処理コマンドの特性に設定します。加算更新をタスクで使用する場合、実際の項目更新処理コマンドには異なる定義が行われます。通常、以下に示されたようなサブレコードを使用して実行した場合、合計の追加や、修正、削除を行うための明示的なロジックを定義する必要があります。



しかし、加算更新を設定することで、レコード後で自動的にすべて処理されます。開発者はただ、追加または、削除される値を指定する必要があります。



この例では、レコードが作成されると、V. **販売金額**は**受注合計額**に追加されます。レコードが削除されると、**受注合計額**から V. **販売金額**の値が削除されます。また、V. **販売数**の値が変更された場合、**受注数**は、変更された V. **販売数**の値にもとづいて変更されます。明細レコードの合計を算出する方法として、加算更新は「スマートな加算方式」といえます。

データベース制約がある場合に、1 対多の関係を実現するには

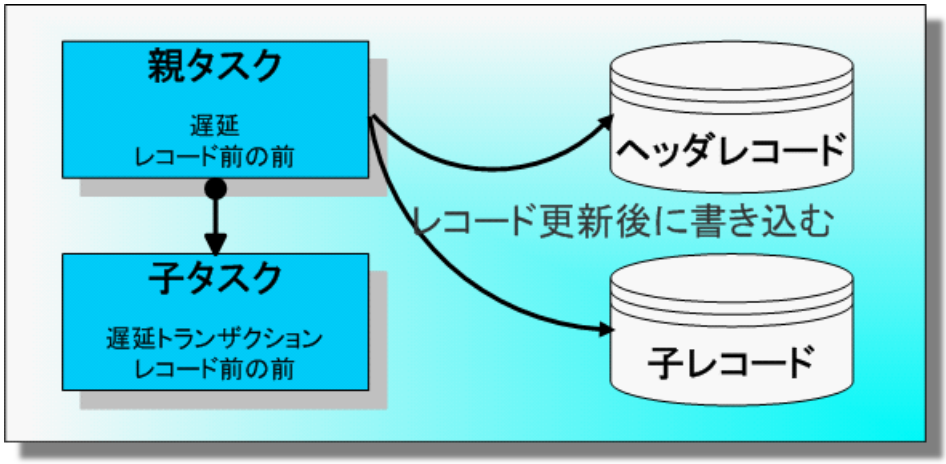
サブタスクが子レコードを表示し更新しながら、親タスクが 1 レコードを表示し更新するようなフォームを定義する場合があります。この場合、子タスクに制御が移っている時に、親レコードは実際にデータベーステーブルに書き込みを行っていない事があります。

このようなことを回避するために、子タスクに制御が移る前に親レコードをコミットする必要があります。以下の方法で、簡単にこのよな処理を実現させることができます。

- 親タスクの**トランザクションモード**特性を、***D= 遅延***または ***W= 親と同一***に設定します。
- 子タスクの**トランザクションモード**特性を、***D= 遅延***または ***W= 親と同一***に設定します。
- **コール**処理コマンド上で、**同期**特性を ***Yes*** に設定します。

コール処理コマンドではなくサブフォームを使用している場合、設定すべき**同期**特性はありません。しかし、親子タスクが、上記の説明のように設定された場合、**同期**特性が ***Yes*** と設定されたかのように、サブフォームタスクが自動的に呼び出されます。

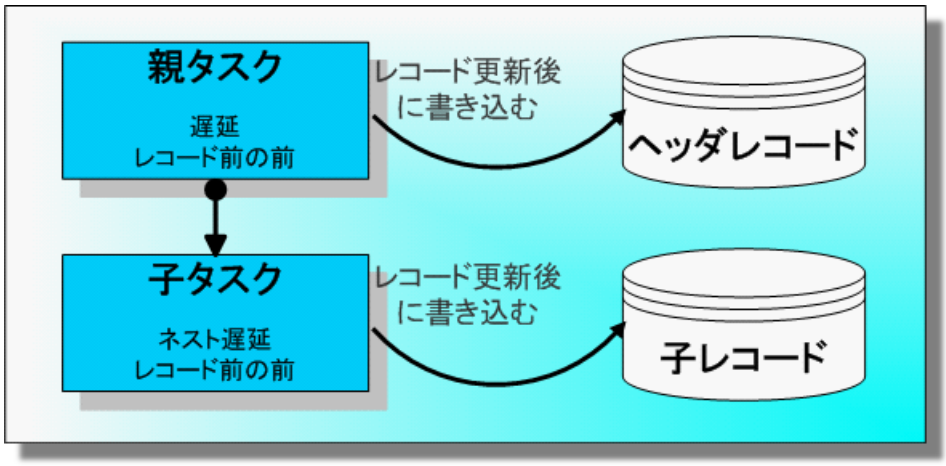
ネストされたトランザクションを実現するには



遅延トランザクションを使用している場合、あるタスクがトランザクションをオープンし、さらにそのタスクがトランザクションをオープンしているタスクを呼び出すような場合があるかもしれません。例えば、**トランザクションモード**特性が ***D= 遅延***で、**トランザクション開始**特性が ***P= レコード前の前***に設定された親タスクがあり、同じ設定の子タスクを起動したとします。

しかし、このような場合のコミット処理は、親タスクのレベルで同時に行われます。子タスクのレベルでロールバックをした場合、子タスクでの変更処理と同様に親タスクでの変更内容もロールバックされます。親子タスクは同じトランザクションを共有しています。

子タスクでの更新内容を親タスクとは独立して扱いたい場合は、以下のような***ネスト遅延***と呼ばれる異なる種類のトランザクションを使用する必要があります。



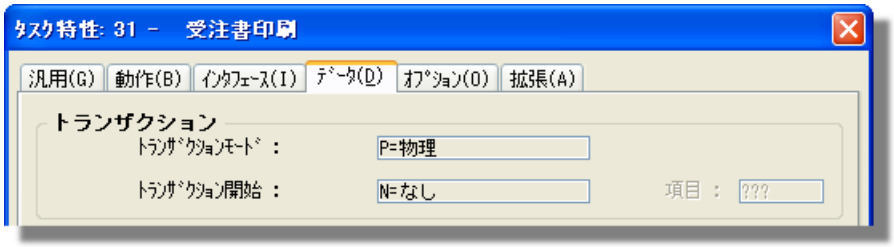
ここでは、親子タスクの両方で作成された更新内容をキャッシュします。子タスクのトランザクションは、子タスクが終了する前にコミットされます。ロールバックが子タスクで実行された場合、子タスクのトランザクションだけがロールバックされます。親タスクのトランザクションは、親タスク内で個別に処理されます。

ここでは、ネイティブな DBMS 内でのネストトランザクションを実行しているわけではありませんが、同じような結果になります。

注： **トランザクションタイプ**特性の ***W= 親と同一***は、物理または遅延トランザクションのどちらかを使用することで子タスクでも同じタイプを使用することができるようになりますが、ネストはされません。子タスクを ***W= 親と同一***に設定した場合、最初の例のように ***D= 遅延***に設定されたように動作します。

トランザクションをオープンしないようにするには

デフォルトでは、ほとんどのタスクはトランザクションをオープンします。しかし、トランザクションをオープンしないようにしたい場合は、以下のように設定します。



オンラインタスクの場合

1. **タスク特性**（*Ctrl+P*）を開きます。
2. **トランザクションモード**特性を、*P= **物理***に設定します。
3. **トランザクション開始**特性は、*N= **なし***に設定します。

ブラウザタスクの場合

1. **タスク特性**（*Ctrl+P*）を開きます。
2. **トランザクションモード**特性を、*N= **なし***に設定します。

明示的にトランザクションをロールバックするには

トランザクションを持つ背景にある主な考えとして、ロールバックが可能であるということが挙げられます。つまり、何か発生した場合、データをコミットしないようにすることができます。データベースエラーが発生した場合など、何らかの状況下で Magic は自動的にトランザクションをロールバックします。また、**Rollback()** 関数を使用することでイベントやデータ条件にもとづいたトランザクションのロールバックを手動で行うこともできます。

関数の詳細は、『リファレンスヘルプ』に記載されていますが、基本的に以下のように動作します。

構文：

Rollback(*Message?, NestingLevel***)**

パラメータ：

- ***Message?:*** 'TRUE'LOG を設定すると、ユーザ用の確認ダイアログが表示されます。
- ***NestingLevel***：どのレベルまでロールバックするかを示す数値を指定します。
  1…最も内側のネストトランザクションをロールバックします。
  2…最も内側のすぐ上位のトランザクションをロールバックします。
  0…最も外側のトランザクションをロールバックします。

データベースオプティマイザの動作を制御するには

通常、SQL データベースを使用して動作させる場合、データベースオプティマイザはステートメントの構文解析方法や、どのインデックスの使用するかを決定します。ほとんどの状況下で、オプティマイザはよい決定をし、可能な限り効率的に検索します。

しかし、オプティマイザの動作を変更するための機能がデータベースオプティマイザには組み込まれています。ヒントを SQL ステートメントに送ることで、これを可能にします。ヒントの正確なコード内容は、使用する SQL エンジンによって異なります。

Magic では、３つの異なるレベルで **SQL ヒント**を設定することができます。

- データベース：データベースレベルでヒントを設定した場合、ヒントはそのデータベースに対応した全ての SELECT ステートメントに含まれます。
- データソース：データソースレベルでヒントを設定した場合、ヒントはそのテーブルの送られるすべての SELECT ステートメントに含まれます。
- インデックス：インデックスレベルでヒントを定義した場合、ヒントはそのテーブルとインデックスに送られるすべての SELECT ステートメントに含まれます。

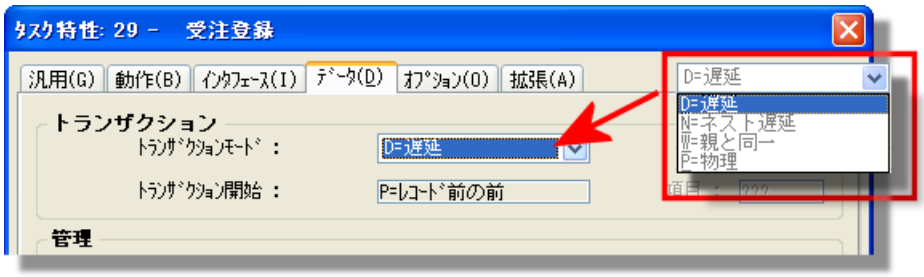
これらのレベルの各特性ダイアログ（***Alt+Enter***）の **SQL** タブ上に**ヒント**特性があります。さらに、SQL タスクでは、ヒントをコード化して指定することができます。

データベーストランザクションを初期化するには

Magic には、トランザクション処理が組み込まれており、デフォルトで自動的に初期化されます。この機能を効果的に使用する方法について理解することが重要です。

ここでは、Magic がどのようにデータベーストランザクションを設定するかについての基本的な説明を行っています。詳細は、Magic の『リファレンスヘルプ』を参照するようにしてください。また、プログラム内でトランザクション処理がどのように実行されているかが明確になっていない場合は、Magic のログファイルやネイティブなデータベースのログファイルをチェックしてください。

トランザクションのタイプの定義



トランザクションモード特性は、各タスクの**タスク特性**（*Ctrl+P*）の**データ**タブで設定します。オンラインまたはバッチタスクの場合、以下の 4 つのオプションがあります。ブラウザタスクの場合は、***P= 物理***の代わりに ***N= なし***を選択することができます。

基本的には、３種類の異なるタイプのトランザクションがあります。

- ***P= 物理*** ：全てのトランザクション処理は、使用する DBMS 側に依存しています。
- ***D= 遅延*** ：Magic 側で処理されます。Magic はデータ操作のステートメントをキャッシュし、必要であればロールバックを行います。データがコミットされたら、Magic はデータ操作ステートメントを DBMS サーバに送ります。
- ***N= なし*** ：トランザクションをオープンしません（詳細は、「トランザクションをオープンしないようにするには」（758 ページ）を参照してください）。

他の２種類のトランザクションのタイプは、物理と遅延の拡張です。

- ***N= ネスト遅延*** ：遅延トランザクションの特別なタイプです（詳細は、「ネストされたトランザクションを実現するには」（757 ページ）を参照してください）。
- ***W= 親と同一*** ：親タスクでトランザクションが使用されていても、物理または遅延トランザクションを開始します。

従って、Magic タスクを設定する場合に最初に決めることは、トランザクションモードを ***P= 物理***か、***D= 遅延***か、***N= なし***のどれにするかになります。遅延トランザクションは、多くの柔軟性があり、遅延トランザクションを使用する場合でのみ利用できるいくつかの機能（ネストトランザクションなど）があります。ブラウザタスクを使用する場合、物理トランザクションは選択できません。

トランザクションツリー

どのタイプのトランザクション処理を使用するか決める場合に、プログラムのツリー構造に留意する必要があります。トランザクションが子タスクでどのように動作するかは、親タスク側のトランザクション設定に依存します。

ここでは、1 つのヘッダレコードと 3 つの子レコードを表示する受注画面を考えてみます。ヘッダレコードを変更すると、すべての 3 つの子レコードが変更されます。

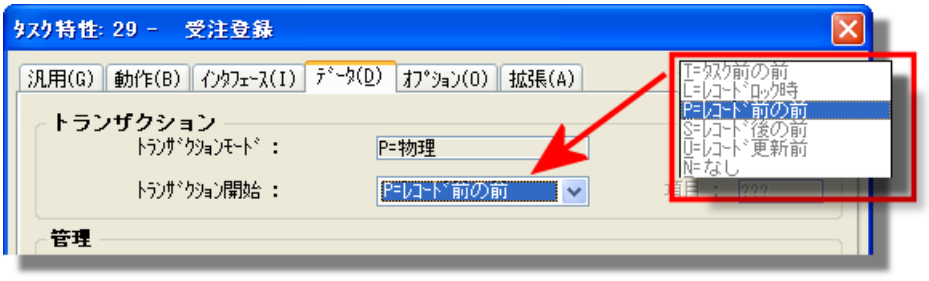
最後の子レコード上でパークしている時に、子レコードのレベルでロールバックをします。そして、親レベルでロールバックをする処理を繰り返します。ここに、各設定に対する結果が表示されています。（全てのトランザクションが***レコード前の前***に設定されています）。

親タスク	子タスク	結果	親でロールバック	子でロールバック
遅延	遅延	子タスクの変更は、親タスクでの変更の一部と見なされます。これらはまとめてコミットされるかロールバックされます。	親タスクの変更と子タスクの全ての変更がロールバックされます。	親タスクの変更と子タスクの全ての変更がロールバックされます。
遅延	親と同一	延期 - 遅延　の場合と同じです。		
遅延	ネスト遅延	各子タスクの変更と各親タスクの変更は独立しているものと見なされます。ユーザがそのレコードから抜けた場合、各レコードはコミットされます。	親タスクの変更（まだレコード上にカーソルがあるためコミットされていません）がロールバックされます。	最後のコミットされていない子タスクの変更だけがロールバックされます。
遅延	物理	各子タスクの変更と各親タスクの変更は独立しているものと見なされます。ユーザがそのレコードから抜けた場合、各レコードはコミットされます。	親タスクの変更（まだレコード上にカーソルがあるためコミットされていません）がロールバックされます。	最後のコミットされていない子タスクの変更だけがロールバックされます。
物理	物理	子タスクの変更は、親タスクでの変更の一部と見なされます。これらはまとめてコミットされるかロールバックされます。	親タスクの変更と子タスクの全ての変更がロールバックされます。	親タスクの変更と子タスクの全ての変更がロールバックされます。
物理	親と同一	物理 - 物理と同じです。		
物理	遅延	エラー		

親タスクでトランザクションがオープンされた場合、子タスクは独自のトランザクションをオープンせず同じトランザクションを共有することは通常よく行われます。例外としては、遅延トランザクションが設定された親タスクが、物理トランザクションかネスト遅延に設定された子タスクを起動する場合です。

また、物理トランザクションとして設定された親タスクが、遅延トランザクションとして設定された子タスクを起動した場合、エラーが発生します。（タスクが複数の異なるプログラムから呼び出される可能性が考えられるため）親プログラムのトランザクションモードがどのように設定されているか確認できない場合、子タスクでは ***W= 親と同一*** を設定することが最も安全な方法です。この場合、物理または遅延トランザクションのどちらかを使用する親タスクと同じ**トランザクションモード**で実行されます。

トランザクションの開始と終了の場所の定義

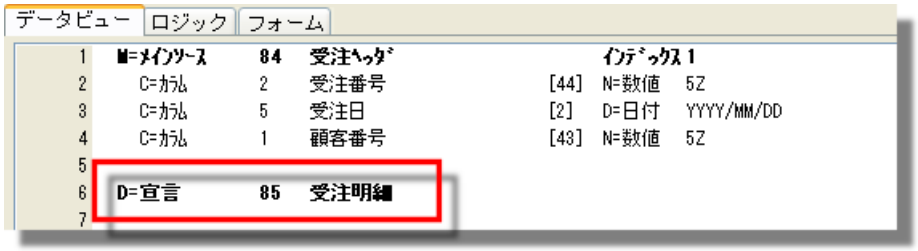


トランザクション開始特性は、トランザクションをいつオープンするかを指定します。物理トランザクションの場合、図のような６つのオプションがあります。遅延トランザクションの場合は、***T= タスク前の前***と ***P= レコード前の前***のみ選択できます。

トランザクションは、開始時と同じレベルで終了します。トランザクションがタスクレベルで開始された場合、タスクの終了時にそれは終了します。トランザクションがレコードレベルで開始された場合、ユーザがそのレコードを抜けた時点で終了します。

しかし、前のセクションで説明したように、現在のタスクが親タスクのトランザクションを共有している場合があります。この場合、トランザクション開始の設定に関係なく子タスクの実行中はオープンされ続けます。

関係するテーブルの定義



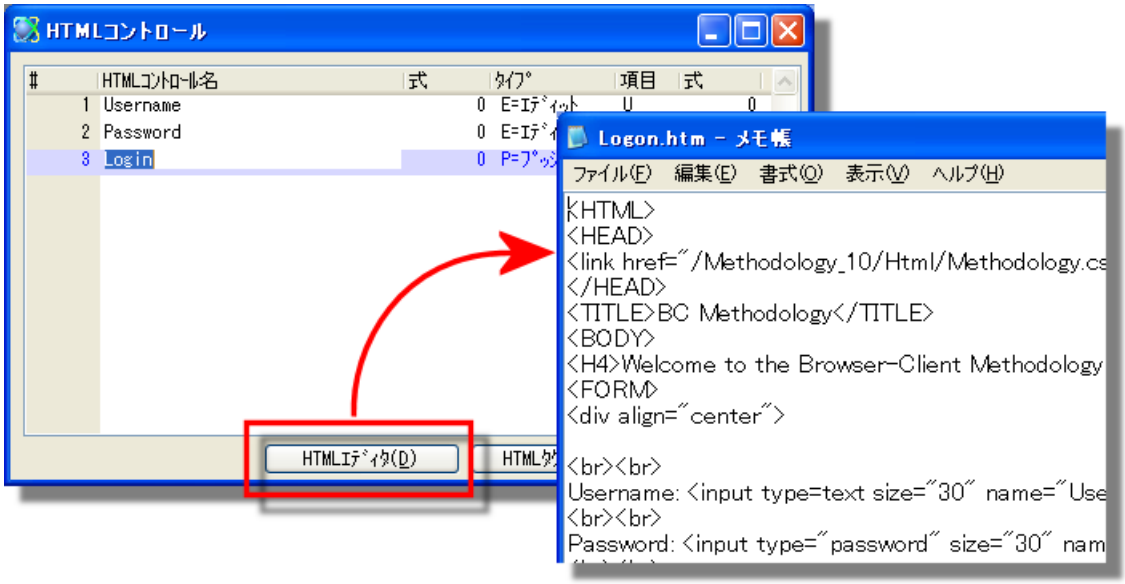
タスクツリー上の親タスクでトランザクションを開始している場合、トランザクションに関与する可能性のあるすべてのテーブルが親タスクで宣言されていることを確認する必要があります。親タスクでテーブルを宣言することで、上位レベルでテーブルがオープンされるため、トランザクションの設定を行うことでデータアクセスがより早くなります。

［このページは意図的に空白にしています。］

第 37 章： ブラウザクライアント

注： Magic V10 では、ブラウザクライアント機能はベータ扱いになっています。

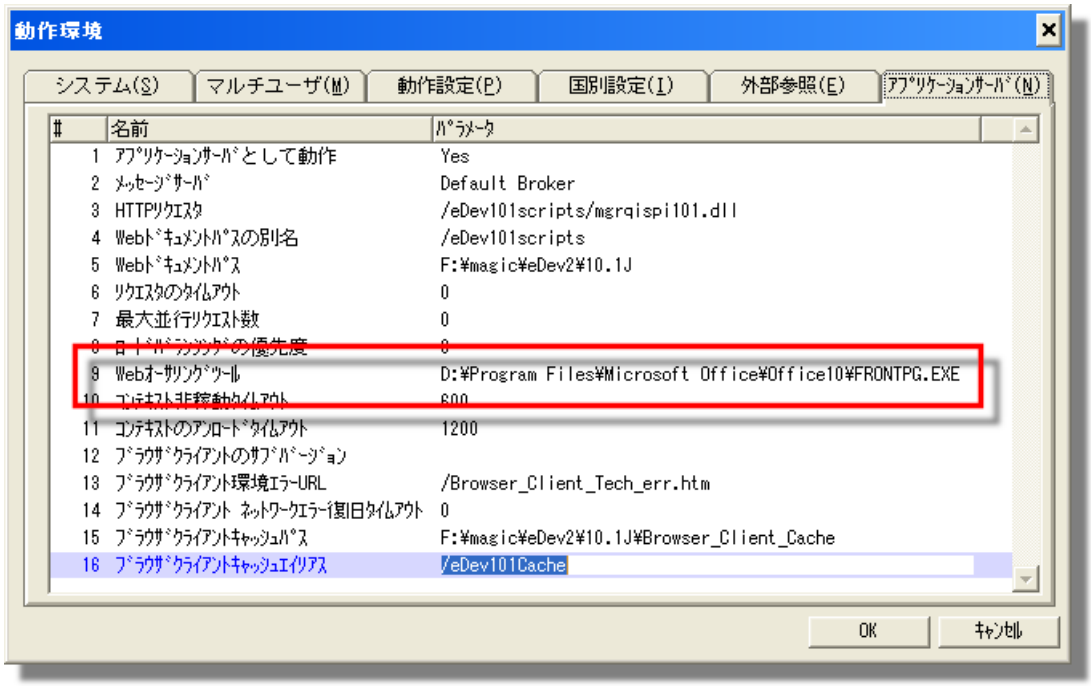
よく利用する HTML エディタを設定するには



マージプログラムのフォームから**ズーム**したり、ブラウザクライアントプログラムの HTML エディタで**クリック**することで直接 HTML を編集することができます。デフォルトでは、メモ帳（ノートパッド）が起動されます。

しかし、使用するエディタを指定することができます。HTML の編集用に設計されたエディタ（Front Page や Dreamweaver など）を使用した方が便利です。

