Magic eDeveloper V10
レコードメインから
イベントへ

Magic Software Japan K.K.

Enabling Business with Superior Technology

初版　2007 年 10 月 12 日

マジックソフトウェア・ジャパン株式会社

目次

第1章 はじめに

1.1 レコードメインからイベント指向プログラムへ

Magic eDeveloper V10 では、イベント指向でプログラム作成を行うようになっており、旧来のレコードメインは、旧バージョンからの移行のために互換レベルを残してあるだけで、新規タスクでは推奨しないようになりました。

この動きは、実は V9 から始まっており、V9 ですでにイベントハンドラによるイベント指向プログラム作成の基本機能が実装されていました。しかし、旧バージョンとの互換性を重視したため、レコードメインも正式サポート機能としてそのまま残されており、V9 および V9Plus で作成されたアプリケーションは、開発者の選択により、レコードメインによるもの、イベント指向によるもの、あるいはその混合という形になっているのが実情でした。

V10 では、この中途半端な状態から一歩進み、レコードメインを使わずイベント指向でプログラムを作成するよう、明確に方向が示されました。

この流れの目指すところは、次のようなものがあります。

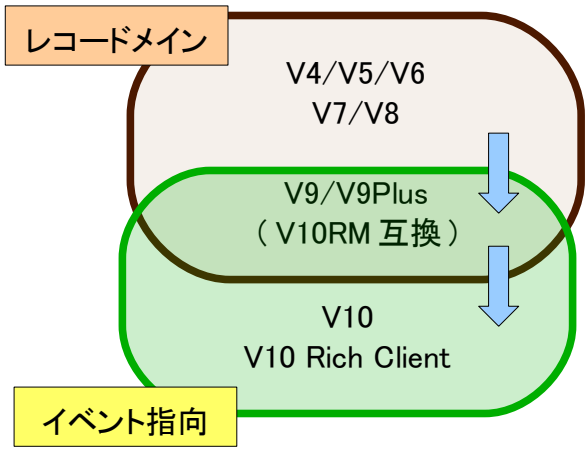
- 書きやすく読みやすいプログラムを作るため。本書で具体的に見ていきますが、レコードメインのプログラムより、イベント指向のプログラムの方が、書くときにも読むときにもすっきりと見通しのよいものにすることができます。
- V10 機能を最大限生かすため。V10 はイベントをベースとした機能が多数あります。その機能を最大限生かすために、イベント指向の考え方を十分に理解することが必要になります。
- 将来の Rich Client をにらみ。本書執筆の時点(2007 年 10 月)で、V10 の大きな新機能として Rich Client (コードネーム) が計画されており、開発作業も進んでいます。Rich Client においては、レコードメインはサポートされておらず、イベント指向プログラミングのみがサポートされています。

このような理由から、V10 に移行してゆく今が、レコードメインからイベントへの考え方の切り替えのよい機会であろうと思われます。

本書の説明では、V8 から V10 への移行を前提として例をとりあげましたが、V9/V9Plus ではレコードメイン、イベントプログラム両方がサポートされているので、本書の考え方は、

- V8 → V9/V9Plus イベント指向プログラム
- V9/V9Plus レコードメイン → V10

という移行においてもほぼそのまま当てはめることができます。



1.2 本書の目的と範囲

本書は、レコードメインによる Magic プログラミングに慣れている Magic 開発者が、イベント指向のプログラミング方法を理解し、移行することができるよう、概念と機能およびプログラム方法について具体例をあげて説明します。

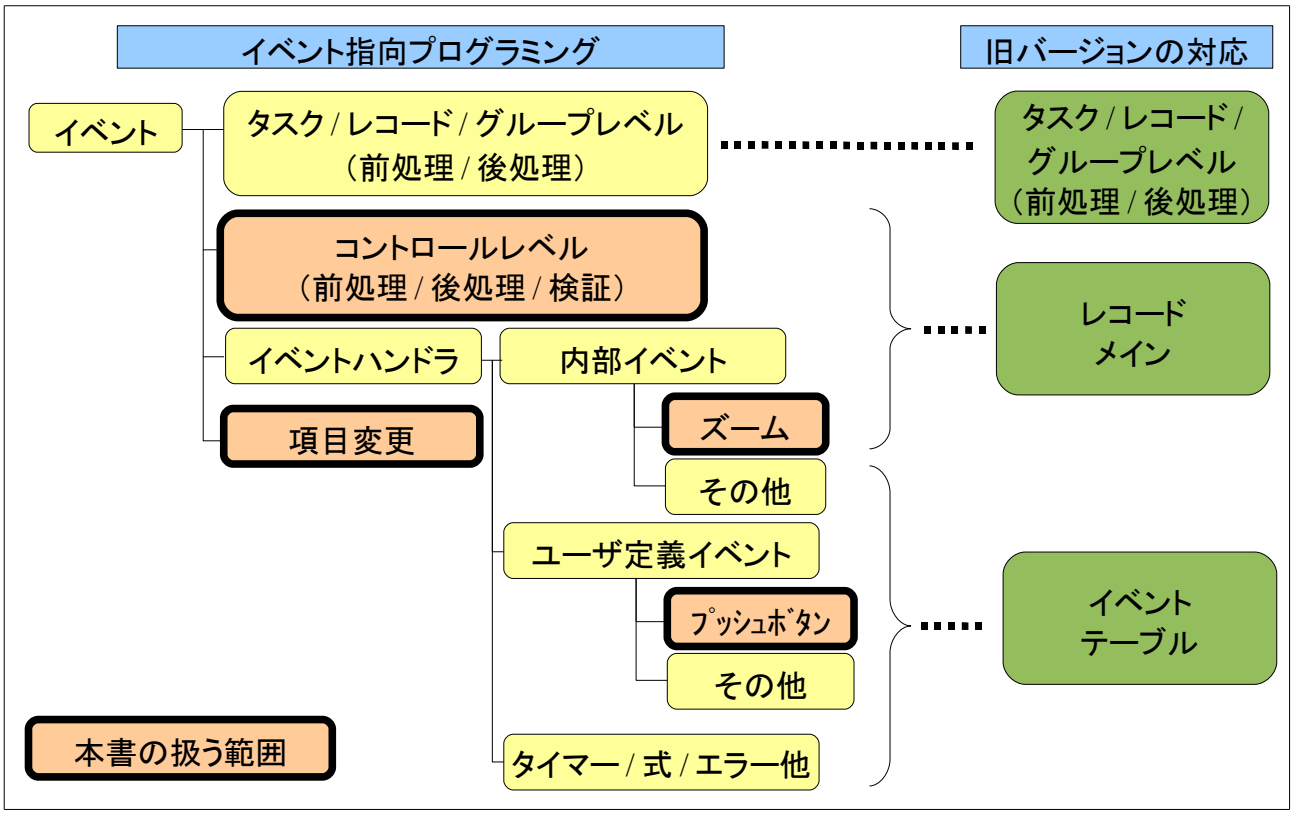
V10 でのイベント関連機能については非常に幅の広い話題がありますが、本書ではレコードメインからの移行に重点を置き、オンラインプログラムの画面制御に関連する話題だけにフォーカスします。具体的には、

- コントロールレベルのハンドラ (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 ハンドラ)
- 項目変更ハンドラ
- プッシュボタンにおけるイベントの利用
- ズームの実現方法

などが含まれます。

逆に、重要なイベント関連機能でありながら、本書に含まれないか、あるいはごく簡単にしか触れない話題には、次のようなものがあります。

- タスク、レコード、グループレベルでのイベントの扱い。
- 「イベント実行」コマンド。
- イベント タイプ (システム、内部、ユーザ定義、タイマー、式、エラー、ActiveX)。イベントとしては、主にユーザ定義イベントのみを扱っています。
- GUI モデルにおけるイベント関連パラメータ。プッシュボタンでの「実行イベント」のみを扱っています。
- ユーザ定義イベントでの「強制終了」パラメータ(N=なし、E=編集、C=コントロール、R=レコード更新前、P=レコード更新後)。ズームの章で、「E=編集」の動作について説明している他は、説明を省略しています。



1.3 前提条件

本書の読者は、次のような知識と経験があることを前提としています。

- dbMAGIC V8 レベルの Magic の知識を持っている。
- 旧バージョン (V9Plus 以下)でレコードメインを使ったプログラムを作ったことがあり、レコードメインの動作仕様について理解している。

この他、Magic Studio V10 のタスクエディッタの基本 (画面構成、データビューやロジック、フォームなどの作成・修正のための操作方法など)を理解していればベターですが、必須ではありません。本書では V10 のイベント指向プログラミングになれていない読者も念頭において、V10 での操作方法も簡単に説明を加えています。V10 の操作にすでに慣れている読者は、その部分は読み飛ばしても構いません。

そして、次のような問題意識を持っている開発者を対象として書かれています。

- レコードメインで書いてきたものを、イベント指向プログラムでどのように書いていけばよいのかよくわからない。
- イベント指向プログラムで作ってみたことはあるが、まだ体系的に理解していないので、不安が残る。

本書がイベント指向という新しいパラダイムに移行していく一助となれば幸いです。

本書で出てくるアプリケーションは、V8 で作られた実際のアプリケーションを V10 に移行したものです。現実の大きなアプリケーションの一部なので、サンプルとしてご提供できませんが、ご了承ください。説明に出てくるプログラムは、主に受注入力を行うタスクです。

第2章　旧バージョンのレコードメインの動作

V10 のイベント指向プログラミングについて説明する前に、本章でまずは旧バージョンでのレコードメインの動作について説明します。この内容は、レコードメインで Magic プログラムを作成されてきた開発者にとっては既知の内容と思いますが、おさらいと頭の整理の意味で一章を割いて解説します。

2.1　タスクエディッタの構成

最初に、タスクエディッタの構成について復習しましょう。

下図は V8 のタスクエディッタの画面です。

V8 のタスクエディッタ

ここで見るように、画面は大きく分けて、上下の二つのテーブルからなります。

上のテーブルは **レベルテーブル** と呼ばれ、レコード、タスク、ブレイクなどのレベルからなります。各レベルにはそれぞれ前処理と後処理とがあり、トランザクションやエラーの設定を行えます。レコードレベルだけは特別に **レコードメイン** のレベルがあります。

下のテーブルは、**処理テーブル** と呼ばれ、各レベルにおいて実行すべきコマンドが記述されます。本書ではレコードメインが話題ですので、レコードメインの処理テーブルを図では表示しています。

レコードメインでは、セレクトコマンド、およびリンクコマンドによって、このタスクで利用するデータ項目を定義しており、その間に、オンライン画面上での操作を、コールコマンド、項目更新コマンド、その他の処理コマンドにより記述しています。

2.2 レコードメインの基本的考え方

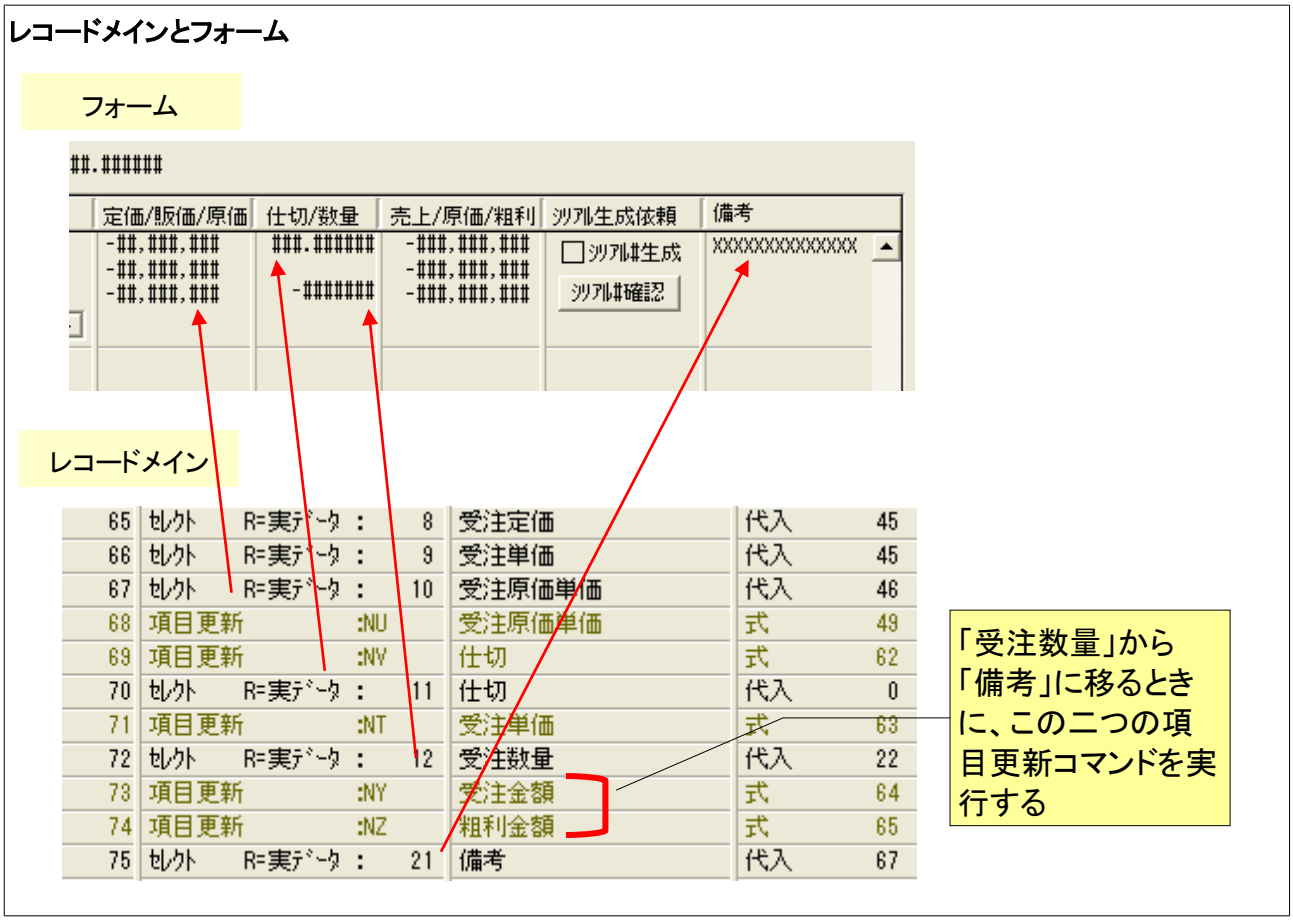
レコードメインでオンライン画面操作を記述する場合の基本的な考え方は、

カーソルの動きに応じて、セレクトコマンド間に処理コマンドを記述する

というものです。ここで、セレクトコマンドは画面上に表示されているデータ項目に対応したものであり、処理コマンドは、項目から項目にカーソルが移動するときに、処理する内容を記述します。

具体的な例を挙げてみると、下図では、フォームとそれに対応するレコードメインを示しています。実行時、カーソルが「受注数量」の項目 (72 行目のセレクトコマンド) にあるとき、ユーザが Enter/Tab キーを押して、その次の「備考」項目 (75 行目のセレクトコマンド)にカーソルが移動するとします。そのときには、この二つのセレクトコマンドの間にある二つの項目更新コマンド (73 行目と 74 行目)とが実行されます。

レコードメインとフォーム

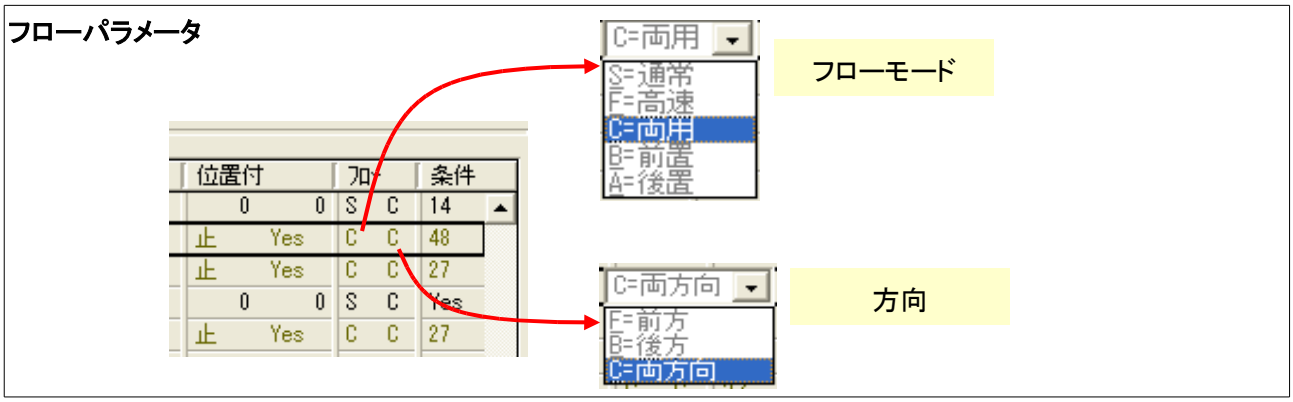


2.3 レコードメインのフローパラメータ

次に、レコードメインでのコマンドについて、もっと細かく見ていきましょう。

2.3.1 フローパラメータ

フローパラメータは、レコードメインの各コマンドで、「フロー」と書かれている欄に設定することのできる、二つのパラメータです。このパラメータで、コマンドが有効になる(実行される)条件を指定することができます。



