イベントドリブン アーキテクチャ Magic xpa

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

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

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

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

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

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

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

Magic は Magic Software Japan K.K. 登録商標です。
Magic xpa は Magic Software Enterprises Ltd. のイスラエルその他の国での商標または登録商標です。
Magic xpa Enterprise Studio、Magic xpa Enterprise Client、Magic xpa Enterprise Server および Magic xpa RIA Server は Magic Software Japan K.K. の商標です。
IBM®, iSeries™, xSeries®, DB2® および WebSphere® は、IBM Corporation の商標または登録商標です。
Microsoft® は、Microsoft Corporation の登録商標です。また、Windows™ は Microsoft Corporation 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。

2020 年 9 月 18 日

1 概要

2 イベント、トリガ、ハンドラ

3 イベントハンドラ

4 イベント実行処理コマンド

5 イベントのポーリング（読み出し）

第 1 章 概要

Magic xpa はイベント駆動型のエンジンです。

Magic xpa はイベントドリブンアーキテクチャを使用してビジネスロジックを定義することができます。イベントドリブンアーキテクチャのパラダイムによるプログラミングでは、より明確なロジックに基づいたアプリケーションを構築することができます。

イベントハンドリングアーキテクチャ

イベントドリブンアーキテクチャによるアプリケーションでは、エンドユーザからのイベントやビジネスロジックの一部として開発者が設計したイベントに対して、応答した処理を行わせることができます。

より明確なコード

イベントドリブンアーキテクチャを使用することにより、ビジネスロジックが、イベントに対して実行されるべき各タスクに整理されて定義されます。これにより、タスクはより明確に、より理解しやすくなり、他の開発者によるメンテナンスも容易になります。

応答型のアプリケーション

イベントドリブンアーキテクチャを使用すると、ユーザやコンポーネントの実行中に発生したイベントに応答して処理を行うことができます。

コードの再利用性

イベントドリブンアーキテクチャを使用すると、ビジネスロジックをグローバルに定義することができます。つまり、一箇所で定義すれば、アプリケーション全体で再利用することができます。このようにすることでロジックを書く時間とメンテナンスする時間を短縮することができます。

第 2 章 イベント、トリガ、ハンドラ

イベント ドリブンアーキテクチャを構成する要素は、イベント、トリガ、ハンドラです。

イベントドリブンアーキテクチャには 3 つの基本的な要素があります。イベント、トリガ、ハンドラです。この章ではこれらの 3 つの要素について説明します。

用語

イベント、トリガ、ハンドラの定義

イベント

イベントとは、実行中のアプリケーションにおいて何かが発生したことを示す、抽象的表現です。実行エンジンはイベントを無視するか、それに対して呼応するか選択することができます。例えば、コントロール上でのマウスクリック操作は、実行エンジンに対してクリックというイベントを発生させます。実行エンジンは、このイベントに対してしかるべき応答を行います。

トリガ

全てのイベントはアプリケーションの実行中に発行されます。いくつかのイベントは、キーの押下やマウスクリックなどの外部動作への応答として発行されます。また、いくつかのイベントは一定時間経過等、アプリケーションが特定の段階に達した時に発行されます。このようにイベントを発行させるための外部動作をトリガと言います。

ハンドラ

ハンドラは、イベントが発生した時に処理が行われるロジックの単位です。ハンドラは特定のイベントを取り扱うように定義されます。

イベントのタイプ

Magic xpa では、システム、内部、タイマー、式、エラー、ユーザのイベントタイプをサポートしています。

システムイベント

システムイベントはキーボード操作によってトリガされます。システムイベントは通常、キーの組み合わせです。

例えば F5 キー押下、Ctrl+R キー押下などは、システムイベントです。

内部イベント

ほとんどの内部イベントは Magic エンジンが扱う内部アクションに対応します。例えば、［次行］イベントがトリガされると、Magic xpa はこのイベントをハンドルし、現在行を出て次行へ入るために必要なロジックが実行されます。

いくつかの内部イベントはアプリケーションインターフェースの操作によってトリガされます。例えば、［マウスオーバー］イベントはエンドユーザがマウスをコントロール上に移動した際にトリガされます。

ノート
Magic xpa ではキーボード割付けファイルを使用して、キーの組み合わせを Magic xpa の内部イベントに割付けられます。これは、システムイベントを内部イベントのトリガとして設定していることになります。

タイマーイベント

タイマーイベントは指定時間毎にトリガされるイベントです。例えば、10 分おきにトリガされるタイマーイベントを定義することが可能です。

> **ノート**
>
> タイマーイベントは設定した時間が経過すると必ずトリガされます。これは以前のバージョンの Magic xpa と動作が異なります。以前はキーボード休止秒数以内に設定時間が経過した時のみタイマーイベントがトリガされていました。

式イベント

式イベントは Magic xpa の式で定義されるイベントです。このイベントは定義された式が True（真）と評価されると直ちにトリガされます。この式の評価は、［オプション／動作環境］の［動作設定］タブの［キーボード休止秒数］で定義された時間間隔で行われます。

エラーイベント

エラーイベントは、データベース関連のエラーが発生した時にトリガされます。例えば、［重複インデックス］エラーは Magic xpa が重複したインデックスをもつレコードを登録しようとした際にトリガされます。

ユーザイベント

Magic xpa のタスクの一部として、独自のイベントをユーザイベントとして定義することができます。このユーザイベントを使用すると、タスクの機能を明確に説明することができます。

.NET イベント

.NET 型項目や .NET クラスを定義することで .NET イベントを選択することができます。.NET 項目に対する［イベント］ロジックユニットを定義した場合、選択されたデータ項目に対するロジックユニットとなります。,NET クラスに対するロジックユニットを定義した場合、タスク内のオブジェクトの任意のインスタンスに対する汎用的なロジックユニットとなります。s

第 3 章 イベントハンドラ

実行時に発生したイベントを扱うロジックユニットを定義することができます

イベントのロジックユニットは、発生したイベントに対する実際の応答処理を定義したものです。この章ではハンドラの定義方法や Magic xpa がどのようにハンドラを実行するのか、そして色々なハンドラの設定について説明します。

イベントハンドラを定義する

［イベント］ロジックユニットは、すべてのタスクで定義することができます。他のロジックユニットと同じようにタスクのロジックエディタで定義されます。新しいハンドラを作成する場合、ロジックエディタ内で新しいヘッダ行を作成し、ユニットのタイプで「E= イベント」を選択します。次のカラムに移動すると、イベントの詳細を設定するためのダイアログボックスが表示されます。

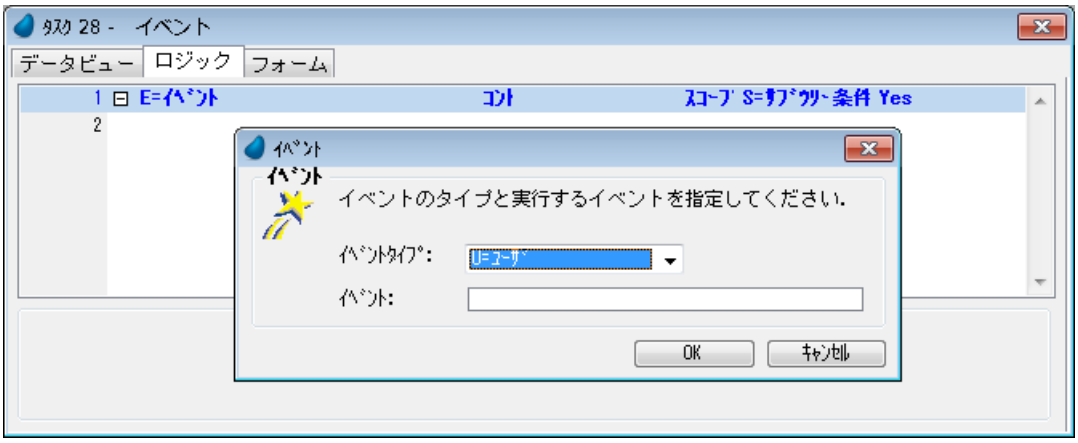

図 3-1［イベント］ロジックユニットで［イベント］ダイアログを表示する

イベントタイプはハンドラの特性において最も重要な設定です。イベントタイプを定義せずにハンドラを定義してはいけません。つまり、ハンドラは全てのイベントタイプに対するグローバルハンドラになることはできません。［イベントタイプ］欄からズームしてイベントタイプを選択してください。

