

MAGIC SOFTWARE JAPAN K.K.

---

Magic eDeveloper

Magic  
**eBusiness**  
**Platform**<sup>TM</sup> v9

イベントドリブン

アーキテクチャ

## 著作権表示と免責事項

The information in this document is subject to change without prior notice and does not represent a commitment on the part of MSE(Magic Software Enterprises Ltd.) and MSJ(Magic Software Japan K.K.).

MSE makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose.

The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms and conditions of the license agreement. It is against the law to copy the software on any medium except as specifically allowed in the license agreement.

No part of this document and/or databases may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or information recording and retrieval systems, for any purpose other than the purchaser's personal use, without the prior express written permission of MSE.

All references made to third party trademarks are for informational purposes only regarding compatibility with the products of Magic Software Enterprises Ltd.

Unless otherwise noted, all names of companies, products, street addresses, and persons contained herein are part of a completely fictitious scenario or scenarios and are designed solely to document the use of Magic.

Magic® is a registered trademark of Magic Software Enterprises Ltd.

PC/TCP® Network Software is a registered trademark of FTP Software Inc.

Microsoft® and FrontPage® are registered trademarks, and Windows™, WindowsNT™ and ActiveX™ are trademarks of Microsoft Corp.

Macromedia® Dreamweaver® is a registered trademark of Macromedia, Inc.

VeriSign® is a registered trademark of VeriSign, Inc.

Clip art images copyright by Presentation Task Force, a registered trademark of New Vision Technologies Inc.

All other product names are trademarks or registered trademarks of their respective holders.

Copyright 2002 by Magic Software Enterprises Ltd.and Magic Software Japan K.K. All rights reserved.

2002年5月22日

# 目次

<b>概要</b> .....	<b>6</b>
イベントハンドリングアーキテクチャ.....	6
より明確なコード.....	6
応答型のアプリケーション.....	6
コードの再利用性.....	6
イベントハンドリング vs. レコードメインのロジック.....	7
レコードメイン - データビューの定義のみ.....	7
レコードメインのロジックのサポート.....	7
<b>イベント、トリガ、ハンドラ</b> .....	<b>8</b>
用語.....	8
イベント、トリガ、ハンドラの定義:.....	8
イベントのタイプ.....	9
<b>ハンドラ</b> .....	<b>10</b>
イベントハンドラを定義する.....	10
イベントハンドラのロジック.....	11
ハンドラの実行.....	11
ハンドラの検索.....	11
ハンドラの階層.....	11
ハンドラスコープ.....	12
伝播.....	13
ハンドラの有効/無効.....	13
コントロールに特定したハンドラ.....	13
<b>イベント実行コマンド</b> .....	<b>15</b>
一般的な使用方法.....	15
Magicの内部ハンドラの実行.....	15
ユーザ定義ハンドラの実行.....	15
ロジックの明確化.....	15
イベント実行コマンド.....	16
内容.....	16
ウェイト.....	16
パラメータ.....	16
参照渡しのパラメータ.....	17
値渡しのパラメータ.....	17
伝播先のハンドラに渡されるパラメータ.....	17
メニューとプッシュボタンからのイベント実行.....	17

ブッシュボタン.....	17
メニューリポジトリ.....	17
<b>イベントのポーリング(読み出し).....</b>	<b>19</b>
ハンドラの即時実行.....	19
同期のイベント実行コマンド.....	19
エラーイベント.....	19
イベントキュー.....	19
非同期のイベント実行コマンド.....	20
システム、内部、タイマー、ユーザイベント.....	20
式イベント.....	20
イベントキューからのイベントのポーリング(読み出し).....	20
アイドル時間.....	20
バッチイベント間隔.....	20
レコードイベント間隔.....	21
イベント可.....	21
<b>コントロールレベル・ハンドラ.....</b>	<b>22</b>
コントロールレベル.....	22
コントロール前処理.....	22
コントロール後処理.....	22
コントロール検証.....	23
コントロール検証 vs. コントロール後処理.....	23
コントロール後処理ハンドラを使用する.....	23
コントロール検証ハンドラを使用する.....	23
<b>ユーザイベント.....</b>	<b>25</b>
ユーザイベントを作成する.....	25
名前.....	25
タイプ.....	25
トリガ.....	25
強制終了.....	26
同期のイベント実行コマンドを使用する.....	27
<b>ハンドラが使用できる 実行環境リソースとタスク情報.....</b>	<b>28</b>
実行環境の情報とリソース.....	28
変数.....	28
I/O(入出力)デバイス.....	29
実行タスクツリー.....	29
ハンドラスコープ.....	30
変数.....	31
実行タスクツリー.....	31
入出力ファイル.....	31
トリガが発生したオブジェクトを参照する.....	31
THIS() 関数.....	31
トリガ発生変数.....	32
トリガ発生タスク.....	32
ハンドラコントロール.....	32