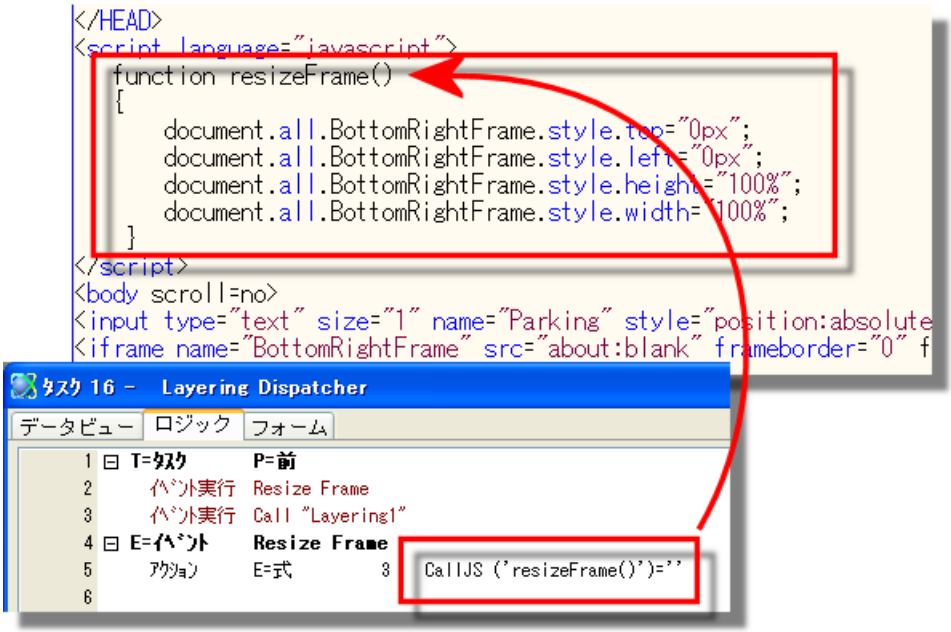
HTML エディタを設定する



1. オプション→設定→動作環境を選択します。
2. アプリケーションサーバタブをクリックします。
3. Web オーサリングツールの行に移動します。
4. 利用したいオーサリングツールの実行モジュールのファイル名前を入力します。ここからズームしてファイルを選択することもできます。

これで、フォームの編集のためにズームを行うと、指定された HTML エディタが起動されます。

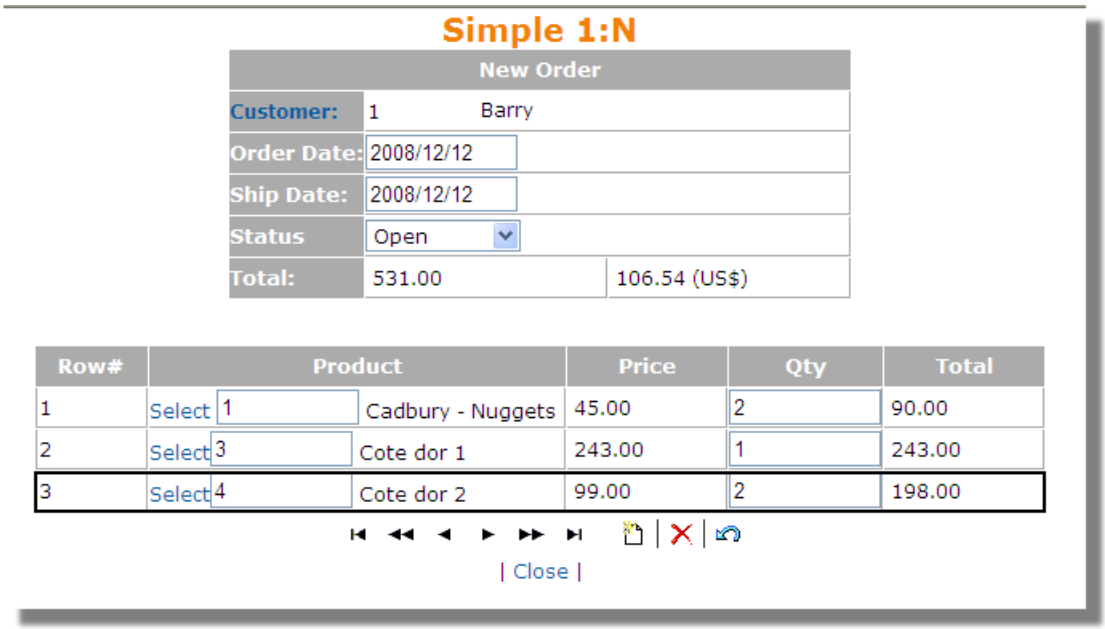
アプリケーションに JavaScript 関数を実装するには



ブラウザクライアント用のプログラムを作成する場合、HTML ページに埋め込まれた JavaScript を呼び出す必要が発生することがあります。このような場合、**CallJS()** 関数を使用することで簡単に実現することができます。プログラム内の任意の場所にアクション処理コマンドを定義することで、この関数を実行させることができます。これにより、JavaScript が実行されます。

実行される JavaScript は、使用する HTML ページ上に直接記述するか、または JavaScript ファイルをリンクするようにして定義する必要があります。

1 対多の関係を構築するには



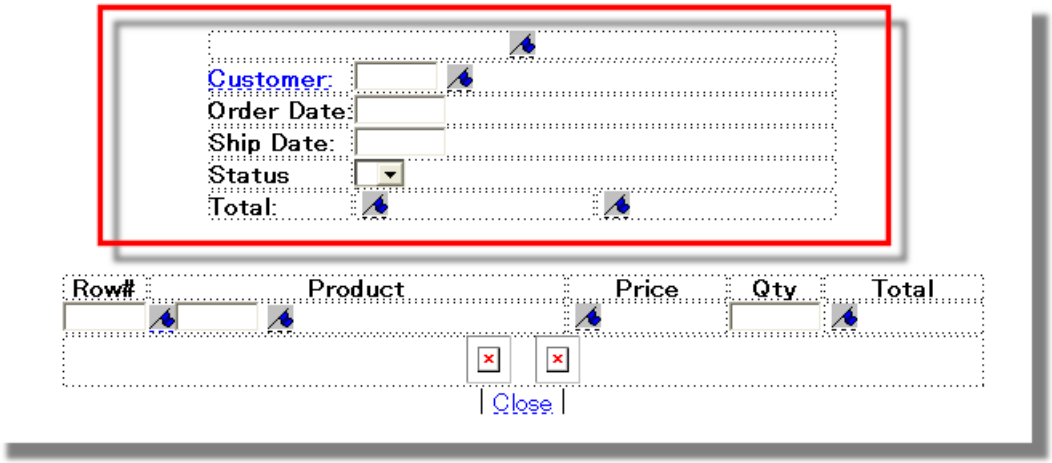
手動で開発する場合、1 つの HTML ページ内に複数のテーブルを設定することは面倒な作業になります。しかし、ブラウザクライアントを使用することで簡単に実現することができます。1 つ以上のサブフォームを設定し、独立した各サブフォームの処理を実行させるために個別のタスクを使用することができます。この例のように、データが接続された場合、1 つのフォーム上のデータが変更されるとサブフォームのデータも自動的に変更されます。

1 対多のタスクを構築するには以下のような 3 つの手順を実行します。

1. ヘッダタスクを作成します。
2. サブタスクを作成します。
3. サブフォームコントロールを使用してヘッダとサブタスクを結合します。

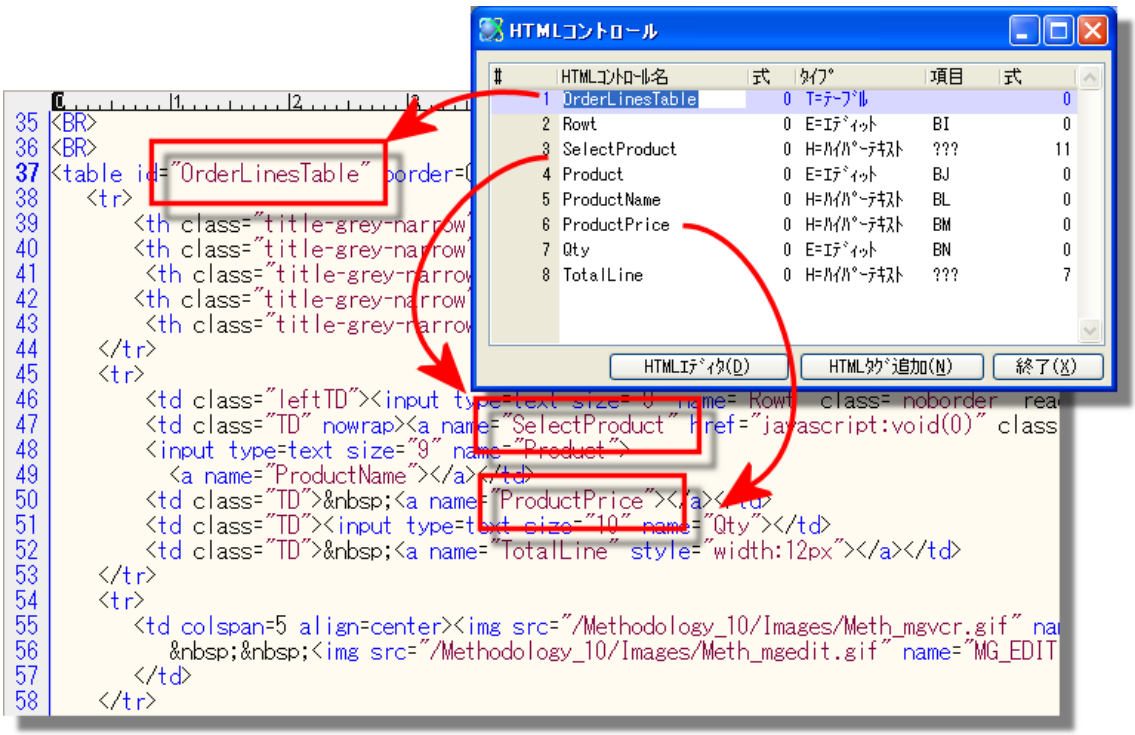
これらの手順の詳細を順番に説明します。

1. ヘッダタスクを作成する

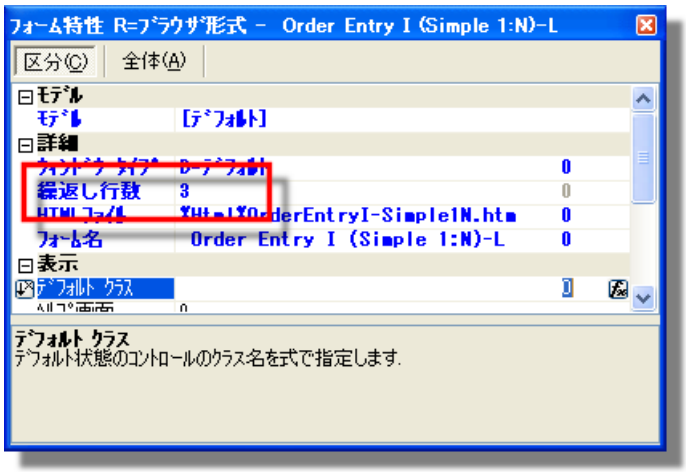


1. ヘッダ用のタスクをブラウザタスクとして作成します。この例では、受注ヘッダを表示するタスクを作成しています。
2. 受注ヘッダフォーム上にて、明細データを表示したい場所にテーブルコントロールを配置します。ここには、次の手順で使用するために ID を設定しておきます。

2. サブタスクを作成する



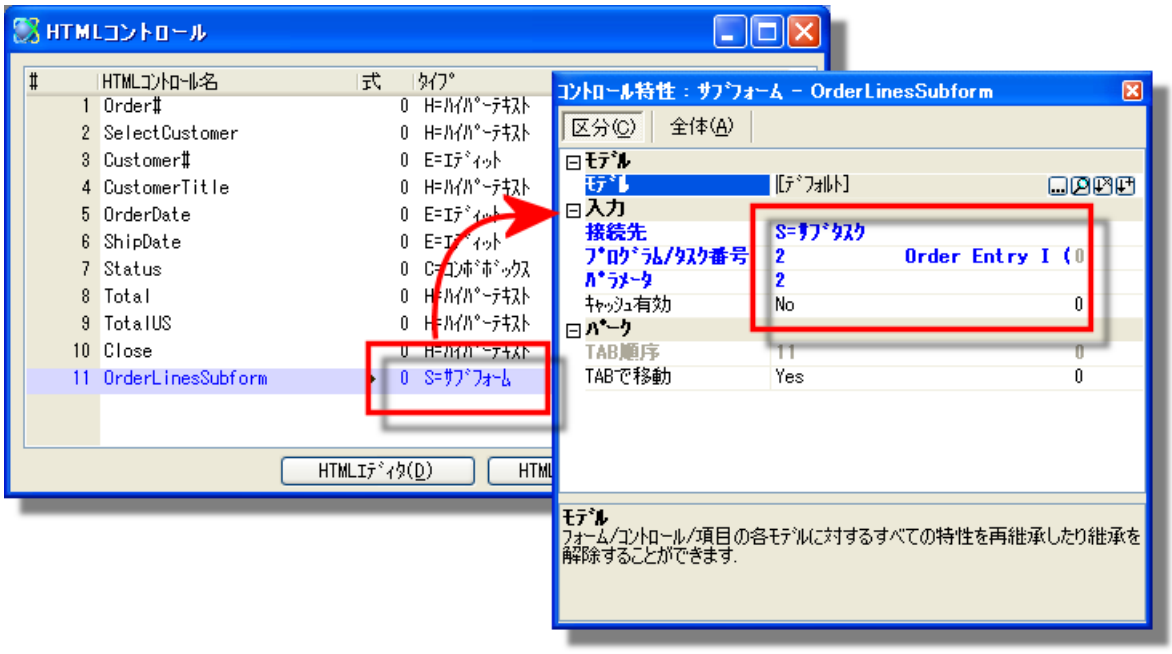
1. サブタスクをブラウザタスクとして作成します。フォームに表示させたい項目を定義します。
2. 親タスクで使用したものと同じ HTML ページを指定します。
3. **タイプ**カラムが***テーブル***に設定された **HTML コントロール**を定義します。このコントロールは、HTML ページ内に id= で指定された名前と同じ名前で **HTML コントロール**テーブルに定義します。
4. テーブルの**コントロール特性**で、**明細行**特性に繰り返し表示させたい行番号を設定します。この例では、HTML テーブルの最初の行はヘッダ行のため、２番目の行の内容が繰り返されます。



5.フォーム特性の繰返し行数特性に、明細行を何回繰り返して表示させるかを指定します。

6. テーブルに表示させたいデータ項目に対してコントロールを配置します。
7. タスクの範囲と位置付を制御するために、パラメータを設定します。この例では、開始モードと注文番号を渡します。これにより受注行には現在の行のみ表示され、親タスクに合わせたタスクモードで実行されます。

3. サブフォームコントロールを使用してヘッダとサブタスクを結合する

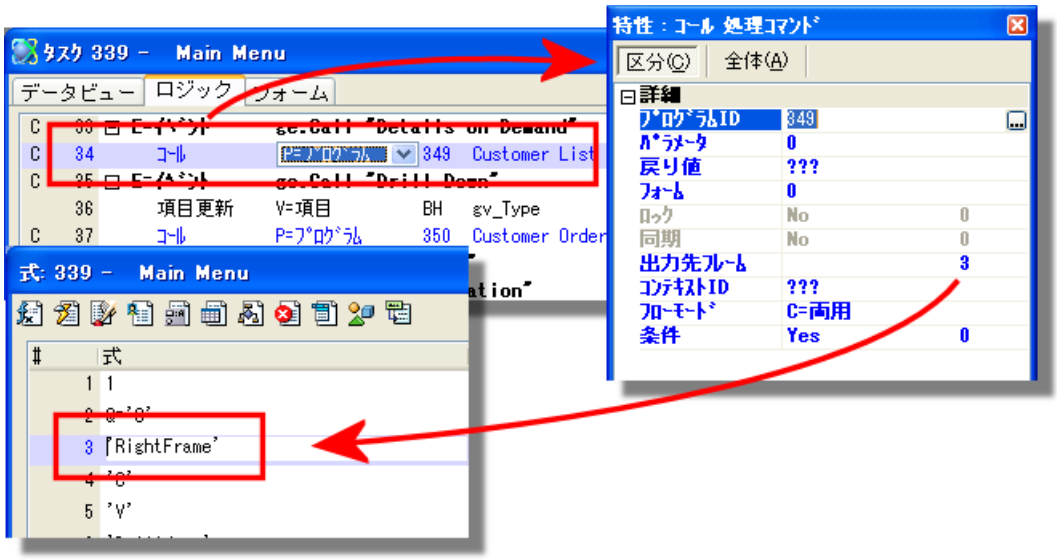


1. 最後に、サブフォームコントロールを親タスクに追加します。*F4* を押下して HTML コントロールを追加し、タイプカラムで***サブフォーム***を選択します。
2. HTML コントロール特性の接続先特性に（サブタスクを呼び出す場合）***S= サブタスク***か、（プログラムを呼び出す場合）***P= プログラム***を指定します。
3. プログラム / タスク番号特性でズームし、呼び出したいサブタスク（またはプログラム）を選択します。
4. パラメータ特性でズームして、呼び出すタスク（またはプログラム）に渡すパラメータを設定します。この例では、サブフォームが親タスクと同じタスクモードで受注明細データを表示するように、タスクモードと受注番号を渡しています。この方法は、第 8 章 :「サブフォーム」（175 ページ）で説明しているようにオンラインタスクでサブフォームを使用した場合と同じです。

これで、メインのタスクが起動されると、自動的にサブタスクも表示されます。

タスク / プログラムが呼び出された時に、新しいウィンドウが開かないようにするには

ブラウザクライアントで新しいタスクを呼び出した場合、デフォルトでは新しいブラウザウィンドウが開きます。しかし、既存のウィンドウの表示を上書きして、前のタスクと同じ IFrame（インラインフレーム）内に新しいタスクの内容を表示させることもできます。

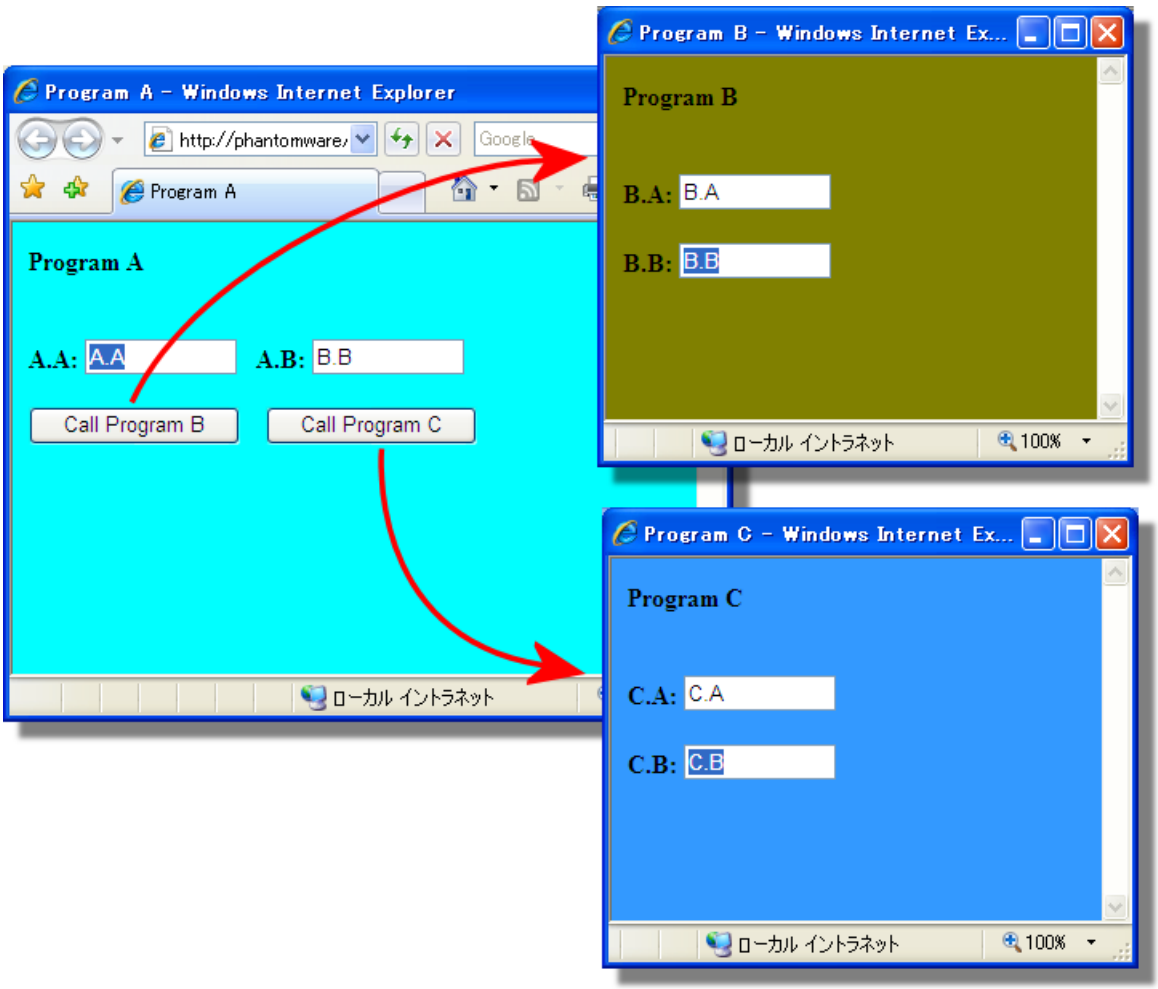


指定したフレームに表示するプログラムを呼び出す

1. コール処理コマンドを作成します。
2. コール処理コマンド特性を開きます。
3. 出力先フレーム特性で、HTML IFrame のタグ名を入力するか、実行時に IFrame タグ名として評価される式を定義します。

これで、実行すると呼び出されたプログラムの内容が指定された IFrame に表示されます。

同じウィンドウ内に異なるプログラムのインタフェースを表示するには

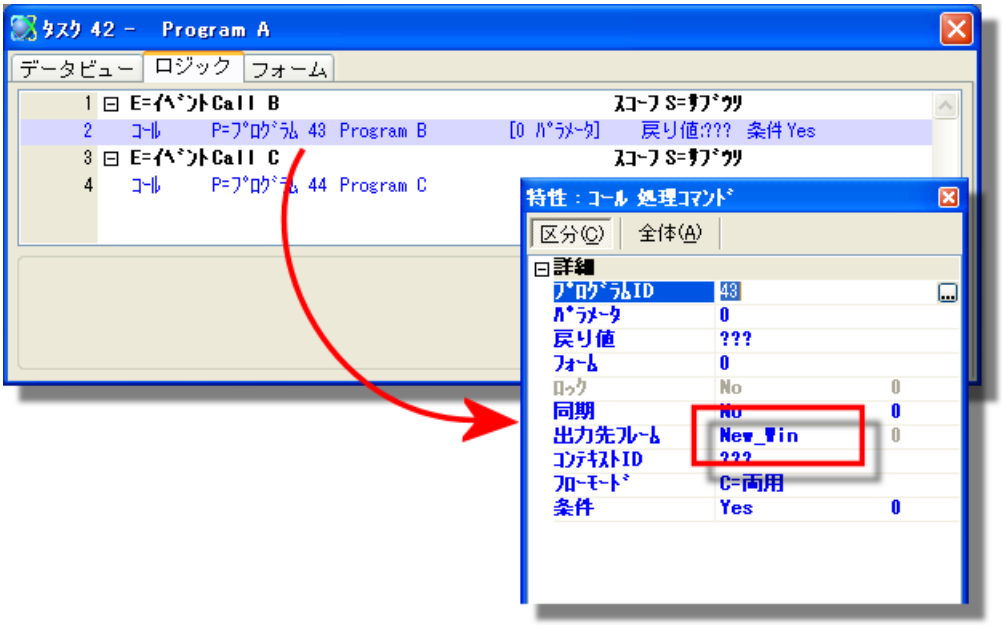


複数の異なるプログラムを表示するために、たくさんのウィンドウを開くことなく 1 つのウィンドウを再利用することができます。IFrames が 1 つのウィンドウを区切って表示させる方法に対し、これは少し異なるやり方になります。IFrame の使い方については、「タスク / プログラムが呼び出された時に、新しいウィンドウが開かないようにするには」（771 ページ）を参照してください。

この例では、ユーザがプログラム B ボタンを**クリック**すると、プログラム B のウィンドウが開きます。そして、ユーザがプログラム C ボタンを**クリック**すると、プログラム C が同じウィンドウ（図では、個別のウィンドウに表示されているように見えますが、実際は、表示が上書きされます）に表示されます。