ハンドラに対するイベントを選択すると、ハンドラはそのイベントがトリガされた時にのみ実行されるようになります。

ノート

ハンドラロジックユニットには他の特性があり、それらはデフォルト値を持っていたり必ずしも設定の必要のないものがあります。これらについては、本ドキュメントにて後述致します。

イベントハンドラのロジック

ハンドラでの処理コマンドは、詳細行に記述します。図 3-2 は、選択されたイベントがトリガされた時に実行されるハンドラのロジック定義の例です。

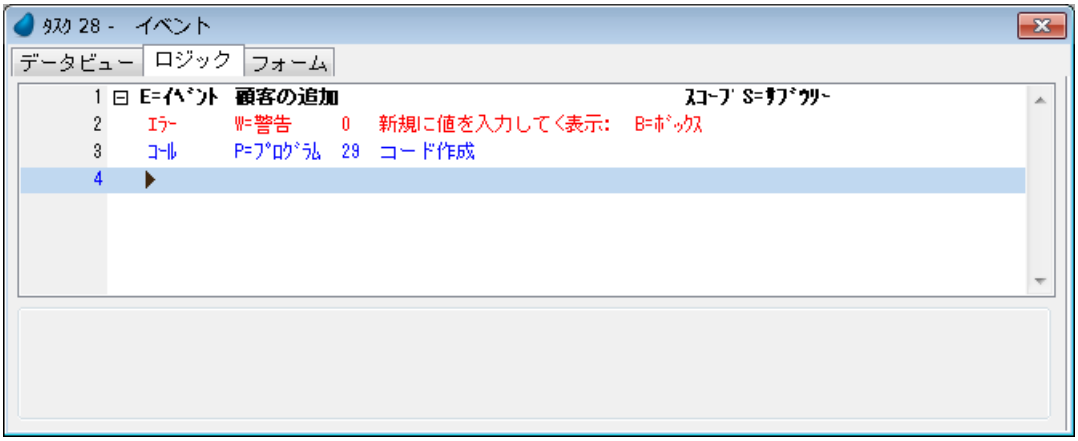

図 3-2 顧客の追加イベントのロジックユニットに定義された処理コマンド

ノート

ハンドラの詳細行に何も処理が定義されていなくてもハンドラは有効ですが、通常は実行したい処理コマンドをハンドラに定義します。

ハンドラの実行

イベントと一緒に定義されているハンドラは有効なハンドラです。対応するイベントがトリガされた時、Magic エンジンはハンドラ内に記述されたロジックを実行します。

ハンドラの検索

イベントがトリガされた時、Magic xpa はそのイベントに呼応するハンドラを、実行タスクツリー※1 の最も下のレベルからメインプログラムへ向かって上向きに検索します。Magic xpa は該当するハンドラを発見すると、そこに定義されている処理を順に行います。ハンドラの詳細行の最後まで処理すると、Magic xpa はタスクツリーの上方へ該当するハンドラがないか検索を続行します。

内部イベントを処理している場合、検索がメインプログラムに到達すると Magic エンジンは該当するイベントに対応する内部ハンドラを検索します。

同一タスク内に定義された、同じイベントに対する複数のハンドラは、タスク定義の記述位置により実行順序が制御されます。タスク定義の最も下方に定義されたものから実行され、上方へ向かう順序でハンドラが実行されます。

※1 : 実行ツリーは、同じコンテキスト内で動作しているすべてのタスクのコレクションです。実行ツリー内のタスクの順序は、それらがオープンされた順序になります。

ハンドラの階層

前のセクションでは同一イベントに対するハンドラがどのように実行されるかを説明いたしました。これらのハンドラはタスクツリーの一部であるいくつものタスク内に定義することが可能です。それらはタスクツリーの階層に従って実行されます。タスクツリーの最下層のタスクに定義されたハンドラが最初に実行され、メインプログラムで定義されたハンドラや Magic エンジンの内部イベントハンドラが最後に実行されます。

階層的なハンドラ構造を使用して、高レベルのハンドラを複数の異なるタスクに対して定義することが可能です。例えば、プログラム A 内で定義されたハンドラは、プログラム A からコールされた全てのプログラムと子タスクで有効となります。

ノート

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プログラムであり、常にタスクツリーの頂点に位置します。従って、メインプログラム内に定義されたハンドラは、プロジェクト全体で有効なハンドラとなります。

例:アプリケーションの何処にいてもエンドユーザが現在の日付を　Ctrl+D を押すことによってチェックできるようにしたいとします。メインプログラムで Ctrl+D のシステムイベントに対するハンドラを定義し、ハンドラ内に現在日付を表示するように適切なロジックを記述します。

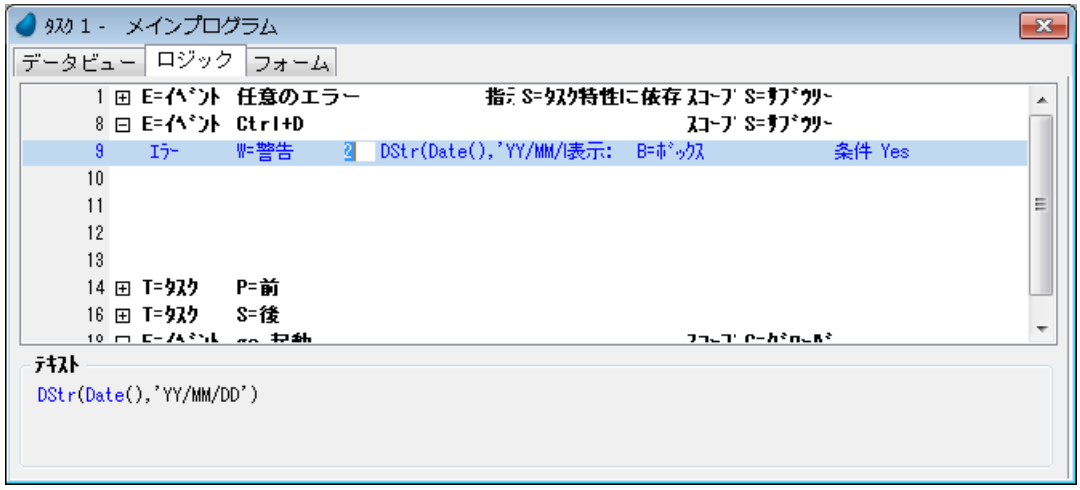

図 3-3 メインプログラムで定義されたグローバルハンドラ

ハンドラの階層システムによって、メインプログラムで作成したロジックはプロジェクト全体で再利用することが可能です。さらにハンドラへの変更は一ヶ所で行われるため、メンテナンスも容易になります。

ハンドラのスコープ

ハンドラを定義する際にタスクツリーの下方全体で有効にするか、現在のタスクでのみ有効にするかを定義することができます。これはロジックユニットの［スコープ］特性で設定します。

［スコープ］特性を「T= タスク」に設定すると、Magic xpa はそのタスク内でトリガされたイベントにのみ反応し、ハンドラを実行します。イベントが下位のタスクでトリガされた場合、Magic xpa はこのハンドラの実行をスキップします。

［スコープ］特性を「S= サブツリー」に設定すると、Magic xpa はそのタスク内、または下位のタスク内でトリガされたイベントに反応し、ハンドラを実行します。

グローバルハンドラ：

メインプログラムにて定義されたハンドラのスコープ特性には、「G= グローバル」というもう一つの設定が可能です。この設定はコンポーネントとして Magic プロジェクトを使用する際に関連するものです。

伝播

トリガされたイベントに対して、一つのハンドラが実行された時、Magic エンジンにそれ以降のハンドラ検索をさせたくない場合があります。これは、ロジックユニットの［伝播］特性を設定することで制御できます。

［伝播］特性を 「No」に設定すると、Magic エンジンは現在のハンドラの実行終了後、更なるハンドラの検索は行ないません。特性を 「Yes」にすると、現在のハンドラ処理の終了後、タスクツリーを上方に向かって該当するハンドラの検索を続行します。