<b>マルチマーキングレコードをハンドリングする</b> .....	<b>33</b>
マルチマーキングレコードをハンドリングする .....	33
上から下へ .....	33
完全なレコードサイクル .....	33
現在パーク中のレコード .....	34
マルチマーキングレコードに対する内部ハンドリング .....	34
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 .....	34
マルチマーキングをサポートする関数 .....	35
選択されているレコード数 .....	35
現在までに処理されたレコード数 .....	35
処理の最初と最後 .....	35
マルチマーキング情報の保持 .....	35
処理の中止 .....	35
<b>あとがき</b> .....	<b>37</b>

## 概要

*Magic eDeveloper* ではイベント駆動型のエンジンが導入されました。

**M**agic eDeveloper 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。

## イベントハンドリング vs. レコードメインのロジック

Magicの以前のバージョンでは、タスクのインタラクティブなロジックはタスクのレコードメインレベルに定義されていました。この方法でのロジック定義は、冗長な処理コマンドのリストをレコードメインレベルに作成することになります。さらに、セレクトコマンドとリンクコマンドによるタスクデータビューの定義が、タスクロジックと並んで定義されていました。

### レコードメイン - データビューの定義のみ

Magic eDeveloperのイベントハンドリングアーキテクチャを使用して、レコードメインはデータビューの定義にのみ使用してください。レコードメインはセレクトコマンドとリンクコマンドのみが含まれることになります。タスクのロジックはタスクのインタラクティブな局面でハンドルされるべきセグメントごとに分離して定義します。

### レコードメインのロジックのサポート

Magic eDeveloperはレコードメイン内に定義されたロジックについても引き続きサポートしますが、レコードメインにロジックを記述する方法はできるだけ避けるようにし、イベントハンドリング機構を使用するようにしてください。これにより明確で、的確に応答するアプリケーションの構築を実現することができます。

**重要：**

レコードメイン中のロジック定義はオンラインタスクでのみサポートされており、ブラウザタスクではサポートされていません。ブラウザタスクではイベントハンドリング機構を使用したロジック定義のみが可能です。

## 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 eDeveloper 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の内部アクションに割付けられます。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、設定/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、重複キーエラーはMagic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。

## ハンドラ

実行環境下で発生したイベントを扱うハンドラを定義する

ハンドラは発生したイベントに対する実際の応答です。この章ではハンドラが定義される手順、Magicがどのようにハンドラを実行するのか、そしてハンドラの各種設定について説明します。

### イベントハンドラを定義する



ハンドラはタスク定義の画面で定義することができます。新しいハンドラを作成する時は、タスク定義に新しい行を作成します。次の画面はハンドラのエントリを示しています。

#	イベント	イベント	詳細	スコープ	伝播	有効	処理
1	T=奴り	P=前処理					0
2	T=奴り	S=後処理					0
3	R=レコト	P=前処理					0
4	R=レコト	M=メイン					9
5	R=レコト	S=後処理					1
6	H=ハンドラ		コントロール:	S=サブツリ	No	Yes	0

図1：タスク定義中のハンドラのエントリ

イベントタイプはハンドラの特長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：

ハンドラには他の特長があり、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処理テーブルに記述します。次の図は、選択し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。

タスク定義:4 - Inet - 会社マスタ							
#	レベル	イベント	詳細	スコア	伝播	有効	処理
1	T=成功	P=前処理					0
2	T=成功	S=後処理					0
3	R=レコード	P=前処理					0
4	R=レコード	M=メイン					9
5	R=レコード	S=後処理					1
6	H=ハンドラ	Ctrl+0	Ctrl+0:	S=呼び出し	No	Yes	2

処理テーブル: ハンドラ On Ctrl+0							
#	処理コマンド	内容	モード:	W=警告	表示:	B=ボックス	条件
1	Enter	0 終了確認	モード:	W=警告	表示:	B=ボックス	Yes
2	Ctrl	P=呼び出し 車両マスタ選択	モード:	0	フォーム:	0	戻:
						戻:	???