最初（左側）のパラメータは、「フローモード」と呼びます。これには、

- S=通常
- F=高速
- C=両用
- B=前置
- A=後置

があります。

このうち、「S=通常」と「F=高速」については、次節以下で説明します。

「C=両用」は「S=通常」と「F=高速」とを兼ねるもので、「常に有効」という意味になります。

また、「B=前置」と「A=後置」は、ズームを実現するために設定するモードであり、第 8 章「ズーム」で説明します。

2 番目（右側）のパラメータは「方向」パラメータであり、これには、

- F=前方
- B=後方
- C=両方向

があります。

これはカーソル移動方向によりコマンド実行の有効/無効を決めるものです。

「C=両方向」というのは「F=前方」と「B=後方」を兼ねるもので、やはり「常に有効」の意味となります。

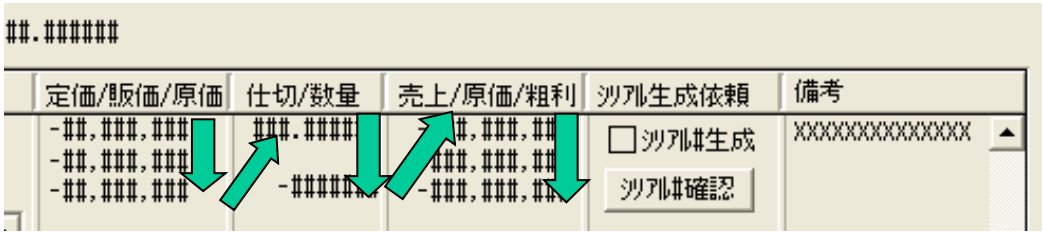
2.3.2 通常モード

フローモードの「S=通常モード」というのは、Enter/Tab キー（「次項目」アクション）で動く場合のモードです。

dbMAGIC V5 以来、「次項目」アクションの標準キー割り当ては Tab キーですが、ユーザの利用の便のために Enter キーを「次項目」アクションに割り当てているアプリケーションが多いので、本書では「Enter または Tab キー」あるいは「Enter/Tab キー」としています。

例えば、下図では 定価、販価、原価、仕切、数量、・・・ などの項目がありますが、カーソルがこれらの項目間を Enter/Tab キーによって一つづつ動いていく場合には、通常モードとなります。

S＝通常モード（Enter/Tabキーで動く場合のモード）



なお、Magic には「次項目」アクションに対応する「前項目」アクションもあり、標準キー割り当てでは Shift+Tab キーなどに割り当てられていますが、「前項目」アクションにより、逆方向に（後ろから前に）カーソルが動く場合にも、通常モードとなります。

2.3.3 高速モード

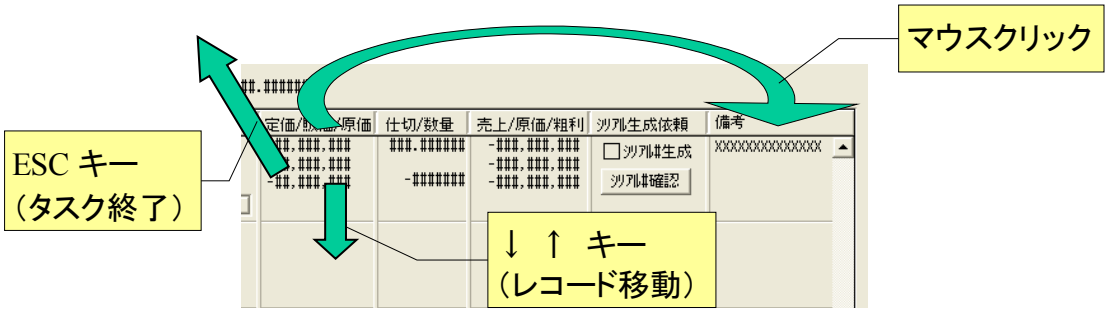
高速モードというのは、「次項目」「前項目」アクション以外の原因によってカーソルが動く場合のモードです。これには、

- マウスクリック
- 「↑」「↓」キーによるレコードの移動
- ESC キーによるタスク終了

などでカーソルが動く場合が該当します。

なお、マウスクリックの場合には、たとえ隣接する項目に移動する場合であっても、高速モードとなります。

F=高速モード（マウス/↑↓キー/タスク終了などでカーソルが動く場合のモード）



2.3.4 方向 パラメータ

「方向」パラメータは、カーソルの動く方向によって、コマンドの有効/無効を制御するパラメータです。

- 「F=前方」に設定されている場合には、順方向にカーソルが動く場合にのみ、コマンドが実行されます。
- 「B=後方」に設定されている場合には、逆方向にカーソルが動く場合にのみ、コマンドが実行されます。
- 「C=両方向」に設定されている場合には、いずれの方向にカーソルが動いてもコマンドが実行されます。

2.3.5 コマンド実行順序

「方向」パラメータに関連して注意しておかなければならないのは、処理コマンドが二つ以上ある場合には、カーソルの移動方向によって、処理コマンドの実行順序が反対になる、ということです。即ち、

- 順方向（上から下）にカーソルが動く場合には、コマンドも上から下に実行される。
- 逆方向（下から上）にカーソルが動く場合には、コマンドも下から上に実行される。

ということです。

例えば、下図のようなプログラムでは、カーソルが「受注数量」と「備考」の間を移動するときに、二つの項目更新コマンドが実行されるように設定されています。このとき、

- カーソルが「受注数量」から「備考」に向けて、順方向に移動する場合には、受注金額の項目更新(73 行目)がまず実行されてから粗利金額の項目更新(74 行目)が実行されます。
- 逆に、カーソルが「備考」から「受注数量」に向けて、逆方向に動く場合には、下にある粗利金額(74 行目)が先に実行され、つぎに上にある受注金額 (73 行目)が実行されます。

コマンド実行順序（カーソルの動く方向により、コマンドの実行順序が異なる）

通常のプログラミング言語では、実行は常に上から下に向けて行われるものですが、Magic のレコードメインのように、カーソルの動きによって上から実行されたり下から実行されたりするのは、Magic に独特な実行方式です。これは Magic 初心者が戸惑う Magic ルールの一つです。

また、アプリケーション開発者にとっては、順方向に実行された場合と逆方向に実行された場合とで結果が異なるようなことになりかねないものであり、カーソルが順方向に動く場合と逆方向に動く場合とで矛盾なく動作するように考慮して設計する必要が出てきます。

2.4 レコードメインの書き方ABC

以上の説明から、レコードメインを使ったプログラミングの方法について、

A） どこで？

B） いつ？

C） 何を？

という観点で整理してみると、次のようになります。

レコードメインの書き方

A） どこで？
→ ある項目（セレクトコマンド）と別の項目（セレクトコマンド）の間で

B） いつ？
→ カーソルの動くとき

- Enter/Tab で動く場合のみ実行させたいなら、「S=通常」 モード
- ↓ ↑ キー、ESC、マウス等で動く場合のみ実行させたいなら、「F=高速」 モード
- 常に実行させるなら、「C=両用」 モード

※ 特定の方向だけに限定したければ、「方向」に「F=前方」 または「B=後方」を設定する。

C） 何を？
→ セレクトコマンドの間に、実行するコマンドを記述。

これは簡略化したもので、実際のレコードメインの動作にはもっと複雑なルールがあります。詳細な説明は省略しますが、本章で説明した以外に、次のような要因がレコードメインの処理の流れに影響を与えます。

- データビューが修正されているかされていないか。
- エラーコマンド（エラーオプション）が実行されたか。
- ブロックコマンド、セレクトコマンド、処理コマンドなどの位置関係と、フローモードや条件式などの組み合わせ。

これらを正確に理解しようとすると、レコードメインフローモードは結構難しいブラックボックスです。Magic 初心者にとっては、学習に時間のかかる壁になり、開発者にとっては、設計やデバッグに時間がかかる原因となります。

2.5 レコードメインの問題点

ここで、レコードメインによるプログラミングの問題点について考えてみたいと思います。

2.5.1 記述性

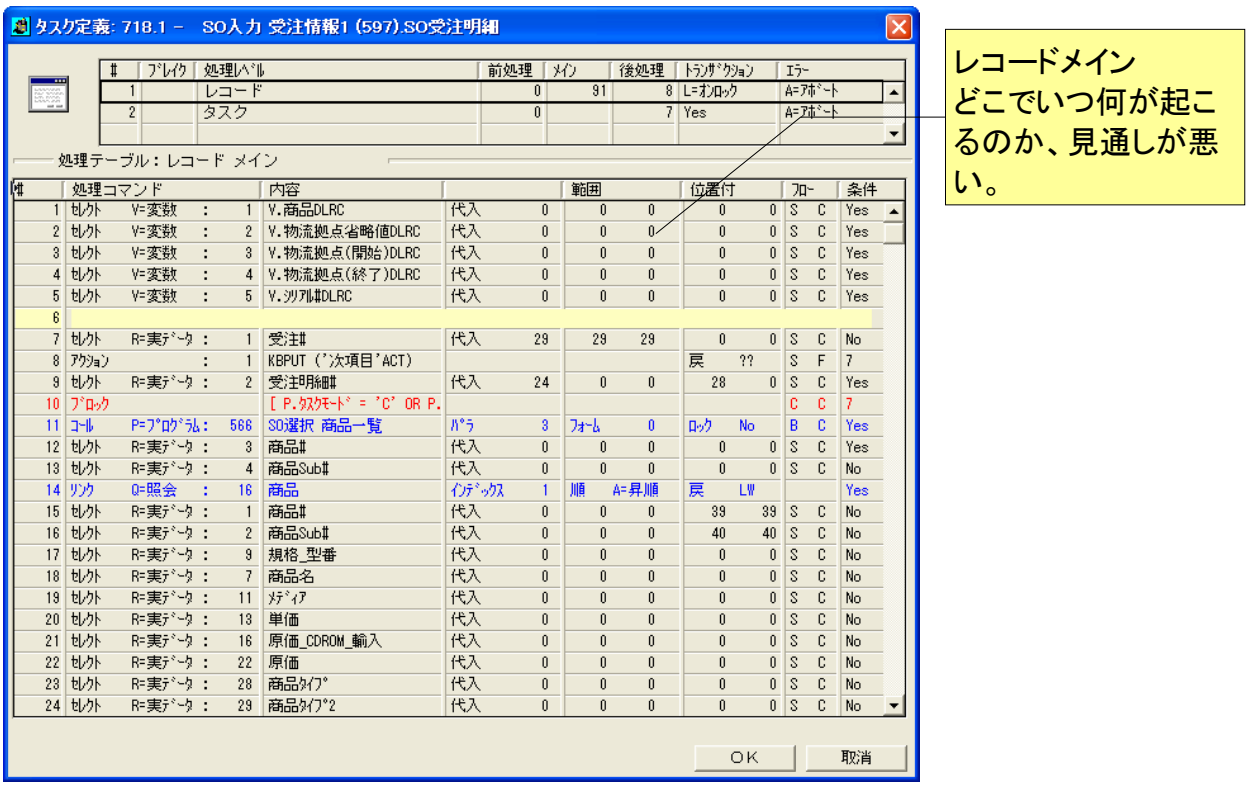
キーボードのみで操作していたDOSの時代には、カーソルの動きを処理の流れと合わせて記述するレコードメインの記述方法は自然で直感的であり、また、一つの「レコードメイン」という処理テーブルの中に多くの意味を多重的に指定することのできる、という意味で、記述性の高い方法でした。

しかし、キーボードに加えてマウスが導入され、WindowsのGUI部品が多用されるようになり、更にはタイマーや式、エラーなど、ユーザ入力以外の要因で割り込み的な処理の記述をしたくなるアプリケーションが増えるに従って、レコードメインでの記述では制御が複雑になって実装が困難な場面が出てくるようになりました。

2.5.2 プログラムの可読性

プログラムが複雑になるにつれ、プログラムの見通しの悪さも目立つようになってきました。例えば、下図はレコードメインの例です。

レコードメインの例

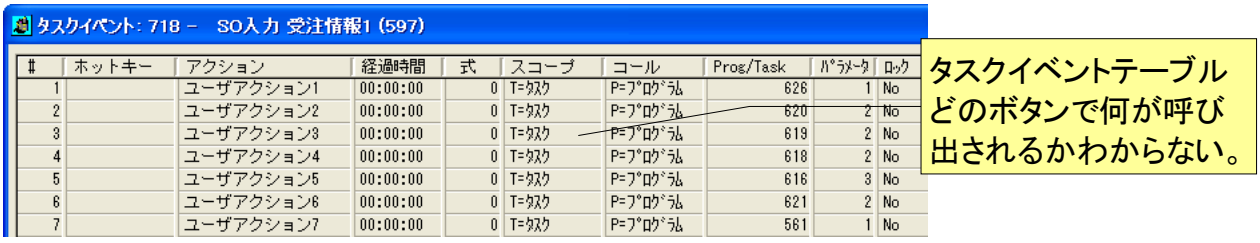


これをぱっと見ただけで、どの項目からカーソルが動くときに、何が実行されるのかわかりにくいものがあります。ユーザの実行時の目から見ると、ある項目の直後に行うべき処理コマンド(例えば、データリンクのエラーチェックなど)が、レコードメイン上では、途中に多くの非パーク項目に挟まれていて、かなり離れたところに設定されているということもよくあります。どの処理コマンドがどの項目で実行されるかは、セレクトコマンドの「条件」欄をチェックして、前後の文脈を見ていかないとわからないことになります。

また、オンライン画面での処理に関連して、イベントテーブルの可読性の悪さも問題になります。次の図は、同

じタスクの タスクイベント テーブルの例です。

タスクイベントテーブル



#	ホットキー	アクション	経過時間	式	スコープ	コール	Prog/Task	ﾊﾟﾗﾒｰﾀ	ﾛｯｸ
1		ユーザアクション1	00:00:00	0	T=ﾀｽｸ	P=ﾌﾟﾛｸﾞﾗﾑ	626	1	No
2		ユーザアクション2	00:00:00	0	T=ﾀｽｸ	P=ﾌﾟﾛｸﾞﾗﾑ	620	2	No
3		ユーザアクション3	00:00:00	0	T=ﾀｽｸ	P=ﾌﾟﾛｸﾞﾗﾑ	619	2	No
4		ユーザアクション4	00:00:00	0	T=ﾀｽｸ	P=ﾌﾟﾛｸﾞﾗﾑ	618	2	No
5		ユーザアクション5	00:00:00	0	T=ﾀｽｸ	P=ﾌﾟﾛｸﾞﾗﾑ	616	3	No
6		ユーザアクション6	00:00:00	0	T=ﾀｽｸ	P=ﾌﾟﾛｸﾞﾗﾑ	621	2	No
7		ユーザアクション7	00:00:00	0	T=ﾀｽｸ	P=ﾌﾟﾛｸﾞﾗﾑ	561	1	No

この図の例を見てわかるように、タスクイベントでは「ユーザアクション *n* 」と、プログラム番号とが見えるだけで、「どのボタンを押したときに、どのような処理が行われるのか」という肝心な内容が、これを見ただけではほとんどわかりません。どのボタンにどのユーザアクションが割り当てられており、何番のプログラムが何をするのか、という記録と対照しない限り、プッシュボタンと処理の関連は一見してはまずわからないといえましょう。

2.5.3 複雑なブラックボックス

2.4「レコードメインの書き方ABC」で簡単に触れましたが、レコードメインの動作は詳細を見ると意外と複雑なブラックボックスです。このため、慣れた開発者であっても、期待通りの動きにならないときに、どうしてそうなるのかすぐにわからないため、修正に時間がかかってしまうことがあります。

第3章 V10 のイベント指向プログラミング

V10 では、レコードメインを基本的に廃止し、全面的にイベント指向でプログラムを作るようになりました。イベント指向のプログラミング方法は、複雑なブラックボックスとなってしまったレコードメインではなく、よりシンプルで直感的にわかりやすい方法でプログラミングを行えるようになることを目指しています。