［伝播］特性は式を使用して設定することも可能です。この式は現在のハンドラの実行終了時に評価され、伝播特性が決定されます。

ヒント：

Magic xpa が内部イベントをハンドリングしないように、内部イベントの伝播を止めるためにハンドラを作成することができます。例えば、［削除］オプションを無効にし、ユーザがレコード削除を行おうとした時にメッセージを表示したい場合は、レコード削除の内部イベントに対してハンドラを作成し、ハンドラ内でメッセージを表示するロジックを記述します。そして［伝播］特性を「No」に設定します。

ハンドラの有効 / 無効

ハンドラの［有効］特性を使用することで、実行中にハンドラの有効／無効を切り替えることができます。［有効］特性を「Yes」に設定すると、ハンドラはアクティブになり、ハンドラが必要な時に実行されます。特性が「No」の場合、Magic xpa はハンドラを無視し、実行しません。

［有効］特性が式で設定されている場合、Magic エンジンがハンドラに到達した時に式が評価されます。

ノート：

ロジックユニットの［有効］特性に設定できる条件には、ロジックユニット内に定義された項目を使用することができません。

コントロールに特定したハンドラ

ハンドラを特定のコントロールから発生したトリガにのみ応答するように条件を設定することが可能です。

コントロール一覧からコントロールを選択することにより、ハンドラを特定のコントロール用に制限することができます。ハンドラエントリの詳細の項目内でズームし、［コントロール一覧］を開いて選択します。コントロール名を直接入力することもできます。

同一タスク内にあるコントロールに対してのみ、ハンドラの限定を行うことが可能です。

ハンドラをコントロールに限定することはできますが、項目に対して限定することはできません。項目に対してハンドラを限定したい場合は、項目をフォームに配置してコントロールを作成した後、そのコントロールを使用します。コントロールがフォームから削除された場合や、フォームが再作成された場合、ハンドラのコントロールに対する参照関係が切れてしまう場合があります。

ノート：

同一タスク内で同一イベントに対するハンドラが複数存在する時、Magic エンジンは最初にコントロールに特定されたハンドラを検索し、実行します。そのハンドラが実行された後、Magic xpa はコントロールに限定されていないハンドラを対象に、同じタスク内のタスクツリー下方からもう一度検索を行います。

［フォーム］エディタからコントロール固有のハンドラにアクセスする

特定のコントロールに対する［イベント］ロジックユニットを作成することができます。既に定義されている［イベント］ロジックユニットに対して、［フォーム］エディタ上に定義されているコントロールを割り当てていることで可能になります。

コントロールからコンテキストメニューを開くと、すでに定義されたハンドラの名前が表示されます。これらのメニューオプションの 1 つを選択すると自動的に［ロジック］エディタ内に定義されている該当するロジックユニットに表示が切り替わります。

「ヘッダ作成 :XXXX」というメニューを選択すると［ロジック］エディタに表示が切り替わり、このコントロールが設定された［イベント］ロジックユニットが自動的に作成されます。

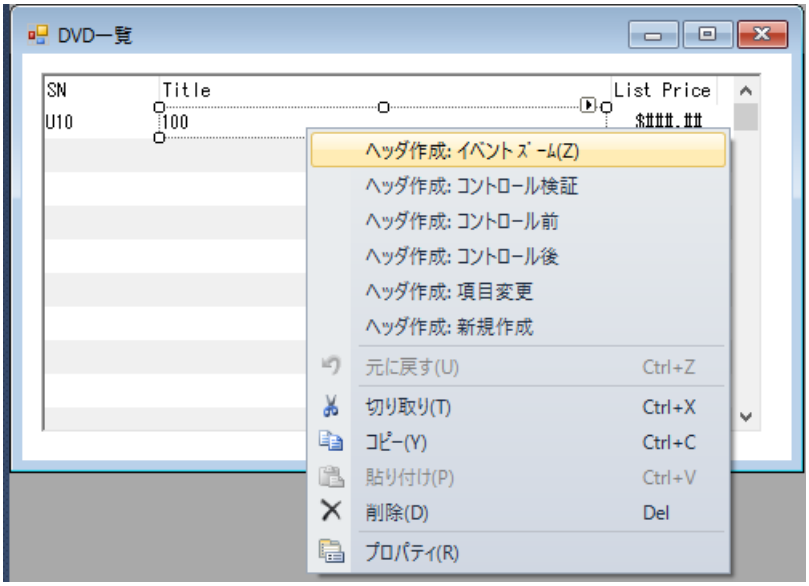

図 3-4「タイトル」エディットコントロールのコンテキストメニューには、このコントロールに対してすでに定義されたハンドラや、「ヘッダ作成：XXXX」というオプションメニューが表示されています。

第 4 章 イベント実行処理コマンド

イベント 実行処理コマンドを利用 することで、明示的にイベント を実行させることができます。

イベント実行処理コマンドを使用すると、アプリケーション内の様々な場所からイベントをトリガすることが可能です。この章では以下の手順について説明します。［イベント実行］処理コマンドの特性と意義、使用方法について説明します。

一般的な使用方法

一般的に、［イベント実行］処理コマンドは、開発者によるハンドラ実行のリクエストと言えます。ユーザ定義のハンドラや Magic xpa 組込みの内部ハンドラを起動することができます。

Magic xpa の内部ハンドラの実行

内部イベントに対する［イベント実行］処理コマンドは、対応する Magic xpa の内部ハンドラを実行させる場合があります。例えば、［次レコード］の内部イベントを実行すると、このイベントに対する内部ハンドラが実行され、現在のタスクは次のレコードに移動しようとします。このような方法を使用して、組込みハンドラを明示的に実行し、タスクを希望の状態に移行させることが可能です。

ユーザ定義ハンドラの実行

一組の処理コマンド群をタスクやプロジェクト内の異なる場所で実行したい場合、各位置にそれらのコマンドを繰返し記述する必要はありません。この一連の処理をユーザ定義イベントに対応させたユーザ定義ハンドラに記述し、希望の位置からこのイベントを実行します。このようにして定義済みロジックの再利用を行うことができます。

ロジックの明確化

実行したい処理コマンドが一箇所からしか実行されない場合でも、それらの処理コマンドはハンドラに記述し、イベント実行処理コマンドによって実行されることが望ましいと考えられます。これにより、一連の処理を分離し、プロジェクトをより論理的に整理し、他の開発者から理解されやすいようにすることができます。

イベント実行処理コマンド

［イベント実行］処理コマンドは、シンプルな処理コマンドです。以下のセクションでは［イベント実行］処理コマンドの特性について説明します。

内容

［イベント］特性は、唯一の必須な特性です。ここには、実行させたいイベントを定義します。ここからズームして［イベント］ダイアログを開きイベントを定義します。

このダイアログボックスはハンドラのダイアログボックスと似ていますが、選択できるイベントタイプが異なります。S= システム、I= 内部、U= ユーザ、P= 公開イベントの 4 つが選択できます。これ以外の、T= タイマー、E= 式、R= エラーのイベントタイプは明示的に実行するものではないため、ここでは選択することはできません。

ウェイト

［ウェイト］特性は［イベント実行］処理コマンドが同期モード（Yes）で実行されるか、非同期モード（No）で実行されるかを指定します。

同期の［イベント実行］処理コマンドは、該当するハンドラに記述されている処理コマンドを即座に実行します。これは Magic エンジンが［イベント実行］処理コマンドに続く処理を実行する前に行われます。

非同期の［イベント実行］コマンドは、ハンドラの処理を即座には行いません。実行されたイベントは Magic エンジンが管理するイベントキューに入力されます。対応するハンドラはイベントがキューから読み出された時に実行されます。Magic エンジンは指定されたタイミングでイベントキューからイベントを読み出します。次の章ではイベント読み出しのメカニズムについて説明します。

重要：

内部イベントは同期的にイベント実行を行うことはできません。内部イベントを実行する場合は［ウェイト］特性を「No」に設定するように注意してください。特性を「Yes」に設定した場合、このイベントに対する内部ハンドラは実行されません。

パラメータ

［イベント実行］処理コマンドは該当するハンドラに対してパラメータを渡すことが可能です。

パラメータのリストは、［イベント実行］処理コマンドの［パラメータ］欄からズームして設定します。

ハンドラに渡されたパラメータは、ハンドラ内で項目処理コマンドによって定義されたパラメータ項目で受け取ることができます。パラメータはハンドラ内に定義された順番に渡されます。従って、最初のパラメータは最初のパラメータ項目に設定されます。