図 2 : クリックタイプのハンドラの処理テーブル

### ノート :

ハンドラの処理テーブル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。ハンドラの処理テーブルの最後まで処理すると、Magicはタスクツリーの上方向へ該当するハンドラがないか検索を続行します。

内部イベントをハンドリングしている場合、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。

### 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

## EVENT DRIVEN ARCHITECTURE

を説明致しました。これらのハンドラはタスクツリーの一部であるいくつか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。

### ヒント：

メインプログラムはアプリケーションの全てのプログラムの親プログラムなので、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、アプリケーション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。

#	レベル	イベント	詳細	スコープ	伝播	有効	処理
1	T=タスク	P=前処理					0
2	T=タスク	S=後処理					0
3	R=ルート	M=メイン					0
4	H=ハンドラ	Ctrl+D	コントロール:	S=サブツリー	No	Yes	1

処理テーブル: ハンドラ On Ctrl+D

#	処理コマンド	内容	モード	表示	条件
1	エラー	1 DSTRT (DATE (), '##/##/##')	モード: W=警告	表示: B=ボック	Yes

図3：メインプログラムで定義されたハンドラ

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アプリケーション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。これはハンドラのスコープの特性で設定します。

スコープの特性をタスクに設定すると、Magic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

がされた場合、Magicはこのハンドラの実行をスキップします。

スコープの特性をサブツリーに設定すると、Magic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、**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アプリケーション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、ハンドラの伝播の特性を設定することで制御できます。

伝播の特性を 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。

伝播の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、**D: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

ハンドラの有効の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。有効の特性を 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はハンドラを無視し、実行しません。

有効の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。

#### 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。

コントロールリスト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。

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、変数に対して限定することはできません。変数に対してハンドラを限定したい場合は、変数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時や、フォームが再作成された時、ハンドラのコントロールに対する参照情報は失われます。

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。

## イベント実行コマンド

*アプリケーション内でイベントトリグンアーキテクチャを使用する場合、イベント実行コマンドを活用*

イベント実行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組込みの内部ハンドラが使用できます。

### Magic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コマンドは、対応するMagic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イベント実行すると、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ています。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アプリケーション内の異なる場所で実行したい時、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イベント実行します。このようにして定義済みロジックの再利用が行えます。

### 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、アプリケーションをより論理的に整理し、他の開発者が理解しやすいようにできます。

## イベント実行コマンド



Magic eDeveloperの他の13の処理コマンドと同様、イベント実行コマンドも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コマンドの特性について説明します。

### 内容

実行するイベントを特定するために必ず設定しなければならない項目です。欄でズームし、イベントダイアログボックスで実行するイベントを選択します。

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良く似ていますが、選択できるイベントタイプはシステム、内部、ユーザの3つです。タイマー、式、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

ウェイトの特性はイベント実行コマンドが同期(ウェイト=Yes)で実行されるか、非同期(ウェイト=No)で実行されるかを指定します。

同期のイベント実行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の特性を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、ハンドラ内の作成されたセレクト（変数）コマンドによって受け取ることができます。パラメータはハンドラ内の変数に定義されている順番に渡されます。従って、最初のパラメータは最初の変数に設定されます。

対応するハンドラに変数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。

### 参照渡しのパラメータ

同期(ウイト=Yes)のイベント実行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変数が変更されると、同時に更新されます。パラメータが式でハンドラに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

非同期(ウイト=No)のイベント実行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変数に変更があっても、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=Yesに設定してある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時(パラメータが変数で、同期のイベント実行コマンドから実行された時)、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ようになります。パラメータが値渡しの時(式で指定した時や非同期のイベント実行コマンドの時)、次のハンドラは常にイベント実行コマンドによって渡された元の値を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変数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用の変数を定義する必要はありません。