本章では、V10 でのイベント指向プログラミングの基礎について簡単に説明し、次章移行で個々のケースについて例を挙げながら説明していきます。

3.1 タスクエディッタの構成

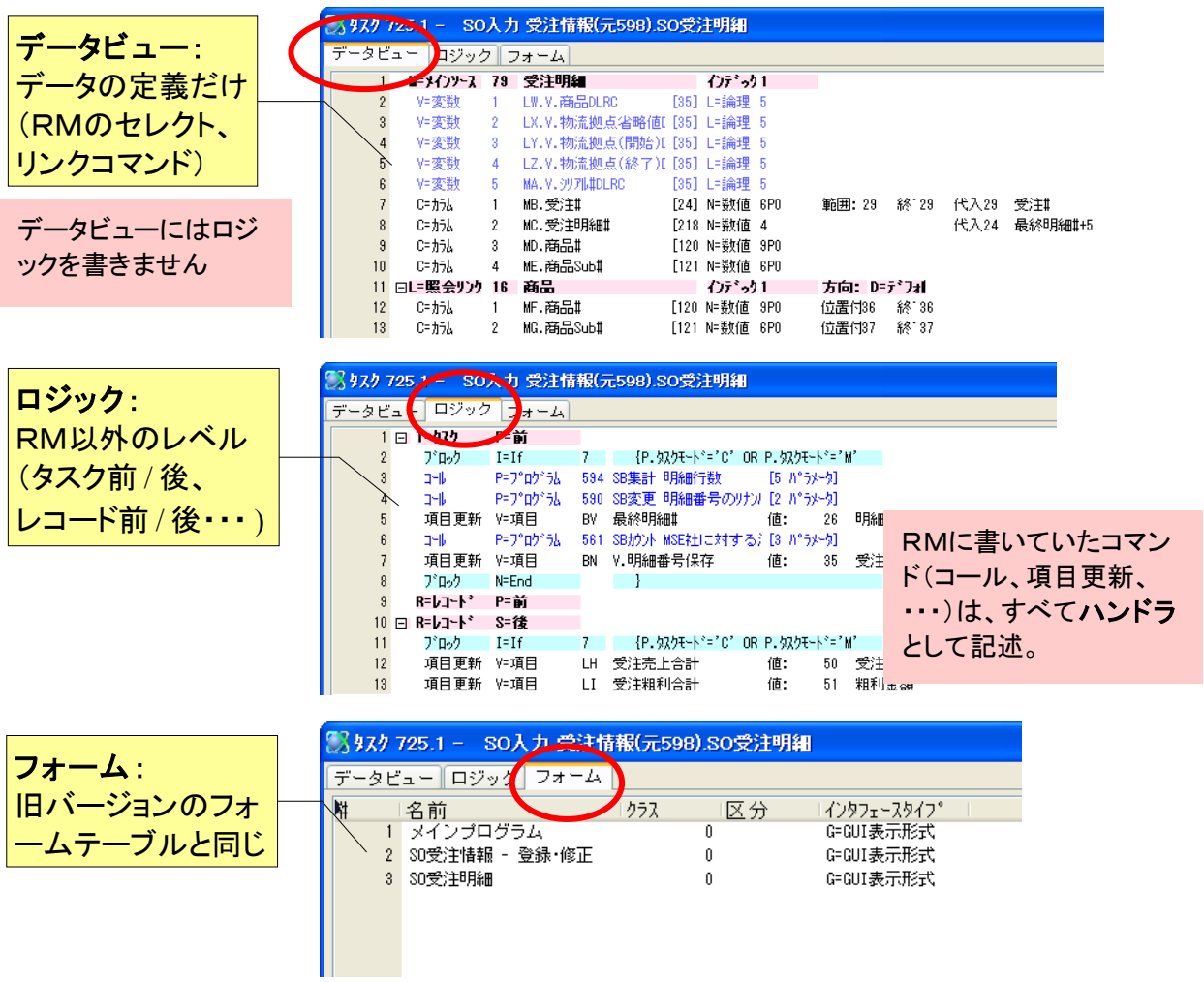
Magic のプログラムの単位であるタスクの作成はタスクエディッタで行いますが、V10 になってタスクエディッタのユーザインターフェースが大きく変更になりました。

まず、タスクエディッタは、

- データビュー
- ロジック
- フォーム

という三つのタブで大分類されています。

V10 タスクエディッタの構成



データビューというのは、このタスクで利用するデータの定義を行うものです。旧バージョンでは、レコードメインで、セレクトコマンドおよびリンクコマンドで行っていたものです。V10 のデータビューでは、データの定義だけを行い、レコードメインで行っていたようなロジックの記述は行いません。

ロジック というのは、文字通りタスクのロジックを記述するものです。

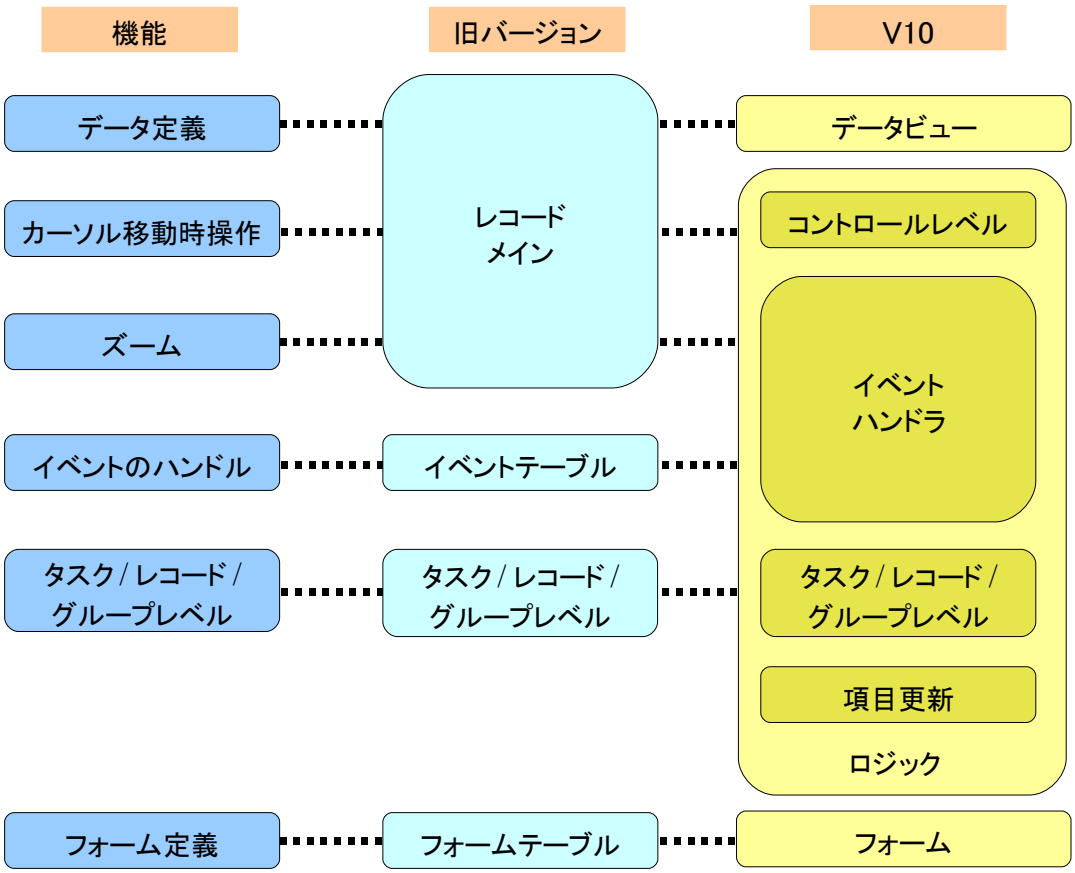
これには、タスクレベル、レコードレベル、ブレイクレベル（バッチタスクのみ）、コントロールレベル（オンラインタスクのみ）のそれぞれ前処理と後処理、コントロール検証、項目変更、およびイベントハンドラを定義します。

これは、旧バージョンのタスクエディッタと比べ、タスク、レコード、ブレイクレベルは同じで、コントロールレベル（前処理、後処理、検証)はレコードメインの中に記述していたオンライン画面での動作に相当します。

フォームは、旧バージョンでのフォームテーブルと同じですので、ここでは説明を省略します。

以上の説明をまとめ、機能別に旧バージョンと V10 との対応を図示すると、下図のようになります。

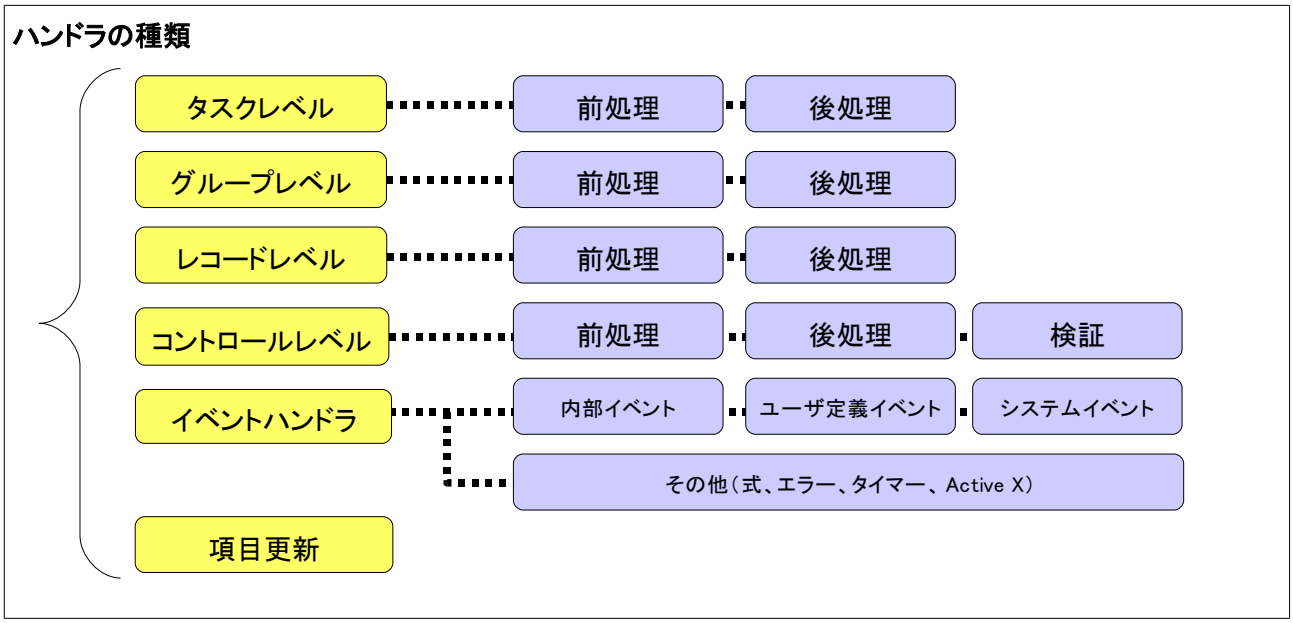
タスクエディッタ: 旧バージョンと V10 の機能別対応



ロジック タブに記述するハンドラの種類については、次節以下により詳しく説明していきます。

3.2 ハンドラの種別とタイプ

ロジック 画面では、ハンドラ を単位として記述します。V10 でのハンドラには次のような種類のものがあります。



タスク/グループ/レコードレベルハンドラ については、以前のバージョンと変わりありませんので、本書では説明を省略します。

コントロールレベルハンドラはオンラインタスクにのみ作成することのできるハンドラであり、前処理、後処理、検証 というタイプがあります。これはオンライン画面上で、カーソルが移動したときに実行されるもので、第 4 章 コントロールレベルハンドラ で詳しく見ていきます。

イベントハンドラは、イベント指向プログラミングの一番重要となるハンドラであり、これには内部イベント、ユーザ定義イベント、システムイベント、その他のタイプがあります。イベントハンドラは V9 ですでに導入されていましたが、V10 でもほぼそのままの形で利用できるようになっています。V8 ではアプリケーションイベントテーブルおよびタスクイベントテーブルで定義されていたものですが、V8 のものにくらべて使いやすい形になっています。

イベントハンドラについては、第 7 章 プッシュボタン、および第 8 章ズーム で、ユーザ定義イベントを使った例についてだけ説明します。

項目変更 ハンドラというのは、V10の新機能であり、項目の値に変更があった場合に実行されます。V9Plus では、「コントロール変更」ハンドラというものがありましたが、項目変更ハンドラはこれを更に広く利用できるよう拡張したものです。項目変更ハンドラの詳細は、第 6 章 項目変更ハンドラ で説明します。

3.3 ハンドラの書き方

V10 でのハンドラは、どの種類のハンドラであっても、すべてロジック タブに定義します。

一つのタスクには一般には複数のハンドラがありますが、必要がなければ一つも書かずに、空のままでもかまいません。例えば、APG で作成したオンライン照会プログラムでは、データビューとフォームの定義だけからない、ロジックが必要ないので、ロジックは空です。

一つのハンドルは、ヘッダ行と詳細行とからなります。

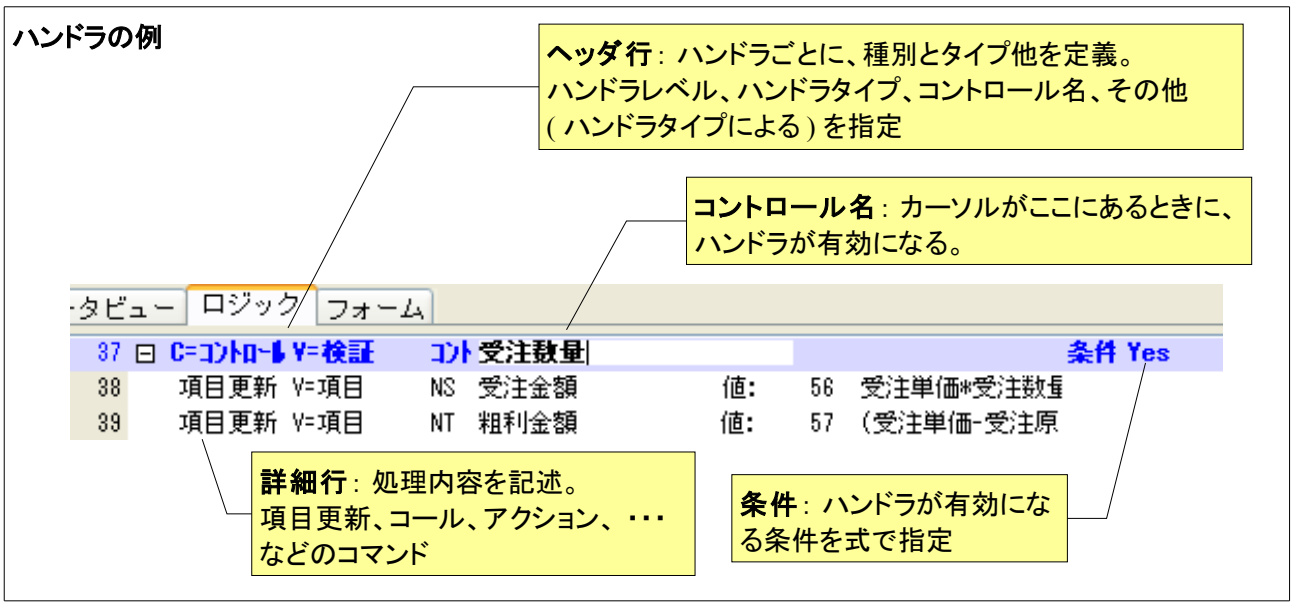
ヘッダ行はハンドラの最初の行であり、ハンドラごとに種別とタイプ他を定義するもので、ハンドラレベル、ハンドラタイプ、コントロール名、その他 (ハンドラタイプによる) を指定します。

ヘッダ行には、**コントロール名** を指定することができます。ここには、フォーム上のコントロールに付けられたコントロール名を指定し、カーソルがそのコントロールにある場合にだけ、そのハンドラが有効となります。

コントロール名の指定は、コントロールレベルハンドラ (コントロール前処理、後処理、検証) では必須であり、イベントハンドラでは任意です。イベントハンドラでコントロール名が指定されなかった場合には、そのイベントハンドラは、カーソルの位置に関わらずいつも有効となります。

また、ヘッダ行には **条件** を式で設定することもできます。条件が設定されていた場合、このハンドラはその条件が真になる場合にだけ有効になります。

詳細行は、このハンドラによって処理される処理内容を、項目更新、コール、アクションなどの Magic の処理コマンドで記述します。詳細行は、一つのハンドラに対して複数行記述することができます。下図で見るように、詳細行は、ヘッダ行よりも数文字分インデントされて表示され、ヘッダ行と詳細行とが見てするわかるようになっています。また、色定義ファイルを使うことによって、ヘッダ行を色分けして表示することもできるようになります。

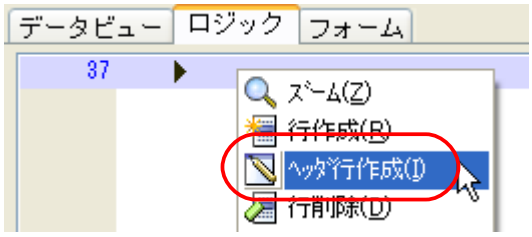


3.4　ハンドラの作成手順

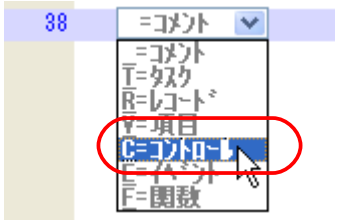
ここでは、上に例に示した「受注数量」のコントロール検証ハンドラを作成してみます。コントロール検証ハンドラの使い方は、次章に解説します。

ハンドラの作成手順

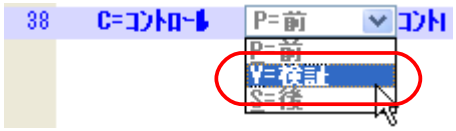
1. タスクエディッタで、ロジック タブをクリックして、ロジック画面を表示させます。
2. 新しいハンドラを作成するには、まずヘッダ行を作成します。
   ヘッダ行を作成するには、ポップアップメニューの「ヘッダ行作成」(Ctrl+H)を選びます。
   → 新しくヘッダ行が作成されます。