対応するハンドラにパラメータ項目が定義されていない場合、ハンドラはパラメータを受け取ることはできません。

ユーザイベントの章では、ロジックをより簡単に管理するためにどのようにユーザイベントパラメータを定義するかを学びます。

参照渡しのパラメータ

［ウェイト］特性が「Yes」に設定された［イベント実行］処理コマンドでハンドラに渡されたパラメータは、ハンドラの対応するパラメータ項目が変更されると同時に更新されます。パラメータが式で渡された場合は値渡しとなり、呼び出したタスクへ更新された値を戻すことはできません。

値渡しのパラメータ

［ウェイト］特性が「No」に設定された［イベント実行］処理コマンドでハンドラに渡されたパラメータは、ハンドラ内で対応するパラメータ項目に変更があっても、呼び出したタスクへは更新されません。

伝播先のハンドラに渡されるパラメータ

イベントを［伝播］特性が「Yes」に設定されたハンドラに渡されたパラメータは、次の該当するハンドラにも渡されます。パラメータが参照渡しの場合（パラメータがパラメータ項目で渡され、同期モードの［イベント実行］処理コマンドから実行された場合）、最初のハンドラがパラメータを更新すると、次のハンドラは更新された値を受け取ることになります。パラメータが値渡しの場合（式で指定された場合や非同期モードの［イベント実行］処理コマンドの場合）、次のハンドラは常に［イベント実行］処理コマンドによって渡された元の値が受け取られることになります。

ノート：

最初のハンドラがパラメータを受け取るためのパラメータ項目を定義していない場合でも、次のハンドラはパラメータを受け取ることが可能です。次のハンドラでのみパラメータを使用するような場合、最初のハンドラにパラメータ項目を定義する必要はありません。

メニューやプッシュボタンからのイベント実行

［イベント実行］処理コマンドは、［ロジック］エディタ以外にも定義することができます。プッシュボタンとメニューのエントリに、それらが選択された時に実行するイベントを定義することが可能です。

プッシュボタン

Magic xpa の［プッシュボタン］コントロールには［実行イベント］特性があります。この特性からズームして、プッシュボタン押下時に実行したいイベントを選択します。定義されたイベントは、プログラムの実行時にボタンが押された時に実行されます。設定可能なイベントとしては、S= システム、I= 内部、U= ユーザ、N= なし があります。

重要：

エンドユーザがプッシュボタンをクリックすることで定義された一連のロジックが実行されるようにしたい場合、以下のような方法で定義してください。

- ユーザイベントを作成します。
- ［ユーザイベント］ロジックユニットにロジックを作成します。
- ［プッシュボタン］コントロールの［実行イベント］特性に、ユーザイベントを定義します。

プッシュボタンの操作を処理するために内部イベントの「クリック」の使用は控えてください。

メニューリポジトリ

［メニュー］リポジトリでは、イベントを実行するためのメニューエントリを定義することが可能です。メニューのエントリタイプをイベントに設定することで、イベント実行用の設定を行うことができます。パラメータの項目からズームしてイベントダイアログを開き、実行するイベントを選択します。選択できるイベントタイプは、S= システム、I= 内部、U= ユーザです。

ノート：

［プッシュボタン］コントロールやメニューエントリから実行されたイベントは、非同期モードで実行されます。

第 5 章 イベントのポーリング（読み出し）

実行エンジンは、いつイベントを処理するのでしょうか？

イベントに対応するハンドラは、イベントがトリガされた時に全てが直ちに実行される必要はありません。ハンドラが実際に実行されるタイミングは、イベントのタイプとそれがトリガされた状況に依存します。

ハンドラの即時実行

いくつかのケースでは、ハンドラはイベントがトリガされると直ちに実行されます。

同期のイベント実行処理コマンド

［イベント実行］処理コマンドで実行した場合、そのハンドラは直ちに実行されます。

エラーイベント

ハンドラがエラーイベントに対して定義されている場合、イベントがトリガされると、そのハンドラが直ちに実行されます。

イベントキュー

その他のイベントはイベントがトリガされると、イベントキューに貯えられます。Magic xpa はこのキューをあらかじめ決められた時間間隔でポーリング（読み出し）します。ハンドラは、イベントがキューから読み出された時に実行されます。

非同期のイベント実行処理コマンド

非同期モードの［イベント実行］処理コマンドでトリガされたイベントは、イベントキューに貯えられ、後で実行されます。

システム、内部、タイマー、ユーザイベント

エラーと ActiveX イベント以外の全てのイベントは、トリガされた時点でイベントキューに貯えられます。

式イベント

式イベントもトリガされた時にはイベントキューに貯えられます。ただし、式イベントはその式が True（真）と評価されると直ちにトリガされるわけではありません。式イベントの式はキーボードがアイドル状態の時に一定間隔で評価されます。式は［キーボード休止秒数］の時間が経過した時点で評価され、True(真) と評価された時にイベントがトリガされ、イベントキューに貯えられます。［キーボード休止秒数］は［オプション／動作環境］の［動作設定］タブの［キーボード休止秒数］で設定します。

イベントキューからのイベントのポーリング（読み出し）

Magic エンジンがイベントキューからイベントを読み出すタイミングは、下記の設定に影響されます。

- アイドル時間
- バッチイベント間隔
- レコードイベント間隔
- イベント可

アイドル時間

オンラインタスクとブラウザタスクでは、イベントキューにあるイベントはコントロールにカーソルがパークしているアイドル状態の時に常に読み出されます。タスクがアイドル状態になると直ちに Magic xpa はキューにある全てのイベントを読み出します。

重要：

非同期モードのイベント実行処理コマンドでは、ハンドラは次にタスクがアイドル状態になった時に実行されるので、予定していた場所（タイミング）とは異なる位置でハンドラが実行される場合があります。例えば、［コール］処理コマンドの直前に実行した非同期の「閉じる」内部イベントは、コール先のタスクで実行され、現在のタスク内では実行されません。これはイベントがキューに貯えられた後の最初のアイドル時間が、コール先のタスクで発生するからです。

バッチイベント間隔

バッチタスクにはアイドル時間がありません。従って、バッチタスクを実行する時は、イベントをキューからポーリング（読み出し）する時間間隔を指定します。この時間間隔は設定／動作環境のシステムタブのバッチイベント間隔で設定します。例えば、この値を1000(ミリ秒) に設定すれば、Magicエンジンは1000 ミリ秒おきにイベントをキューから一つずつポーリングします。

この設定がゼロの時は、時間間隔によるイベントのポーリングは行われません。この設定は全バッチタスクで共通の設定です。

レコードイベント間隔

特定件数のレコード処理後にイベントキューからイベントのポーリング（読み出し）を行うようにバッチタスクを設定することができます。［タスク特性］の［レコードイベント間隔］特性の項目で設定します。例えば、この値を 100 に設定した場合、Magic エンジンはメインソースで 100 レコード処理する毎にイベントをポーリングします。設定を 1 にすると、1 レコード毎にイベントがポーリングされます。

この特性の値は式で設定します。式はレコード間隔を表す数値に評価されることが必要です。

ノート：

Magic エンジンはバッチイベント間隔とレコードイベント間隔の両方でイベントのポーリングを行います。これらの条件のうち、どちらかの条件が成立した時、ポーリングが行われます。

イベント可

イベントのポーリングを完全に無効にして、バッチタスクを中断することなく実行させることができます。そのようにするには、［タスク特性］の［イベント可］特性を「No」に設定します。

第 6 章 ロジックユニット

タスク、グループそしてタスクのコント ロールのサイク ルで処理できます。

タスクの一部としてあらかじめ設定されたタイミングに対応したロジックユニットを作成することができます。タスクが指定されたレベルに達すると、常にこのロジックユニットが自動的に実行されます。

タスクレベル

すべてのタスクは、タスク前とタスク後という 2 つの主要な処理レベルを通過します。

タスク前

タスクがオープンされると、タスク前を通過します。［タスク前］ロジックユニットが定義された場合、この時点で実行されます。［タスク前］ロジックユニットでは、このタスクを実行する上で必要な準備のためのロジックを定義することができます。