## メニューとプッシュボタンからのイベント実行



イベント実行コマンドは、処理テーブル以外にも実行できる場所があります。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実行コマンドを定義することが可能です。

### プッシュボタン

Magicのプッシュボタンコントロールにはイベント実行の特性があります。この特性の項目からズームして、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、実行モードで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、システム、内部、ユーザ、なしが選択できます。

### 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

## EVENT DRIVEN ARCHITECTURE

で、イベント実行を行います。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、イベント実行を行うイベントを選択します。使用できるイベントタイプはシステム、内部、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 (ウェイト=No) に設定された時と同様に、非同期で実行されま  
す。

## イベントのポーリング(読み出し)

*Magic*エンジンがイベントを読み出すメカニズム

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コマンド

イベント実行コマンドで(ウェイト=Yes)にして同期にて実行した場合、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時、使用中のデータベースゲートウェイからエラー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*はこのキューを予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コマンド

非同期のイベント実行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

エラーイベント以外の全てのイベントは、トリガされた時に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設定 / 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：

非同期のイベント実行コマンドでは、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、コールコマンドの直前に実行した非同期の「C:クローズ」の内部イベントは、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。タスク制御のレコードイベント間隔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エンジンはメインテーブルで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の特性をNoに設定します。

## コントロールレベル・ハンドラ

*Magic eDeveloper* ではコントロールレベルと呼ばれる新しいタスクレベルが使用できるようになりました。

Magicのタスクは定義済みのタスク動作用にハンドラが定義されています。以前のバージョンでは、これらの定義済みハンドラには、タスク前、タスク後、レコード前、レコード後、グループ前、グループ後、がありました。Magic eDeveloperではこれに追加して、コントロールレベルと呼ばれるハンドラレベルが追加されました。

### コントロールレベル

各タスクに自動的に定義されるタスクレベルやレコードレベルのハンドラと異なり、コントロールレベルのハンドラは手動で作成する必要があります。コントロールレベルではコントロール前処理、コントロール後処理、コントロール検証の3種類のハンドラが用意されています。

#### コントロール前処理

コントロール前処理はタブの方向に関係なく、コントロールに入る際に実行されます。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。

#### コントロール後処理

コントロール後処理はタブの方向の関係なく、コントロールから出る際に実行されます。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：

コントロール後処理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

コントロール検証ハンドラは以下の状況の時に実行されます：

- コントロール検証ハンドラが設定されたコントロール、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、コントロール検証ハンドラは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出る時に、コントロール検証ハンドラはコントロール後処理ハンドラの直前に実行されます。
- ユーザがコントロール検証ハンドラが設定されたコントロールを超えてスキップをする際に実行されます。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。
- コントロール検証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する場合、新しいレコードにおいて該当するコントロール検証のハンドラが実行されます。

コントロール検証ハンドラは、コントロールを出る際、新しく入力されたデータが更新され、関連する変数とそれに依存する式の再計算が一度行われた後に、コントロール後処理の直前に実行されます。

### ノート：

コントロール検証ハンドラ中に定義したエラーコマンドで、モードをエラーにすると、処理コマンドを中断し、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. コントロール後処理

コントロール検証とコントロール後処理の主な相違は、コントロール後処理がユーザがそのコントロールにパークし、出る時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出る時と、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時にも実行されるということです。

### コントロール後処理ハンドラを使用する

コントロール後処理は、ユーザが目的のコントロールにパークして、そこから出る時のみに実行すべき処理に使用します。

### コントロール検証ハンドラを使用する

コントロール検証ハンドラは、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のに使用します。

### 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ハンドラ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。

**ヒント：**

コントロールレベルハンドラはタスクのフロー方向に関らずに実行されます。FLOW()関数を使用して、ハンドラの処理をタスクフローに応じて切り替えることができます。FLOW()関数についての詳細はMagicリファレンス9章の式の章をご覧ください。

## ユーザイベント

*Magic eDeveloper*では独自のイベントを作成することができます。

Magic eDeveloper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ハンドルされるものもありますが、適切なハンドラを作成することによってハンドルすることができます。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