このようなタイプの動作を簡単に実現させることができます。

ブラウザウィンドウを再利用する



同じウィンドウを使用して異なるプログラムを呼び出す場合、同じ**出力先フレーム**特性を使用します。この例では、２つのイベントが定義されており、それぞれ異なるプッシュボタンに割り当てられています。各イベントは異なるプログラムを呼び出し、各プログラムは表示のための個別の HTML ページが定義されています。しかし、**コール処理コマンド**特性の**出力先フレーム**特性には、どちらも ***New_Win*** という名前のフレームが定義されています。

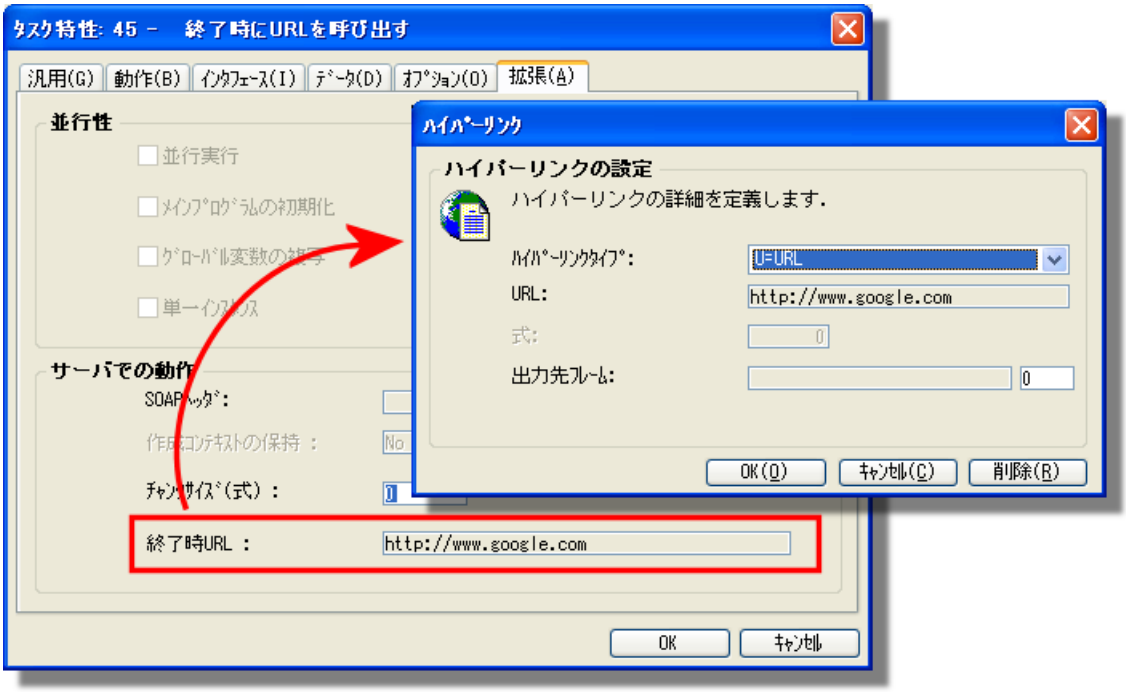
New_Win は IFrame ではなく、使用する HTML ファイル内にはどこにも定義されていません。任意の名前が指定できます。２つのプログラムの呼び出し処理で同じ名前を指定することにより、Magic に対し同じウィンドウを使用するように指示しています。**出力先フレーム**特性に別の名前を指定した場合、別の新しいウィンドウが開きます。指定された名前が今まで開いていたフレーム名と同じ場合、同じウィンドウを再利用して表示されます。

最上位のプログラムが終了する際にオープンする URL を指定するには

Magic のブラウザクライアント機能を使用している場合、ほとんどのブラウザウィンドウは対話型で、ウィンドウはユーザからの操作を待ちます。しかし、ユーザが処理を終わらせすことで通常の HTML 表示に切り替わり、ブラウザクライアントとしてのタスクループを行わせたくない場合があります。または、異なる Magic アプリケーションを起動させたり、ユーザが処理を終了させる場合にプログラムを起動させるような場合も考えられます。このような場合、タスク特性の**終了時 URL** 特性を使用することで実現できます。

注： **終了時 URL** 特性は、ブラウザウィンドウの  をクリックしてウィンドウを閉じた場合には実行されません。このような終了方法は、開発者が制御できないものです。このようなことを防止するために閉じるボタンを表示させず、Magic の終了イベントを使用して終了させるようにできます。

終了時に URL を表示する

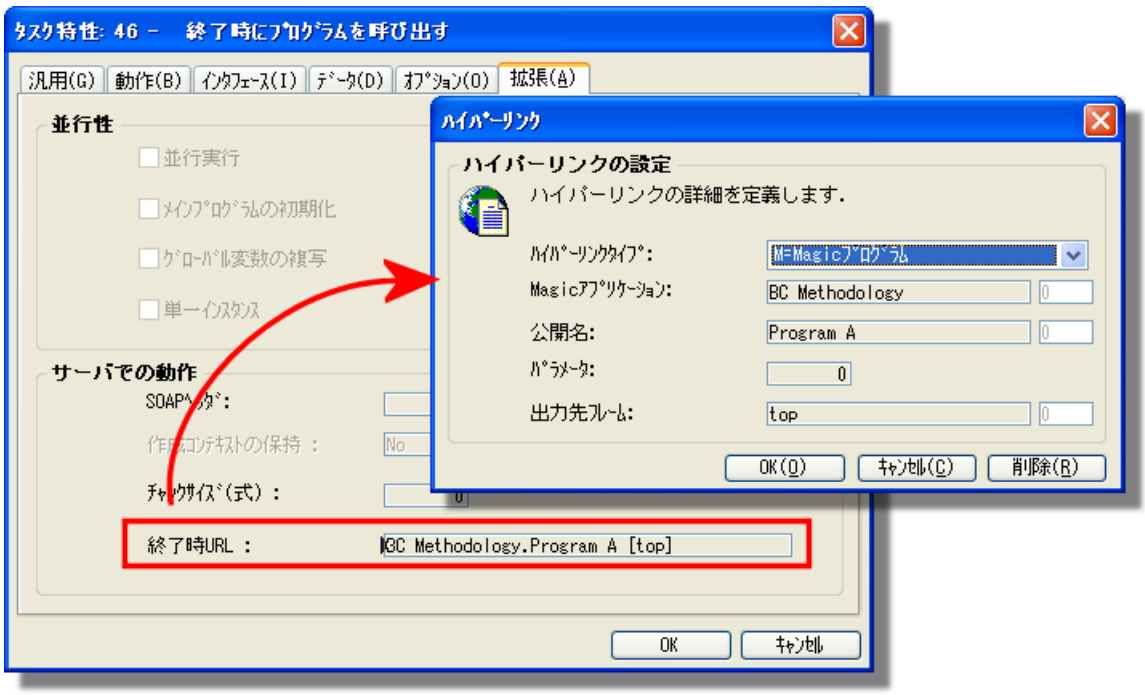


プログラムの終了時に URL を呼び出すには、以下の手順を行います。

1. **終了時 URL** 特性（タスク特性→拡張タブ）でズームします。ハイパーリンクダイアログが表示されます。
2. **ハイパーリンクタイプ**で ***U=URL*** を選択します。
3. 呼び出したい URL を入力します。
4. 呼び出す URL を別のウィンドウまたはフレームで出現させたい場合、**出力先フレーム**特性にフレーム名を入力します。指定されない場合、現在のウィンドウまたはフレームに表示されます。

これで、プログラムが終了する際に、指定された URL に制御が移ります。

終了時に Magic プログラムに移る



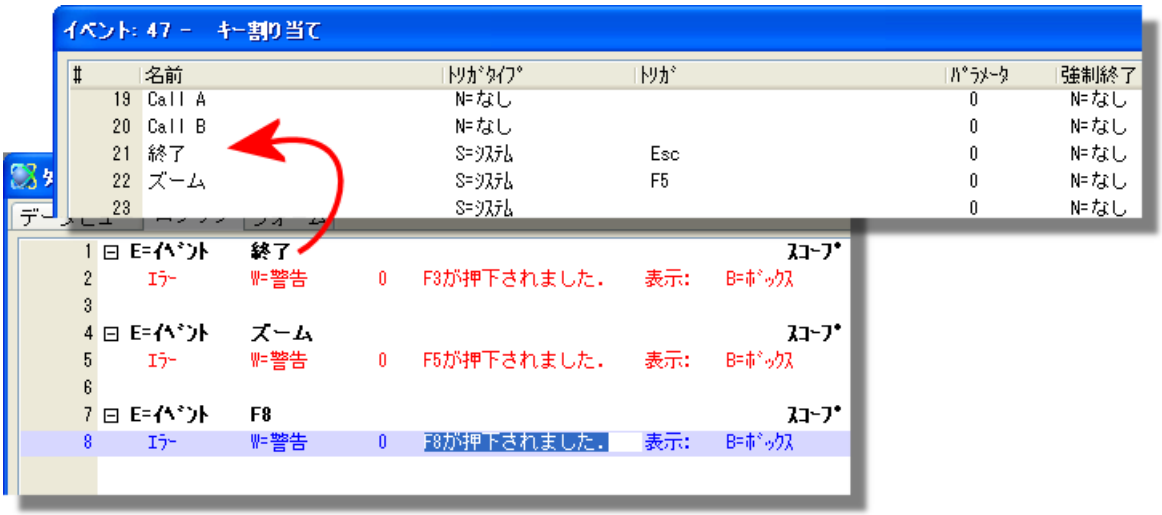
プログラムの終了時に別の Magic プログラムを呼び出すには、、以下の手順を行います。

1. **終了時 URL** 特性（**タスク特性→拡張タブ**）で**ズーム**します。**ハイパーリンク**ダイアログが表示されます。
2. **ハイパーリンクタイプ**で ***M=Magic プログラム***を選択します。
3. Magic のアプリケーション名を入力します。呼び出したいプログラムは、現在のアプリケーションでも異なるアプリケーションでもどちらでも指定できます。**Magic アプリケーション**から**ズーム**してプロジェクトファイルかキャビネットファイルを選択して指定することもできます。
4. **公開名**に呼び出したいプログラムの公開名を指定します。同じプロジェクト内であればここから**ズーム**して公開名を選択することができます。
5. 呼び出すプログラムを別のウィンドウまたはフレームで出現させたい場合、**出力先フレーム**特性にフレーム名を入力します。指定されない場合、現在のウィンドウまたはフレームに表示されます。

これで、プログラムが終了する際に、指定された Magic プログラムに制御が移ります。

Magic の内部イベントにキーボードを割り当てるには

Web ブラウザで実行している場合は、通常の Magic のデフォルトでのキー操作は使用できません。例えば、ほとんどの Web ブラウザでは、**F5** は**ズーム**ではなく**再表示**に割り当てられています。



しかし、イベントを使用することで、任意の動作をキーボード操作に割り当てることができます。ちょうど、オンラインのプログラムを実行するときのように、これらのイベントを設定することができます。

ロジックユニット内でキー操作を使用する

以下のようにすることで、キー操作を**イベント**ロジックユニットに定義することができます。

1. *Ctrl+H* を押下して**ロジックユニット**のヘッダ行を作成します。
2. *E* を入力して**イベント**を選択します。**イベント**ダイアログが表示されます。
3. **イベントタイプ**では**システム**を選択します。*Tab* 移動すると、**キー定義**ダイアログが表示されます。
4. このイベントに割り当てたい組み合わせキーを押下します。

これで、ユーザが定義された組み合わせキーを押下すると、**イベント**ロジックユニットが実行されます。

ユーザイベントでキー操作を使用する

ユーザイベントを発行させるためにキー操作を使用することもできます。特定のユーザイベントが複数の方法で発行される場合、この方法は便利かもしれません。例えば、特定のプログラムをプッシュボタンや標準的なファンクションキー、他のプログラムからの呼び出しによって起動させたい場合が考えられます。

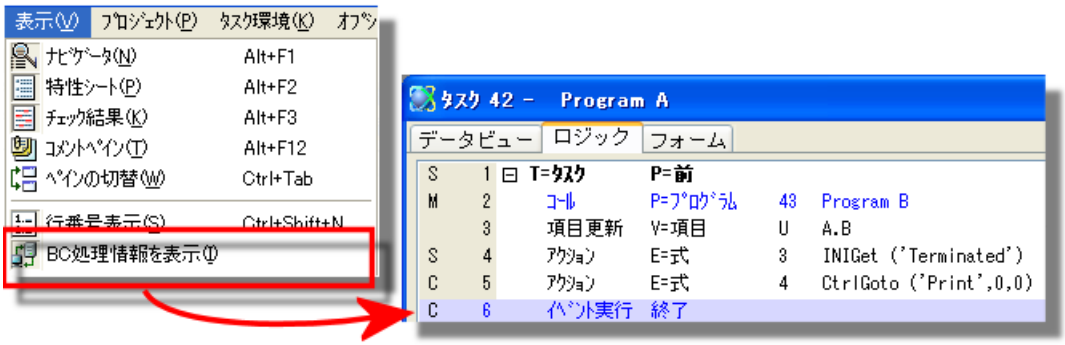
1. *Ctrl+U* を押下して**ユーザイベント**テーブルを開きます。
2. *F4* を押下して、１行追加します。
3. ユーザイベント名を入力します。次のカラムに *Tab* 移動します。
4. **トリガータイプ**カラムで *S= システム*を選択し、*Tab* 移動します。
5. **トリガー**欄から**ズーム**して**キー定義**ダイアログを開きます。
6. このイベントに割り当てたい組み合わせキーを押下します。

これで、ユーザが定義された組み合わせキーを押下すると、ユーザイベントが発行されます。

サーバ側とクライアント側の処理コマンドと関数を区別するには

ブラウザクライアントのプログラムを作成する場合、処理コマンドの中には Magic のアプレットを使用しているクライアント上で実行されるものや、サーバ側で実行されるものがあります。各処理コマンドがどちらで実行されるかを知ることは、アプリケーションのパフォーマンスを向上させる上で参考になります。

『リファレンスヘルプ』の、「サーバ側で実行される関数」と「クライアント側で実行される関数」の各トピックを参照することで、どの関数がどちらで実行されるかを確認することができます。また、**BC 処理情報を表示**オプションを使用することで、各処理コマンドがどちらで実行されるかを確認することもできます。

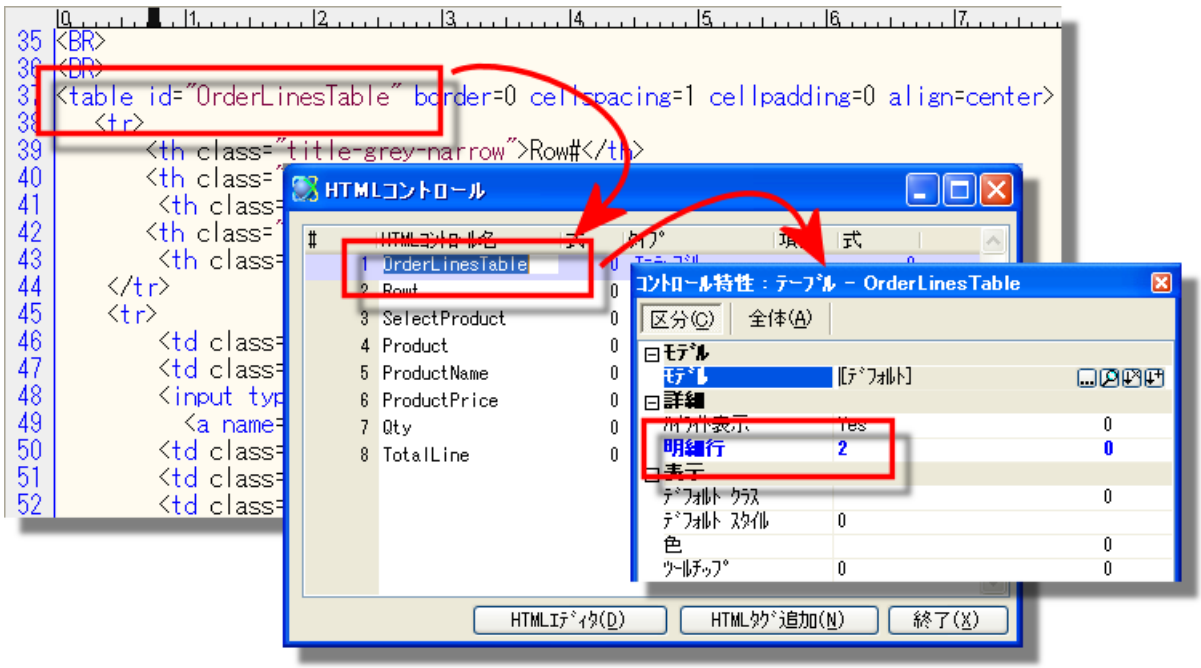


BC 処理情報を表示（**表示**メニュー）が設定されていない場合は、まず設定する必要があります。

設定が有効になっている場合、**ロジックエディタ**内の行番号カラムに S、M、または C が表示されます。

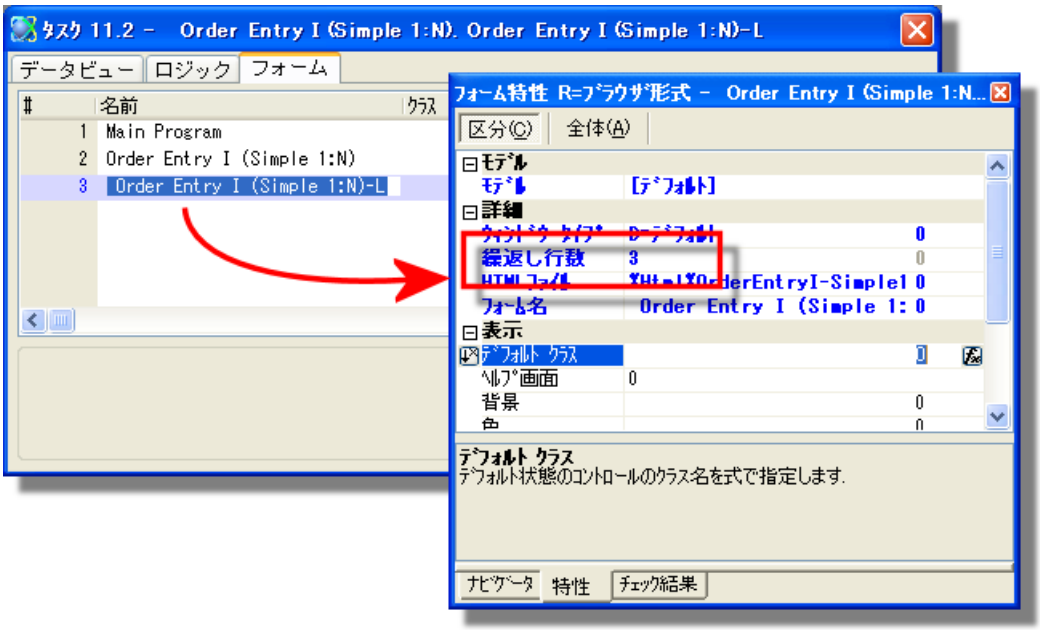
- S：処理コマンドがサーバ上で実行されます。
- C：処理コマンドがクライアント上で実行されます。
- M：混合モードで処理されます。
- 空白：処理コマンドがクライアントまたはサーバのどちらかで実行されます。

テーブルの繰り返し行を指定するには



ブラウザクライアントのプログラムに定義されるテーブルは HTML テーブルです。HTML テーブルは、id= の HTML 属性によって Magic 側の **HTML コントロール**テーブルで参照することができます。

テーブルコントロールの**コントロール特性**内の**明細行**特性は、どの行を繰り返すかを指定します。この例では、テーブル内に <tr> タグで定義された２つの行があります。最初の行は、テーブルヘッダになります。２番目の行に繰り返し行が含まれています。従って、**明細行**は ***2***に設定します。



次に、テーブルを含むフォーム内で、**繰返し行数**特性に明細行を繰り返す数を設定します。この例では、行は３回繰り返されます。

Row#	Product	Price	Qty	Total
1	Select 1 Cadbury - Nuggets	45.00	2	90.00
2	Select 3 Cote dor 1	243.00	1	243.00
3	Select 4 Cote dor 2	99.00	2	198.00

ここに実行結果が表示されています。２番目の HTML 行が３回繰り返されています。

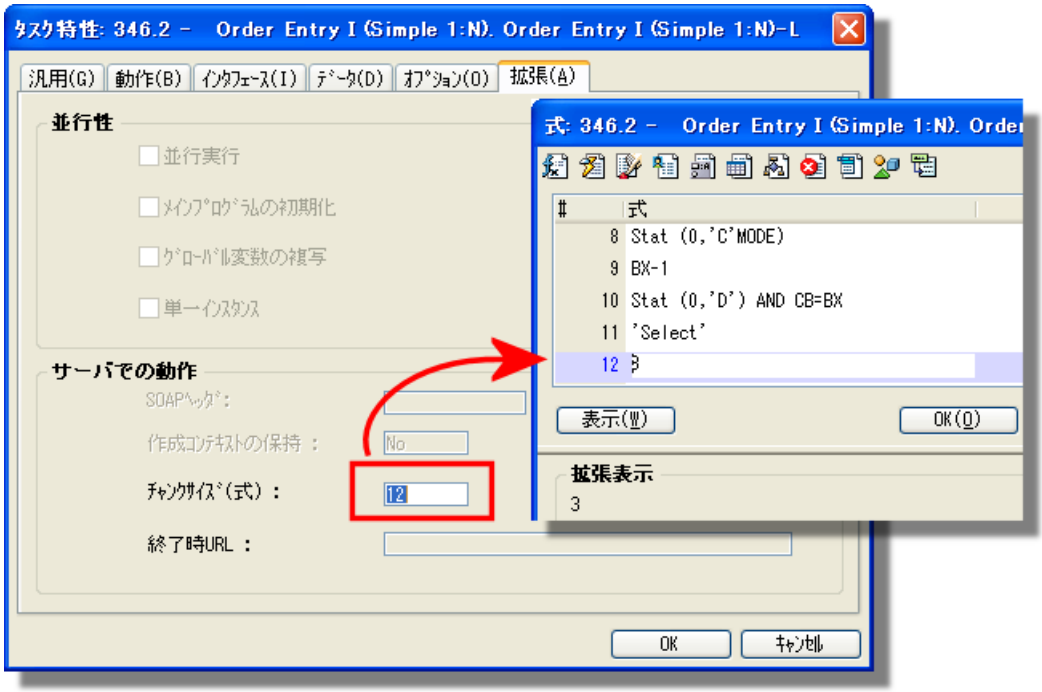
ブラウザに渡されるレコードセット内のレコード数を設定するには

ブラウザクライアントの応答の良さの理由として、現在表示されているレコードだけでなく、データの「チャンク（大きな塊）」がクライアントに送られることが挙げられます。これは、ユーザがテーブルを下にスクロールする場合に、即時に対応できることを意味しています。

比較的に小さなテーブルの場合は、すべてのレコードがすぐに送られためあまり問題にはなりません。レコードは暗号化され、圧縮され、迅速に送られます。しかし、テーブルが非常に大きい場合、すべてのレコードを送るには時間がかかることがあります。クライアント側のキャッシュに保持されるレコードの「チャンク」の理想的なサイズは、各レコードのサイズとユーザのスクロール量など含めたいくつかの要因に依存します。

チャンクサイズが設定されていない場合、メインソースが定義されたタスクに対して 30 がデフォルト値になります。メインソースが定義されていないタスクの場合は、1 がデフォルト値です。

チャンクサイズを設定する



1. タスク特性の高度な設定タブを開きます。
2. チャンクサイズ（式）特性でズームして、式エディタを開きます。
3. *F4* を押下して、1 行追加します。各チャンク毎に取得したいレコード数を指定します。
4. OK をクリックして式エディタを閉じます。式番号が表示されます。
5. OK をクリックしてタスク特性を閉じます。