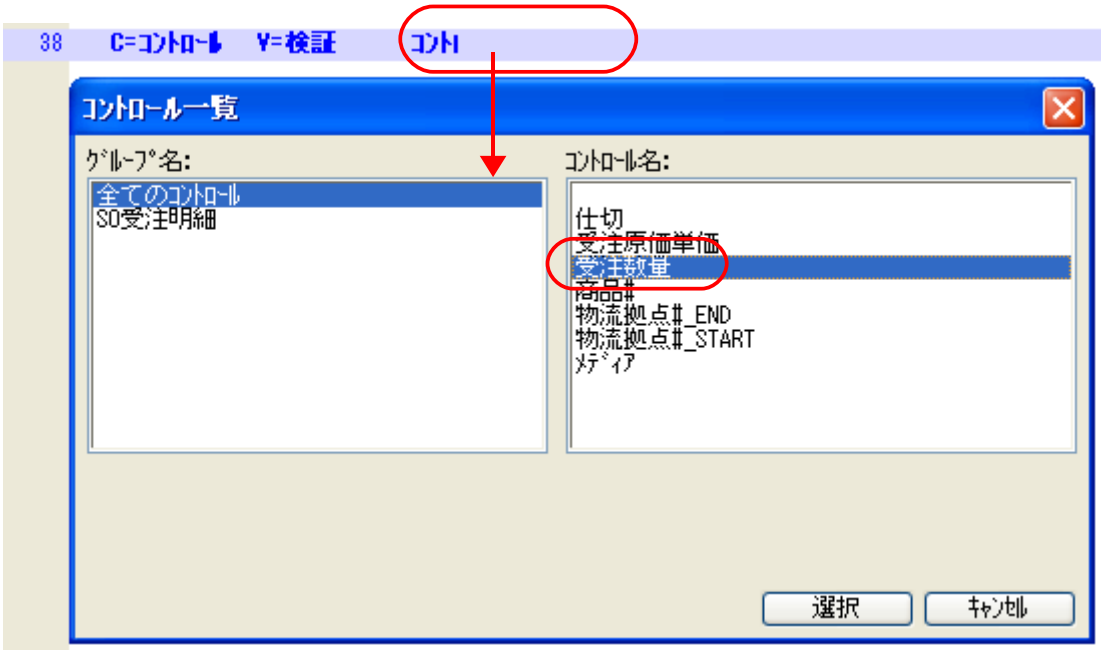
3. ヘッダ行の先頭で、ハンドラの種類として、コンボボックスから「C=コントロール」を選びます。



4. ヘッダ行の 2 番目のカラムに進み、ハンドラのタイプとして、コンボボックスから「V=検証」を選びます。



5. ヘッダ行の「コントロール名」欄に進み、ズームします。
   → このタスクのフォームで定義されているコントロール名の一覧が表示されます。



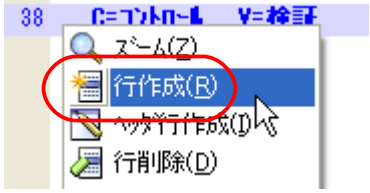
6. この中から「受注数量」を選択します。

   → 「コントロール名」欄に、選択したコントロール名が表示されます。

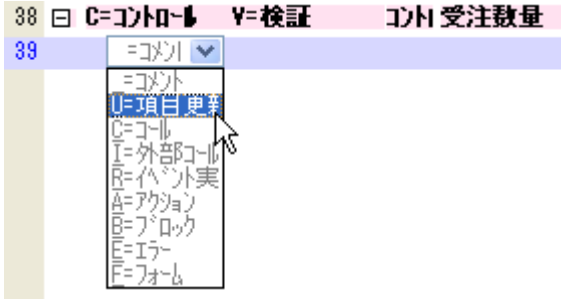
   以上で、ヘッダ行ができあがりました。

```
38    C=ｺﾝﾄﾛｰﾙ    V=検証        ｺﾝﾄ 受注数量                                    条件 Yes
```

7. 次に詳細行を作成します。
   詳細行を作成するには、ポップアップメニューの「行作成」(F4)を選びます。
   → 新規詳細行が作成されます。



8. 詳細行の先頭のコマンドのところで、ズームします。
   → コマンドを選択するためのコンボボックスが出てきます。



9. この中から、「項目更新」を選びます。

10. あとは、旧バージョンでのコマンドの場合と同じ方法で記述することができます。

```
38 ⊟ C=ｺﾝﾄﾛｰﾙ    V=検証        ｺﾝﾄ 受注数量
39     項目更新    V=項目        NS   受注金額              値:     56   受注単価*受注数量   条件: Yes
```

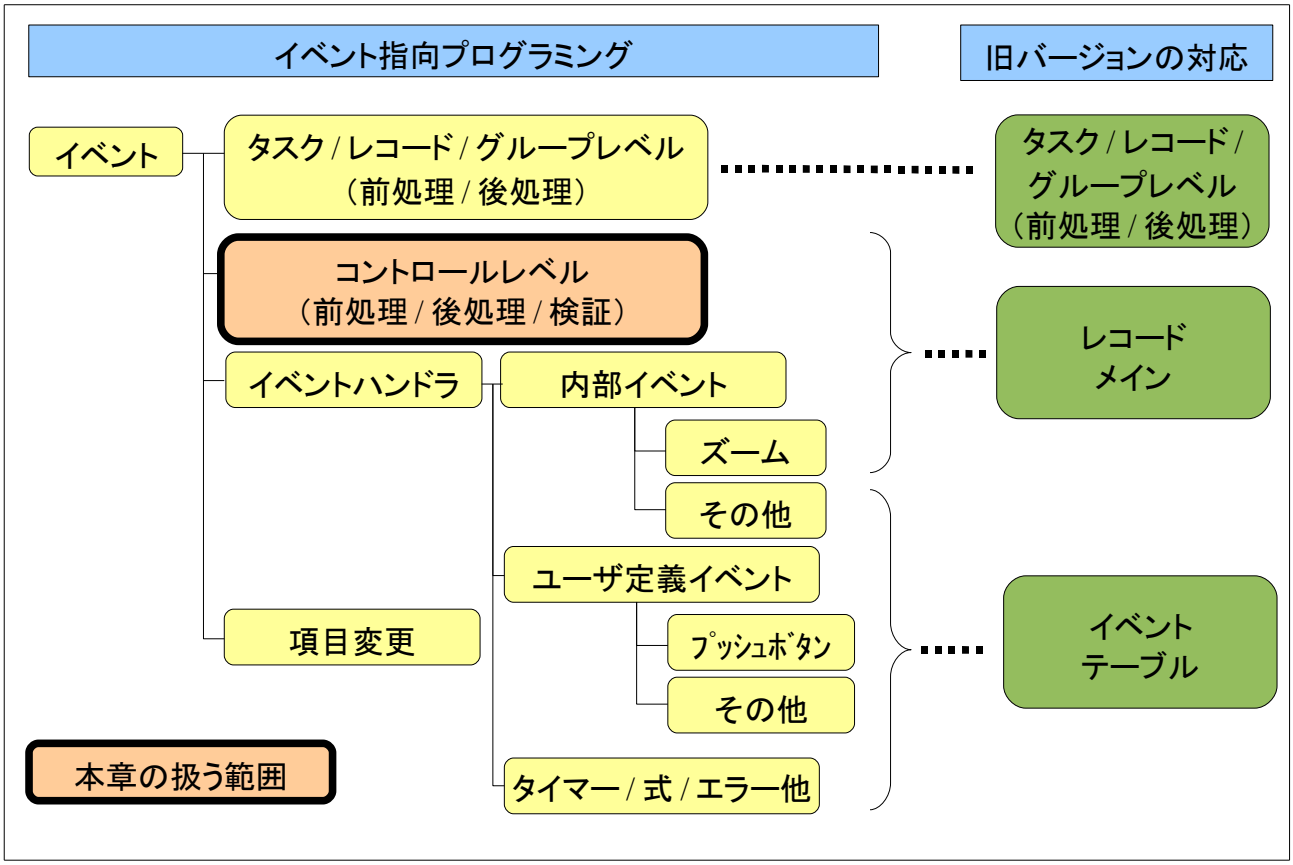
11. 必要に応じ、「行作成」(F4)を繰り返し、一連のコマンドを記述してください。

```
38 ⊟ C=ｺﾝﾄﾛｰﾙ    V=検証        ｺﾝﾄ 受注数量
39     項目更新    V=項目        NS   受注金額              値:     56   受注単価*受注数量
40     項目更新    V=項目        NT   粗利金額              値:     57   (受注単価-受注原価)*条件: Yes
```

第4章　コントロールレベルハンドラの基本

本章と次章では、コントロールレベルハンドラについて扱います。

本章ではまずコントロールレベルハンドラの動作の基本について説明します。次章ではそれをもとにして、具体的な例を使って、レコードメインで書いていたものをコントロールレベルハンドラで書き直していきます。



4.1 コントロールレベルのハンドラの動作

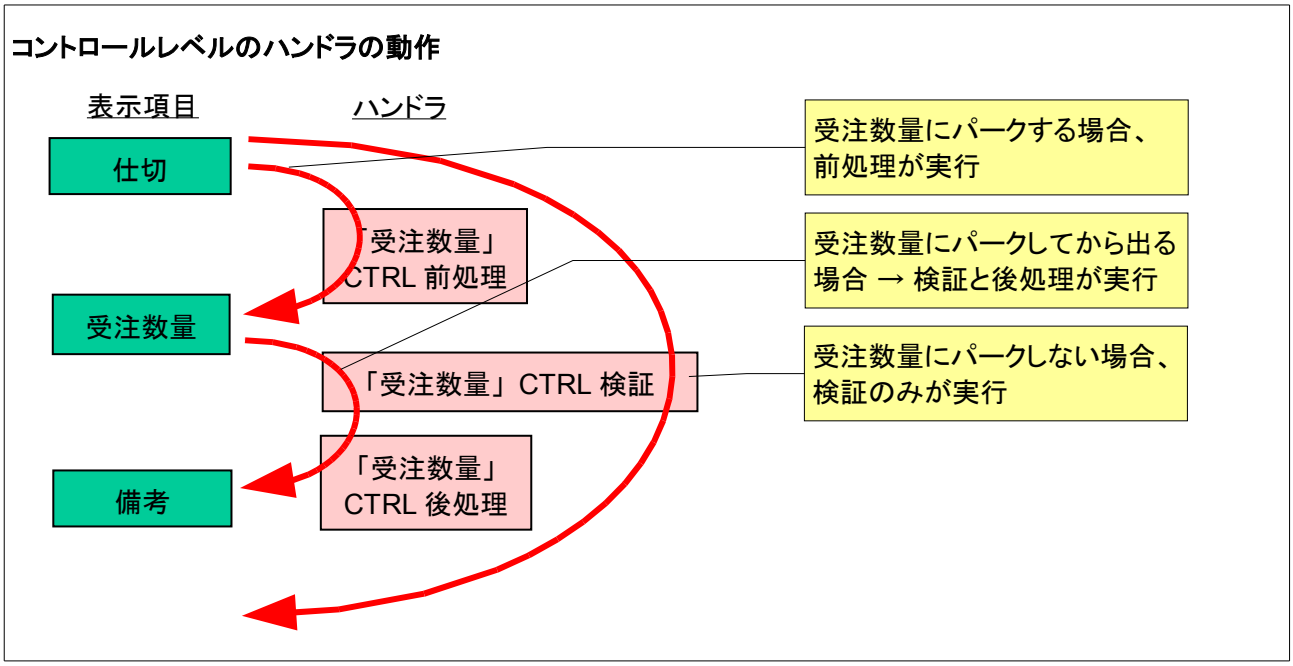
オンライン画面のカーソル移動時の操作は、以前のバージョンではレコードメインの中で、セレクトコマンドの間に処理コマンドを入れて定義していましたが、V10 のイベント指向プログラムでは、コントロールレベルのハンドラに記述します。

コントロールレベルハンドラは、各コントロール(表示項目)ごとに、

- コントロール前処理
- コントロール検証
- コントロール後処理

の 3 種類のハンドラを設定できます。

これらのハンドラは、実行されるタイミングが異なります。下図は、「受注数量」というコントロール名が付けられたエディットコントロールに、それぞれコントロール前処理、コントロール検証、コントロール後処理が定義されていた場合に、どのタイミングでそれぞれのハンドラが実行されるかを描いたものです。(「CTRL」は「コントロール」の意味です。)



最初にカーソルが「仕切」項目にあり、Enter キーなどにより「受注数量」項目にカーソルが移動したとすると、そのタイミングでコントロール前処理が実行されます。

次に、再度 Enter キーにより、カーソルが「受注数量」から「備考」項目に移動したときには、コントロール検証が実行され、ついでコントロール後処理が実行されます。

このように、「受注数量」にカーソルがパークする場合には、コントロール前処理 (入るとき)、検証・後処理 (出るとき) がすべて実行されます。

一方、カーソルが「受注数量」にパークしない場合には、前処理および後処理は実行されません。すなわち、最初にカーソルが「仕切」(あるいは、タブ順序で言ってそれ以前の項目)にあったとき、マウスなどで「備考」項目(あるいは、タブ順序で言ってそれ以降の項目)をクリックして、カーソルが「受注数量」にパークせずに飛び越して移動した場合には、前処理および後処理は実行されません。しかしこのような場合でもコントロール検証は実行されます。

4.2 コントロールレベルハンドラの書き方ABC

以上のような動作をすることから、ある項目（正確には、その項目に対応するフォーム上のコントロール）に着目して、コントロールレベルハンドラの書き方は以下のようになります。

- どこで？
  - ある表示項目(例: 受注数量)において・・・
- いつ？
  - カーソルが項目にパークするときにだけ実行するなら
    - 入るときの処理 → コントロール前処理に書く
    - 出るときの処理 → コントロール後処理に書く
    - ズームのときの処理 → ズームハンドラに書く（第 8 章ズームで説明）
  - カーソルが動くときに常に実行するなら
    - コントロール検証に書く
- 何を？
  - ハンドラの中にコマンドを記述

4.3 レコードメインとコントロールレベルハンドラとの違い

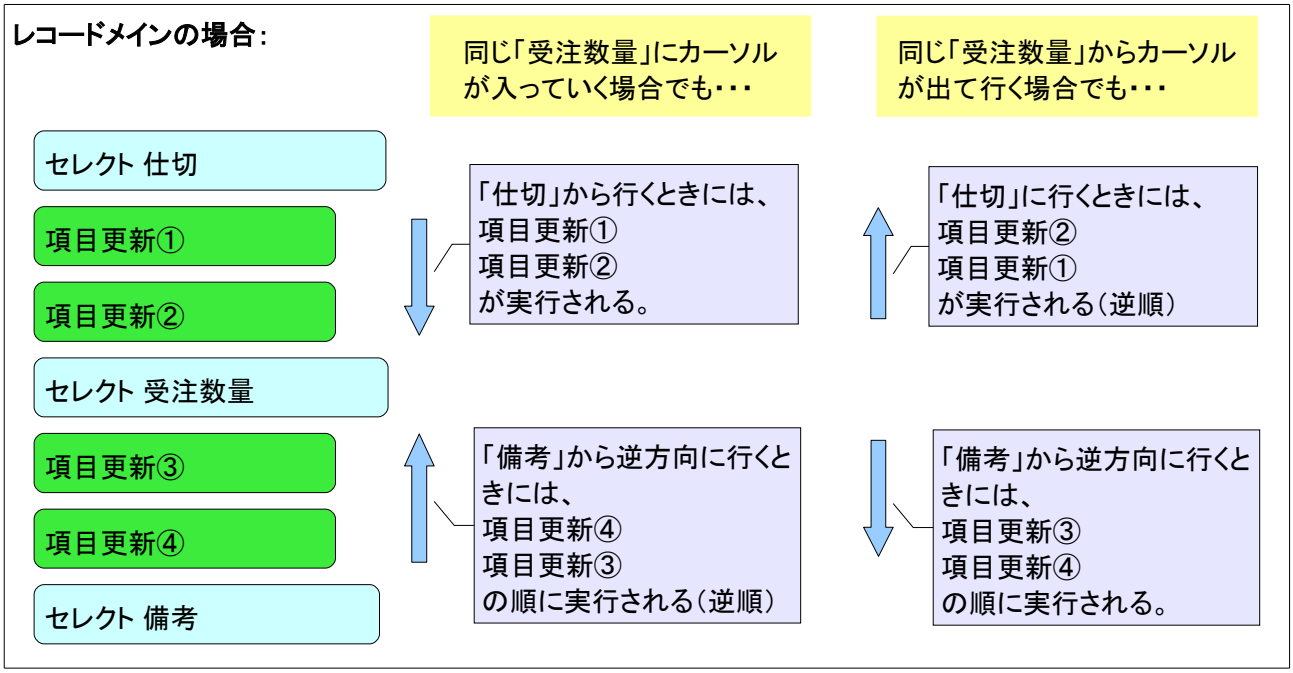
ここで、コントロールレベルのハンドラの動作と、旧バージョンでのレコードメインの動作との違いについて、一言加えておきます。

4.3.1 カーソルの移動方向

旧バージョンのレコードメインでは、カーソルの移動順序(タブ順序)がセレクトコマンドの順序と対応しており、セレクトコマンドの間に、カーソル移動時の処理内容を記述していました。