ハンドラと同じように、Magicのタスクに対して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+K（またはタスク環境 / ユーザイベント）で表示されるイベント一覧にて定義します。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。この欄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、いくつか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。

#### 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、システム、内部、タイマー、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。

**強制終了**

この特性は、エンジンがイベントをハンドルする前に、脱出すべきタスクレベルを指定します。

**強制終了:N=なし**

ユーザイベントの強制終了がN=なしに設定されている時は、ハンドラを実行する前にいかなるタスクレベルからの脱出も要求しません。ハンドラはタスク、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コマンドで (Weight=Yes) で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時は、次のアイドル時間にハンドラが実行されます。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を含む) がアイドル時間です。

**強制終了:C=コントロール**

ユーザイベントの強制終了がC=コントロールに設定されている時は、ハンドラを実行する前に、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。その後ハンドラが実行されます。ハンドラ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。

**ノート:**

エディットコントロールで編集集中にトリガされた非同期イベントはエディットモード中でハンドルされます。エディットコントロールに対応する変数はまだ更新されておらず、入力された値は反映されていません。編集集中に実行されるハンドラには、パーク中の値に入力値が反映されないだけでなく、タスク全体での入力値による再計算もまだ行われていません。

コントロールに対応する変数への入力値の反映が行われ、タスクでその値による再計算が行われた後でハンドラを実行するためには、強制終了をC=コントロールに設定し、コントロールレベルの処理を完全に終了してからハンドラが実行されるようにします。

**強制終了:R=レコード**

ユーザイベントの強制終了がR=レコードに設定されている時は、ハンドラを実行する前に、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、レコード後処理が実行されます (強制レコード後処理 = Yesの場合または、レコードの値に変更がある場合)。ユーザイベントのハンドラ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、レコード前処理が実行されます。

つまり、ハンドラはレコード後処理の後で、データベースへの更新の直前に実行されることとなります。ハンドラの処理終了後、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、ハンドラ内で実行した更新コマンドも実行されます。

同期のイベント実行コマンドを使用する

強制終了の設定がC=コントロール、またはR=レコードに設定されているユーザーイベントは同期的に実行することはできません。

同期 (Wait=Yes) のイベント実行コマンドでトリガされたユーザーイベントは、強制終了がコントロールまたはレコードの場合、この特性が無視され強制終了がN=なしのユーザーイベントとして扱われます。

## ハンドラが使用できる 実行環境リソースとタスク情報

ハンドラは実行環境中の様々なリソースやタスク情報を使用できます。

ハンドラはアプリケーション内で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。この章では実行環境のリソースやタスク情報へのハンドラのアクセス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変数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変数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を参照し、使用することができます。

#### 変数

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変数を参照することができます。アプリケーション中の全てのアクティブタスクの変数を参照 / 操作するために様々な関数が用意されています。変数に関連する関数には次のようなものがあります: VARATTR, VARCURREN, VARINDEX, VARMOD, VARNAME, VARPIC, VARPREV, VARSET

下図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変数（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の変数は、プログラムCからアクセスすることが可能です。変数を参照するにはタスクツリーの順序または変数名を使用します。