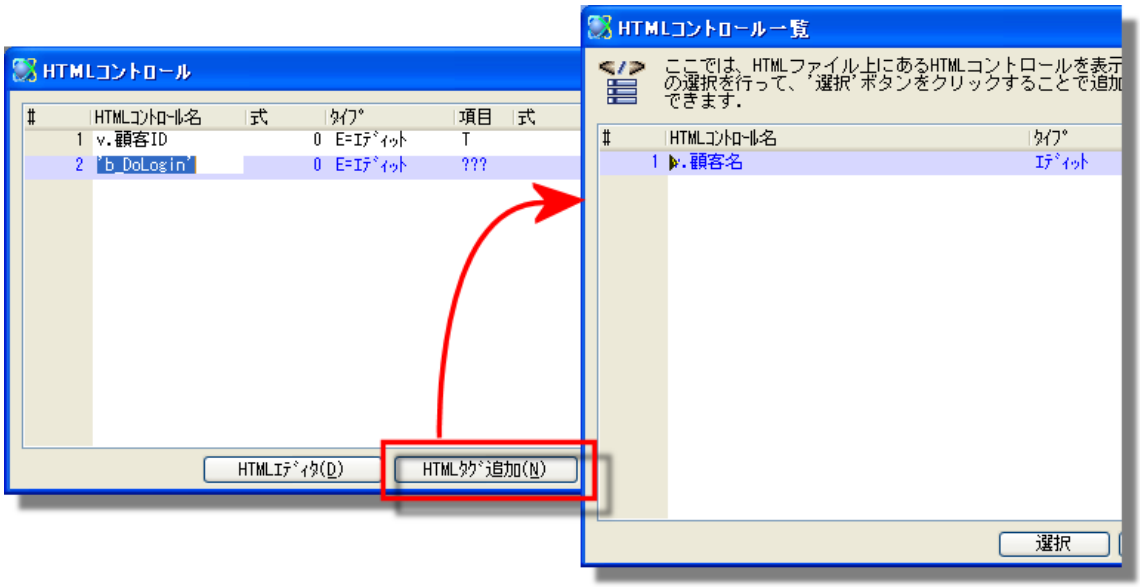
コントロールを既存のフォームに追加するには

ブラウザクライアントのプログラムを作成後に、コントロールを HTML ページに追加する必要があるかもしれません。このような場合、新しいコントロールが認識できるように Magic プログラムを修正する必要があります。

新しいコントロールを追加する

```
1  <HTML>
2  <HEAD>
3  </HEAD>
4  <BODY>
5  <FORM NAME ="Customer Login">
6  <b>顧客のログイン</b><br>
7  <br><b>v.顧客ID: </b><input type=text size="10" name="v.顧客ID"> <b>
8  <br><b>v.顧客名: </b><input type=text size="10" name="v.顧客名"> <b>
9  <br><br>
10  <input type='button' name='b_DoLogin' value="ログイン">
11 </FORM>
```

1. 新しいコントロールを HTML に追加します。name= 属性が、Magic のオブジェクトにリンクされるために設定されています。この例では、name は ***v.顧客名*** になっています。
2. この項目にリンクされる変数または実項目を追加します。



3. Magic プログラムの**フォームエディタ**に移動します。
4. **フォーム名**から**ズーム**し、既存の **HTML コントロール**を開きます。
5. **HTML コントロール**のリストで追加したい行にカーソルを移動します。そして、HTML タグ追加ボタンを**クリック**します。**HTML コントロール一覧**には、**HTML コントロール**に定義されていない全てのタグが表示されます。追加したいタグにカーソルを置き選択ボタンを**クリック**します。選択されたコントロールが **HTML コントロール**に追加されます。必要であれば、***Ctrl+ クリック***で複数のコントロールを選択することもできます。
6. **項目**カラムから**ズーム**してこのコントロールに割り当てる項目を設定します。項目の代わりに**式**カラムから**ズーム**することで式を指定することもできます。

これで、新しいコントロールがブラウザタスクの一部として動作します。

スタイルとクラスを実装するには

HTML 内に、スタイルとクラスを実装するにはいくつかの方法があります。例えば、以下のようにすることができます。

- 色とフォント属性を各フィールドタグに追加する。
- 複数のタグに適用されるスタイルにアクセスするクラスを使用する。
- 複数の HTML ファイルにスタイルを適用するためのスタイルシートを使用する。
- フォントと色をインタラクティブに変更するために、スクリプトを使用する。

Magic を使用している場合、これらの HTML オプションが存在しており、さらに Magic でコード化された HTML 機能にアクセスすることができます。また、HTML コードとは独立して Magic 内でスタイル用に使用することもできます。

どのオプションを使用するかは、どの程度の柔軟性が必要であるか、またはシステムの一般的なデザインに依存します。一般的な原則として、システムは可能な限り保守しやすいように要求されるため、フォーマットが高いレベルで維持できるように、Magic のモデルや HTML のスタイルシートを使用することが良い方法になります。

しかし、フィールドによっては特定のスタイルを使用してフォーマット化できない場合もあります。このような場合は、フィールドレベルでフォーマットすることになります。

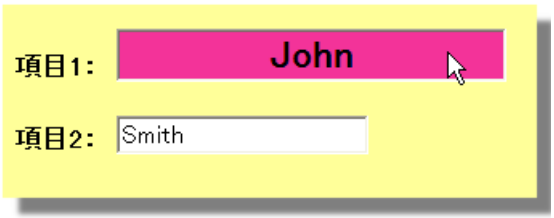
ここでは、Magic でスタイルとクラスの実装方法をいくつか説明します。

スタイルシート

```
.formreg {border:thin dashed silver;font-family:"ＭＳ Ｐゴシック";background-color:#FFFF99}
.bighdr{text-align:center; color:back; background-color:red font-family:'Century Schoolbook',serif; font-size:18pt;}
.regtext{text-align:left; font-size:12pt;background-color:#FFFFFF;}
.overtext{text-align:center; font-size:18pt;font-weight:bold;background-color:#F33399;}
.outtext{text-align:right; font-size:12pt;background-color:#FFFF99;}
```

通常、スタイルシートは個別の CSS ファイル内に定義されます。CSS ファイルは複数の Web ページで使用することができます。この例では、すべてのプログラムが上記の簡単なスタイルシートを使用しています。

ここで定義されているスタイルは、カーソルが入力フィールド上にある場合に、フィールドが大きくなったり、ピンク色に変わったりするような表示効果を定義しています。この例では、カーソルが 項目 1 の上にある場合の実行結果を示しています。



次に、この効果をコード化する３つの異なる方法について説明します。

HTML クラスを使用する

表示効果をコード化するためにスタイルを使用する最初の方法は、HTML 内にクラスを定義することです。ここでは、以下の３つのクラスが定義されています。

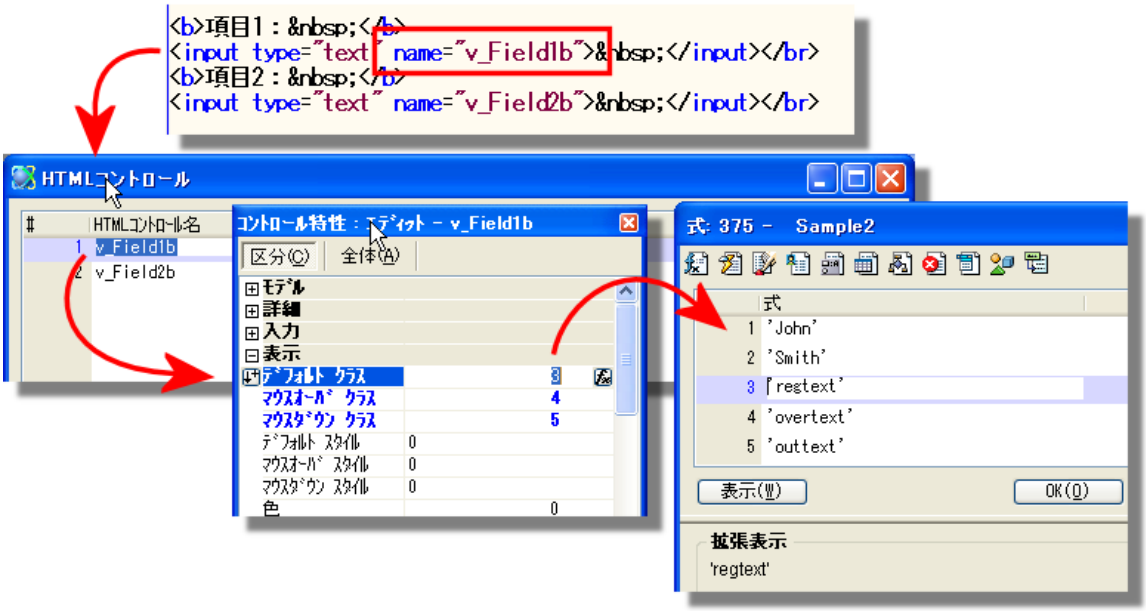
- regtext
- overtext
- outtext

タグ属性として追加されます。

これで正しく動作しますが、必要な場所にすべてのタグの設定を繰り返し行う必要があります。また実行時に Magic プログラム内で値を変更することができません。

```
<b>項目1： </b>
<input type="text" name="v_Field1b"
    size="20"
    value="John"
   class="regtext"
   onmouseover="this.className='overtext'"
   onmpuseout="this.className='outtext'"> 
</input></br>
<b>項目2： </b>
<input type="text" name="v_Field2b"
    size="20"
    value="Smith"
   class="regtext"
   onmouseover="this.className='overtext'"
   onmpuseout="this.className='outtext'"> 
</input></br>
```

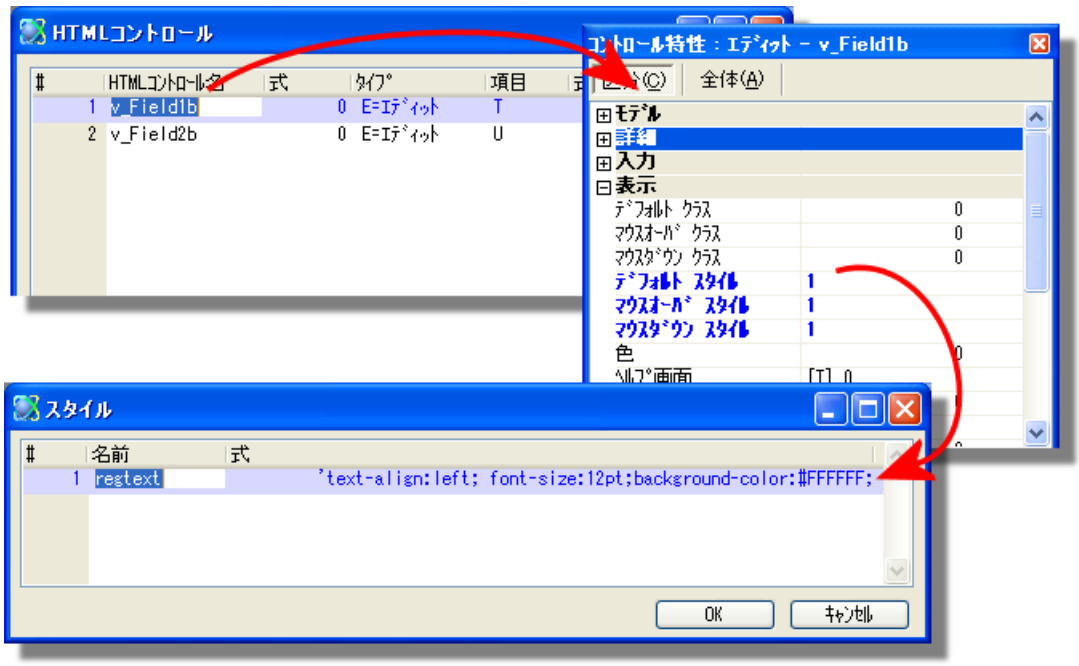
Magic でクラスを使用する



この例では、HTML 内にクラス名を定義する代わりに　入力項目のコントロール特性にクラスを定義しています。各特性は、式でクラス名を指定しています。

式は、スタイル名として評価されるグローバル変数で指定することもできます。これにより正確なスタイル名をプログラム毎に指定する必要がありません。

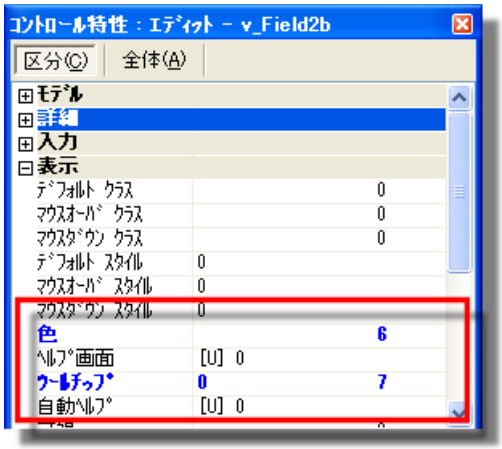
Magic 内でスタイルをコード化する



ここには、ちょうど前の２つの方法のように動作するプログラムがあります。しかし、ここではスタイルシートを使用していません。その代わりに、Magic プログラム内でスタイルがコード化されています。

デフォルトスタイル特性や、**マウスオーバスタイル**特性、**マウスダウンスタイル**特性で**ズーム**するすることで、実際のスタイルを式として定義するためのダイアログが表示されます。ここには、前の説明で示された CSS スタイルシート内で定義されたものと同じ文字列をスタイルとして定義します。スタイル定義は、グローバル変数内に格納することもできます。左のスタイル名はテキストでのみ指定します。

他のコントロール特性を使用する



HTML スタイルではなく他の Magic 特性を使用することもできます。これらの特性によって、HTML 内には特別なコードを入力することなく様々な実行制御を行うことができます。

この特性の設定処理は、同じ特性が Magic のクライアント／サーバ環境で実行された場合と同じように、Web ブラウザ上で実行されます。

例えば、**色**特性を式で指定することで、他のフィールドの値にもとづいてフィールドの色を設定することができます。このオプションは、HTML ファイル側の色設定を使用しません。Magic の**基本色**テーブル（**オプション→設定→色**）に定義された色が使用されます。

ヘルプ画面特性には、表示させる URL ヘルプが指定できます。ユーザが *F1* を押下すると、指定された URL の内容が新しいウィンドウで表示されます。**ツールチップ**特性には、表示させたいツールチップや、ツールチップとして表示させたい文字を式で定義することができます。

可視特性を使用して、特定の状況下でフィールドを表示させるかどうかを指定することもできます。

ページ上に ActiveX コントロールを実装するには

HTML ページ上に簡単に ActiveX コントロールを実施することができます。ActiveX コントロールを使用するには以下の３つの手順を実行します。

1. HTML ページに ActiveX コントロールを配置します。
2. このコントロールを使用する VB スクリプトか JavaScript を使用する必要があります。
3. スクリプトを実行させるために Magic プログラム内で **CallJS** 関数を使用し、コントロールを処理します。

各手順の詳細を説明します。

注： ここで使用される ActiveX オブジェクトは、Microsoft のカレンダコントロールです。既にクライアント側の PC にインストールされていることを前提に説明しています。

HTML ページに ActiveX コントロールを配置する

```
<form name = "BC External Event">
<p align="center">
<br><font size="5">日付： <input type="text" size="10" name="Date" style="font-size:14pt"
onclick=document.Calendar1.Day=8></br>
</font>
</p>
<p align="center">
<object classid="clsid:8E27C92B-1264-101C-8A2F-040224009C02" id="Calendar1" width="288" height="192">
  <param name="_Version" value="524288">
  <param name="_ExtentX" value="7620">
  <param name="_ExtentY" value="5080">
  <param name="_StockProps" value="1">
  <param name="BackColor" value="-2147483633">
  <param name="Year" value="2007">
  <param name="Month" value="8">
  <param name="Day" value="21">
  <param name="DayLength" value="1">
  <param name="MonthLength" value="2">
  <param name="DayFontColor" value="0">
  <param name="FirstDay" value="1">
  <param name="GridCellEffect" value="1">
  <param name="GridFontColor" value="10485760">
  <param name="GridLinesColor" value="-2147483632">
  <param name="ShowDateSelectors" value="-1">
  <param name="ShowDays" value="-1">
  <param name="ShowHorizontalGrid" value="-1">
  <param name="ShowTitle" value="-1">
  <param name="ShowVerticalGrid" value="-1">
  <param name="TitleFontColor" value="10485760">
  <param name="ValueIsNull" value="1">
</object>
```

最初に、ActiveX オブジェクトを HTML ページに挿入する必要があります。これは <object> タグを使用することで実現できます。classid 属性はオブジェクトの一般的な ID を指定します。<param> タグはオブジェクトの様々な特性を指定しています。

各オブジェクトは、どのように記述されたかによって異なります。ActiveX オブジェクトの扱い方についての詳細は、オブジェクトに添付されるドキュメントを参照してください。

注： HTML エディタによっては、ウィザード形式でオブジェクトを挿入することで上図のような定義を自動的に行うことができます。

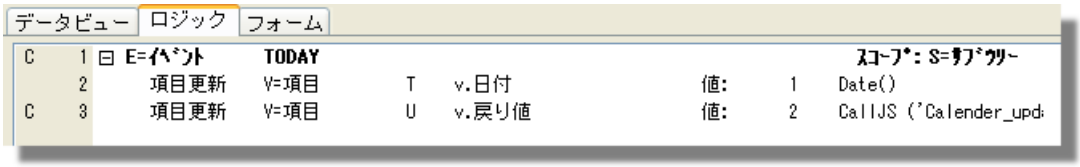
スクリプトを使用して ActiveX コントロールを動作させる

```
<SCRIPT LANGUAGE="JavaScript">
//This JavaScripts segment provides a function that manipulates the embeded Calender object
//This allows the Magic engine to easily interct with exterminal modules on the page.

function Calender_update(MGDay,MGMonth,MGYear)
{
document.Calendar1.Day=MGDay;
document.Calendar1.Month=MGMonth;
document.Calendar1.Year=MGYear;
}
```

ActiveX オブジェクトが定義されたら、スクリプトを使用することでオブジェクトを実行させることができます。スクリプトのほとんどは HTML ページ内でローカルに処理されますが、Magic から呼び出すこともできます。また、オブジェクトで発生したイベントを Magic で処理することもできます。

このスクリプト、**Calendar_Update** は３つのパラメータを受け取り、カレンダーオブジェクトの日付を変更することができます。

Magic からスクリプトを呼び出す



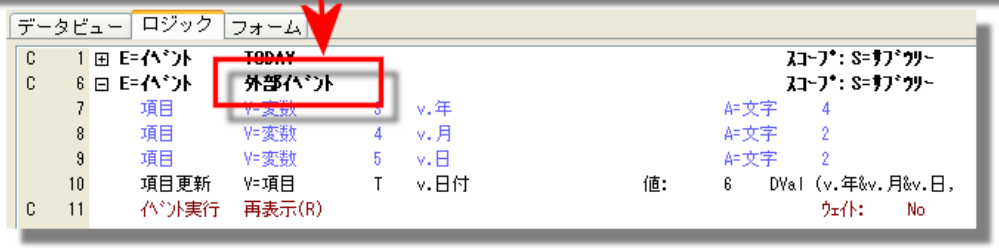
ここでは、**CallJS** 関数を使用して JavaScript 関数を呼び出すプログラム例を示しています。ユーザが今日ボタンをクリックすると **TODAY** イベントが発行されます。このイベントハンドラ内で前述の JavaScript 関数 **Calendar_Update** を起動しています。

Magic 内のスクリプトからイベントを処理する

スクリプトは、Magic 内で処理されるイベントを発行させることもできます。この説明については、「HTML ページ内の外部スクリプトからの Magic のロジックを起動するには」（787 ページ）を参照してください。

HTML ページ内の外部スクリプトからの Magic のロジックを起動するには

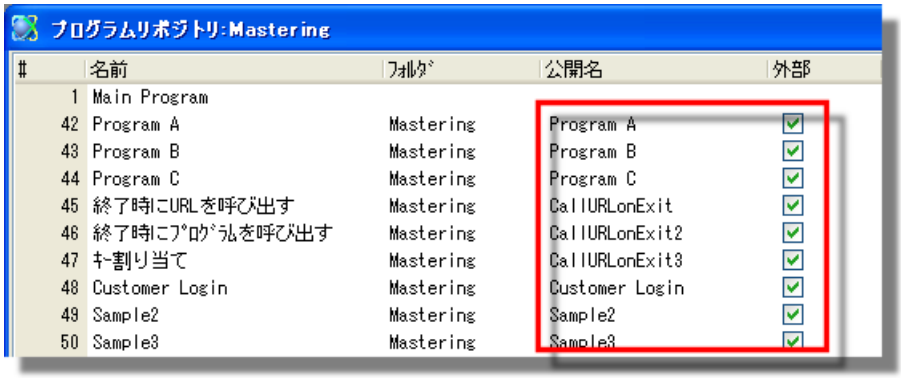
```
<SCRIPT LANGUAGE="VBScript">
//This VBScripts segment handles the click event of the Calender object.
//Upon a click on the object the MGExternalEvent function is called with the expected arguments.
//This allows external modules on the page to interact with Magic engine.

Sub Calendar1_click()
cal  window.MGExternalEvent(document.Calendar1.Year,document.Calendar1.Month,document.Calendar1.Day)
end sub
</SCRIPT>
```



HTML ページ内で MGExternalEvent と呼ばれる特別なスクリプトを定義することができます。このスクリプトを呼び出すことで、外部イベントと呼ばれる Magic の内部イベントを発行させ、指定されたパラメータを渡すことができるようになります。

この例では、VB スクリプトを使用して MGExternalEvent を呼び出しています。その際、３つのパラメータ（年数、月数、および日付）を渡します。イベントハンドラはこれらの３つのパラメータを受け取り、項目を更新し、画面を再表示するという処理を実行します。これによって、ウィンドウ上のカレンダの日付をクリックするとカレンダの上に表示された日付が更新されます。

プログラムを外部から呼び出せるようにするには



Magic プログラムを Web ブラウザなど Magic アプリケーション外から呼び出すことを可能にするために、以下の２つの処理が必要になります。

1. プログラムに**公開名**を設定する。
2. **外部**カラムのチェックボックスをチェックする。

以上です。

プログラムリポジトリを参照するだけで、どのプログラムを呼び出すことができるかを確認することができます。

第 38 章： ブローカ

Magic がリクエストを受信できるように Web サーバ（IIS）を設定するには

MRB（Magic　Request Broker）を使用して動作させる前に、Web サーバを適切に設定する必要があります。設定するには、以下のような作業を行います。

1. Magic をインストールする前に、あらかじめ IIS Web サーバをインストールしておく必要があります。
2. Magic のインターネットリクエスタを一緒にインストールします。
3. Magic のインストールの後にインストール内容を確認し、Web サーバが動作していることを確認します。

これらの手順を順に説明します。

1.Microsoft IIS をインストールする

IIS（Internet Information Services）は、Web サーバとして動作する Windows OS のコンポーネントです。コンピュータの管理ウィンドウ（**スタートメニュー→設定→コントロールパネル→管理ツール**）を参照することで確認することができます。

IIS がインストールされていない場合、Windows コンポーネントウィザード（**スタートメニュー→設定→コントロールパネル→プログラムの追加と削除→ Windows コンポーネントの追加と削除**）を使用してインストールすることができます（その際、OS のインストール CD が必要になります）。