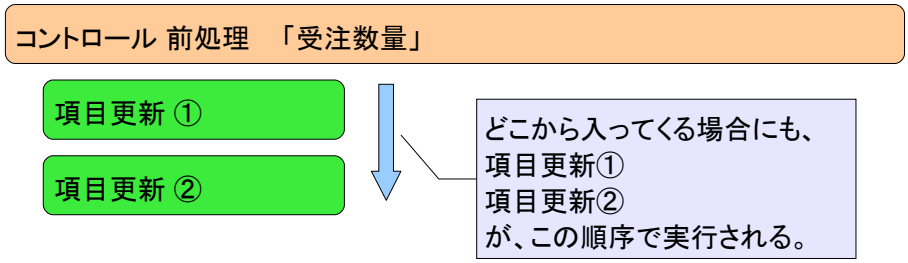
例えば、カーソルが「受注数量」にパークする際に、上（仕切）から来る場合には「仕切」のセレクトコマンドと「受注数量」のセレクトコマンドの間の処理コマンド（項目更新① と 項目更新②）が実行されますが、下（備考）から来る場合には、「備考」と「受注数量」の間のセレクトコマンドの間の処理コマンドが逆方向に（つまり、項目更新④、項目更新③の順序で）実行されました。

逆に、カーソルが「受注数量」から出て行く場合には、下（備考）に向けて移動する場合には、「受注数量」と「備考」の間の処理コマンド（項目更新③と項目更新④）が順方向に実行されました。一方、上（仕切）に向けて逆方向に移動する場合には、「仕切」のセレクトコマンドの間にある項目更新② と項目更新① とが逆順に実行されます。



これに対し、V10 のコントロールハンドラでは、カーソルの移動の方向に関わらず、つまり上から入る場合にも下から入る場合にも、コントロール前処理が実行され、出る場合には上に行くときも下に行くときも、コントロール検証とコントロール後処理とが実行されます。また、各ハンドラ中のコマンドはいつも上から下に実行され、レコードメインの時のように、逆方向に下から上に実行されることはありません。

コントロールハンドラの場合：



※ コントロール後処理、検証も同様

この例外として、「エラー」コマンドを「R=復帰」モードで実行する場合があります。このときには、エラーコマンドが実行された後、エラーコマンドからハンドラの最初のコマンドまでが逆方向に実行されます。これは旧バージョンとの互換性を持たせるための機能で、詳細はリファレンスヘルプの　ロジックエディタ > 処理コマンド > エラー　を参照してください。

4.3.2　カーソル移動の原因

旧バージョンのレコードメインでは、カーソルが移動する原因（Tab/Enter キー、マウスクリック、↑ ↓ キー、ESC キーによるタスククローズ・・・）によって、フローモード（通常モード、高速モード）が区別されていました。

V10 のコントロールハンドラでは、カーソルの移動の原因に関わらず、例えば Tab/Enter キーで入る場合にも、マウスクリックで入る場合にも、コントロール前処理が実行され、コントロールから出る場合には、キーの種類やマウスクリック、その他の要因の如何に関わらず、コントロール検証、コントロール後処理が実行されます。

	旧バージョンのレコードメイン	V10 のコントロールハンドラ
カーソルの移動方向	影響される	関係ない
カーソル移動の原因	フローモードに影響	関係ない

旧バージョンとの互換性のために、V10 のハンドラの中でもコマンド単位でフローモードを指定できるようになっています。具体的な例は 5.1「例① リンクの成功チェック」で説明します。リファレンスマニュアル ロジックエディタ > 処理コマンド から、各コマンドの特性から、「フローモード」「フロー方向」に関するページを参照してください。

第5章 コントロールレベルハンドラの例

本章では、具体的な例をあげて、コントロールレベルハンドラへ移行する例を見ていきます。

レコードメイン→コントロールハンドラの移行ポイントとしては、次のようなことがあります。

前章の説明にあったように、レコードメインとコントロールレベルハンドラとは1対1に対応しているものもあるけれども、対応していないものも数多くあります。特に、カーソルの移動方向に関して、レコードメインは実行タイミングが大きく依存するのに対し、コントロールレベルハンドラは依存しませんので、カーソルの移動方向に依存した処理をレコードメインで記述していた場合には、コントロールレベルのハンドラに移行しようとするときに面倒なことになります。

レコードメインからコントロールレベルハンドラに機械的に変換しようとすると、一見すると必要以上に複雑で冗長な結果となってしまい、また、100%の完全な変換は根本的には不可能であるため、いずれにしても手作業が入ってしまうようになります。

このため、レコードメインからコントロールレベルハンドラに移行しようとする場合には、機械的に変換することよりも、「本来何をしたいのか？」の原点に立ち帰って、ハンドラ設定を考える方がはるかに良いプログラムとなります。

次節以下で、レコードメインからコントロールレベルハンドラへの移行について、いくつか例を挙げて説明していきますので、これを参考にして、プログラムを見直していってみてください。

以下に説明する内容は「考え方」を説明するために簡略化したもので、レコードメインの複雑な詳細に依存した作りになっている場合には、全く同じ動きにすることが困難な場合もあります。

5.1 例① リンクの成功チェック

最初の例では、「『商品＃』で商品マスタへのリンクを行い、マスタに登録されていない商品番号が入力された場合にはエラーメッセージを表示させる」という、リンクの成功チェックを行うことを考えて見ます。

5.1.1 レコードメインでのやりかた

レコードメインの書き方 ABC「どこに？いつ？なにを？」に当てはめて考えてみると、次のようになります。

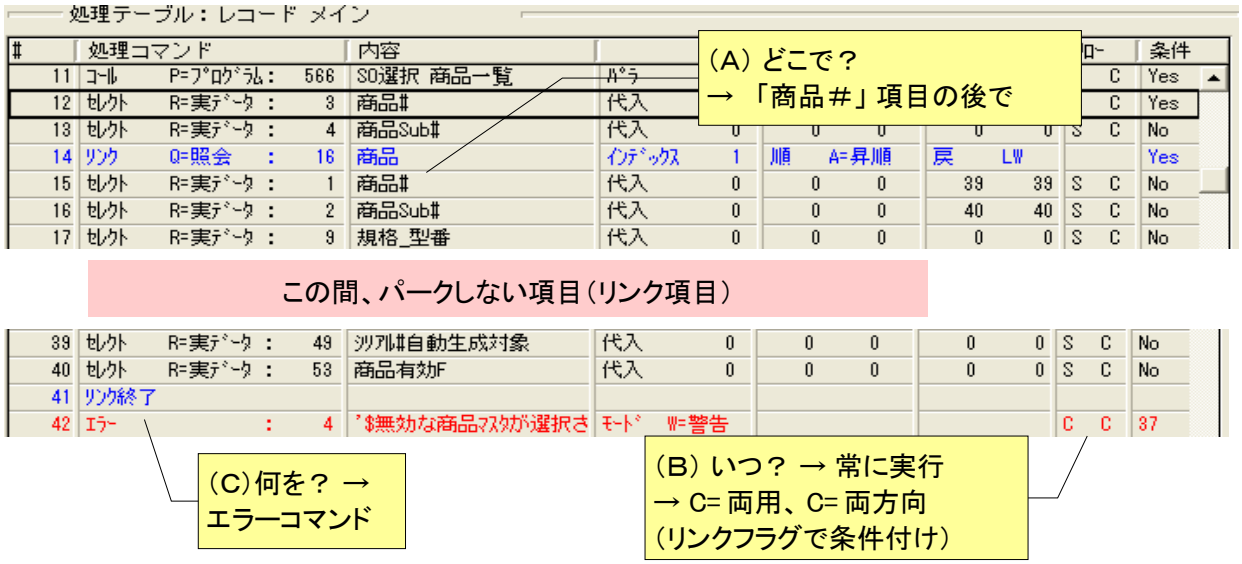
A) どこに？ 「商品＃」項目から次の項目に行くところで。

B) いつ？ カーソル移動の原因に関わらず（つまり、Enter/Tab キーでカーソルが移動する場合にも、マウスで移動する場合にも、ESC でタスクを終了させようとする場合でも）、リンクに失敗した場合には、常に実行しなければなりません。

C) 何を？ 商品＃が不正の由、エラーメッセージを表示します。（リンク結果を見て、条件付けします）

プログラム上では、次のようになります。

レコードメインでのやりかた



- レコードメインでは、リンク項目「商品＃」を使ってリンクを行い、リンクの成功を戻り変数（下図では、14 行目のリンクコマンドの「戻」に設定されている変数 LW ）に格納しています。
- 「どこで？」が「『商品＃』項目から次の項目に行くところで」なので、「商品＃」のセレクトコマンドと、次にパーク可能なセレクトコマンドの間に書くことになります。
- 「何を？」が「商品＃が不正の由、エラーメッセージを表示」なので、「エラー」コマンドを使います。これは、「リンク終了」コマンド（41 行目）の後に設定しています。このエラーコマンドには、「条件」欄に式 37 番が設定されていますが、これは戻り変数 LW でリンク成功をチェックする式です。

  リンク項目「商品＃」（12 行目）から、エラーコマンド（42 行目）まで、多くのセレクトコマンドがありますが、これらはすべてパークしない（「条件」欄が No になっている）ため、「商品＃」から Enter/Tab で次に進むと、エラーコマンドがすぐに実行されます。
- 「いつ？」は「カーソル移動の原因に関わらず常に実行」なので、フローモードは「C=両用」となります。

5.1.2 コントロールレベルハンドラでの書き方

これを V10 のコントロールレベルハンドラで書き直すと、次のようになります。

まず、コントロールレベルハンドラの書き方 ABC「どこで？いつ？何を？」に当てはめてみると、

A) どこで？ 商品＃のエディットコントロールで

B) いつ？ このコントロールを通り過ぎるとき（カーソルがパークしてもしなくとも）常に。

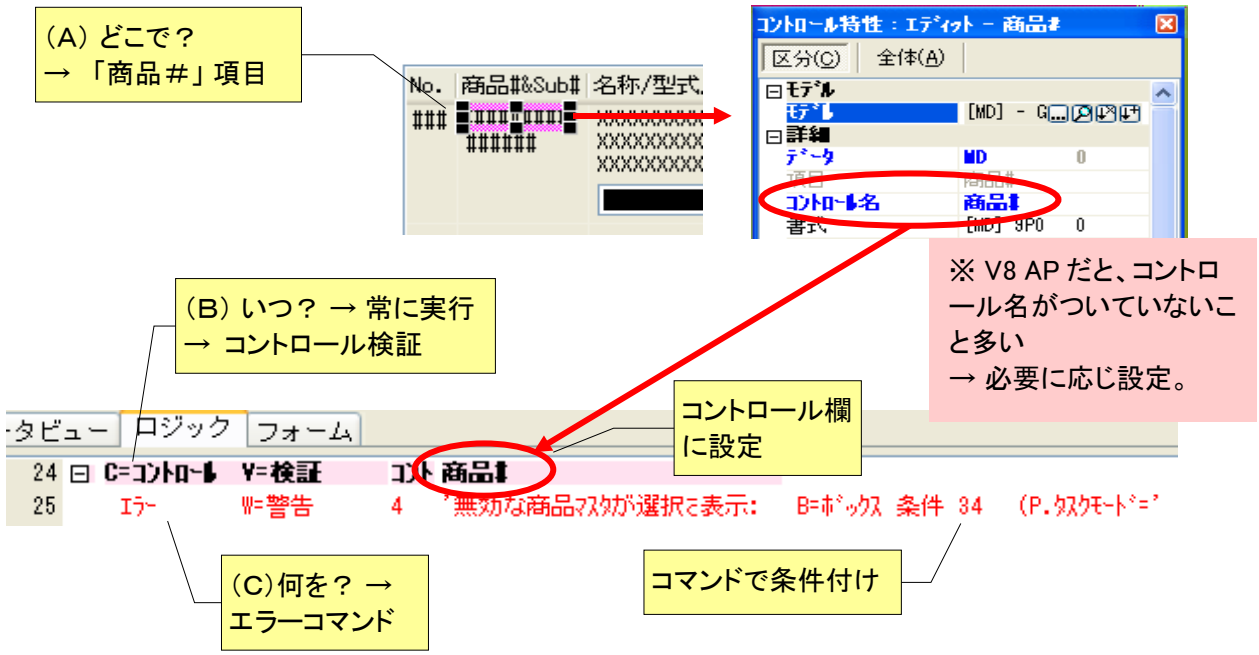
C) 何を？ 商品＃が不正の由、エラーメッセージを表示します。（リンク結果を見て、条件付けします）

となります。

従って、プログラム上は次のようになります。

- データビュー画面において、リンク項目「商品＃」を使ってリンクを行い、リンクの成功を戻り変数に格納します。
- 「いつ？」は「コントロールを通り過ぎるとき常に」ですので、コントロール検証 ハンドラを使います。
- 「どこで？」が「『商品＃』のエディットコントロールで」ですので、このコントロール検証ハンドラの「コントロール名」欄には、コントロール名「商品＃」を指定します。
- コントロール検証ハンドラの詳細行には、「何を？」に相当するエラーコマンドを指定します。エラーコマンドの「条件」で、リンク結果を格納する変数を使って条件付けします。

V10 でのやりかた



V9 以降でプログラムを作成した場合には、コントロール名は自動的にデフォルトで項目名と同じものがつけられますが、V8 ではデフォルトで付けられないので、V8 からの移行アプリケーションではコントロール名が付いていない場合が多いです。この場合には手作業でコントロール名を指定してください。

5.1.3 微妙な動作の違い

レコードメインの場合、エラーコマンドは「商品＃」のセレクトコマンドと、次にパークする項目のセレクトコマンドの間に書いたので、次項目に進む場合はエラーチェックがされましたが、前の項目に戻る場合にはエラーチェックがされませんでした。このため、間違って「商品＃」に行ってしまったときにも、カーソルを元に戻してやりなおし、ということができました。

一方、コントロール検証の場合には、カーソルが「商品＃」から出るときにはカーソルの移動方向に関係なく（前方向でも後方向でも）エラーチェックを行うようになります。従って、いったん「商品＃」に入ったら、正しい値を入力するまでは前にも後ろにも行けない、ということになってしまいます。

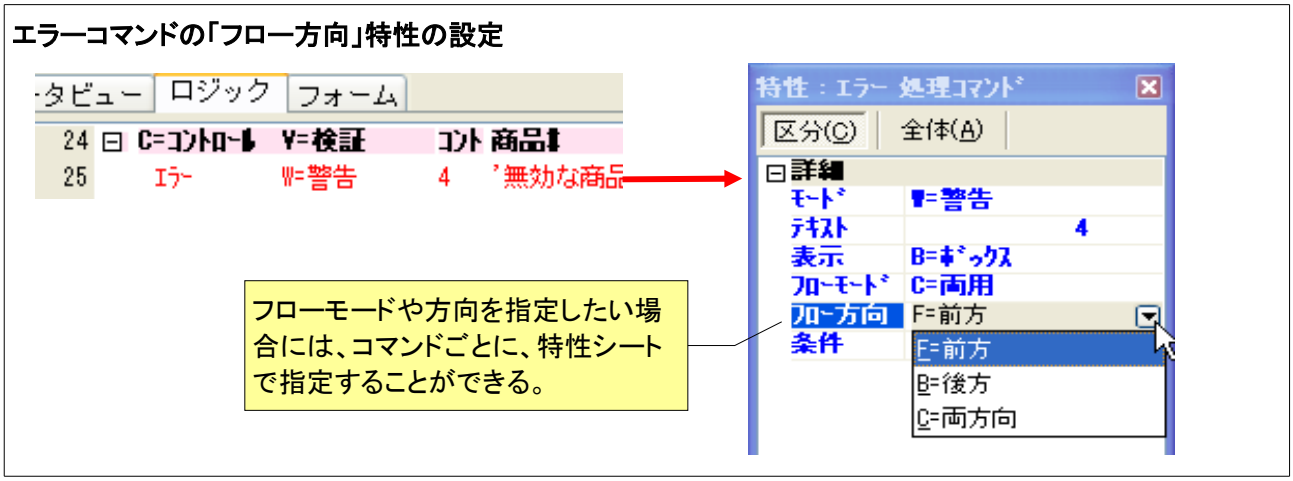
このような問題に対応するため、また旧バージョンとの互換性を保つため、V10 でも、旧バージョンのフローパラメータの概念が残っており、各コマンドごとに「フローモード」と「フロー方向」を設定できるようになっています。

- 「フローモード」パラメータの設定値
  - S=通常
  - F=高速
  - C=両用
- 「フロー方向」パラメータの設定値
  - F=前方
  - B=後方
  - C=両方向

それぞれのパラメータの意味は、レコードメインの場合と同じです。

例えば、上記のリンクエラーチェックの問題に対応するには、カーソルが前方に移動する場合にだけチェックするようにすればよいので、エラー コマンドの「フロー方向」パラメータを「F=前方」に設定してやればよいことになります。