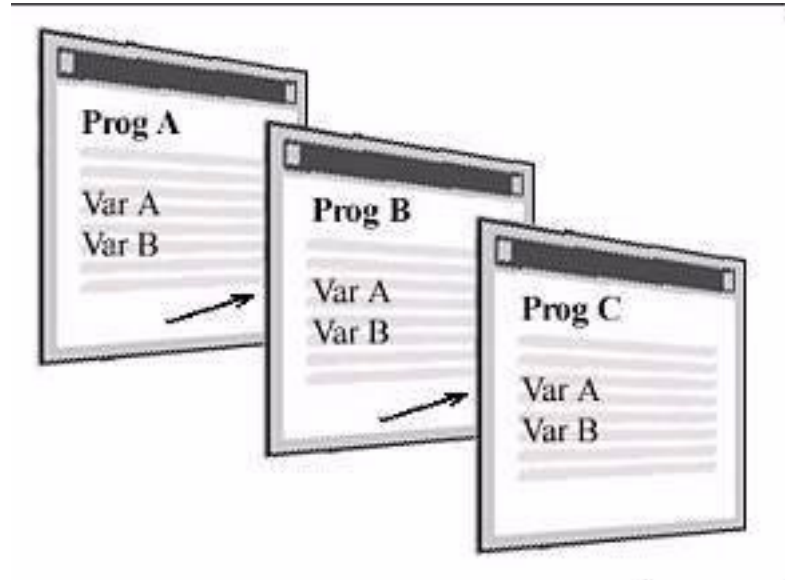


図4：この例では、プログラムAがプログラムBをコールし、それがさらにプログラムCをコールする縦列型のプログラムコールを示しています。

### I/O（入出力）デバイス

開発モードにおいては互いに見ることの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。現在アクティブなタスクツリー中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を使用して入出力を行うには、現在タスクの入出力ファイルの**使用する入出力名**の特性を設定します。この欄で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名前指定します。詳細については、Magicリファレンスガイド7章-プログラムの「使用する入出力名」の項目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コールすると、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トップに位置し、コール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上位の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**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します。タスクの世代とは、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。

上記の図では、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.

### ハンドラスコープ



上位のタスクで定義されたハンドラが、下位のタスクで実行されることは多々あります。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ハンドラでも、それはイベントをトリガしたタスク下で実行されたと考えます。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、全てのタスクツリー中のリソースや情報はハンドラのロジックで使用できます。

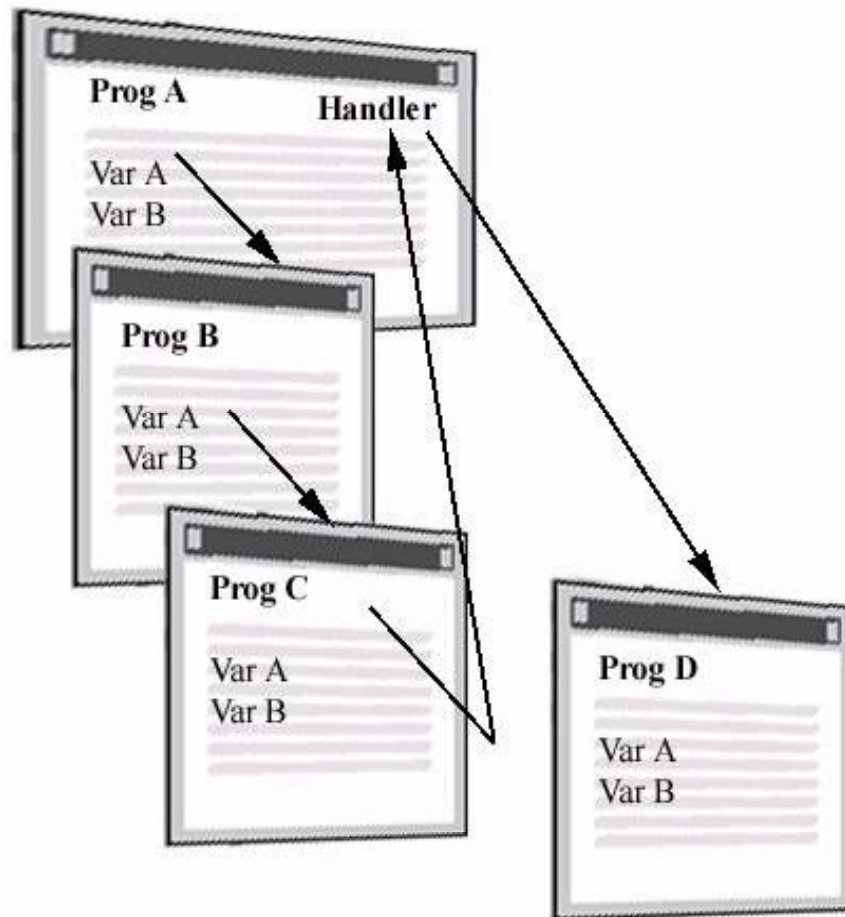


図5：この例では、A B Cとコールし、Cでイベントがトリガされています。

上図では、プログラムAがプログラムBをコールし、それがさらにプログラムCをコールしています。プログラムCではイベント実行を行い、プログラムA中に定義されたハンドラを実行します。このハンドラ中でプログラムDをコールして