重要：

いくつかの内部の初期設定処理（例えば、入出力デバイスのオープン、オープンするファイル名の指定、データソースのオープン、データビューを定義するための範囲設定）は、タスク前が実行される前に実行されます。

これは、タスク前で、入手力ファイルの名前を変更したり、ファイルを削除したりすることができないことを意味しています。このような処理は、タスクを実行する前の段階（親タスク）で実行する必要があります。

タスク後

タスクが終了する際に、タスク後を通過します。［タスク後］ロジックユニットが定義された場合、この時点で実行されます。［タスク後］ロジックユニットでは、実行したタスクの後処理のために必要なすべてのロジックを定義することができます。

重要：

タスク前の前に内部的に初期設定された内容は、［タスク後］ロジックユニットの実行の後にクローズされます。タスク前と同様に、タスク後で入出力ファイル名を変更したりファイルを削除することはできません。

レコードレベル

エンドユーザがタスクのレコードを参照する場合、Magic エンジンはレコード前とレコード後のロジックユニットを実行しています。

レコード前

レコードがパークされる直前に、［レコード前］ロジックユニットが実行されます。レコード前は、エンドユーザがレコードに入る前に、必要とされる準備処理を実行します。例えば、潜在的な不一致をエンドユーザに通知するために新規の入力レコード上で何らかの計算処理を実行させることができます。

レコード後

レコードを更新した後、そのレコードを抜けることで［レコード後］ロジックユニットが実行されます。レコードの修正処理を完了するために必要なロジックを実行させるためにレコード後を使用することができます。

レコードが更新されたかどうかにかかわらずレコード後の実行を行わせたい場合は、、［タスク特性］の［強制レコード後］特性を「Yes」に設定してください。

グループレベル

バッチタスクを実行する場合、処理されたレコードをグループ分けすることができます。グループはデータビューエディタで定義された項目を使用して定義することができます。定義された項目が同じ値を持っているレコードはすべて同じグループとしてみなされます。

この項目の値が変わると、あるグループから別のグループに移動する場合となり、ループ後とグループ前の各ロジックユニットが実行されます。

実行時期

グループ前

グループ項目がその値を変更したり、最初のグループ処理が実行される前に、グループ前が実行されます。新しいグループのための準備処理用の処理コマンドを実行させるために使用することができます。例えば、グループレベルでの集計値を加算するための項目を初期化するような場合に使用できます。

グループ後

グループ項目が変更した場合、つまりグループの最後となるレコードが抜け別のグループ用のレコードに入った時点でグループ前が実行されます。グループでの集計結果をフッタとして出力するための処理コマンドを実行するためにグループ後を使用することができます。

ノート：

- **インデックスセグメントの順番は、［グループ］ロジックユニットの順番と一致させる必要があります。インデックスのより上位のセグメントは、より高位の［グループ］ロジックユニットにする必要があります。**
- **［グループ後］は、［グループ前］の後に定義してください。それらの間に新しい［グループ］ロジックユニットを定義しないでください。**

コントロールレベル

エンドユーザがオンラインタスク上でレコード上にパークし、コントロールの周辺でカーソルを動かす場合、エンドユーザの様々な操作を受け取って処理することができます。コントロールのサイクルには、3 つのタイプ（コントロール前、コントロール後、コントロール検証）の［コントロール］ロジックユニットとして定義することができます。

コントロール前

コントロール前は Tab の移動方向に関係なく、コントロールに入る直前に実行されます。このハンドラ内のいかなる処理コマンドも、タスクがコントロールにパークする前に実行されます。

コントロール後

コントロール後は Tab の移動方向に関係なく、コントロールから出る直後に実行されます。このハンドラ内のいかなる処理コマンドも、タスクの終了時、またはコントロールから出た後で次のコントロールにパークする前に実行されます。

ノート：

コントロール後が実行される直前に、Magic エンジンはコントロールの値を新しく入力された値に更新し、この値に関連する再計算処理を行います。

コントロール検証

［コントロール検証］ロジックユニットは以下の状況の場合に実行されます：

- 検証対象のコントロール、またはそれよりも前にあるコントロールから、他のレコードにスキップする際に実行されます。この場合、レコードに変更がある場合にのみ実行されます。
- ユーザがコントロールから抜ける場合に、コントロール後の直前に実行されます。

- ユーザが検証対象のコントロールを超えてスキップした場合に実行されます。これは同一レコード内で該当するコントロールより前にあるコントロールから後にあるコントロールへ、あるいはその逆にスキップが行われた時に実行されます。
- 検証対象のハンドラが設定されたコントロールより後ろにある他のレコードのコントロールへスキップした場合、新しいレコードの該当コントロールに対してロジックユニットが実行されます。
- ［コントロール検証］ロジックユニットは、コントロールを抜ける場合、新しく入力されたデータが更新され、関連するデータ項目とそれに依存する式の再計算が一度行われた後に、コントロール後の直前に実行されます。

［コントロール検証］ロジックユニットは、以下の状況で Web Client タスクで実行されます。

- Magic xpa は、［レコード後］を実行する直前に、現在のタスク内のすべての［コントロール検証］ロジックユニットを実行します。［コントロール検証］ロジックユニットは、タスク内で定義された順に実行されます。（サポートバージョン：4.6.1）

ノート：

［コントロール検証］ロジックユニット内に定義されたエラー処理コマンドで、モードを E= エラーに設定すると、処理コマンドは中断され、カーソルは検証しているコントロールに再びパークします。

コントロール検証 vs. コントロール後

コントロール検証とコントロール後の主な相違は、コントロール後がユーザがそのコントロールにパークし、抜ける場合にのみ実行されることに対して、コントロール検証はユーザがコントロールを抜ける場合と、他のコントロールやレコード間移動でコントロールを通過した場合にも実行されるということです。

コントロール後ロジックユニットを使用する

コントロール後は、ユーザが目的のコントロールにパークして、そこから抜ける場合のみ実行すべき処理を定義します。

コントロール検証ロジックユニットを使用する

コントロール検証は、タスクがコントロールを通過する際に必要な処理を実行する場合に使用します。

重要：

コントロールレベルのハンドラはコントロールを特定したものである必要があります。つまり、コントロールレベルのロジックユニットに対してはコントロール名を必ず選択する必要があります。

第 7 章 ユーザイベント

Magic xpa では独自のイベントを作成することができます。

Magic xpa では定義済みのものを含め、様々なタイプのイベントをサポートしています。これらのイベントは Magic エンジンによって自動的に処理されるものもありますが、適切なロジックユニットを作成することによって処理することもできます。この章ではユーザイベントについて、その定義方法と特性を説明します。

ユーザイベントを作成する

ロジックユニットで利用する、Magic xpa のタスクに対して有効なイベントを定義することができます。タスクイベントは Ctrl+U（またはタスク環境／ユーザイベント）で表示されるユーザイベントテーブルにて定義します。イベントには、名前、タイプ、トリガ、強制終了の 4 つの特性を定義します。

名前

イベントの名前を定義します。このカラムは必ずしも設定が必要な項目ではなく、名前の無いイベントを定義することもできますが、いくつものユーザイベントから明確に区別できる分かり易い名前を付けることを推奨致します。

トリガタイプ

ユーザイベントを自動的にトリガさせるイベントを定義することができます。タイプとしては、S= システム、I= 内部、T= タイマー、E= 式が指定できます。ユーザイベントのトリガを指定しない時はトリガタイプを「N= なし」に設定します。

ノート：

タイプが「N= なし」の場合、イベント実行処理コマンドがユーザイベントをトリガする唯一の方法となります。

トリガ

タイプに応じたトリガの値を設定します。

パラメータ

イベントを実行させる場合、イベント実行処理コマンドとともにパラメータを渡すことができます。渡されるパラメータの値は、対応するロジックユニットに定義されたパラメータ項目で受け取ることができます。

パラメータを実行させるロジックユニットに渡す必要がある場合、ユーザイベントにあらかじめ必要なパラメータの名前と書式を定義しておいてください。

パラメータ項目

パラメータカラムには、定義されたパラメータの数が表示されます。このカラムからズームすると図 7-1 のようなパラメータテーブルが開き、任意の数の必要なパラメータを定義することができます。各パラメータに対して、名前や、モデル、または型や書式を設定してください。