エラーコマンドの「フロー方向」特性の設定



エラーコマンドのフロー方向特性を設定するには、次のようにします。

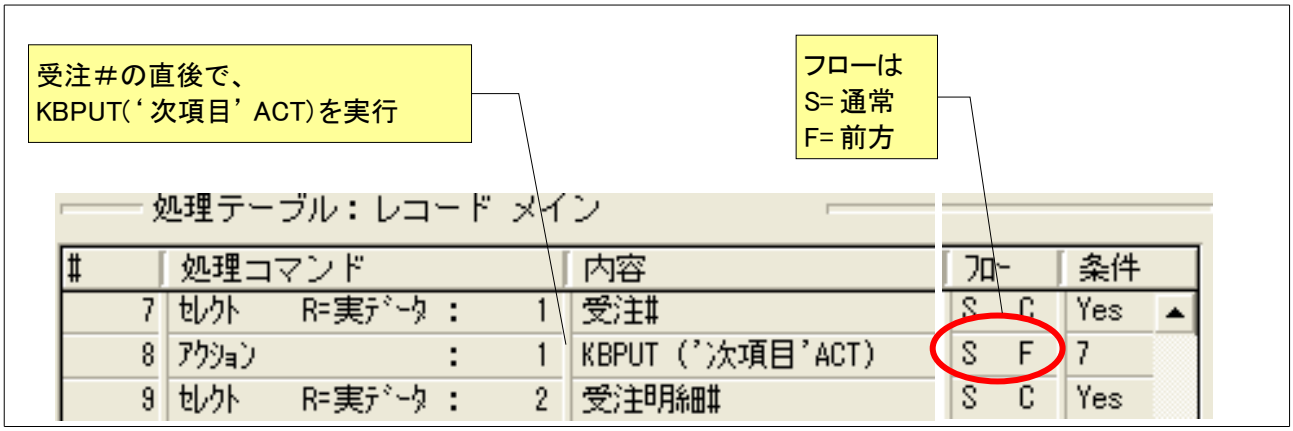
1. 詳細行の「エラー」コマンドにカーソルを置きます。
2. 特性シートを開きます（Alt-Enter）。→ エラー処理コマンドの特性が表示されます。
3. 「フロー方向」特性を「F=前方」に設定します。

「フロー方向」や「フローモード」特性の他に、Flow という関数があり、現在のフローモードをチェックすることができますので、条件式に使うことができます。Flow 関数の詳細については、リファレンスヘルプ 式エディタ > 関数ディレクトリ > Flow を参照してください。

5.2 例② 受注番号の後、条件により次の項目をスキップ

5.2.1 レコードメインでのやりかた

下図は、V8 で作成したタスクのレコードメインの一部です。



ここでは、「受注＃」から「受注明細＃」に移動するときに、式7番で定義される条件が成立した場合に、KBPUT('次項目') を実行します。条件式7番の内容についてはここでは立ち入りませんが、ここで、やりたい意図は何かというと、

- 受注＃から Enter /Tab で移動するとき、条件式 7 番 が成立したら、次の項目（受注明細#）をスキップする。

ということです。フローパラメータの設定を見てみると、

- 「S=通常モード」となっているので、Enter/Tab キーで移動する場合だけ実行し、マウスやレコード移動時には実行しない。
- 「F=前方」となっているので、カーソルが前方向に移動する場合にだけ実行し、逆方向に(「受注明細#」から「受注#」に）戻る場合には実行しない。

ということになります。

5.2.2 コントロールレベルハンドラでのやりかた

これと同じことをコントロールレベルハンドラで実現しようとするとどうなるでしょうか？「どこで？いつ？何を？」で考えてみましょう。

A） どこで？ 「受注＃」のエディットコントロールにおいて。

B） いつ？ カーソルがパークした後、別項目に出て行く時。（ただし、前方向に進む場合のみ）

C） 何を？ 項目をひとつスキップさせる。（条件式で条件付けする）

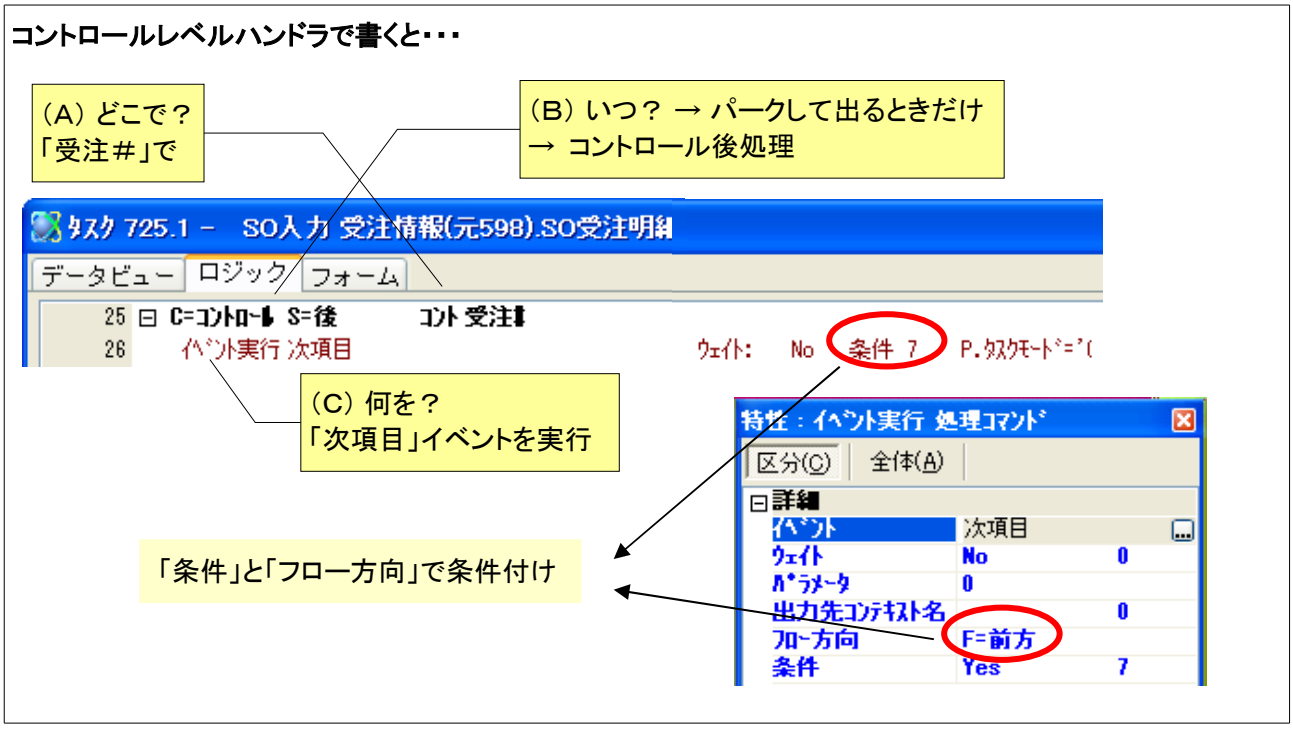
ということになります。

従って、コントロールレベルハンドラで書くと、次のようになります。

- 「いつ？」は「カーソルがパークした後、別項目に出て行く時」なので、コントロール後処理 ハンドラに書きます。
- 「どこで？」は「『受注＃』のエディットコントロールにおいて」なので、ヘッダ行の「コントロール」欄には、「受注＃」 を設定します。
- 「何を？」は「項目をひとつスキップさせる」なので、**イベント実行** コマンドで「次項目」イベントを発生さ

せます。

- 前方向にカーソルが移動するときにだけ実行させたいので、イベント実行 コマンドの「フロー方向」特性に「F=前方」を設定します。



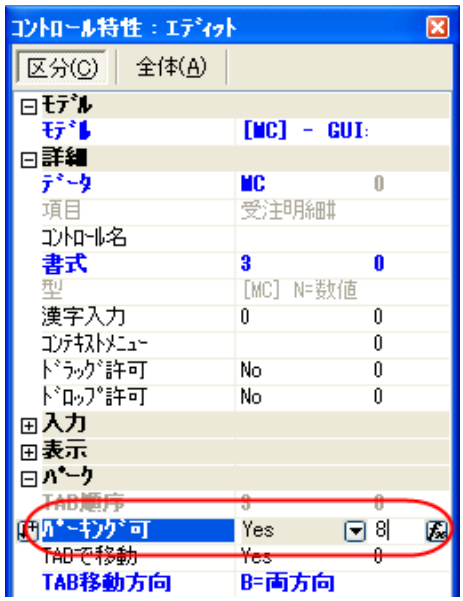
V10 でも KBPUT 関数はありますが、イベントを発生させるためには、イベント実行コマンドを利用することを推奨します

5.2.3 別のやりかた

このように、レコードメインで行っていたことをコントロール後処理で書き換えることが可能ですが、この例についてもっとよく考えてみると、本質的には「受注明細＃」のパークの有無を制御したい、ということなので、「次項目」イベントを発行するよりは、そのものずばり、「受注明細#」項目のパーク条件で制御した方がよいと言うことができましょう。

パークの有無は、レコードメインではセレクトコマンドの条件として記述していましたが、V10 ではフォームエディッタで、コントロールの「パーキング可」特性に条件を記述します。(右図)

このように、レコードメインに書いていた方法を V10 で書き直す場合、コントロールレベルのハンドラにする以外に、全く別の方法によるより直接的な方法もありえます。V10 にするときには、別のよりよい方法がないかを検討する良い機会でしょう。



5.3 例③ 項目更新による再計算ロジック

次の例は、「受注数量」の値をもとに、「受注金額」と「粗利金額」とを再計算するロジックを実現するものです。再計算は、「代入式」を使えば自動的に行うことができるのですが、場合によっては、計算の条件などがあったりして、代入式ではなく、条件付きの項目更新コマンドで行うことがよくあります。ここでは、代入式ではなく、項目更新コマンドで行う場合を考えて見ます。

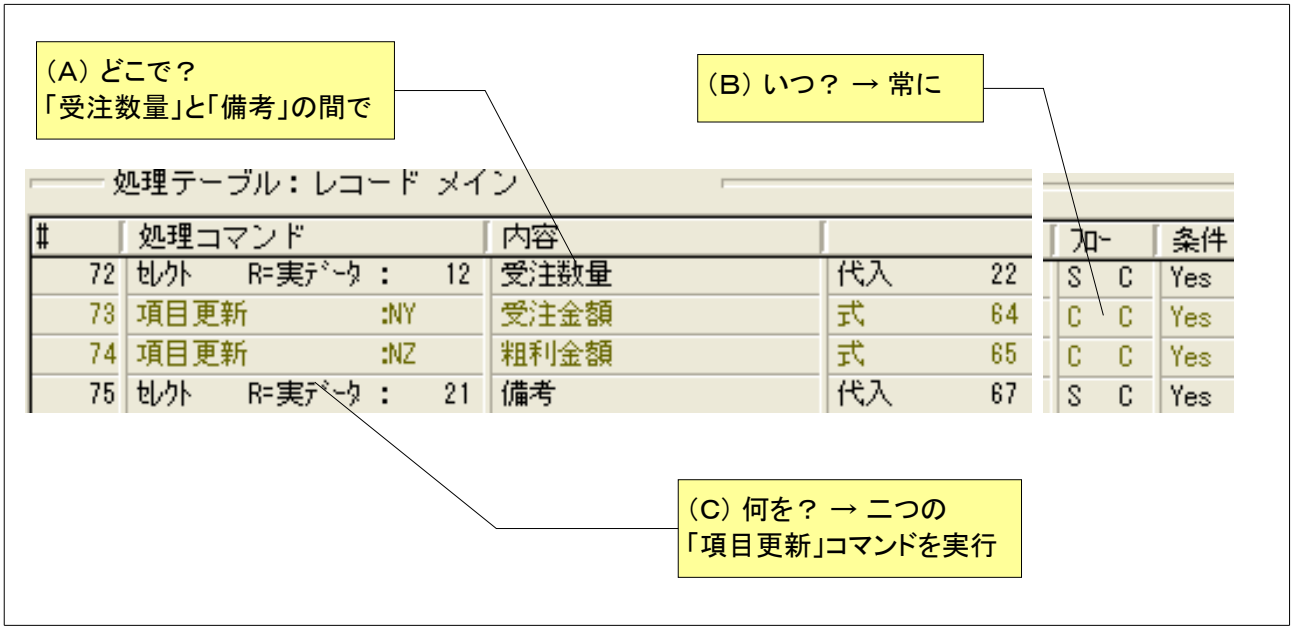
5.3.1 レコードメインでのやりかた

レコードメインを使ったときには、次のように設計しました。

A) どこで？「受注数量」と、次のパーク項目である「備考」の間で実行しますので、それぞれに対応するセレクトコマンドの間に記述します。

B) いつ？ 項目更新コマンドは、フローモードやパークの有無に関わらず常に実行しなければなりません。従って、フローモードは「C=両用」とします。

C) 何を？「受注金額」と「粗利金額」を計算する項目更新コマンドを実行します。



5.3.2 コントロールレベルハンドラでのやりかた

コントロールレベルハンドラを使う場合には、

A) どこで？ 「受注数量」において行います。

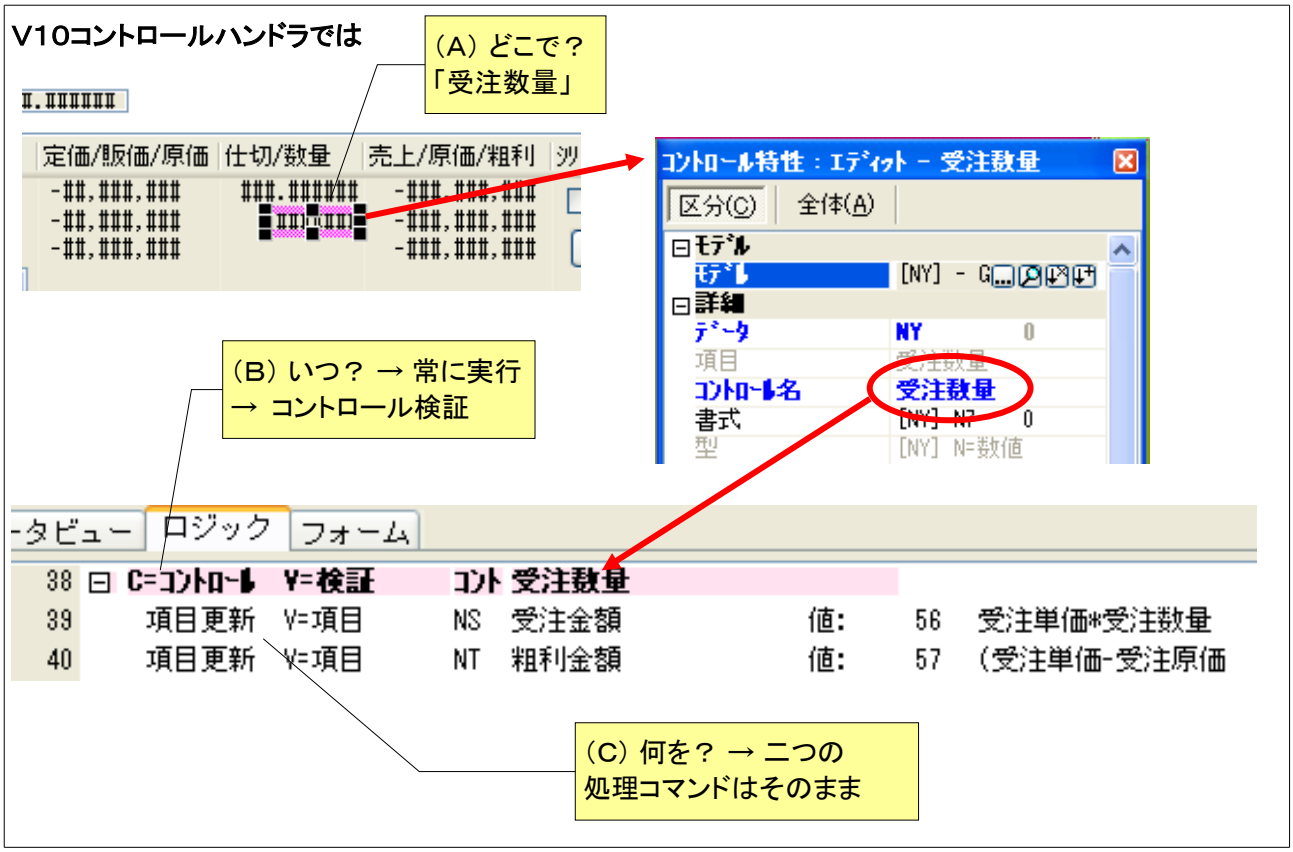
B) いつ？ 項目を通り越した時点で。(カーソルのパーク有無には関係なく)

C) 何を？「受注金額」と「粗利金額」を再計算します。

としたいのですから、次のように書きます。

- 「いつ？」は「カーソルのパーク有無には関係なく 項目を通り越した時点」ですから、コントロール検証ハンドラにします。
- 「どこで？」は「『受注数量』において」ですから、ヘッダ行の「コントロール」欄には、コントロール名「受注数量」を指定します。
- 「何を？ 」は「「受注金額」と「粗利金額」を再計算」ですから、レコードメインの場合と同様、項目更新コ

マンドで再計算を行います。



5.3.3 微妙な動作の違い

この例でも、レコードメインで行った場合と、コントロールレベルハンドラで行った場合とでは、表面上は出てこないけれども、微妙に動作が違うところがあります。

二つの項目更新コマンドの実行順序

第一には、「受注金額」の計算の項目更新コマンドと、「粗利金額」計算の項目更新コマンドの実行される順序についての違いです。

レコードメインの時は、カーソルの移動方向により、コマンドの実行順序が異なりました。すなわち、カーソルが順方向(「受注数量」から「備考」へ)に動く場合には、上から下に実行され、「受注金額」が先に計算された後に「粗利金額」が計算されました。一方、カーソルが逆方向(「備考」から「受注数量」へ)に動く場合には、逆に、下から上に実行され、「粗利金額」が先に計算されてから、「受注金額」が計算されます。