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。

### 変数

プログラムDはタスクツリー中の全ての変数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の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:

ハンドラがコールしたプログラムDからではなく、ハンドラ内から直接タスクツリーを参照する場合、世代はハンドラがあたかもプログラムCから実行されている場合の指定となります。例えば、ハンドラから直接参照できる世代は、プログラムC = 0、プログラムB = 1、プログラムA = 2となります。

#### ブラウザクライアント・アプリケーション

ブラウザタスクにおいて、モーダルでないウィンドウを使用してプログラムコールを行った場合、コール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



ハンドラやハンドラからコールしたタスクは全タスクツリーを参照できますが、時には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があります。イベントがトリガされたオブジェクトとは変数、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ハンドル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。トリガ発生オブジェクトはトリガ発生タスク、またはトリガ発生変数です。この関数は変数関連の関数で変数インデックスの代わりに使用するか、世代関連関数で世代の代わりに使用します。

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。

#### トリガ発生変数

トリガ発生変数とは、イベントがトリガされた時にタスクがパークしていた変数を表します。トリガ発生変数には、変数関連関数にて変数インデックスに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、トリガ発生変数の属性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())

#### ハンドラコントロール

イベントがトリガされた時にハンドル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、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。ハンドル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。

## マルチマーキングレコードを ハンドリングする

*Magic eDeveloper*ではマルチマーキングされたレコードでトリガされたイベントをハンドリングすることができます。

*Magic eDeveloper*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ハンドラ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ハンドリング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*Magic*リファレンスガイド11章のフォーム表示の項目を参照してください。

### マルチマーキングレコードをハンドリングする



テーブルコントロールで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時、イベントに対応するハンドラ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ハンドルされます。

#### 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

各レコードがハンドルされる前に、エンジンは各レコードに入り、レコード前処理を実行し、その後ハンドラを実行します。レコードに変更があった場合や強制レコード後処理が設定されていれば、レコード後処理が実行されます。

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処理を実行します。

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ままとなります。

#### 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、レコード前処理、ハンドラ、レコード後処理が実行された後、そのレコードにパークするために再びレコード前処理が実行されます。

#### マルチマーキングレコードに対する内部ハンドリング

マルチマーキングレコードに対してハンドラが有効な内部イベントはD: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時は、Magicエンジンは選択をキャンセルした後、内部イベントハンドラを実行します。

#### 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を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、全てのイベントがハンドル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 (Wait=Yes) にトリガされたイベントのみをハンドルします。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 eDeveloper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を用意してあります。

### 選択されているレコード数

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。この関数はタスク世代を指定し、上位タスク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時、この関数はゼロを返します。この機能を使用して、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の数を返します。この関数はタスク世代を指定し、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ハンドラは全ての選択レコードに対して同じハンドラが実行されます。しかし、ハンドラが最初に実行される時にだけ必要なロジックもあります。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ハンドラ処理の最後でのみ必要なロジックもあります。

このような場合、MMCURR()関数とMMCOUNT()関数を使用することで、ハンドラの処理段階を特定することができます：

- MMCURR()関数が1の値を返す時、複数レコード処理の最初の段階と判断できます。
- 選択レコード総数と処理済みレコード数が等しい時、つまりMMCURR()関数 = MMCOUNT()関数となる時、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MMCLER()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

## EVENT DRIVEN ARCHITECTURE

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ハンドラの実行を停止することができます。

プロセス実行中にMMSTOP()関数が評価された時、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。

## あとがき

*イベント駆動型アプリケーションの開発をはじめる前に*

このドキュメントでMagic eDeveloperのイベントドリブンアーキテクチャについて十分な情報をご提供できましたでしょうか。この技術資料では、いくつかのイベントハンドリング機構の定義を説明し、開発モードでの使用方法と、実行モードでの動作について説明しました。

Magic eDeveloperはイベントハンドリング機構と他の優れた機能を組み合わせ、様々な規模のベストな開発プラットフォームを提供します。