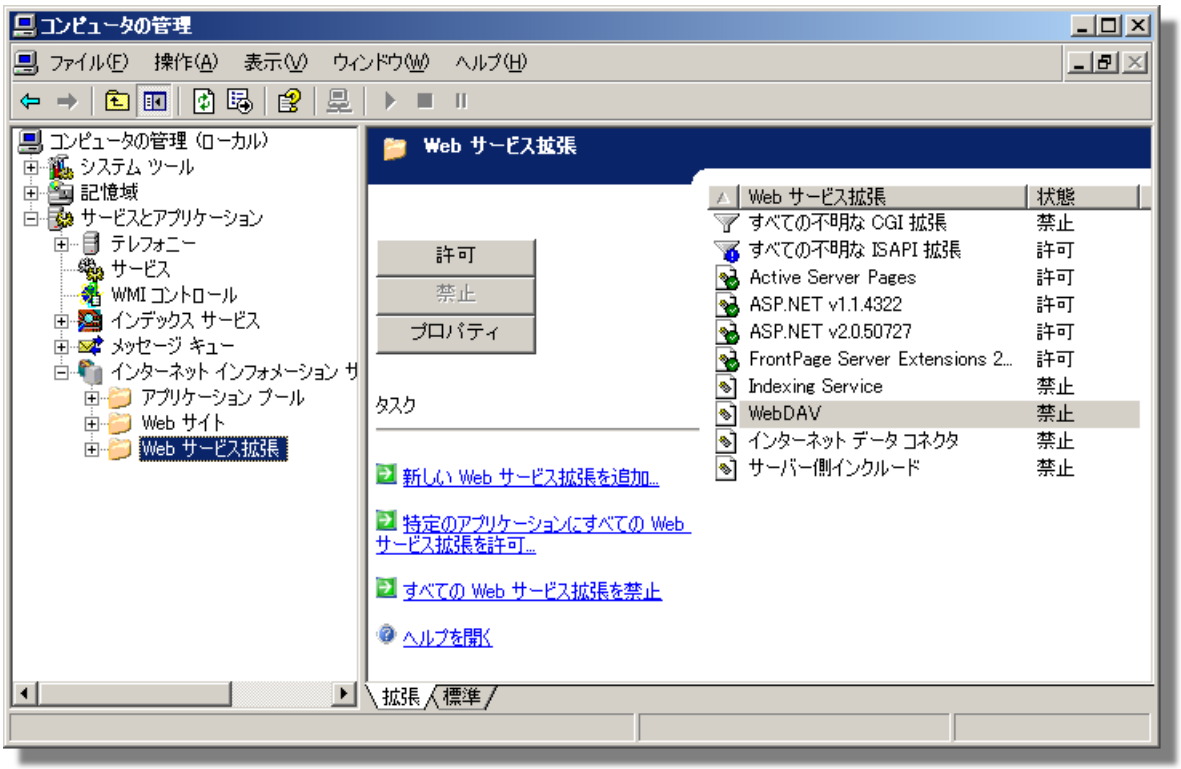
Windows 内に IIS が既にインストールされている場合は、Magic のインストール時に自動的にインターネットリクエスタのセットアップを行うように選択されます。その際、リクエスタのコピー先のエイリアスも自動的に設定されます（カスタムインストールにて変更することもできます）。

注：　IIS のインストール方法などは、使用する Windows のバージョンによって異なる場合があります。

IIS のセキュリティ機能の対応

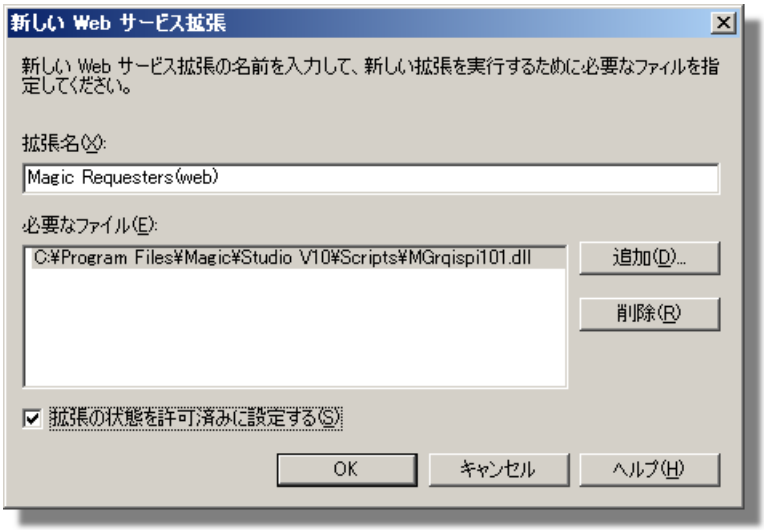
IIS の 6.0 以降におけるセキュリティ機能の拡張によって、DLL ファイルが保護され、これらが明示的に実行を許可された場合でない限り実行されないようになっています。

IIS6.0 の場合



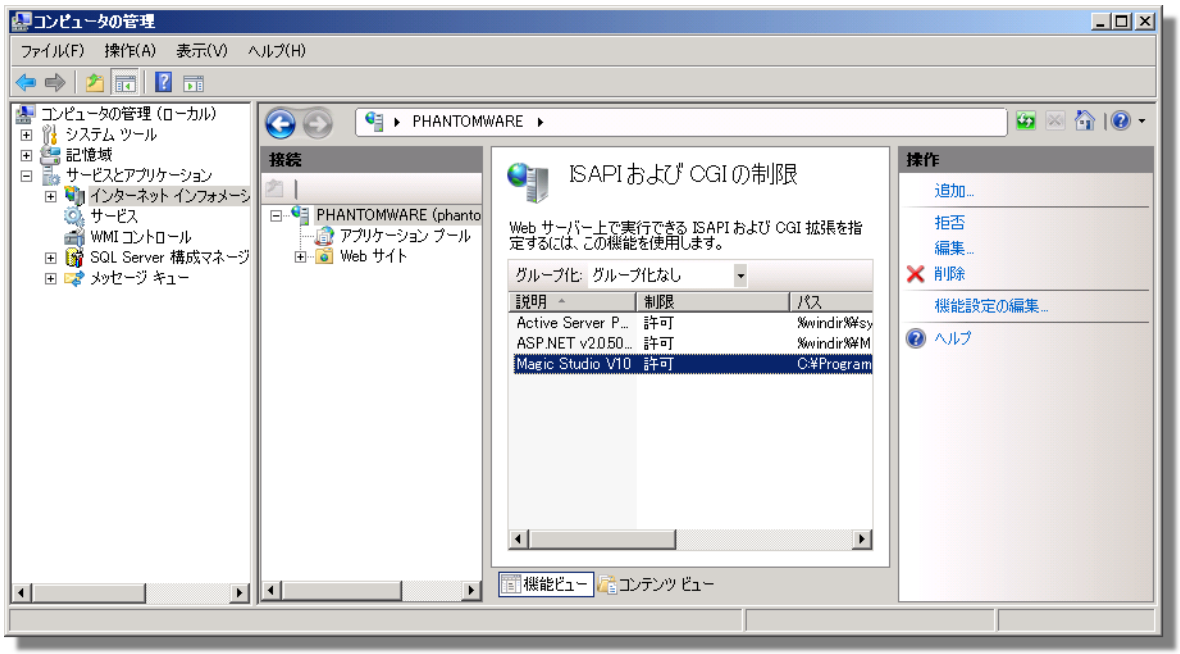
以下の手順は、ISAPI と CGI のリクエスタを IIS 6.0（Windows Server2003）の環境で実行させるための設定について説明しています。

1. マイコンピュータのアイコン上で**右クリック**を行い、**管理**を選択します。
2. **サービスとアプリケーション→インターネットインフォメーションサービス→ Web サービス拡張**を選択します。
3. 新しい Web サービス拡張を追加を**クリック**します。
4. **拡張名**に任意の名前を入力します。追加ボタンを**クリック**し、MGrqispi101.dll ファイルを追加します。
5. **拡張の状態を許可済みに設定する**のチェックボックスをチェック状態にします。



上記の操作が実行されると、**Web サービス拡張**のリストに Magic リクエスタが***許可***の状態で表示されるようになります。

IIS7.0 の場合

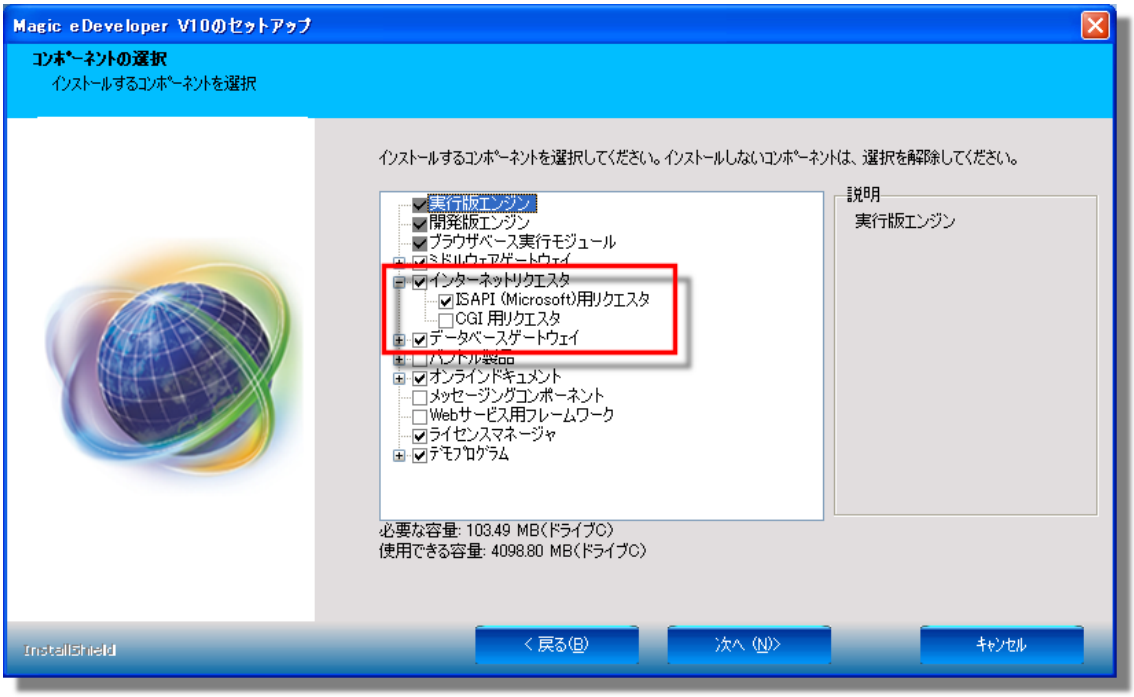


IIS 7.0（Windows Vista および Windows Server 2008）の場合は、以下のように設定します。

1. マイコンピュータのアイコン上で**右クリック**を行い、**管理**を選択します。
2. **サービスとアプリケーション→インターネットインフォメーションサービス**を選択します。
3. 右側に接続ウィンドウが表示されます。 ***サーバ名***をクリックします。
4. さらに右側のウィンドウ内の IIS グループに表示されている ISAPI および CGI の制限を**クリック**します。
5. 右側の操作ウィンドウの追加をクリックします。
6. **ISAPI または CGI パス**で参照ボタンを**クリックし**、MGrqispi101.dll ファイルを追加します。
7. **説明**に任意の名前を入力します。**拡張パスの実行を許可する**のチェックボックスをチェック状態にします。

注： 上記の設定処理は、Magic のインストール処理で行われます。

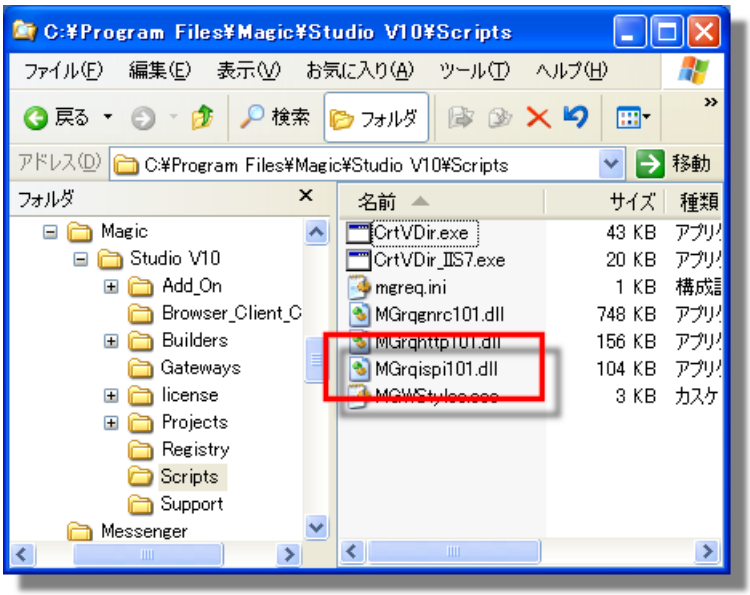
2.Magic をインストールする



次に、Magic をインストールし、インターネットリクエスタがセットアップされることを確認します。Microsoft IIS の場合は ISAPI リクエスタのチェックボックスをチェックし、Apache の場合は CGI リクエスタのチェックボックスをチェックします。

すでに Magic がインストールされている状態で、リクエスタトタのみ追加したい場合は、以下のようします。

1. **コントロールパネル→プログラムの追加と削除→ Magic Studio**（または、**Magic Enterprise Server**）**→変更と削除**をクリックします。
2. 修正のインストールオプションを選択します。
3. 前述のように、ISAPI リクエスタのチェックボックスをチェックして追加します。
4. 他のオプションのチェックを外さないようにしてください。



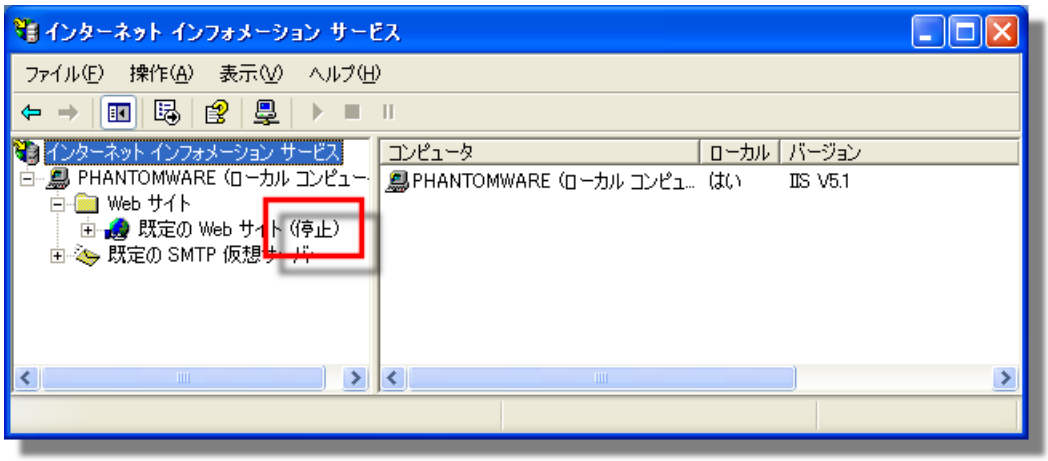
リクエスタが適切にインストールされると、Magic の Scripts サブディレクトリ内に MGrqispi.dll というファイルがコピーされます。これは、IIS 環境下で実行されるインターネットリクエスタと呼ばれる DLL です。

3.MRB が実行していることを確認する

Magic のインストール方法によって、MRB はサービスまたは実行形式のどちらかで実行されます。Windows のサービスとして実行する場合、OS の起動時に自動的に開始されます。実行形式でインストールされた場合は、手動で起動

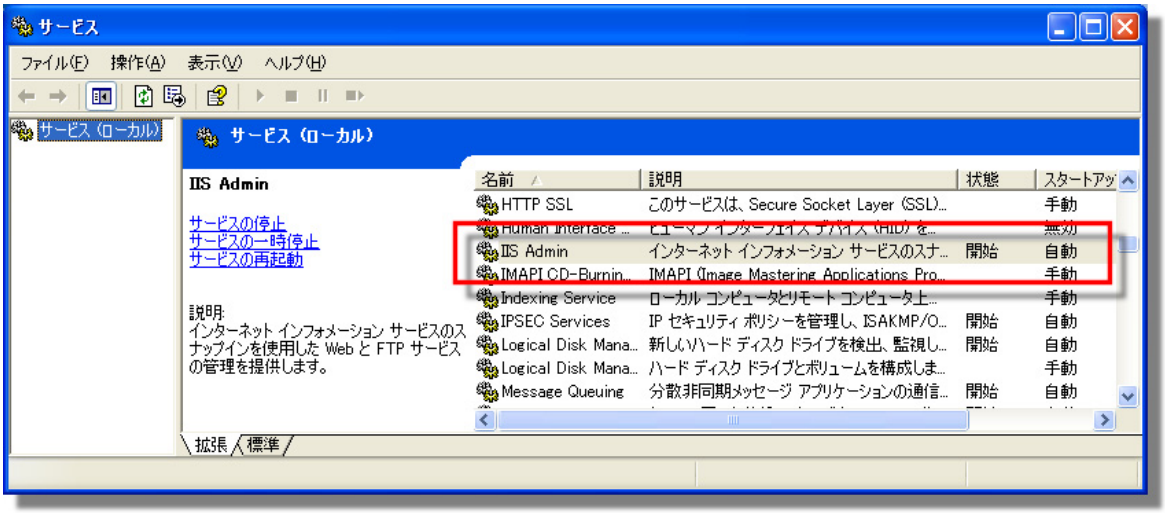
させる必要があります。MRB の起動／停止は、**ブローカとリクエスタ**メニュー（スタートメニュー→ **Magic Studio V10**）上にあります。

Broker モニタを使用することで MRB の実行状態を確認することができます。モニタに関する詳細は、「MRB の動作を監視するには」（798 ページ）を参照してください。



Web サーバの実行状態は、IIS サービスの管理ウィンドウで確認することができます。サービスが停止している場合、起動する必要があります。コンテキストメニューから Web サイトの起動を選択することができます。

4.Web サーバが実行していることを確認する



OS のサービス設定によって、PC の起動時に Web サーバを自動的に起動させるかどうかを指定することができます。IIS の場合、サービスウィンドウ（**スタートメニュー→コントロールパネル→管理ツール→サービス**）で設定することができます。

Magic がリクエストを受信できるように Apache Web サーバを設定するには

ここでは、Magic 用に Apache サーバを設定するために必要な手順を説明しています。

1. Magic をインストールする前に、あらかじめ Apache サーバをインストールしておく必要があります。
2. Magic のインターネットリクエスタをインストールします。
3. Apache のディレクトリを設定します。
4. インストール後にインストール状態を確認し、サーバが動作していることを確認してください。

これらの手順を順に説明します。

注：　ここでは Apache2（Ver2.n.n）を前提に説明しています。Apache1（Ver1.n.n）の場合は、インストール方法や起動方法等が異なります。

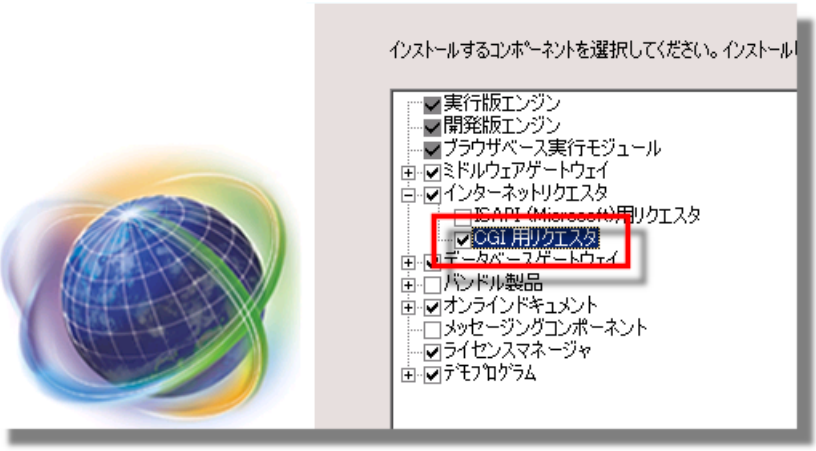
1.Apache サーバをインストールする

Apache サーバをダウンロードし、インストールします。http://www.apache.org から必要なバージョンをダウンロードすることができます。Apache Projects の左下の HTTP Server を選択してください。このサイトには、Windows セキュリティに関する問題をどのように扱うかなどを含めた、さまざまなインストール情報があります。

ダウンロードされたファイルでインストールします。表示されるメッセージに従って操作してください。Network Domain と Server Name の指定欄では、開発用 PC の場合、localhost とだけ入力してください。Apache がインストールされると、PC のシステムトレイに小さな Apahce アイコン（羽）が表示されます。このアイコンをクリックすることで、Apache サービスを管理したり監視することができます。

Apache のインストール内容を確認するには、Web ブラウザを開き、http:\\localhost という URL を入力してください。

2.Magic のリクエスタがインストールされていることを確認する



次に、Magic をインストールし、インターネットリクエスタがセットアップされることを確認します。Apache の場合は、CGI リクエスタのチェックボックスをチェックします。

すでに Magic がインストールされている状態で、リクエスタトタのみ追加したい場合は、以下のようします。

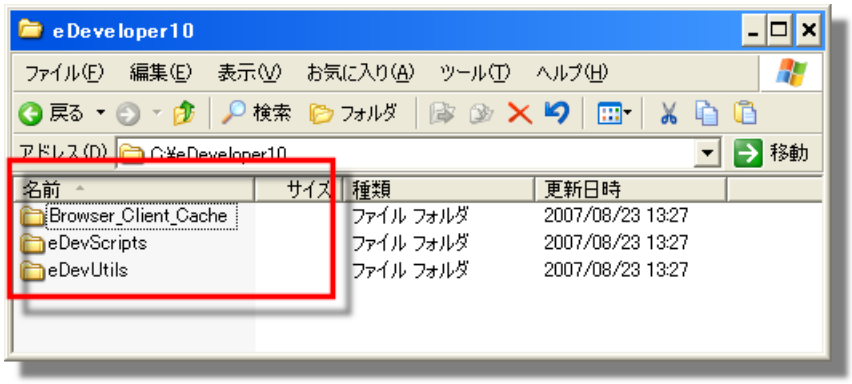
1. コントロールパネル→プログラムの追加と削除→ Magic Studio（または、Magic Entereprise Server）→変更と削除をクリックします。
2. 修正のインストールオプションを選択します。
3. 前述のように、CGI リクエスタのチェックボックスをチェックして追加します。
4. 他のオプションのチェックを外さないようにしてください。

リクエスタが適切にインストールされると、Magic の Scripts サブディレクトリ内に Mgrqcginnn.exe (nnn は Magic のバージョン番号）というファイルがコピーされます。

3.Apache のディレクトリを設定する

次に、Apache 用のディレクトリ環境を設定する必要があります。以下の手順で行います。

1. スクリプト用のディレクトリを作成します。
2. これらのディレクトリが認識できるように Apache のコンフィギュレーションファイルを修正します。
3. Magic.ini でこれらのディレクトリを定義します。



ディレクトリの作成

以下の３つのディレクトリを作成する必要があります。

- スクリプト用
- ユーティリティ用
- キャッシュ用

この例では、C:¥eDeveloper10 というディレクトリ内にこれらの３つを作成しています。以下の例ではこれらのディレクトリを使用しています。

スクリプト用ディレクトリ

例では、eDevScripts です。このディレクトリ内に以下の Magic ファイルをコピーします（”xxx” は、Magic のバージョンによって異なります)。

- mgrqgnrcxxx.dll
- mgrqhttpxxx.dll
- mgrqcgixxx.exe
- MGREQ.INI

注： ファイル名は、Magic のバージョンによって異なる場合があります。

ユーティリティ用ディレクトリ

例では、eDevUtils です。このディレクトリ内に Magic の BC モジュール用ファイルをコピーします。.