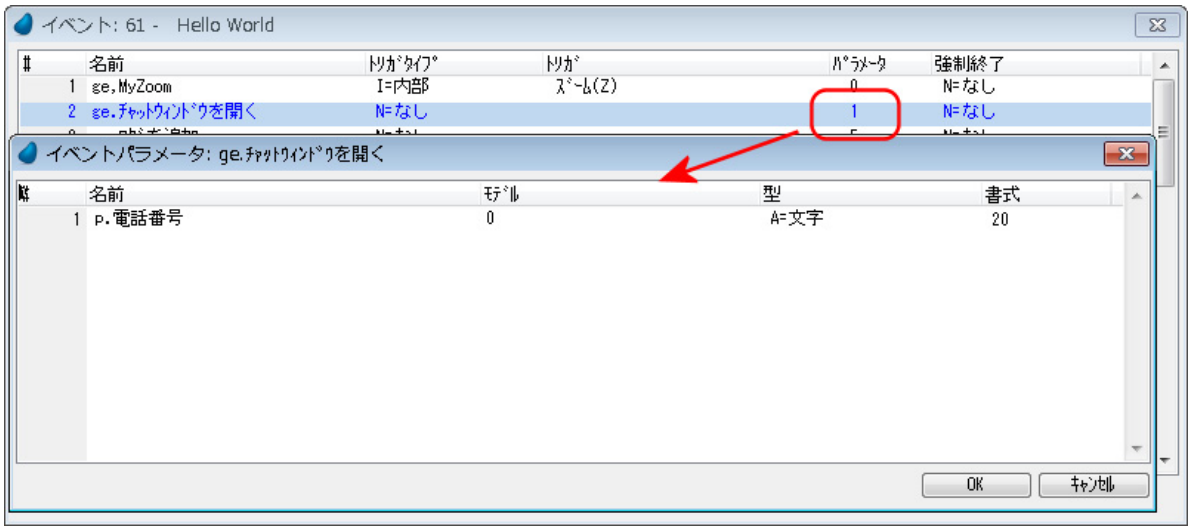

図 7-1 パラメータテーブル

パラメータの照合

パラメータが定義されたユーザイベントを実行すると、［イベント実行］処理コマンドの［パラメーター一覧］ダイアログに適切なパラメータや対応する型と書式が表示されます。

パラメータ項目の自動作成

パラメータが定義されたユーザイベントを選択して新しいロジックユニットを作成すると、ロジックユニットにパラメータ項目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。

強制終了

この特性は、エンジンがイベントを実行する前に、抜け出るタスクレベルを指定します。

N= なし

ユーザイベントの［強制終了］特性が「N= なし」に設定されている場合は、ロジックユニットを実行する前にいかなるタスクレベルからも抜け出ることはありません。ロジックユニットはタスク、レコード、コントロールなどのどのレベルからでも実行することが可能です。イベントは［イベント実行］処理コマンドで［ウェイト］特性を「Yes」に設定してトリガすることで同期的に実行されます。

イベントが非同期的にトリガされている場合は、次のアイドル時間にロジックユニットが実行されます。オンラインタスクとブラウザタスクでは、コントロールにパークしている時（［エディット］コントロールでの編集中を含む）がアイドル時間になります。

E= 編集

ユーザイベントの［強制終了］特性が「E= 編集」に設定されている場合は、イベントを処理する前にコントロールの編集モードを終了します。これは、［イベント］ロジックユニットが実行する場合に、コントロールに入力された値によってコントロール項目内容が更新されることを意味しています。イベントのすべての処理が終了した後で、タスクはこのコントロールの編集モードに戻ります。

［強制終了］特性のこのレベルは、現在のコントロールに入力された値をロジックユニット内で参照する必要がある場合に使用します。例えば、あるコントロールに対するズームイベントを処理し、ズームの［イベント］ロジックユニットから起動されたタスク内で、コントロールに入力された値を参照したい場合、ズームの内部イベントをトリガとするユーザイベントを定義し、［強制終了］特性を「E= 編集」に設定したものを使用します。

ノート：

［強制終了］特性が「N= なし」に設定されたイベントを処理するロジックユニットによって現在の項目を更新した場合、入力された値はコントロールから Tab で抜けた場合に表示時が切り替わります。これは、コントロールが最後に編集した時の値のまま編集モードにとどまっているためです。しかし、［強制終了］特性が「E= 編集」の場合、エンジンが編集モードを終了させ、再度編集モードでコントロールに入るため、更新された値で表示されるようになります。

C= コントロール

ユーザイベントの［強制終了］特性が「C= コントロール」に設定されている場合、ロジックユニットを実行する前に、現在パークしているコントロールレベルから抜けることが要求されます。その後ロジックユニットが実行されます。ロジックユニットの実行が終了すると、タスクは同一コントロールに戻ります。

これは、現在のコントロールに対応するすべてのコントロールレベルを実行することを意味しています。これは、コントロールを一端抜けるため、コントロール検証とコントロール後が実行され、再度コントロールに入るためコントロール前が実行されるからです。

R= レコード

ユーザイベントの［強制終了］特性が「R= レコード」に設定されている場合は、ロジックユニットを実行する前に、現在のレコードレベルから抜けることが要求されます。ユーザイベントによって、レコード後が実行されます（強制レコード後特性＝ Yes の場合または、レコードの値に変更がある場合）。ユーザイベントのロジックユニットはレコードの更新と共に、アクティブなトランザクション中に実行されます。一連の処理終了後、タスクは同一レコードに戻り、その際、レコード前が実行されます。

つまり、ロジックユニットはレコード後の後で、データベースへの更新の直前に実行されることになります。ロジックユニットの処理終了後、データベースのレコードに対する更新が実行されます。

ノート：

現在のトランザクション中で、イベントのハンドリング終了後直ちにレコードを更新したい場合、強制終了を「R= レコード」に設定します。

ノート：

強制終了をレコードに設定した場合、エンジンがレコードを出てトランザクション中でレコードを更新する際、ロジックユニット内で実行した［項目更新］処理コマンドも実行されます。

R= レコード更新前

ユーザイベントの［強制終了］特性が「R= レコード更新前」に設定されている場合、イベントが実行された直後に現在のレコードが更新されます。

ノート：

イベントの強制終了を「R= レコード更新前」に設定した場合、Magic エンジンがレコード処理を終了し、トランザクション内のレコードを更新した時に、このイベントによって実行されるロジックユニット内で実行される［項目更新］処理コマンドによってまた更新されることになります。

P= レコード更新後

ユーザイベントの［強制終了］特性が「P= レコード更新後」に設定されている場合、イベントを処理する前にレコードの更新処理を終了するという点では「レコード更新前」と同じです。しかし、Magic エンジンは最初に修正されたレコードを更新し、更新済みのレコードに入り直し、その際［レコード前］ロジックユニットを実行します。その後、そのイベントに対応したロジックユニットを実行することになります。

［イベント］ロジックユニットが、現在のレコードを参照する別のタスクを呼び出すように定義され、そのタスクを呼び出す前にレコードを更新する必要がある場合、［強制終了］特性を「P= レコード更新後」に設定します。

この方法で呼び出されたタスクは、新しく更新されたレコードを参照することができます。

ノート：

イベントの強制終了を「P= レコード更新後」に設定した場合、現在のレコード上で実行されたロジックユニット内で実行されるあらゆる［項目更新］処理コマンドがレコード更新後に実行され、Magic エンジンが再度レコードを抜けた後に、エンドユーザによって暗黙的または明示的にデータベース内に反映されます。

同期のイベント実行処理コマンドを使用する

［強制終了］特性が「E= 編集」、または「C= コントロール」、「P= レコード更新前」、「U= レコード更新後」に設定されているユーザイベントは同期的に実行することはできません。これは、実行エンジンが、処理コマンドの実行セット内で要求されたレベルを終了するための一連の処理を実行することができないためです。

同期モードの［イベント実行］処理コマンドでトリガされたユーザイベントは、［強制終了］特性が「N= なし」以外に設定されていても、「N= なし」のユーザイベントとして扱われます。

第 8 章 実行環境リソースとタスク情報

様々な実行環境と情報がイベント レベルのロジック ユニット で有効になります。

［イベント］ロジックユニットは、実行アプリケーション内の様々なリソースやタスク情報を使用／参照することができます。この章では［イベント］ロジックユニットを使用して、実行環境のリソースやタスク情報にアクセスする方法について説明します。

実行環境の情報とリソース

実行環境では、各タスクが使用できるリソースとタスク情報には特定の範囲が定義されています。実行環境リソースとはデータ項目と入出力デバイスを指します。また実行環境タスク情報とはタスクツリー中でアクティブなタスクのデータ項目や定義済みの入出力デバイスを含む全ての情報を指します。タスクはリソースとタスク情報（開発環境では必ずしも必要ではありませんが）を参照し、使用することができます。

項目

位置記号や名前を使用して、アクティブなタスクツリー中の全てのデータ項目を参照することができます。アプリケーション内の全ての有効なタスクの項目を参照／操作するために様々な関数が用意されています。項目に関連する関数には次のようなものがあります：

VarAttr, VarDbName VarCurr, VarCurrN, VarIndex, VarMod, VarName, VarPic, VarPrev, and VarSet.