従って、もし「粗利金額」の計算が、「受注金額」の値に依存するような形で計算式が作られていれば、項目更新コマンドの実行順序によって計算結果が変わってくることになります。すなわち、カーソルの動きが順方向か逆方向かで、計算結果が異なることになります。

ここで使った例では、たまたま、お互いの依存関係がないように計算式が立てられていたので、どちらを先に実行しても結果は変わりませんでしたが、一般的には、開発者は上から下に実行することを大前提として処理を記述していることが通常で、逆方向の動きについては考慮し忘れることが間々あります。

レコードメインに書いた場合には、このように、上から下に処理が進む場合と下から上に処理が進む場合とがあるので、開発者は常に、どちらから実行がされても、結果に矛盾が起こらないように考慮を払いながらプログ

ラムを作らなければならないことになります。これが、レコードメインで書く場合の難しさを増やす要因の一つともなっていました。

コントロール検証の時は、カーソルの移動方向に関わらず、常に上から下に実行されますので、逆方向の動きに煩わされることなく、プログラムを作成することができます。

実行タイミング

レコードメインの場合には、ユーザが「受注数量」で数量を入力した後、Enter キーで「備考」へカーソルを動かすことを念頭においており、そのタイミングで「受注金額」と「粗利金額」を計算していました。

ところが、ユーザが「受注数量」を入力した後、先に進まずに、前の「仕切」項目に戻ったとすると、項目更新コマンドは実行されませんので、「受注数量」および「粗利金額」はもとのままです。

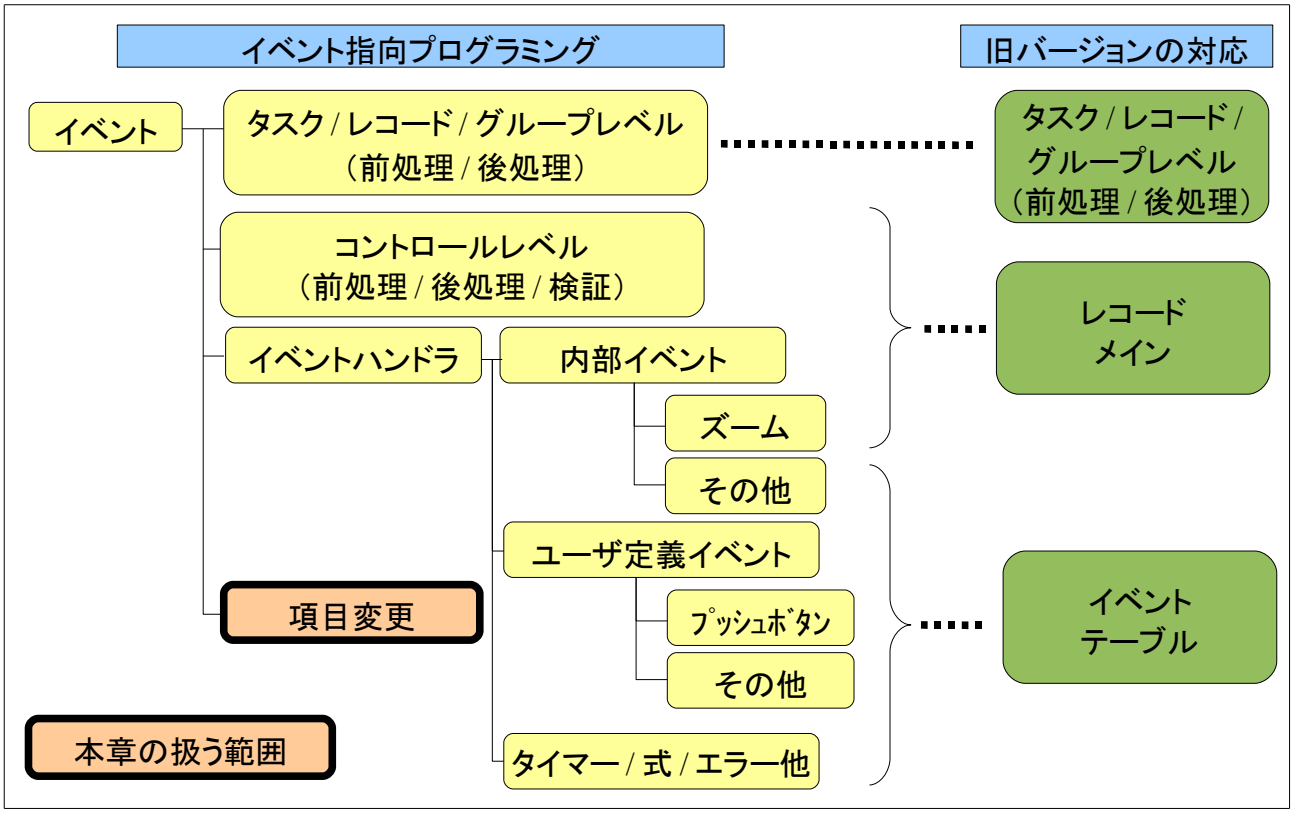
最終的には、タスクを終了するか、次レコードに移るタイミングで（F=高速モードで）この項目更新コマンドが実行されるため、データベースには正しい値が格納されるのですが、ユーザの画面表示上ではそのことが見えないままとなってしまいます。

コントロール検証で書いた場合には、カーソルの移動方向に関わらず、「受注数量」項目を出るタイミングで項目更新コマンドが実行されますから、「備考」に進む場合も、「仕切」に戻る場合も、常に「受注金額」と「粗利金額」が更新されます。

第6章　項目変更ハンドラ

前章では、コントロールレベルハンドラについて説明しました。本章以後では、それ以外のハンドラについて説明していきます。

まず、本章では、「項目変更」ハンドラについて説明します。



6.1 項目変更ハンドラとは？

項目変更ハンドラというのは、ある特定の項目の値に注目して、その値に変更があった場合に実行されるハンドラのことです。この場合、値の変更の理由としては、ユーザの入力による場合、項目更新コマンドや VarSet 関数の実行による場合、コールプログラムのパラメータとして渡され呼び出し先のプログラムで変更があったため、などがありえますが、項目変更ハンドラは変更の理由に関わらず、値に変化のあった場合に常に実行されます。

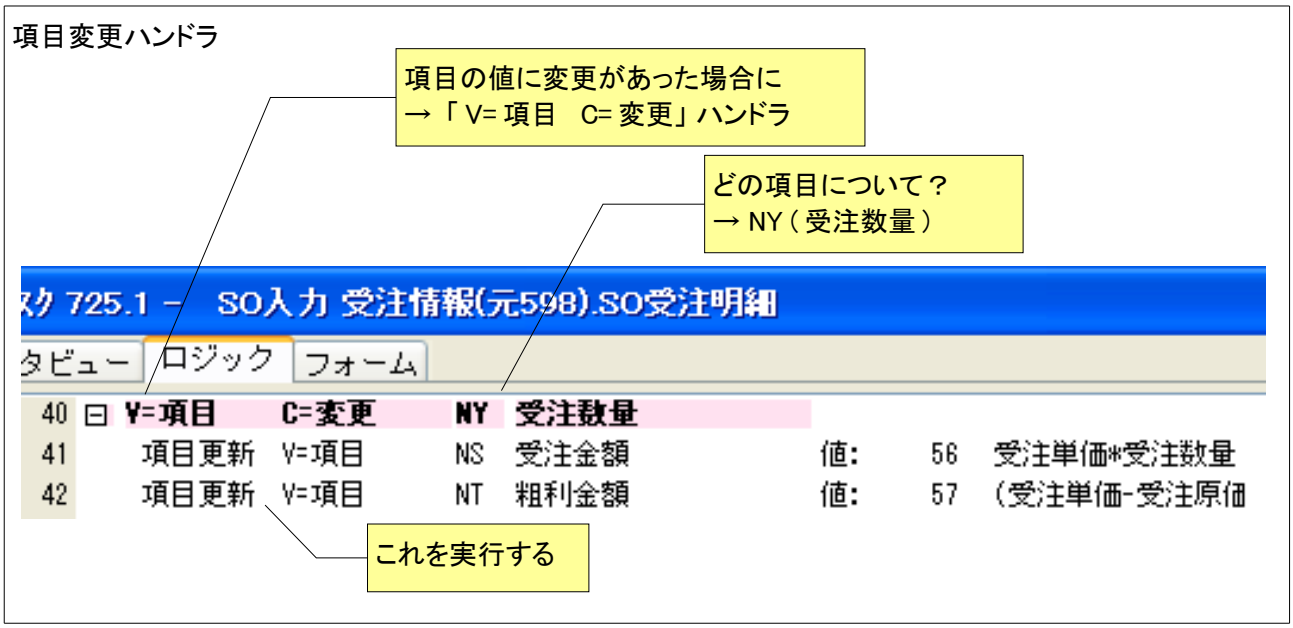
V9/V9Plus においては、「**コントロール変更**」というハンドラがありました。これはオンラインフォーム上で、ユーザ入力によって値が変更された場合に実行されるハンドラでした。V10 の項目変更ハンドラと似ていますが、V10 の項目変更ハンドラとは異なり、ユーザ入力による値の変更の場合にだけ実行されます。項目更新コマンドによる値の変更などの場合には、コントロール変更ハンドラは実行されませんでした。

V10 では項目変更ハンドラがあるので、コントロール変更ハンドラはなくなりました。ただし V9/V9Plus でのコントロール変更ハンドラとの互換性をとるために、変更の理由（ユーザ入力によるものか、それ以外か）を、ハンドラのパラメータとして渡すようになっています。詳しくは、リファレンスマニュアル Magic エンジン > エンジン実行レベル> 項目レベル を参照してください。

6.2 再計算ロジックの例

前章 5.3「例③ 項目更新による再計算ロジック」では、「受注数量」のコントロール検証イベントハンドラを使って、受注金額と粗利金額とを計算させていましたが、このようなあるデータ項目の値の変更をきっかけとして実行すべきコマンドは、「項目変更」ハンドラにした方がわかりやすくなります。

項目変更ハンドラは、次のように記述します。



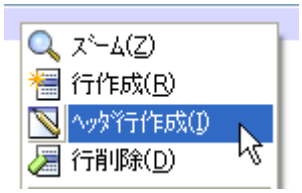
1. 項目変更ハンドラでは、ヘッダ行で「V=項目」、「C=変更」を指定します。
2. 変数項目を指定する欄がありますが、ここには、値の変化を監視すべき変数のシンボルを設定します。
   この例では、「受注数量」（シンボルは NY）を指定しています。
3. ハンドラの明細行には、実行すべき処理コマンドを記述します。
   この例では、「受注金額」と「粗利金額」を計算する項目更新コマンドを記述しています。

このように設定しておけば、項目 NY（受注数量）の値に変化があった場合にはいつでも、この二つの項目更新コマンドが実行されます。

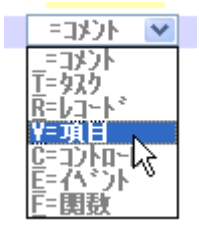
項目変更ハンドラ作成の手順

上の例の項目変更ハンドラは、次のように作成します。

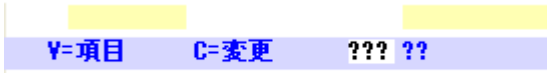
1. ロジック タブをクリックします。→ ロジック画面が開きます。
2. コンテキストメニューから「ヘッダ行作成」(Ctrl+H) を選びます。→ ヘッダ行が作成されます。



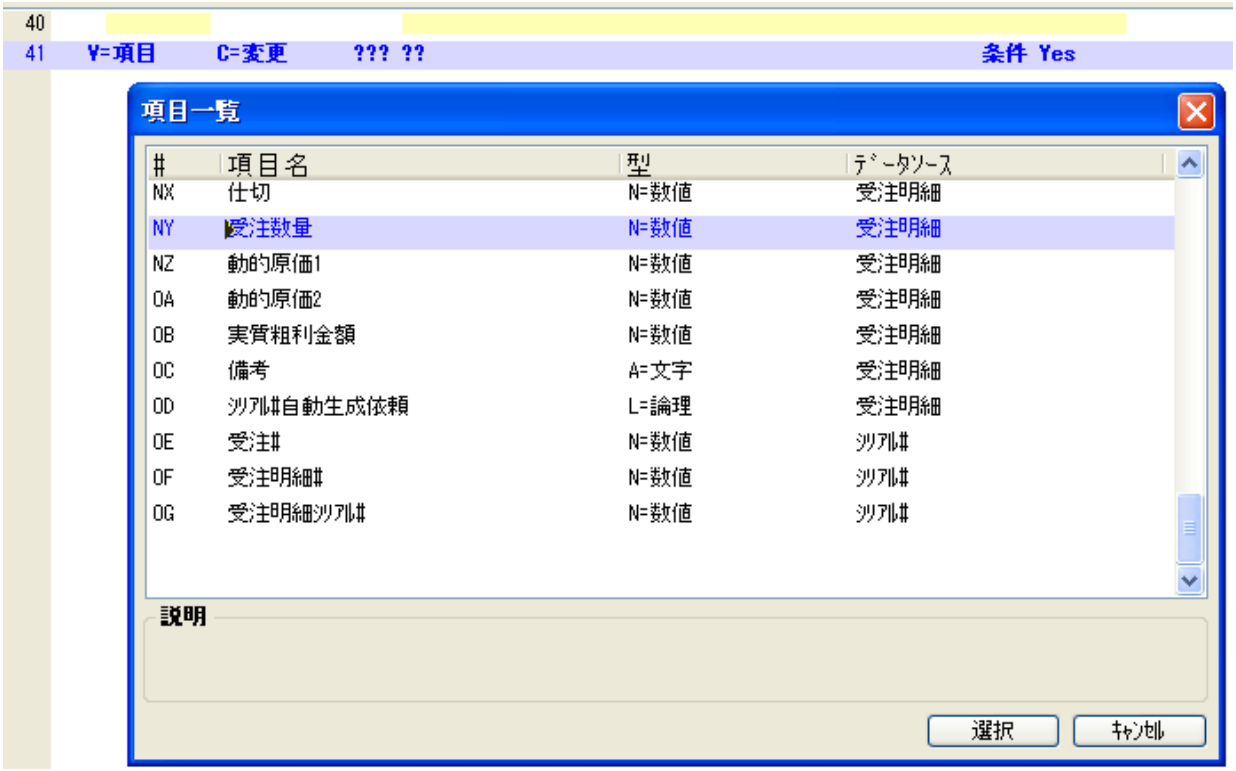
3. ハンドラの先頭のコンボボックスから「V=項目」を選択します。



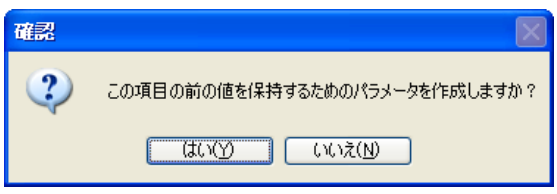
→「V=項目」が設定されるとともに、タイプが「C=変更」に自動的に設定されます。



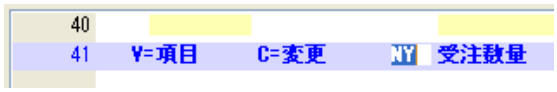
4. 変数欄に進み、ズームします。→ 変数一覧が表示されます。



5. 「受注数量」(NY) を選択します。
→ 右図のようなダイアログが出ます。

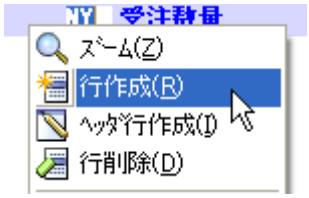


6. この例では、「いいえ(N)」で答えてください。
   → ヘッダ行の変数欄に、NY が設定され、その右に変数名「受注数量」が表示されます。
   これでヘッダ行の設定は終了です。

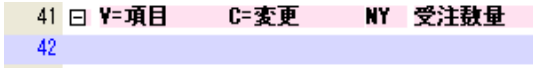


次に、詳細行の記述をします。
詳細行では処理コマンドを記述します。処理コマンドの設定方法は、旧バージョンの場合とほとんど同じです。
例えば、項目更新コマンドを設定する場合には次のようにします。

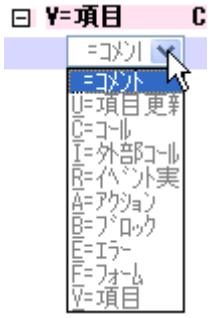
7. ポップアップメニューから「行作成(R)」(F4)を選びます。



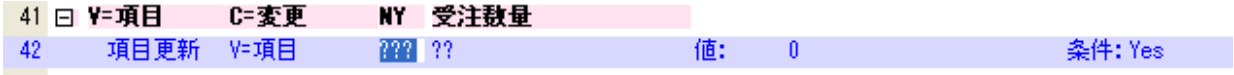
   → 空の詳細行が 1 行できます。



8. 詳細行の先頭で、ズーム(F5 またはダブルクリック)します。
   → コマンドのコンボボックスが現れます。



9. 「U=項目更新」を選びます。
   →「項目更新」が設定されると共に、項目更新コマンドのパラメータ入力欄が現れます。
   ※ コンボボックスから選ぶ代わりに、文字「U」をキー入力しても、項目更新コマンドが設定されます。