- MGBC*.cab.
- MGBC*.js.
- *.jpg.
- *.css

キャッシュ用ディレクトリ

例では、Browser_Client_Cache です。このディレクトリは、ブラウザクライアントの実行時にキャッシュファイルを作成するためのディレクトリです。

Apache のコンフィギュレーションファイルの変更

Apache のコンフィギュレーションファイルは httpd.conf という名前のテキストファイルで、Apache のインストール先の ¥conf サブディレクトリにコピーされます。

ここに、前述で作成した３つのディレクトリに対するエイリアスを定義する必要があります。以下の内容をコピーしディレクトリ名だけを実際のインストール環境に合わせて変更することで利用できます。

```
ScriptAlias /eDevScripts/ "C:/eDeveloper10/eDevScripts/"
<Directory "C:/eDeveloper10/eDevScripts">
```

```
Options None
AllowOverride None
Order allow,deny
Allow from all
</Directory>

Alias /eDevUtils/ "C:/eDeveloper10/eDevUtils/"
<Directory "C:/eDeveloper10/eDevUtils">
Options None
AllowOverride None
Order allow,deny
Allow from all
</Directory>

Alias /eDevBCCache/ "C:/eDeveloper10/Browser_Client_Cache/"
<Directory "C:/eDeveloper10/Browser_Client_Cache">
Options None
AllowOverride None
Order allow,deny
Allow from all
</Directory>
```

Magic.ini の変更

Magic.ini を以下のように変更します。

```
InternetDispatcherPath = /eDevScripts/mgrqcgixxx.exe
WebDocumentPath = C:¥eDeveloper10¥eDevUtils
WebDocumentAlias = /eDevUtils
CTLCacheFilesPath = C:¥eDeveloper10¥Browser_Client_Cache
CTLCacheFilesAlias = /eDevBCCache
```

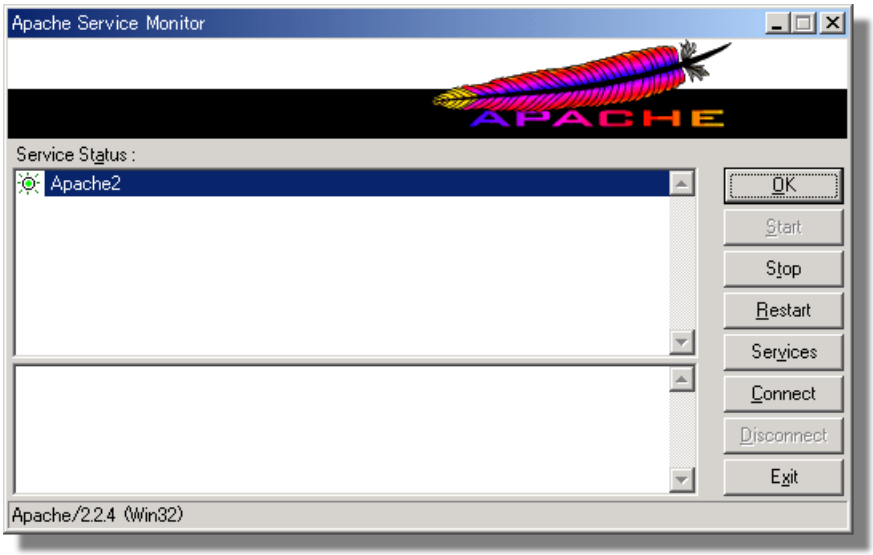
（"xxx"：は、Magic のバージョンによって異なります）

4.MRB が実行していることを確認する

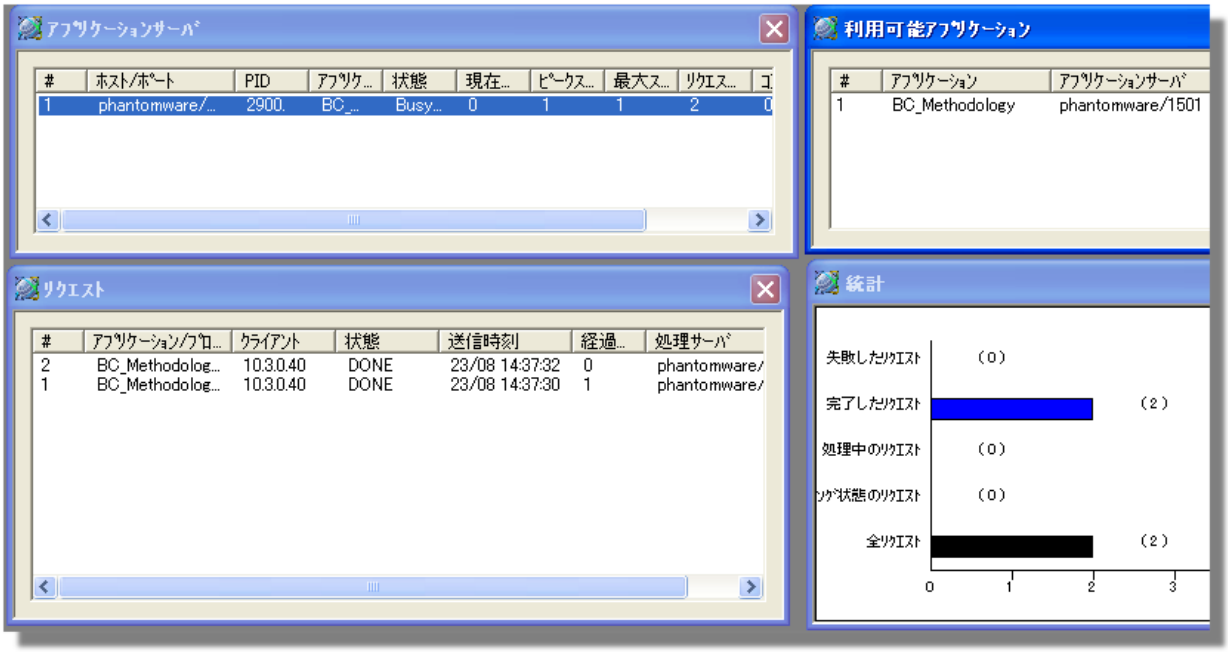
Magic のインストール方法によって、MRB はサービスまたは実行形式のどちらかで実行されます。Windows のサービスとして実行する場合、OS の起動時に自動的に開始されます。実行形式でインストールされた場合は、手動で起動させる必要があります。MRB の起動／停止は、**ブローカとリクエスタ**メニュー（**スタートメニュー→ Magic Studio V10**）上にあります。

ブローカモニタを使用することで MRB の実行状態を確認することができます。モニタに関する詳細は、「MRB の動作を監視するには」（798 ページ）を参照してください。

Apache サーバが実行していることを確認するには、Apache サービスモニタを使用するか、Windows のタスクバーにアイコンが表示されていることを確認します。

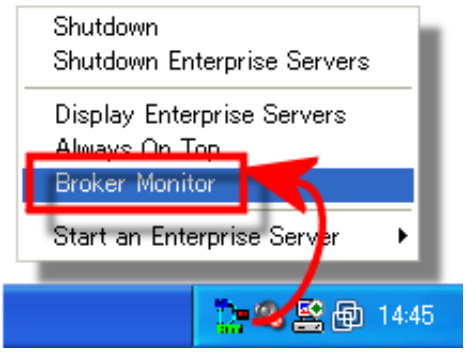


MRB の動作を監視するには



MRB を使用する場合、起動された MRB の統計情報を取得する必要があるかもしれません。Magic には、MRB の実行状況を監視するためのいくつかのツールがあります。

上図のような Broker モニタは、MRB の操作状況をビジュアルでかつリアルタイムに表示することができます。



Windows のタスクバーの MRB アイコンでコンテキストメニューを開き、**Broker Monitor** をクリックすることで Broker モニタが起動されます。

Magic のリクエスト関数

Broker モニタに表示される情報は、Magic に組み込まれているリクエスト関数を使用することでも取得できます。例えば、**RqLoad** 関数は、リクエストの総数や、各状態毎のリクエスト数（ペンディング中、処理中、処理が成功、処理が失敗）を返します。

さらにこれらの関数の中には、指定した MRB の処理を制御することができるものもあります。例えば、**RqRtBlock** 関数は、特定のサーバまたはサービスにリクエストが送られることを防止します。

これらの関数の多くは、Mgrb.ini ファイルに設定されたパスワード（**Supervisor Password** や **Query Password**）を指定する必要があります（「MRB のパスワードを定義するには」（802 ページ）を参照してください）。

コマンドラインの情報

mgreqcmdl.exe を使用することで、Magic プログラム外から MRB の情報を取得することができます。例えば、以下のように実行した場合、現在のアプリケーションサーバでサポートされている全てのアプリケーションが表示されます。

```
mgrqcmdl -query app
```

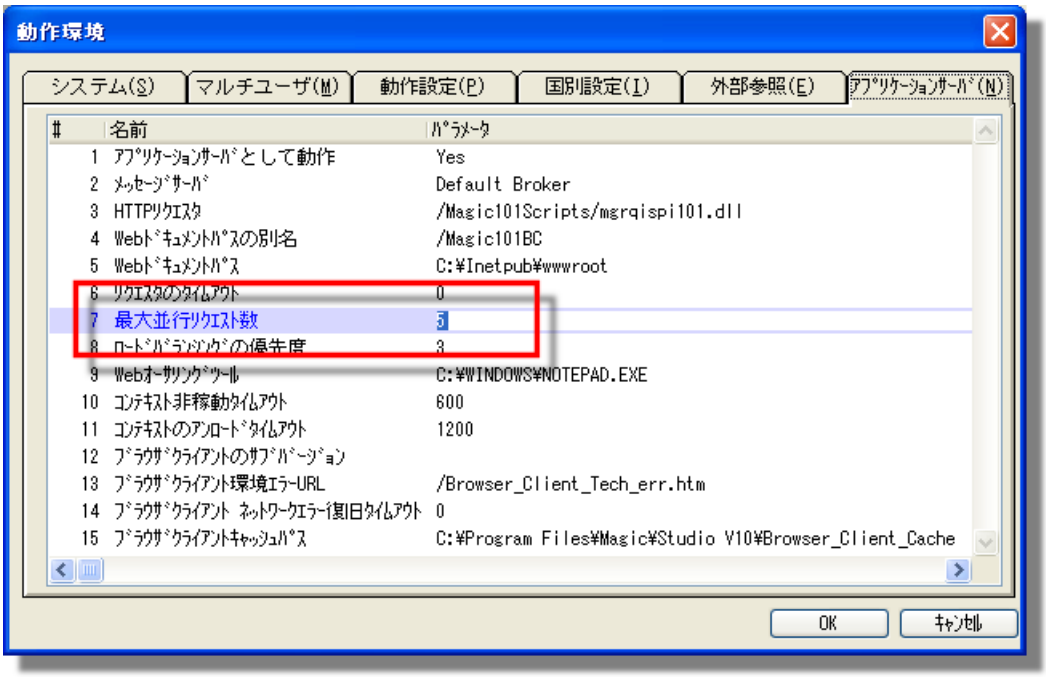
コマンドラインリクエストのパラメータ情報を確認する場合は、パラメータ無しでコマンドを実行してください。

MRB のメインログ

```
3212   12:59:30,81312  27/06/2007   Enterprise Server phantomware/1501 : Notified termination
3212           ,81312  27/06/2007   Enterprise Server phantomware/1501 : Removed : : "OK" (0)
3756   13:30:54,55250  27/06/2007 Startup  (Version eDeveloper 10.1 SP2, build Apr 12 2007)
3756           ,55296  27/06/2007   pid 3692  - C:¥PROGRAM FILES¥MAGIC¥STUDIO V10¥EDEVSTUDIO.EXE
3756           ,55296  27/06/2007   pid 1528  - C:¥PROGRAM FILES¥MAGIC¥STUDIO V10¥EDEVRTE.EXE
3756           ,55296  27/06/2007   pid 1604  - C:¥PROGRAM FILES¥MAGIC¥STUDIO V10¥MGRQMRB.EXE
3756           ,55296  27/06/2007   pid 3752  - C:¥PROGRAM FILES¥MAGIC¥STUDIO V10¥MGRQMRB.EXE
3756           ,55296  27/06/2007   BrokerPort = /4000
3756           ,55296  27/06/2007   CommTimeout = 1000
3756           ,55296  27/06/2007   ReLoad     = TRUE
3804   13:30:55,55796  27/06/2007   Enterprise Server phantomware/1501 : Inserted  (pid .1528  ,
e 1 threads | 0 requests, "")
```

mrb_event.log から情報を取得することもできます。このログファイルは、デフォルトで Magic のインストールディレクトリに作成されます。

Magic が同時に処理するリクエスト数を制限するには



Magic が使用するライセンスの内容にもとづいて、アプリケーションサーバが同時に処理することができるスレッド数が決定されます。しかし、**最大並行リクエスト数**（**オプション→設定→動作環境→アプリケーションサーバ**）の設定によってスレッド数の上限を制限することができます。エンジンが使用するスレッド数は、ライセンス数の上限内までの値を指定することができます。

0 が指定された場合、スレッド数はライセンス上の上限まで使用できます。

MRB が自動的にアプリケーションを起動するようにするには


```
[APPLICATIONS_LIST]
Online = eDevStudio.exe /DeploymentMode=T,C:¥Program Files¥Magic¥Studio V10,,,0,0
MyApp1 = eDevRTE.exe /DeploymentMode=R /StartApplication=C:¥MAGIC¥Projects¥Masterring
eDev¥Mastering.ecf,C:¥Program Files¥Magic¥Studio V10,,,0,0
```

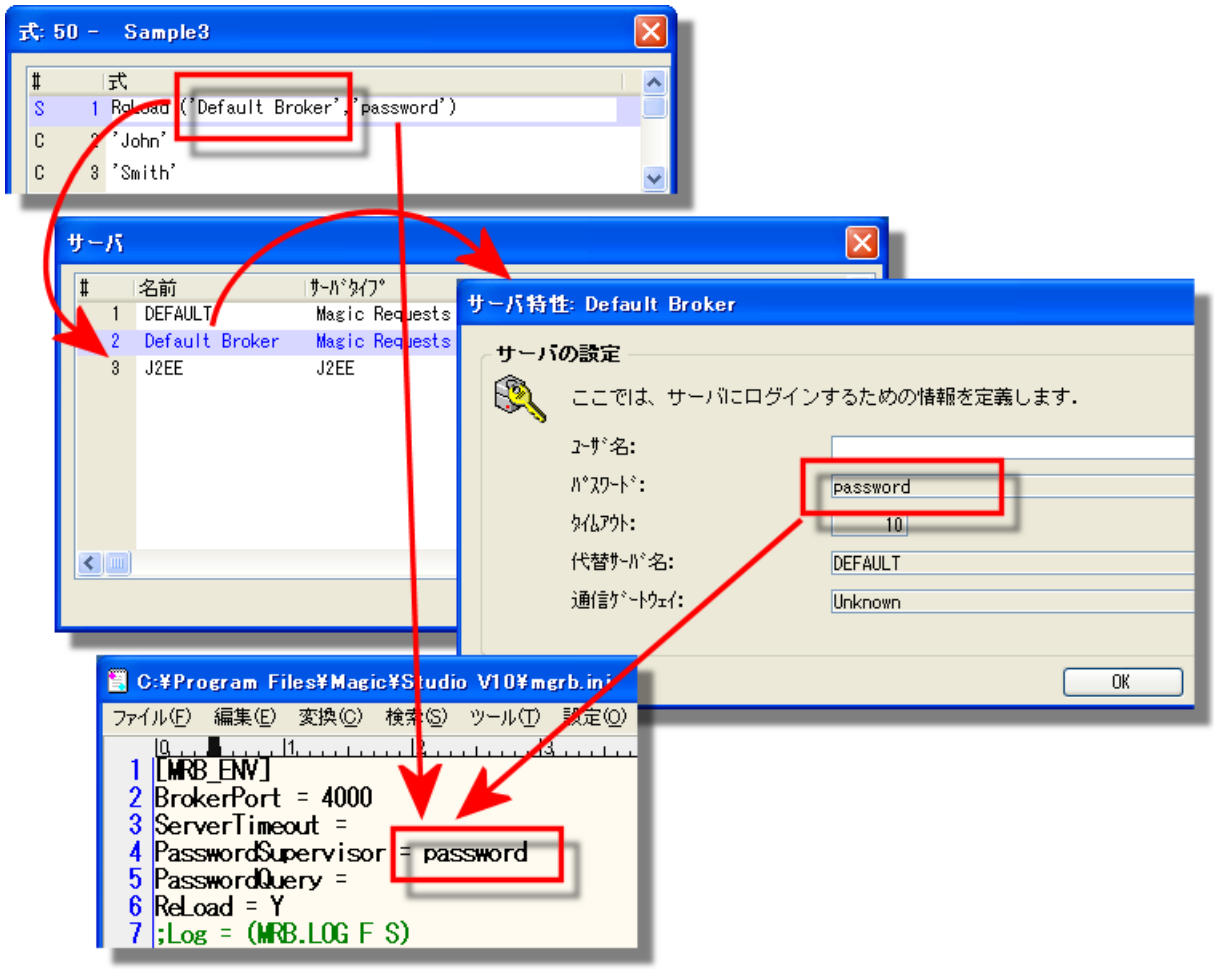
アプリケーションを自動的に起動するようにしたい場合は、mgrb.ini の [APPLICATIONS_LIST] セクションで指定します。上記の例は、Magic Studio と Mgaic Client を起動するように指定したものです。

構文は以下の通りです：

エントリ名 =< コマンド >［< 作業フォルダ >］，［< ユーザ名 >］，［< パスワード >］，［<MRB の初期化時に実行される回数 >］,［< 起動されるエンジンの最大数 >］

<MRB の初期化時に実行される回数 > に ***1*** が設定された場合、指定されたアプリケーションを 1 インスタンスだけ起動するようになります。

MRB のパスワードを定義するには



MRB のパスワードは、mgrb.ini で定義します。ここには、２つのパスワード（**Supervisor Password** と **Query Password**）が設定されています。**Supervisor Password** は MRB を管理する場合、**Query password** は MRB に問い合わせをする場合に必要です。

Supervisor Password と同じパスワードを**サーバ特性**（**オプション→設定→サーバ→特性**）に指定する必要があります。

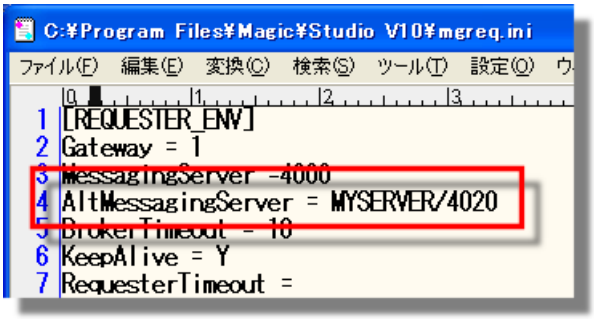
Supervisor Password は、Magic のインストール時に設定されます。カスタムインストールを行うことでパスワードを任意に指定することができます。

Query password はインストール処理では設定されません。これは、エンジンや実行アプリケーションの状況を確認するためだけに限定して使用されるためです。

代替えの MRB を定義するには

Magic のリクエスタ機構は、メインで使用される MRB に加え、代替えの MRB を使用することができます。これによりリクエスタは、メインの MRB が動作していない場合、代替えの MRB を使用して処理を行うことができます。

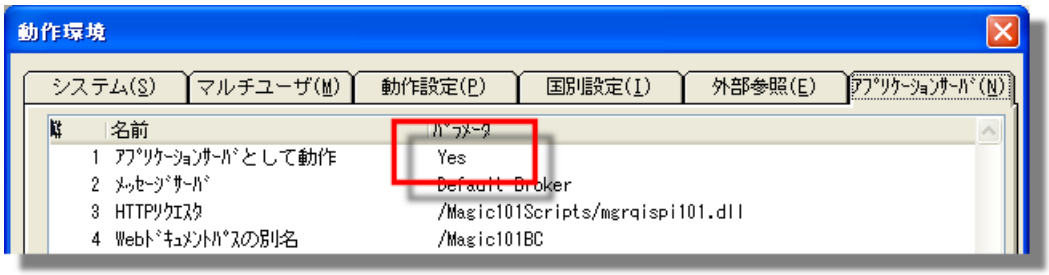
Mgreq.ini で MRB を定義する



リクエスタの動作環境は、mgreq.ini という名前のファイルで定義します。

代替えの MRB を追加するには、**AltMessagingServer** に使用する MRB のホスト名 / ポート番号を指定するだけです。

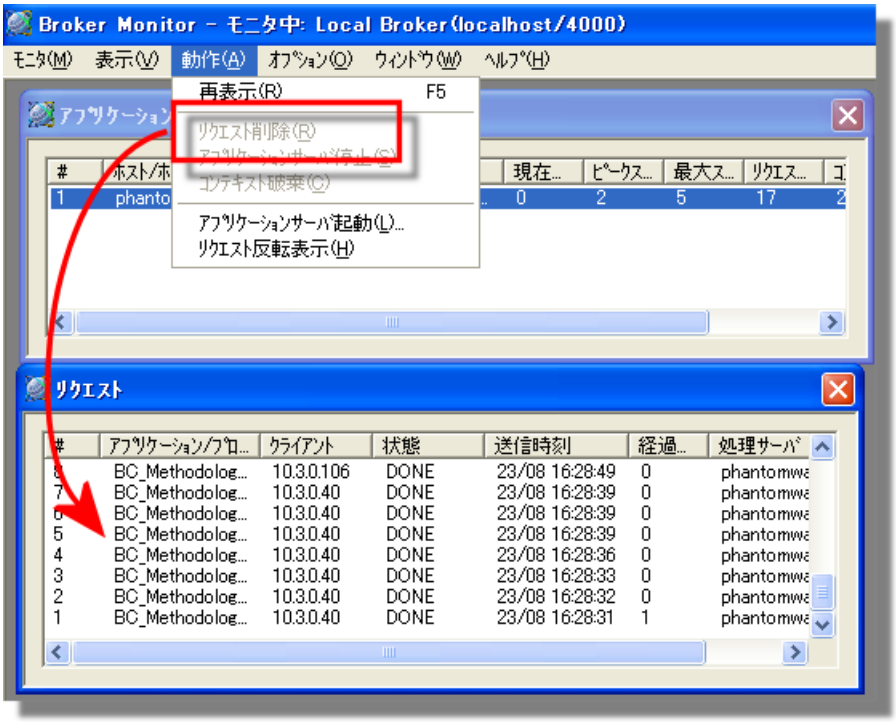
実行エンジンが MRB のリクエストを受けないようするするには



Magic エンジンがリクエストを受け付けないようにするには、**アプリケーションサーバとして動作**（**オプション→設定→動作環境→アプリケーションサーバ**）を *No* に設定します。

ただし、一時的にリクエストを受けないようにする場合は、**RqRtBlock** 関数を使用します。再びリクエストを受信するようにする場合は、**RqRtResume** 関数を使用します。

キュー内で待ち状態のリクエストを削除するには



リクエストがまだリクエストキュー内でペンディング状態の場合、Broker モニタからリクエストを削除することができます。

- リクエストパネルでリクエストを選択します。
- **動作→リクエスト削除**を選択します。

これで、リクエストは削除されます。

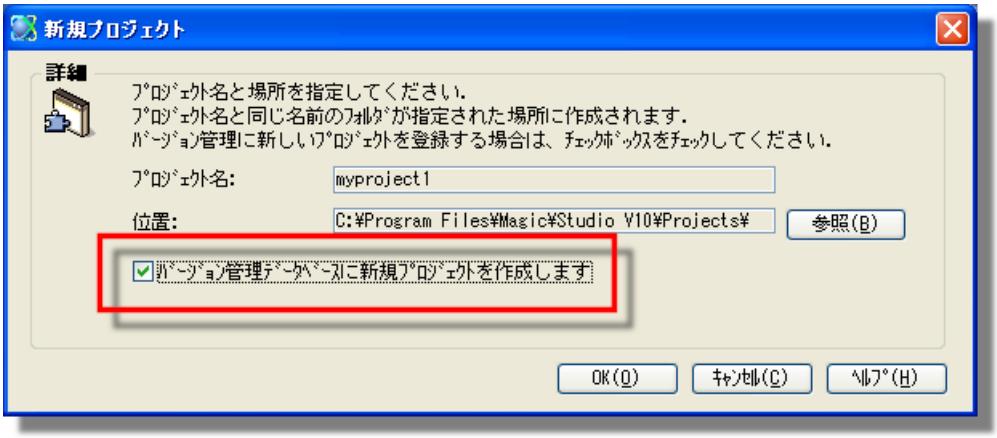
ロードバランシングを実装するには

簡単なロードバランシング構成を実現するには、少なくとも２つのアプリケーションサーバが同じアプリケーションを実行し、同じ MRB に接続する環境が必要です。MRB は自動的に Magic サーバ間でリクエスト処理を平準化するように動作します。