図 8-1 に示されているような、縦列型のプログラムコールでは、各プログラムが独立した項目（下図の各 Var A、Var B）を持っていますが、実行環境においてプログラム A とプログラム B の項目は、プログラム C からアクセスすることが可能です。項目を参照するにはタスクツリーの順序または項目名を使用します。

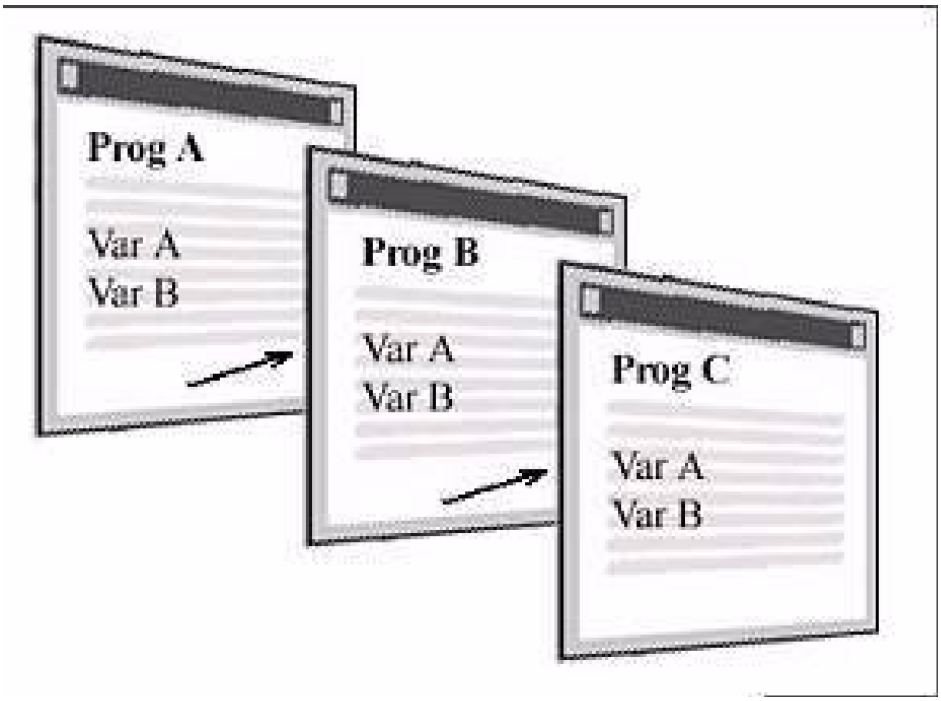

図 8-1 この例では、プログラム A がプログラム B を呼び出し、それがさらにプログラム C を呼び出す縦列型のプログラムコールを示しています。

（入出力）デバイス

開発モードでは、互いに参照することができない他のプログラム中に定義された入出力デバイスも、実行環境中で使用することが可能です。実行時にアクティブなタスクツリー内のプログラムの入出力であれば、入出力に使用することができます。

アクティブな上位タスクで定義した入出力デバイスを使用して入出力を行うには、現在のタスクの入出力ファイルの［使用する入出力名］特性を設定します。この特性には、上位タスクによって既にオープンされている入出力名を文字列で指定します。詳細については、『リファレンスヘルプ』のプログラムのセクションの「使用する入出力名」のトピックを参照してください。

実行タスクツリー

プログラムが他のプログラムや子タスクを呼び出すと、実行タスクツリーが構成されます。このツリーは分岐の無い単一ツリーで、メインプログラムが最上位に位置し、呼び出した順序でつながっており、現在実行中のタスクが一番下に位置します。様々な関数を使用して、現在実行中のタスクからツリーの上位に位置するタスクを参照することができます。これらの関数ではタスクの世代番号を指定することでタスクツリー中の特定のタスクを指定することができます。タスクの世代番号とは、タスクツリー中で現在（最下位）のタスクを０、一つ上位のタスクが１、と上位に向かって１ずつ順に増えていく番号です。

図 8-1 では、プログラム C の世代が０、プログラム B が１、プログラム A が２という世代になります。

次の関数は世代を指定して、タスク情報を参照することができます：

CHeight, CLeft, CLeftMDI, Counter, CTop, CTopMDI, CurRow, CurrPosition, CWidth, DbCache, EOF, EOP, LastPark,Level, Line, Page, Rollback, Stat, VarInp, ViewMod, WinBox, WinHWND.

イベントロジックユニットのスコープ

上位のタスクで定義されたロジックユニットを、下位のタスクで実行されることができます。実行タスクツリーを考える場合、上位レベルで実行されたロジックユニットでも、それはイベントをトリガしたタスクで実行されたものとして扱われます。従って、ロジック自体は上位タスクで定義 / 実行されたものであっても、すべてのタスクツリー内のリソースをロジックユニットで利用できることを意味しています。

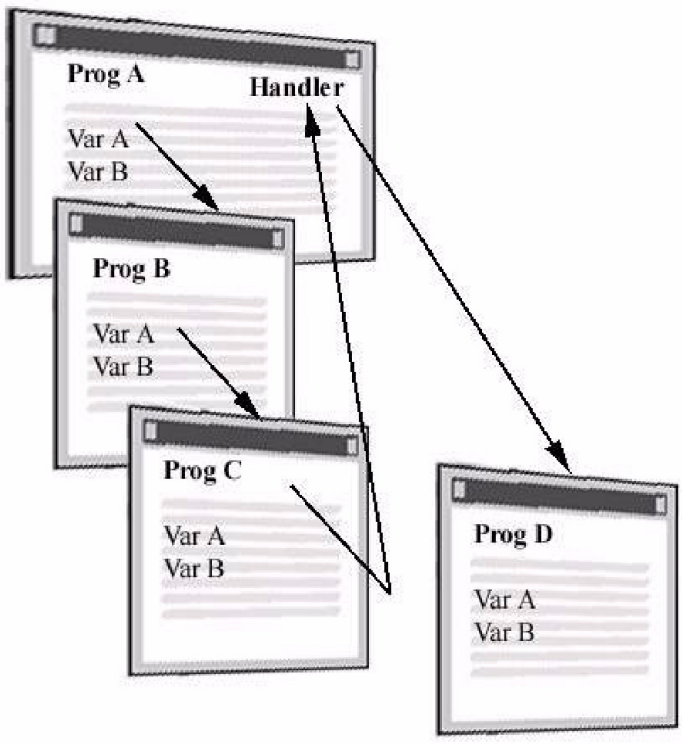

図 8-2 この例では、A → B → C と呼び出し、C でイベントがトリガされています。

図 8-2 では、プログラム A がプログラム B を呼び出し、それがさらにプログラム C を呼び出しています。プログラム C ではイベントを実行し、プログラム A に定義されたロジックユニットを実行します。このロジックユニット内でプログラム D を呼び出しています。