複数の異なる PC 上で複数の異なる Magic サーバを実行させることで、より複雑なロードバランシングを実現することができます。さらに、**ロードバランシングの優先度**（**オプション→設定→動作環境→アプリケーションサーバ**）を設定することで、各サーバに対し異なる優先順位を指定することができます。

第 39 章： ソース管理

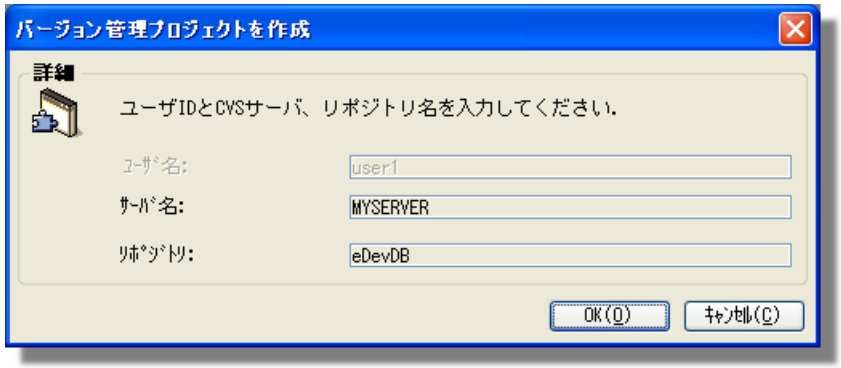
バージョン管理ソフトによって管理されるプロジェクトを作成するには



Magic で新しいプロジェクトを作成する際に、最初からバージョン管理で管理するように指定することができます。ここではその方法について説明します。

必要条件： あらかじめバージョン管理データベースを設定しておく必要があります。詳細は、「バージョン管理プロバイダを決めるには」（819 ページ）を参照してください。

1. **ファイル→新規作成**を選択するか、**ウェルカムスクリーン**上の新規作成ボタンを**クリック**することで、手動で新規プロジェクトを作成します。**新規プロジェクト**ダイアログが表示されます。
2. 通常と同じように**プロジェクト名**と**位置**を入力します。そして、**バージョン管理データベースに新規プロジェクトを作成する**チェックボックスをチェックします。



3. その際、使用するサーバ名と VC データベースの確認ダイアログが表示されます。Magic のインストール時に CVS のバージョン管理サーバも選択している場合、CVS にはデフォルトで eDevDB と呼ばれるリポジトリが作成されます。ただし、別の名前のリポジトリを定義していたり、別のバージョン管理サーバを使用することもできます。
4. OK をクリックします。Magic がデータベース内にプロジェクトを登録している間、処理スクリーンが表示されます。

これで、新しいプロジェクトがバージョン管理ソフトによって管理されることになります。

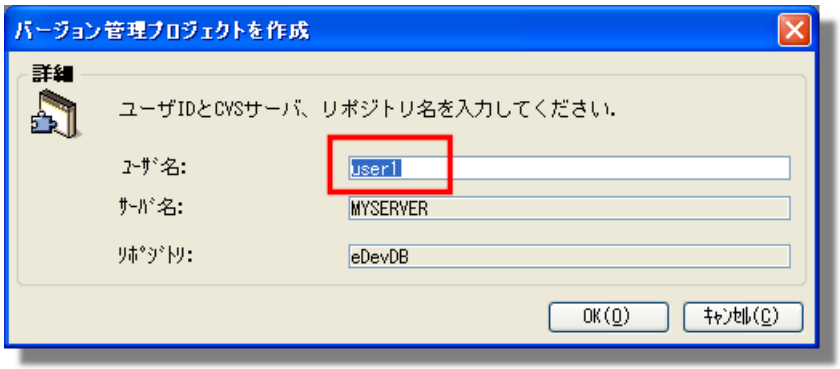
CVS を使用した操作手順の詳細は、「CVS を使用して開発するには」（821 ページ）を参照してください。

CVS サーバへのログインユーザを指定する

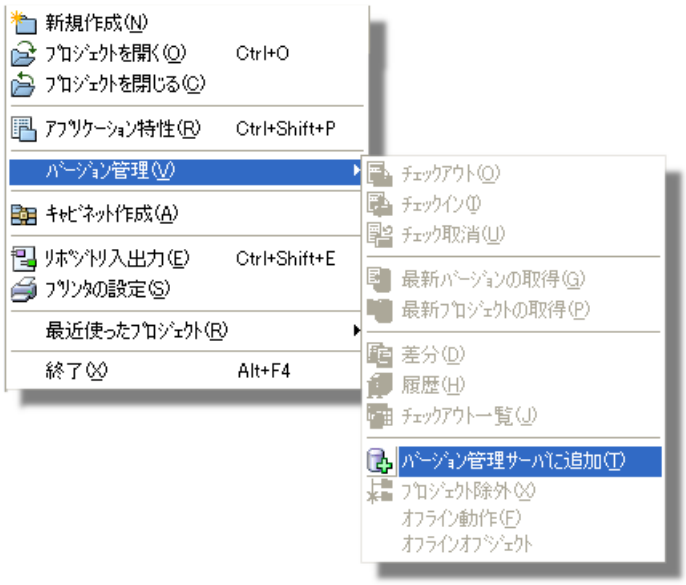
CVS サーバへのログインは、デフォルトでは Windows のログインユーザに固定されていますが、変更することもできます。

1. Magic.ini ファイルの **[MAGIC_SPECIALS]** セクションに **SpecialCVSLogin** キーワードを設定します。
   ```
   [MAGIC_SPECIALS]
   SpecialCVSLogin = Y
   ```

これにより、ユーザ ID の指定欄が有効になります。



既存のプロジェクトをバージョン管理データベースに追加するには

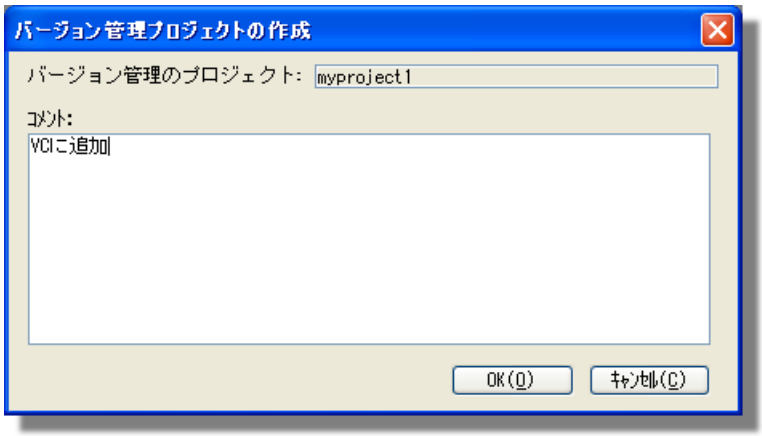


すでに開発中のプロジェクトが存在している場合、必要であれば VC サーバに登録することができます。

必要条件： あらかじめバージョン管理データベースを設定しておく必要があります。詳細は、「バージョン管理プロバイダを決めるには」（819 ページ）を参照してください。

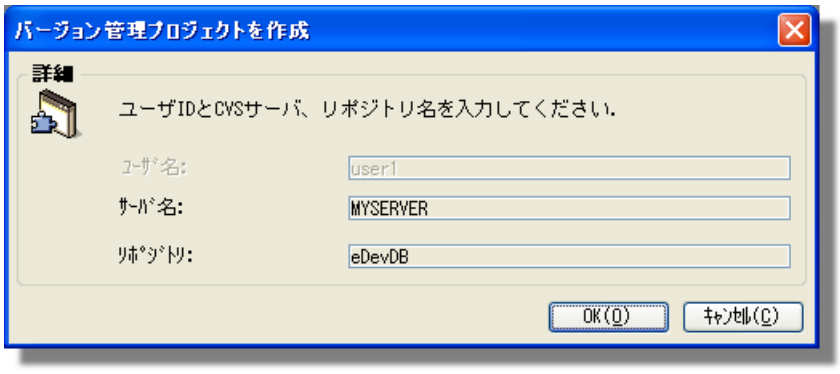
1. バージョン管理を行いたいプロジェクトを開きます。

Source Safe の場合



Source Safe を使用している場合は、このようなウィンドウが表示されます。プロジェクトを作成する時にコメントを追加することができます。

CVS の場合



CVS を使用している場合は、サーバ名とリポジトリを定義するダイアログが表示されます。この場合はコメントを入力することができません。

2. バージョン管理プロジェクトの作成ダイアログが表示されます。上記のように、使用しているバージョン管理製品に応じて表示されるウィンドウが異なります。必要なデータを入力し OK をクリックします。
3. しばらく処理を実行し、プロジェクトは VC データベースに登録されます。

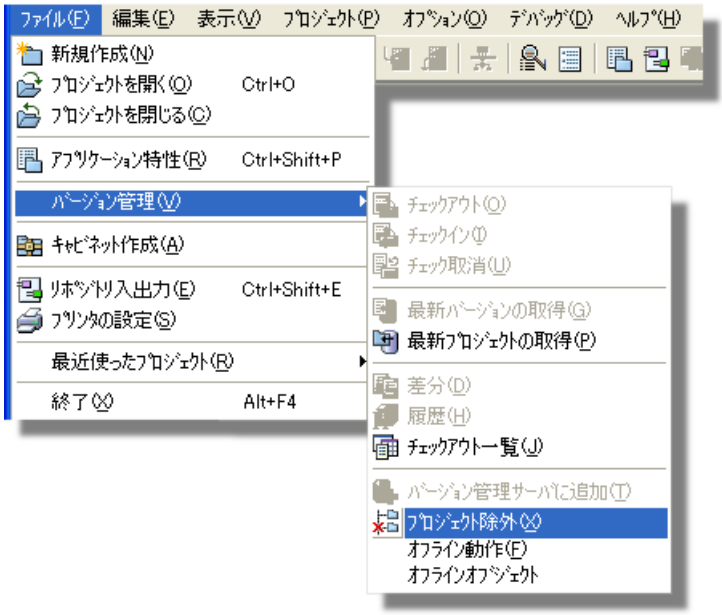
VC データベース内のプロジェクト

プロジェクトが VC データベースに登録されると、VC ツールで内容を確認することができます。上記の図は、プロジェクトが Visual SourceSafe でチェックインされた場合に Visual SourceSafe エクスプローラで表示される例を示しています。

VC データベースに定義した場合の効果

プロジェクトを VC データベースに登録すると、最初にプログラムをチェックアウトしない限り変更することはできません。チェックアウトしないでプログラムを開こうとすると、警告メッセージが表示され読み込み専用モードで開きます。

プロジェクトをバージョン管理から削除するには



プロジェクトを VC サーバで管理しなくなった場合、**ファイル→バージョン管理→プロジェクト除外**を選択することで、VC サーバの管理から除外することができます。

プロジェクトを除外しても、プロジェクトファイルのコピーはまだ VC サーバ上に残っています。再びバージョン管理を使用し始める場合は、VC サーバ上のコピーかクライアント側のコピーのどちらかを削除する必要があります。

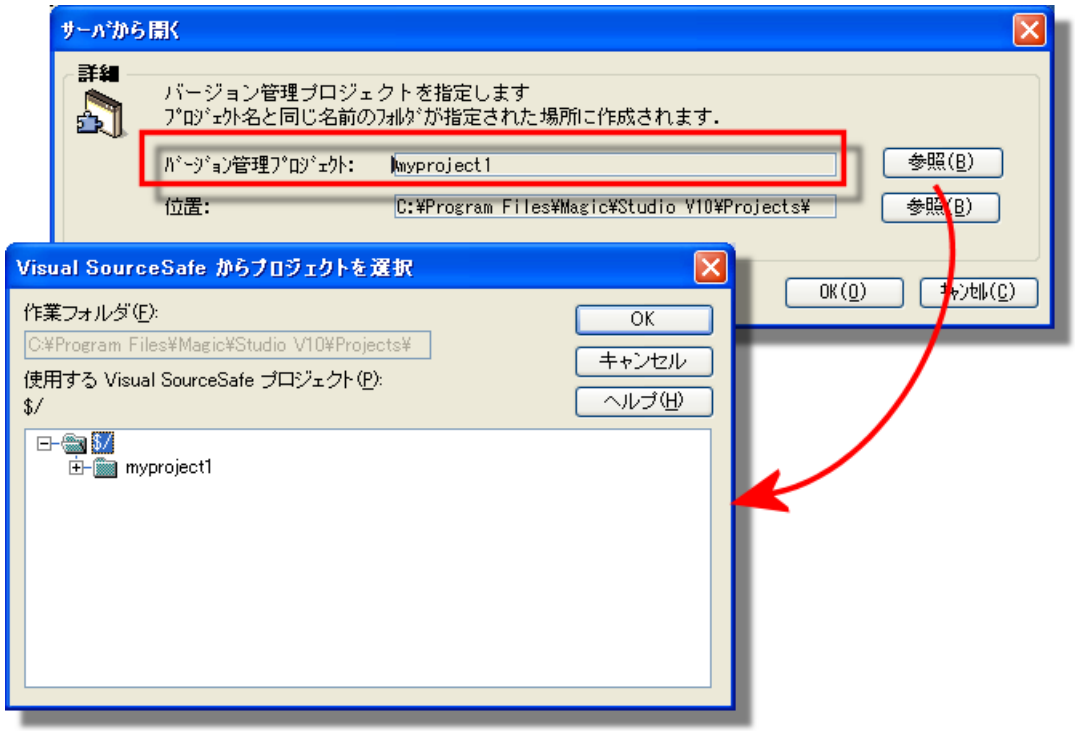
バージョン管理プロジェクトを新たにオープンするには

バージョン管理されたプロジェクトを開発者が初めてオープンする場合、一旦、開発者は作業用としてプロジェクトのローカルなコピーを作成します。

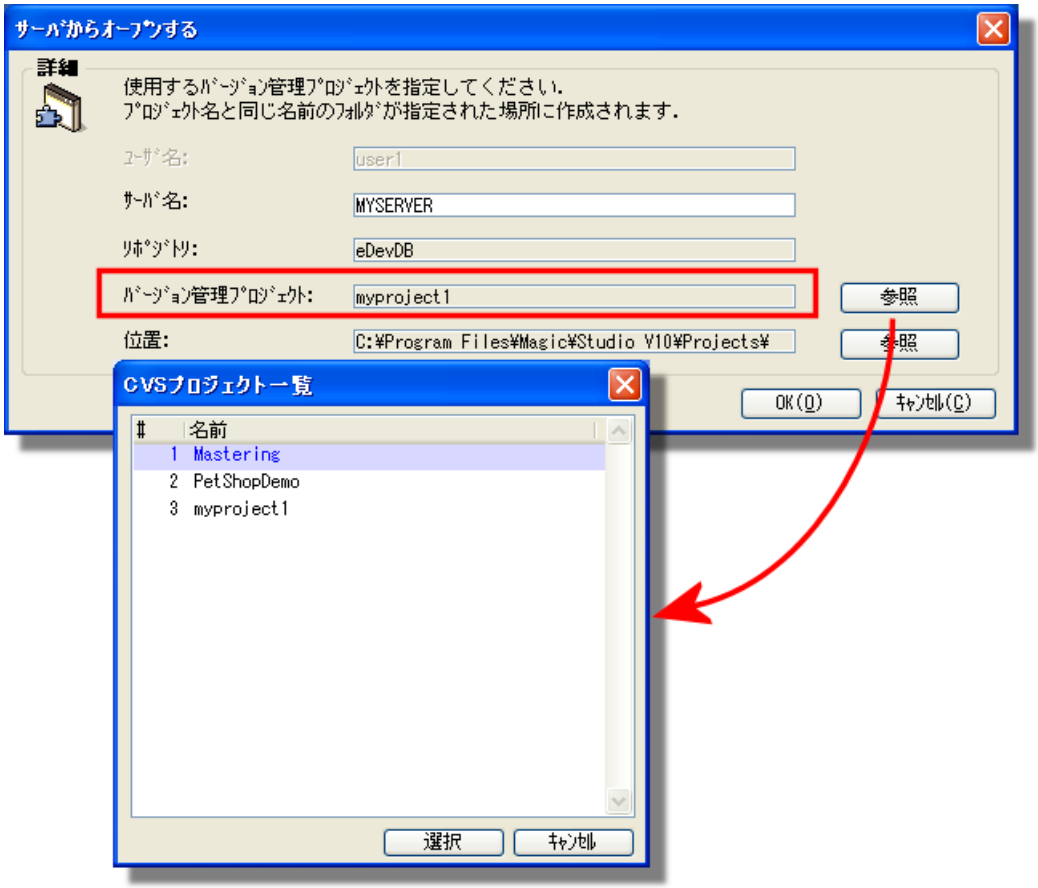
1. 現在のプロジェクトを閉じます。

2. ファイル→バージョン管理→サーバから開くを選択します。

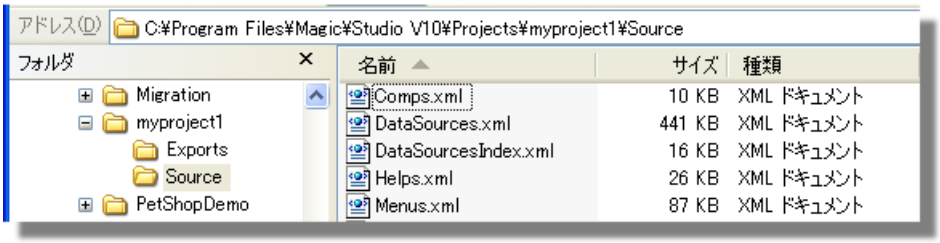
Source Safe の場合



CVS の場合



3. サーバからオープンするダイアログが表示されます。
4. 参照をクリックし、オープンするプロジェクトを選択します。この例では myproject1 を選択しています。
5. プロジェクトのコピーは位置で指定されたパスに作成されます。

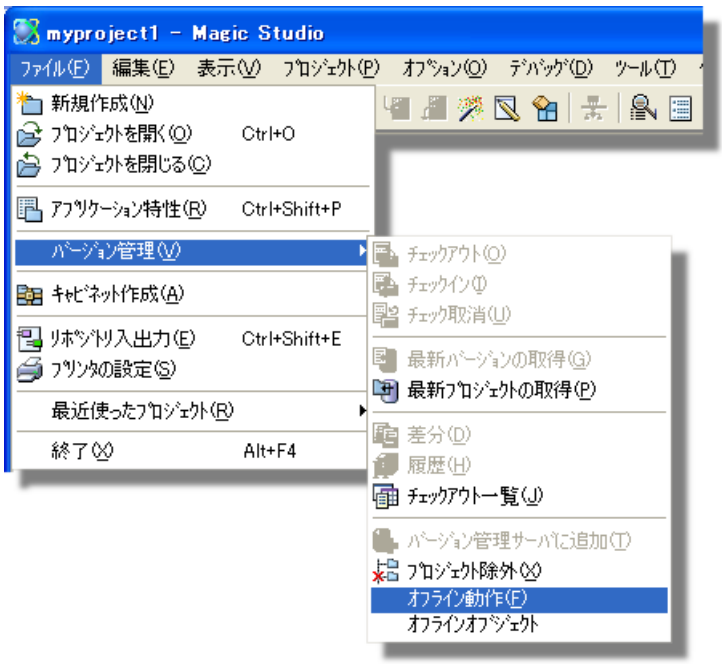


注： 1 台の PC 上で複数のユーザ ID を使用してプロジェクトをオープンするような場合は、一覧に指定するコピー先のパスをユーザ毎にユニークなパスにする必要があります。

バージョン管理サーバが無効な場合に、バージョン管理プロジェクトを開発するには

バージョン管理を使用している場合、VC データベースが何らかの理由でオフラインになる可能性があります。このような場合に開発を止めないようにしたいのであれば、オフラインの状態で VC 環境のまま password 開発を継続させることができます。

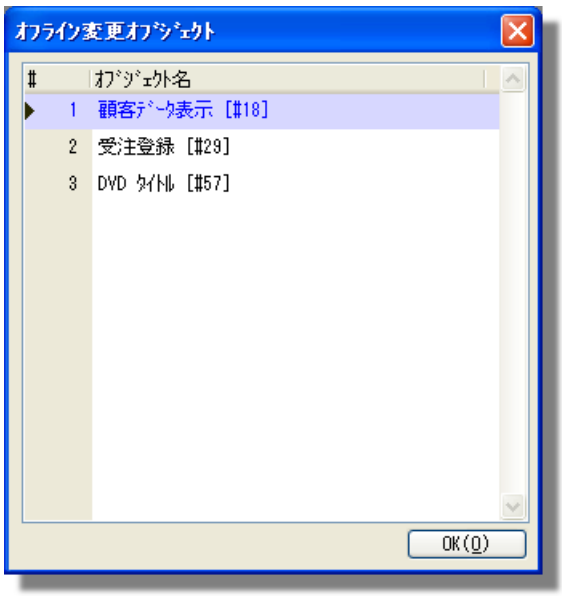
オフライン状態で開発する場合もオブジェクトを変更することができます。Magic は変更されたオブジェクトを追跡し、VC サーバに再接続したときこれらの変更を反映するようにします。



プロジェクトは、一度オフラインモードになると VC プロジェクトで管理されていないかのように動作します。チェックアウトの操作を行うことなくリポジトリやリポジトリ内のオブジェクトを修正することができます。

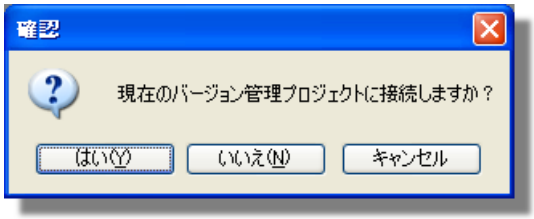
オフラインの場合に変更される項目

ファイル→バージョン管理→オフラインオブジェクトを選択することで、オフラインで作業するように選択してから変更された全てのオブジェクト一覧が表示されます。

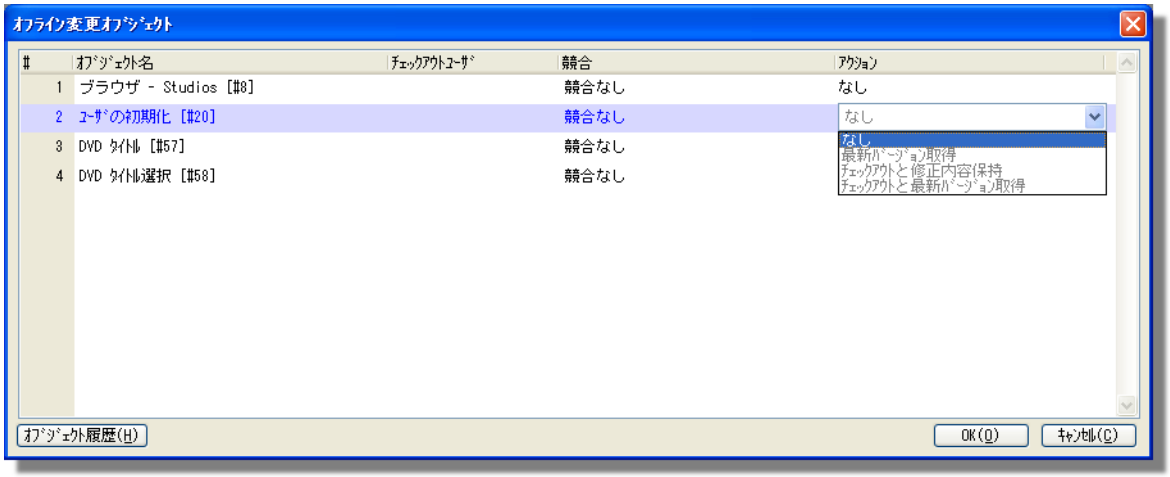


このモードでは、変更されたオブジェクトを見るために VC サーバをオンラインにする必要がありません。しかし、取得できる情報がないためオブジェクト名だけが表示されます。

VC への再接続



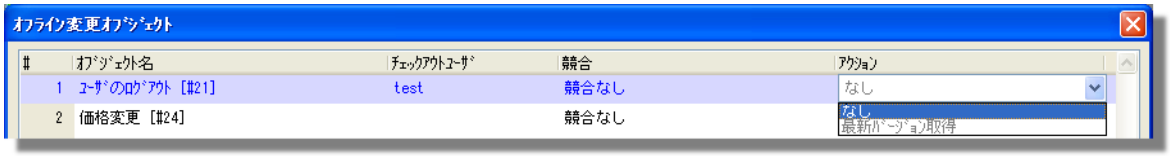
オフラインモードで作業している場合、新しい **Magic Studio** のセッションを開始するたびに、VC サーバに接続するかどうかを確認するダイアログが表示されます。



その際はい（Y）を**クリック**すると、変更されたオブジェクトの一覧が表示されます。ここで変更された各オブジェクトに対してどのような対応を行うかを指定することができます。

1. **オブジェクト名**カラムでは、Magic 上の名前が表示されます。この名前には、バーレン（丸括弧）内にプログラム番号が表示されます。
2. **チェックアウトユーザ**カラムには、チェックアウトしたユーザ ID が表示されます。
3. **競合**カラムには、競合しているかどうかが表示されます。２人の異なるユーザが同じオブジェクトに異なる変更を加えた場合、競合していることになります。
4. **アクション**カラムでは、競合している場合の対応方法を選択することができます。

すでに他のユーザによってチェックアウトされているオブジェクトがオフラインモードで修正されている場合、以下の２つのオプションのみ表示されます。



- なし …… 何もしません（変更内容を有効にします）。
- 最新バージョン取得 …… 最新のバージョンによって変更内容を上書きします。

更新内容を追跡するには

開発者がプログラムをチェックアウトしたりチェックインするたびに、VC ツールは変更内容を追跡します。開発者は、変更理由について簡単なコメントを書き込むことができます（これは必ず行ってください）。

変更内容を確認したい場合は、ファイル→バージョン管理→履歴を選択することで参照できます。どのような内容が表示されるかは、使用している VC ソフトに依存します。ここでは、Source Safe と CVS での例を紹介します。

モデルとデータソースをチェックアウトする

モデルとデータソースは個別にチェックアウトすることができません。これらのリポジトリ内でファイル→バージョン管理→チェックアウトを選択すると、リポジトリ全体がチェックアウトされます。

プログラムのチェックイン / チェックアウトを行う

リポジトリ内のオブジェクトを使用して開発する場合、ファイル→バージョン管理→チェックアウトを選択します。その際、コメントを入力するダイアログが表示され、その後に編集可能な状態で開きます。

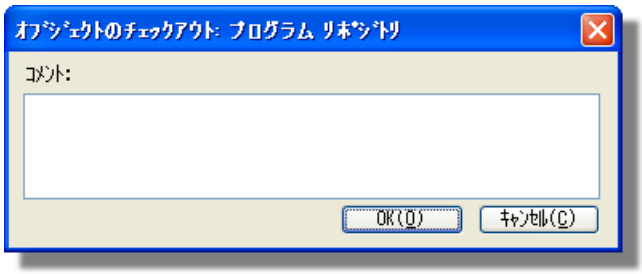
修正が終了したら、オブジェクト内でファイル→バージョン管理→チェックインを選択します。その際もコメントを入力するダイアログが表示されます。

これらのコメントは、オブジェクトの変更履歴として表示されます。

自動的にプログラムリポジトリをチェックアウトする

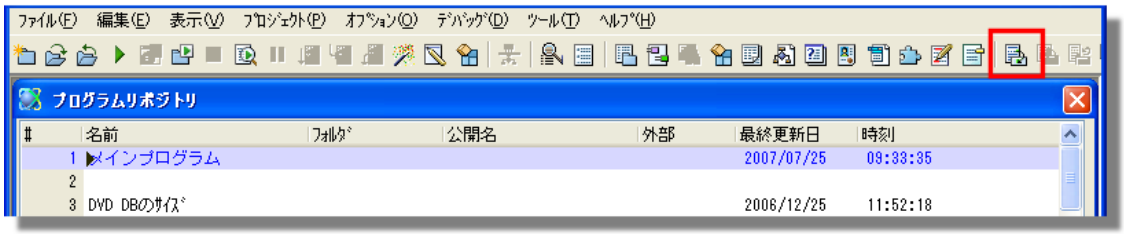
オブジェクトの追加や削除などの変更を行う場合、リポジトリ全体をチェックアウトする必要があります。

開発者がこのような処理を行おうとした場合、Magic は自動的にリポジトリをチェックアウトしようとします。その際、コメントを入力するダイアログが表示され、リポジトリはチェックアウトされます。



注： CVS の場合、チェックアウト時にコメントは入力できません。

手動でプログラムリポジトリをチェックアウトする



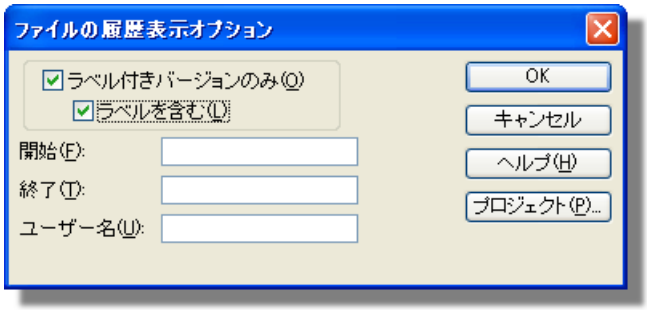
手動でプログラムリポジトリをチェックアウトするには以下のようにします。

1. ヘッダ行にカーソルを置きます。
2. リポジトリのチェックアウトアイコンをクリックします。

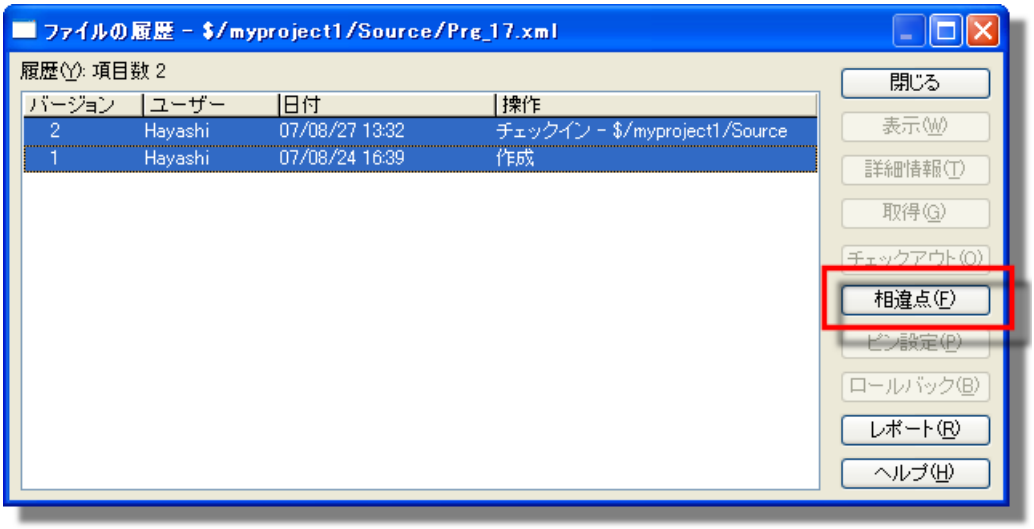
ヘッダ行（カラム名が表示されている行）に位置付けた状態でチェックアウトアイコンをクリックすることで、手動でリポジトリのチェックインやチェックアウトを行うことができます。

Source Safe での履歴

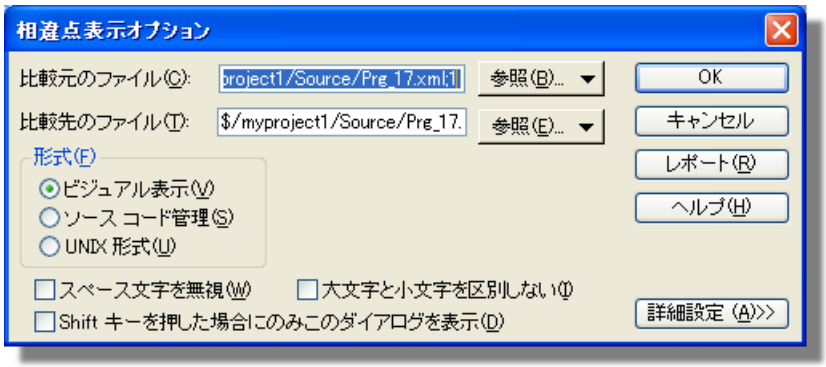
変更内容を確認する場合、ファイル→バージョン管理→履歴を選択することで変更履歴を参照することができます。



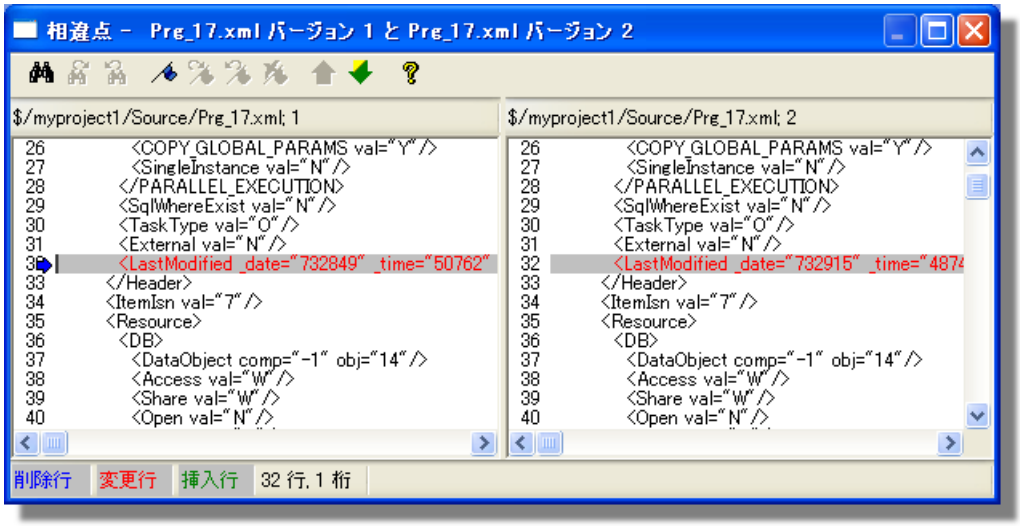
ファイルの履歴ダイアログでは、いくつかのフィルタリング条件を入力することができ、任意の履歴だけを表示させることができます。必要なオプションを入力し、**OK** をクリックします。



これで変更履歴が表示されます。２つの変更履歴を選択し、相違点をクリックします。

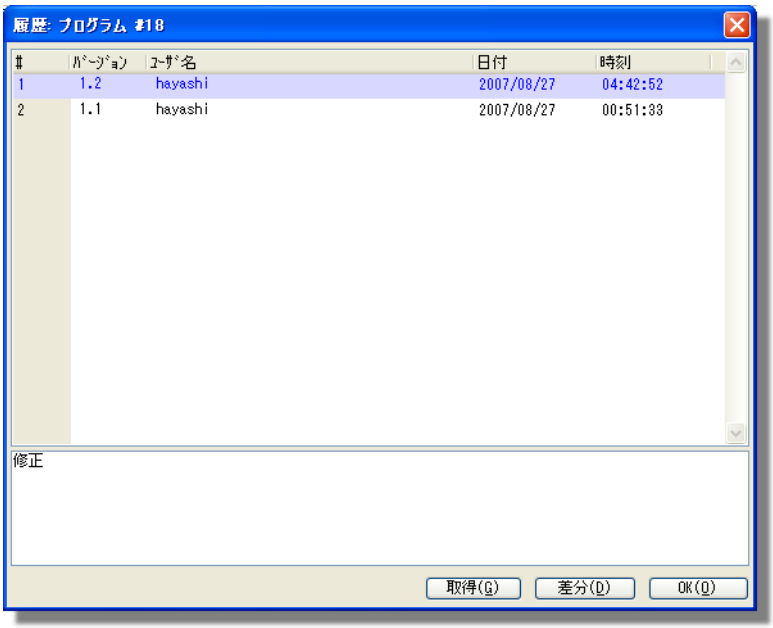


相違点表示オプションダイアログが表示されます。このダイアログは、どのようにしてバージョンを比較するかを決めることができます。オプションを選択したら OK をクリックします。



２つのファイルの比較結果が並べた状態で表示されます。変更、挿入、削除の行が色分けされて表示されます。

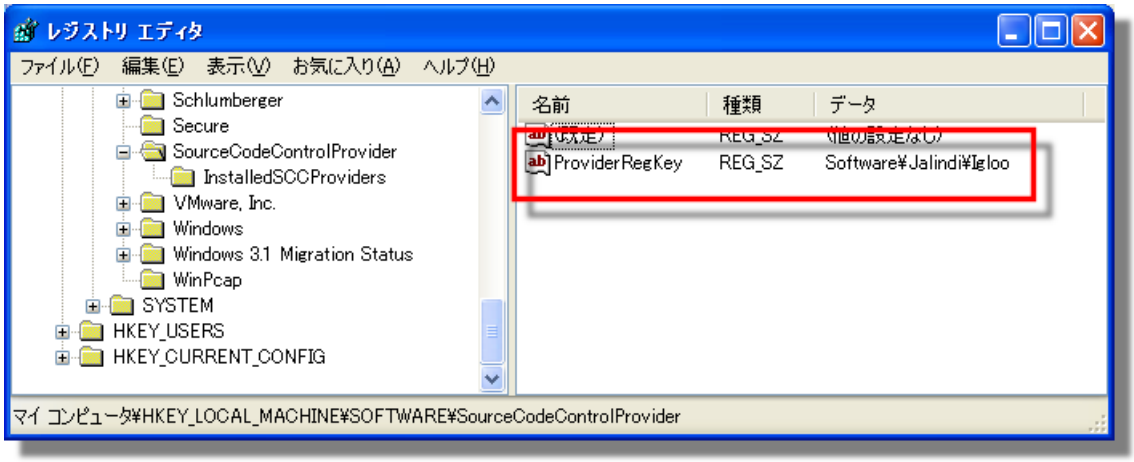
CVS での履歴



CVS を使用している場合、ファイル→バージョン管理→履歴を選択することで、パークしているオブジェクトの変更履歴が表示されます。カーソルがヘッダ行のある場合は、リポジトリのチェックイン／チェックアウト履歴が表示されます。

差分をクリックすると差分ツールコマンドライン（オプション→動作環境→外部参照）で指定された差分ツールが起動されます。指定されていない場合は、エラーになります。

バージョン管理プロバイダを決めるには



現在使用されているバージョン管理プロバイダは、OS のレジストリに登録されています。登録内容は以下のようにして確認できます。

1. スタートメニュー→ファイル名を指定して実行を選択します。
2. Regedit を入力し、**OK** をクリックします。レジストリエディタが起動されます。
3. 次のパスを選択します。

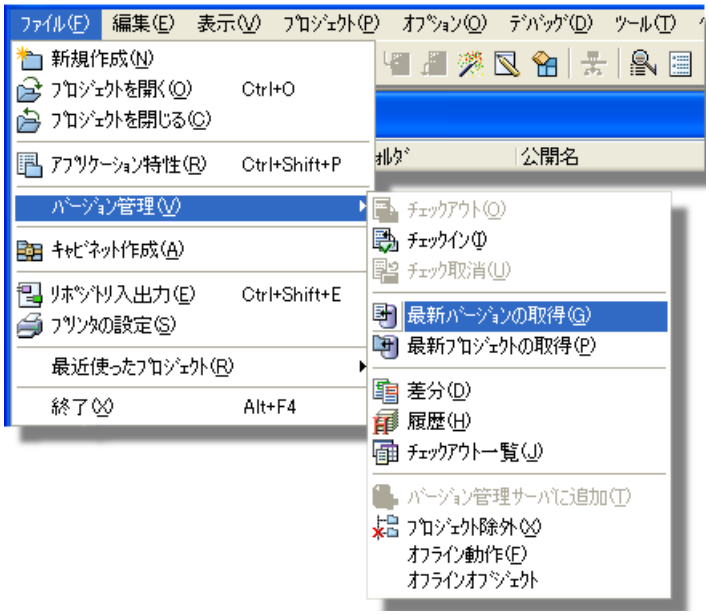
   HKEY_LOCAL_MACHINE¥SOFTWARE¥SourceCodeControlProvider¥ProviderRegKey

ここに Magic が使用するソースコード管理プロバイダが定義されています。他のバージョ管理システムがインストールされている場合もあります。しかし、使用できるものはここに定義されているものだけです。

以下のキー内にインストールされているすべてのプロバイダのリストが定義されています。

HKEY_LOCAL_MACHINE¥SOFTWARE¥SourceCodeControlProvider¥InstalledSCCProviders

オブジェクトを前の状態にロールバックするには



この説明における***ロールバック***とは、オブジェクトまたはプロジェクト全体を前のバージョンに戻すことを意味しています。

1 つのオブジェクトの最新バージョンを取得するには

1 つのオブジェクトの最新バージョンを取得するには以下のようにします。

1. ロールバックしたいオブジェクト上にカーソルを置き、**ファイル→バージョン管理→最新バージョンの取得**を選択します。
2. 確認ダイアログが表示されるので、**OK** を押下します。
3. これでそのオブジェクトの以前にチェックインされたバージョンが取得されます。

プロジェクト全体をロールバックする

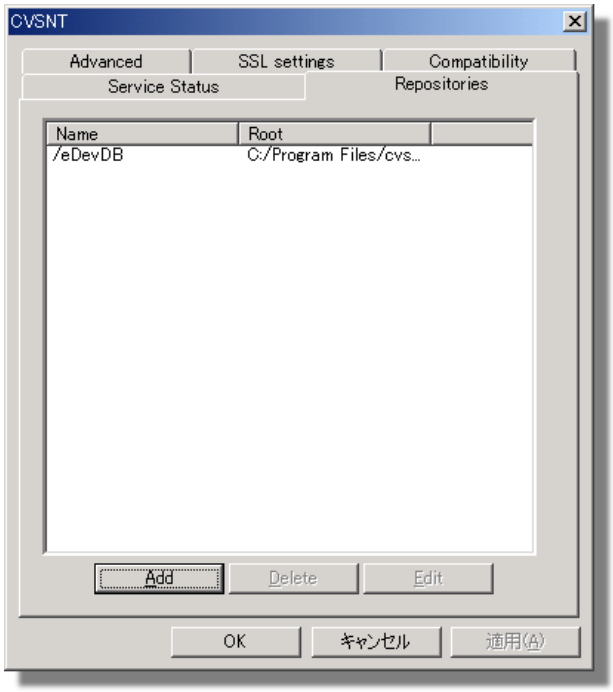
プロジェクト全体をロールバックするには、以下のようにします。

1. **ファイル→バージョン管理→最新プロジェクトの取得**を選択します。
2. 差し変わる各オブジェクトに対する確認ダイアログが表示されます。
3. 処理が終了すると、差し変えられたすべてのオブジェクトのリストが表示されます。

CVS を使用して開発するには

CVS は、**Magic Studio** にバンドルされたオープンソースのソース管理ツールです。CVS に対して精通する上で、いくつか知っておく必要がある重要事項があります。

CVS コントロールパネル



最初に、CVS には設定用に使用されるコントロールパネルがあります。**スタートメニュー→コントロールパネル（またはスタートメニュー）→ CVSNT → Service Control Panel** を選択することでアクセスできます。

ここから、CVS の起動や停止、インストール環境の変更を行うことができます。内容を変更した場合、CVS を一旦止めて再起動する必要があります。

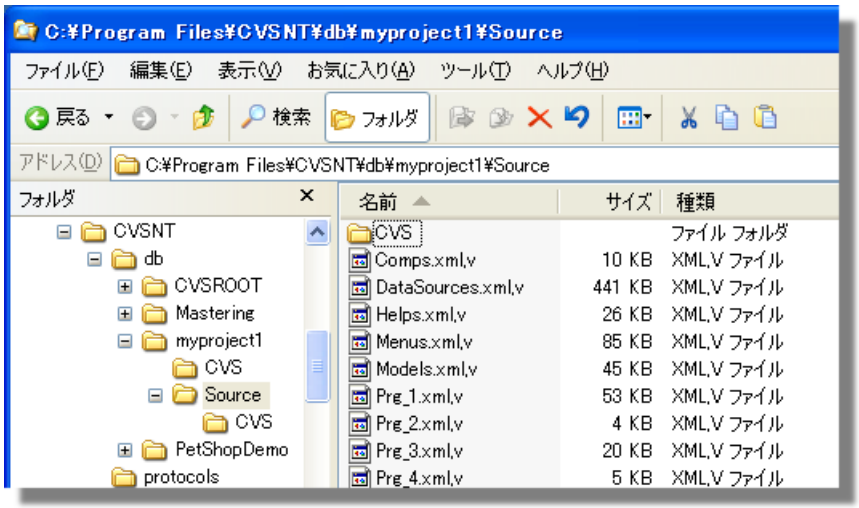
ここには、リポジトリの設定を行うこともできます。eDevDB は Magic がデフォルトで使用するリポジトリ名で、Magic のインストール処理の中で CVS をインストールするとこのリポジトリ名が自動的に登録されます。

CVSNT は、リポジトリ名の前に「\」を指定する必要があります。

右側には **Root** カラムがあります。これは、CVS のデータが実際に格納される場所で、有効なファイルシステムの位置を指定する必要があります。ここでは C:¥Program Files¥cvsnt¥db 内にデータが格納されることになります。

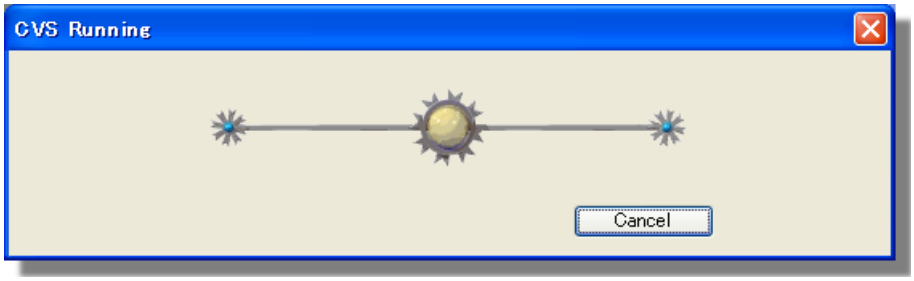
注：　コントロールパネルの図は、Magic にバンドルされている Ver2.0.58 のものです。バージョンによっては、表示内容が異なる場合があります。

ファイルの位置



プロジェクトが CVS でチェックインされると、プロジェクトの XML ファイルが CVS 内に登録されます。ここでは、CVSNT/db の下の myproject1¥Soruce 内に XML ファイルが登録されます。これらのファイルは読み込み専用になっています。これらのファイルは、元の myproject1¥Soruce ディレクトリ内にあるものと同じ XML ファイルに、CVS ユーザが追跡するためのデータが追加されています。

CVS の実行



CVS が実行している場合、上に表示されているような独自のウィンドウが表示されます。処理内容によっては、完了にかなりの時間がかかることがあります。

Magic eDeveloper V10 マスタリング eDeveloper

Magic Software Japan K.K.

第五版 2009 年 1 月 23 日
発行 〒 151-0053 東京都渋谷区代々木三丁目二十五番地三号
あいおい損保新宿ビル 14 階

Magic Software Japan K.K.