プログラム D から見た実行タスクツリーは、プログラム D がプログラム C から直接コールされた場合と全く同じ扱いになります。

項目

プログラム D はタスクツリー中の全ての項目を使用することができます。

実行タスクツリー

この場合のタスクの世代番号は次のようになります：プログラム D = 0、プログラム C = 1、プログラム B = 2、プログラム A = 3

入出力ファイル

プログラム D は、使用する入出力名特性を使用することにより、全ての他の上位タスクによってオープンされた入出力デバイスに対して入出力を行うことができます。

ノート：

ロジックユニットが呼び出した「プログラム D」からではなく、ロジックユニット内から直接タスクツリーを参照する場合、世代番号はロジックユニットがあたかも「プログラム C」から実行されているように指定されます。

例えば、ロジックユニットから直接参照できる世代番号は、「プログラム C」が 0、「プログラム B」が 1、「プログラム A」が 2 となります。

並行プログラムとモードレスなブラウザタスク：

並行プログラムやブラウザタスクを呼び出す際に、モーダルでないウィンドウを使用して呼び出した場合、コール処理コマンドを実行したプログラムを先頭にコールされたプログラムまでの新しいタスクツリーの階層を作成します。

トリガが発生したオブジェクトを参照する

ロジックユニットやロジックユニットから呼び出されたタスクはタスクツリー全体を参照できますが、トリガが発生したオブジェクトを参照することが必要な場合もあります。イベントがトリガされたオブジェクトとは項目やコントロール、またはタスクを表します。トリガが発生したオブジェクトに対する参照機能は、トリガが発生したオブジェクトを正確に処理する必要のあるグローバルハンドラにおいて必要な機能です。

This() 関数

This() 関数はトリガ発生オブジェクトを参照する関数です。このオブジェクトには、トリガ発生タスクやトリガ発生項目があります。この関数は項目に関連した関数で、項目のインデックス（シンボル名）の代わりに使用するか、世代関連関数では世代番号の代わりに使用することができます。

This() 関数はパラメータを持ちません。また This()+1 のような複雑な式には使用できません。

トリガ発生項目

トリガ発生項目とは、イベントがトリガされた時にタスクがパークしていた項目を表します。トリガ発生項目には、項目関連関数にて項目インデックスに This() 関数を使用することによって参照することができます。例えば、トリガ発生項目の型を取得するには次のように記述します：VarAttr(This())

トリガ発生タスク

トリガ発生タスクとは、イベントがトリガされたタスクを表します。トリガ発生タスクには、世代関連関数にて世代番号に This() 関数を使用することによって参照することができます。例えば、トリガ発生タスクのレベルを取得するには次のように記述します：Level(This())

処理されるコントロール

イベントがトリガされた時に処理されていたコントロールを参照することも可能です。これは通常、クリックやダブルクリック、マウスオーバー、マウスアウト、などのマウスに関連したイベントで使用されます。処理されていたコントロールは HandledCtrl() 関数を使用すると、コントロール名を取得することができます。この関数はパラメータを必要としません。

第 9 章 マルチマーキングレコードを処理する

バッチタスクのようにマルチマークされたレコード上でトリガされたイベントを処理することができます。

Magic xpa の［テーブル］コントロールでは、エンドユーザがテーブル中に表示されているレコードをマルチマーキングすることをサポートしています。アプリケーションで選択された行に対して、自動的に定義されたロジックを実行するように設計することができます。例えば、エンドユーザが複数の請求書レコードを選択し、各請求書の合計を計算した結果を表示するような［イベント］ロジックユニットを実行させることができます。

この章ではマルチマーキングレコードの処理について説明します。マルチマーキングとレコード選択方法の詳細については、『リファレンスヘルプ』の表示フォームのトピックを参照してください。

マルチマーキングレコードを処理する

［テーブル］コントロールにおいて複数レコードを選択している状態でイベントがトリガされた場合、イベントに対応するロジックユニットが選択中の各レコードに対して実行されます。選択されているレコードは一つずつ順に自動的に処理されます。

上から下へ

選択されたレコードは、タスク中で表示されている順序で上から下へと処理が行なわれます。レコードが選択された順序とは関係ありません。

完全なレコードサイクル

各レコードが処理される前に、エンジンは各レコードに入り、レコード前を実行し、その後ロジックユニットを実行します。レコードに変更があった場合や強制レコード後特性が「Yes」に設定されている場合は、レコード後が実行されます。Magic エンジンはその後、次の選択レコードに移動し、同様の処理を行います。処理は全ての選択レコードが処理されるまで繰り返されます。

現在パーク中のレコード

タスクは選択しているレコードの先頭、末尾、あるいは途中のレコード以外にもパークすることが可能です。

Magic エンジンが複数レコードに対する処理を行う前に、エンジンは現在パーク中のレコードを抜けます。従って Magic エンジンはレコードが更新されている場合は、コントロール検証とレコード後を実行します。

複数レコードの処理が終了すると、タスクは選択行の末尾のレコードにパークした状態で留まります。

ノート：

複数レコードの処理が終了すると、Magic エンジンは選択された最後のレコードにパークします。従って、選択行の最後にタスクが到達すると、レコード前、イベントに対応したロジックユニット、レコード後が実行された後、そのレコードにパークするために再びレコード前が実行されます。

マルチマーキングレコードに対する内部処理

マルチマーキングレコードに対して処理できる内部イベントは、「削除」イベントのみです。複数レコード選択後、削除イベントを実行することにより、選択中の全レコードの削除を行うことができます。

その他の内部イベントが実行された場合は、Magic エンジンは選択状態を取消した後、内部イベントに対応したロジックユニットを実行します。

マルチマーキングレコード処理中に他のイベントをトリガする

マルチマーキングレコードの処理中にトリガされた別のイベントを処理することができます。ただし、全てのイベントが処理されるわけではありません。複数レコード処理中の場合、Magic エンジンは同期的（ウェイト =Yes）にトリガされたイベントのみを処理します。複数レコード処理中にトリガされた非同期イベントは全て無視されます。

マルチマーキングをサポートする関数

Magic xpa では、複数選択レコード処理の制御を向上するためのいくつかの関数が用意されています。

選択されたレコード数

現在選択中のレコードの件数を照会するには MMCount() 関数を使用します。この関数はタスクの世代番号を指定することで、上位タスクでで選択されているレコード数を参照することも可能です。

ヒント：

レコードが選択されていない場合、この関数は 0 を返します。この機能を利用して、レコードが選択されているかどうかの判定を行なうことができます。

現在までに処理されたレコード 数

選択されたレコードのうち、幾つのレコードが処理されたかを参照することができます。MMCurr() 関数は選択されたレコードのうち、現在までに処理されたレコード数を返します。この関数はタスクの世代番号を指定し、上位タスクの選択レコード中の処理されたレコード数を参照することも可能です。

処理の最初と最後

複数選択レコードに対して実行されるロジックユニットは全ての選択レコードに対して同じロジックユニットが実行されます。しかし、ロジックユニットが最初に実行される時にだけ必要な処理もあります。例えば、変数の初期化や処理開始の確認ダイアログの表示などです。一方、処理結果の表示などロジックユニットの最後でのみ必要な処理もあります。

このような場合、MMCurr() 関数と MMCount() 関数を使用することで、ロジックユニットの処理段階を特定することができます:

- MMCurr() 関数が 1 の値を返す場合、複数レコード処理の最初の段階と判断できます。
- 選択レコードの総数と処理ずみレコード数が等しい場合、つまり MMCurr() 関数＝ MMCount() 関数となる場合は、複数レコード処理の最後の段階と判断できます。

マルチマーキング情報の保持

マルチマーキングレコードに対して処理を行った後、処理されたレコードは選択されたままの状態になっています。これにより、エンドユーザはマルチマーキングレコードに対して複数のタスク処理を行うことができます。

レコードの選択を解除するには MMClear() 関数を使用します。この関数が評価されると、複数レコードに対する処理終了後に選択が解除されます。

処理の中止

マルチマーキングレコードに対する自動処理を停止するには MMStop() 関数を使用します。この関数を使用すると、未処理の選択レコードに対するロジックユニットの実行を停止することができます。

プロセス実行中に MMStop() 関数が評価された場合、現在実行中のレコードに対する処理は完全に行なわれます。

このレコードの処理が終了すると、タスクはそのレコードにパークし、その後の処理を停止します。MMStop 関数を使用して処理を中止した時でも、レコードの選択状態は保持されます。

ノート：

MMClear() 関数と MMStop() 関数は、共に現在のタスクでのみ有効です。これらの関数を上位のタスクで実行することはできません。