10. 更新する項目、式、条件等を設定します。

11. 同様にして、2 つ目の項目更新コマンドも設定します。

6.3 項目変更ハンドラの利点

項目変更ハンドラを使うと、コントロールレベルのハンドラを使うことに比べて次のような利点があります。

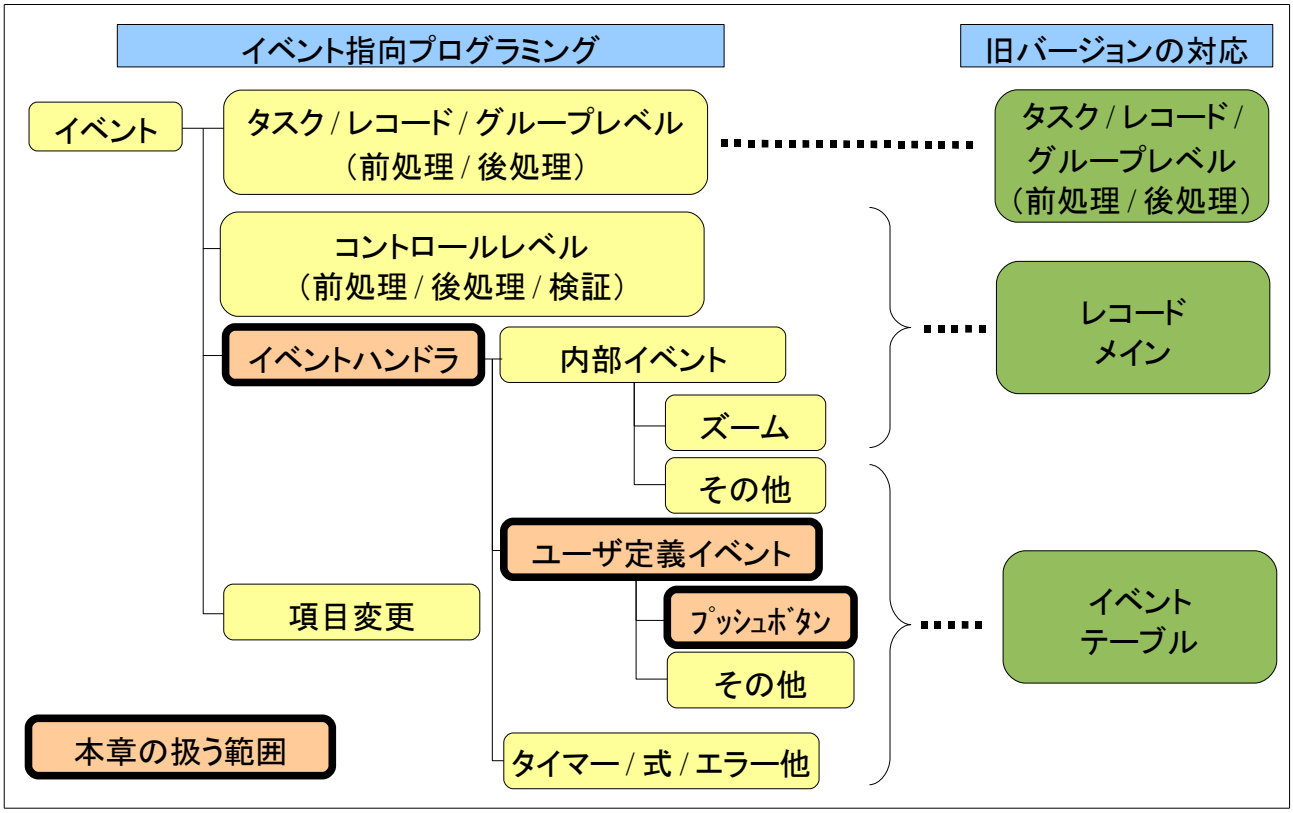
- 意図がわかりやすい。ここに挙げた例では、開発者の意図としては、「受注数量をもとにして受注金額と粗利金額を常に正しい値に更新しておきたい」というものがあります。この意図をプログラム上で表現するときに、「『受注数量』コントロールを通りすぎるときにこれこれをせよ」という意味のコントロール検証ハンドラよりは、「『受注数量』に変化があったときに、これこれを実行せよ」という意味の項目変更ハンドラのほうが、開発者の意図をより直接的に明確に反映していると見ることが出来ます。これにより、プログラムを作りやすくなるし、読む側も意図を明瞭に知ることができるようになります。
- 設定忘れがなくなる。項目変更ハンドラは、指定された項目の値に変更があった場合には、いつ、どこで、どのようにして（例： ユーザ入力、項目更新コマンド、コールプログラムなど）変更されたかに関わりなく、常に実行されます。そのため、何箇所かで変更される可能性がある項目の場合には、同じ再計算ロジックをあちこちに繰り返す必要がなく、また、開発者にとっても、漏れが出る可能性がなくなります。
- 値の変化がない場合には実行されない。コントロールレベルのハンドラを使ったやりかたでは、「受注数量」の値の変化の有無に関わらず、項目更新コマンドが常に実行されます。ここの例のように、四則演算だけの計算ならば、毎回計算してもオーバーヘッドは無視できる程度ですが、時間のかかるプログラムの呼び出しなどを伴う処理であればオーバーヘッドは無視できない場合もあります。項目変更ハンドラでは、値に変化のない場合には、ハンドラが実行されません。例えば、ユーザがカーソルを「受注数量」に移動したのち、そのまま別の項目に移動した場合、あるいは、ユーザが「受注数量」にデータを入力しても、前と同じ値だった場合などには、ハンドルが実行されません。従って、再計算は必要最低限にだけ限られることになり、オーバヘッドも最小にすることができます。

項目更新ハンドラは Magic V10 に備わった強力な機能です。ぜひ活用して簡潔明快なプログラムを作るようにしてください。

第7章 プッシュボタン

本章と次章では、イベントハンドラを使った例を説明します。

イベントハンドラを利用する最も一般的なケースはプッシュボタンです。プッシュボタンを押したときに、ある処理を行う、というユーザインターフェースは、昨今の GUI インターフェースでは非常に多用されますので、本章ではまず、プッシュボタンについて説明をしていきます。



7.1 旧バージョンでのやりかた

旧バージョンでは、プッシュボタンの処理を行うのに、タスクイベントテーブルをよく使っていました。

例えば、「ｼﾘｱﾙ＃確認」というプッシュボタンを押したときに、ｼﾘｱﾙ＃確認プログラムを呼び出す、というプログラムを考えてみます。より正確には、「どこで？いつ？何を？」に従って書いてみると、次のような仕様になります。

A） どこで？「ｼﾘｱﾙ#確認」というラベルのあるプッシュボタンにおいて。

B） いつ？ プッシュボタンが押されたとき。

C） 何を？ ｼﾘｱﾙ＃確認処理を行う。

これを実現するためのプログラムは、内部アクション「ユーザアクション１」を使って、

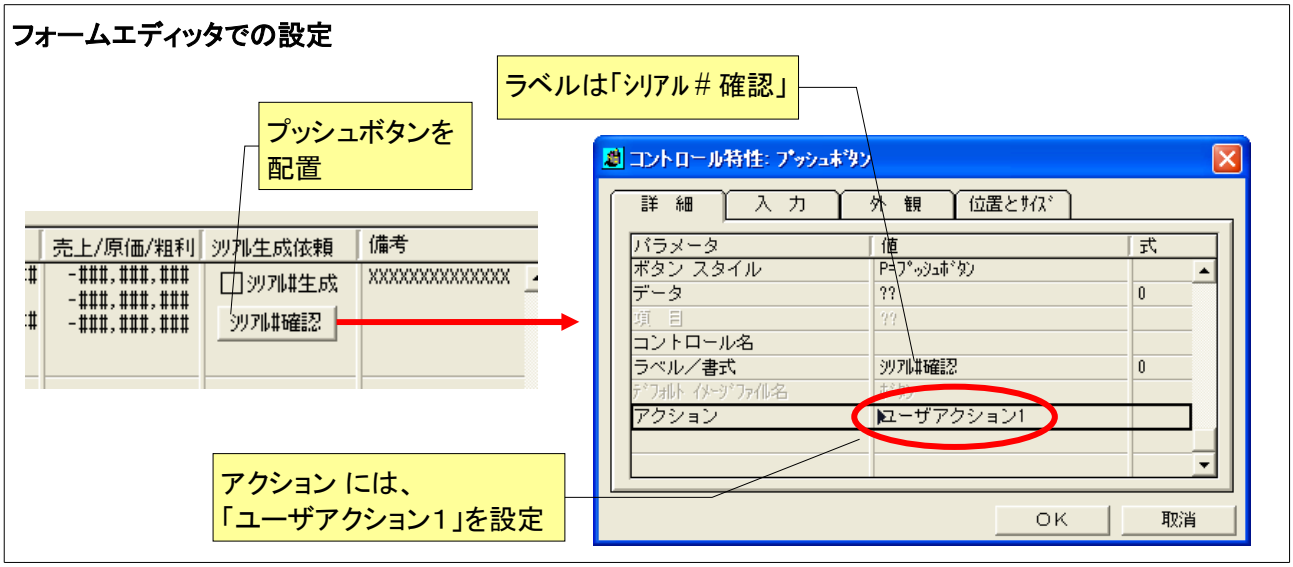
- フォーム上のプッシュボタンの設定：実行時、ユーザがプッシュボタンを押したときに、「ユーザアクション１」を発生させる。
- イベントテーブルでの設定：ユーザアクション１が発生したときに、それをキャッチして、プログラム 588 番「SO 照会 ｼﾘｱﾙ#」を呼び出すようにする。

という 2 箇所での設定が必要です。

フォーム上のプッシュボタンの設定は、次の通りです。

- フォームエディッタ上にプッシュボタンを配置します。
- このプッシュボタンのコントロール特性を開きます。
- 「ラベル/書式」特性に「ｼﾘｱﾙ#確認」を設定します。
- 「アクション」特性に「ユーザアクション１」を設定します。

フォームエディッタでの設定

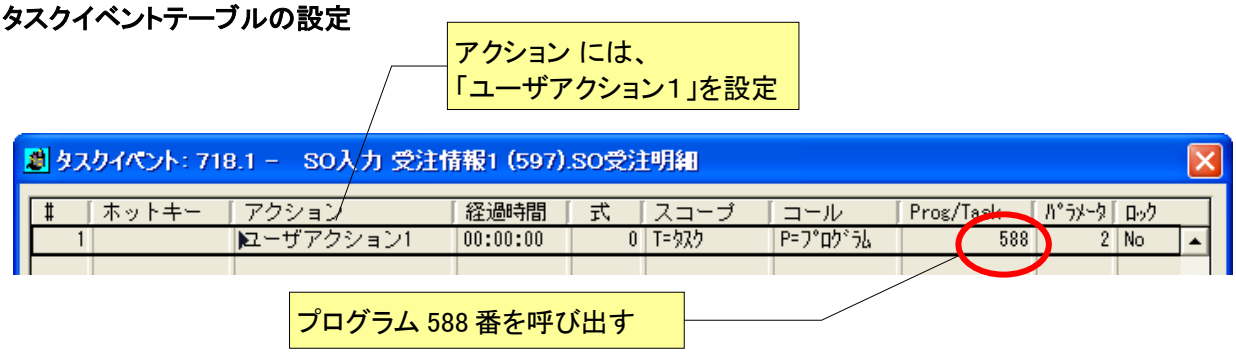


一方、タスクイベントテーブルでの設定は、次のようになります。

- タスクイベントテーブルを開きます。
- 「アクション」欄に、同じく「ユーザアクション１」を指定します。
- 「Prog/Task」欄には、呼び出すプログラムの番号（588 番）を指定します。

- 「パラメータ」欄には、呼び出すプログラムに渡すパラメータを設定します。

タスクイベントテーブルの設定



7.2 イベントハンドラでのやりかた

V8 でのタスクイベントテーブルは、「ベントが起こった場合に、どのプログラムを呼び出すか」という関連付けを定義するものですが、V10 ではこれを**イベントハンドラ** で行います。

今回の例で使うイベントとしては、V8 の場合と同じく「ユーザアクション1」などでも良いのですが、ユーザ定義イベントを使った方が、アプリケーションにとって意味のある名前でイベントを扱うことができ、プログラムがわかりやすくなるので、ここではユーザ定義イベントを使います。

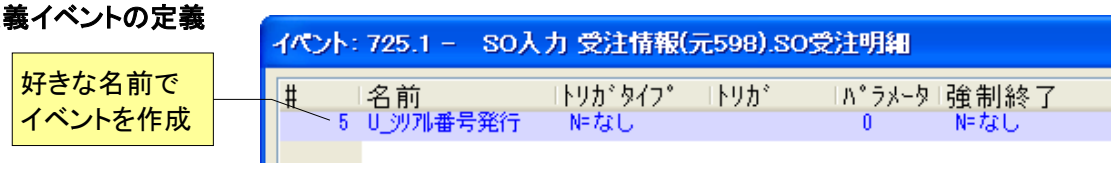
7.2.1 ユーザ定義イベントの利用

ユーザ定義イベントというのは、

- V9 で導入された機能で、V10 でもサポートされています。
- 文字通り、開発者が必要に応じて新しく作成することができるイベントで、自由な名前をつけることができます。
- V8 での「ユーザアクション *n*」と同様、それ自体では何の機能を持ちません。必ず、イベントハンドラと共に用います。
- タスクの**イベントテーブル**で定義します。
- ひとつのタスク内で定義できる数に制限はありません。
- スコープ（有効範囲）があります。あるタスクに定義したユーザ定義イベントは、そのタスクおよびその子孫タスクで有効です。ただし、「メインプログラム」(プログラム番号1番）に定義したユーザ定義イベントは、そのアプリケーション全体で有効なグローバルなものとなります。

例えば、次の図は、あるタスクのイベントテーブルに「U_ｼﾘｱﾙ番号発行」という名前のユーザ定義イベントを定義した例です。

ユーザ定義イベントの定義



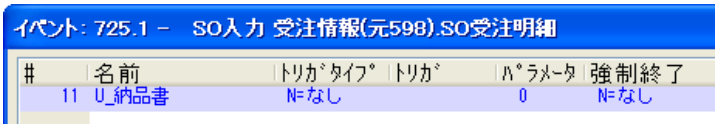
V8 の「（タスク）イベントテーブル」と、V9Plus/V10 での「イベントテーブル」とは、名前は同じですが内容は異なります。

V8 の「（タスク）イベントテーブル」というのは、上記の例のように、イベントが起こった場合に、どのプログラムを呼び出すか、という関連付けを定義するものです。

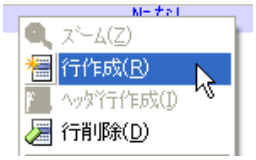
V9Plus/V10 での「イベントテーブル」は、ユーザ定義イベントを定義するものです。V9Plus/V10 で、イベントとそれに対する処理とを関連づけるのは、イベントハンドラになります。

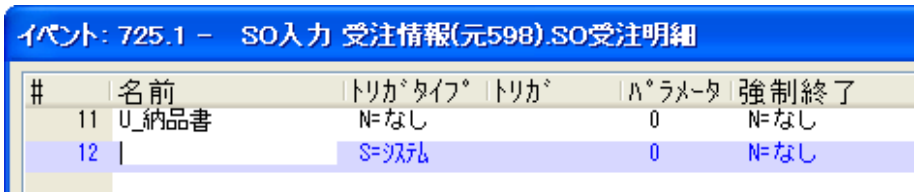
ユーザ定義イベントは次のようにして新規作成します。

1. タスクを開きます。
2. イベントテーブルを開きます (Ctrl+U)。



3. ポップアップメニューから「行作成(R)」(あるいは F4 キー)で新規行を作成します。

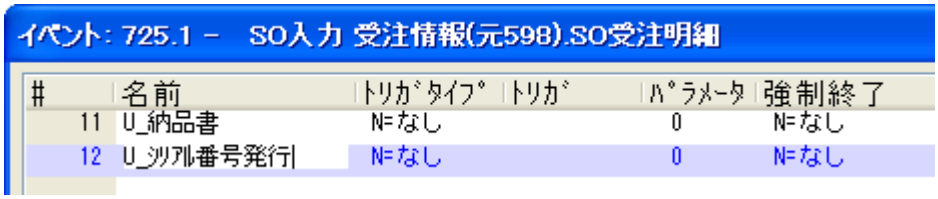




4. 「名前」欄には、利用目的がわかるような任意の名前をつけます。
   ここでは、「U_シリアル番号発行」とします。

5. その他のパラメータについては、次のように設定します。
   - トリガタイプ:「N=なし」
   - トリガ:(空欄のまま)
   - パラメータ:(0 のまま)
   - 強制終了:「N=なし」

   これらのパラメータの意味については、第 8 章「ズーム」で、ズーム機能に関連したものについて簡単に説明します。



V8 で「終了」「取消」「選択」「削除」等々、Magic エンジンに何らかの動作を起こさせるものは「アクション」と呼んでいましたが、V9 から「内部イベント」と呼ばれるようになりました。

「ユーザアクション *n*」は、エンジンに何らの動作を起こさせるものではありませんが、既定義のアクションとして、内部イベントとして分類されています。

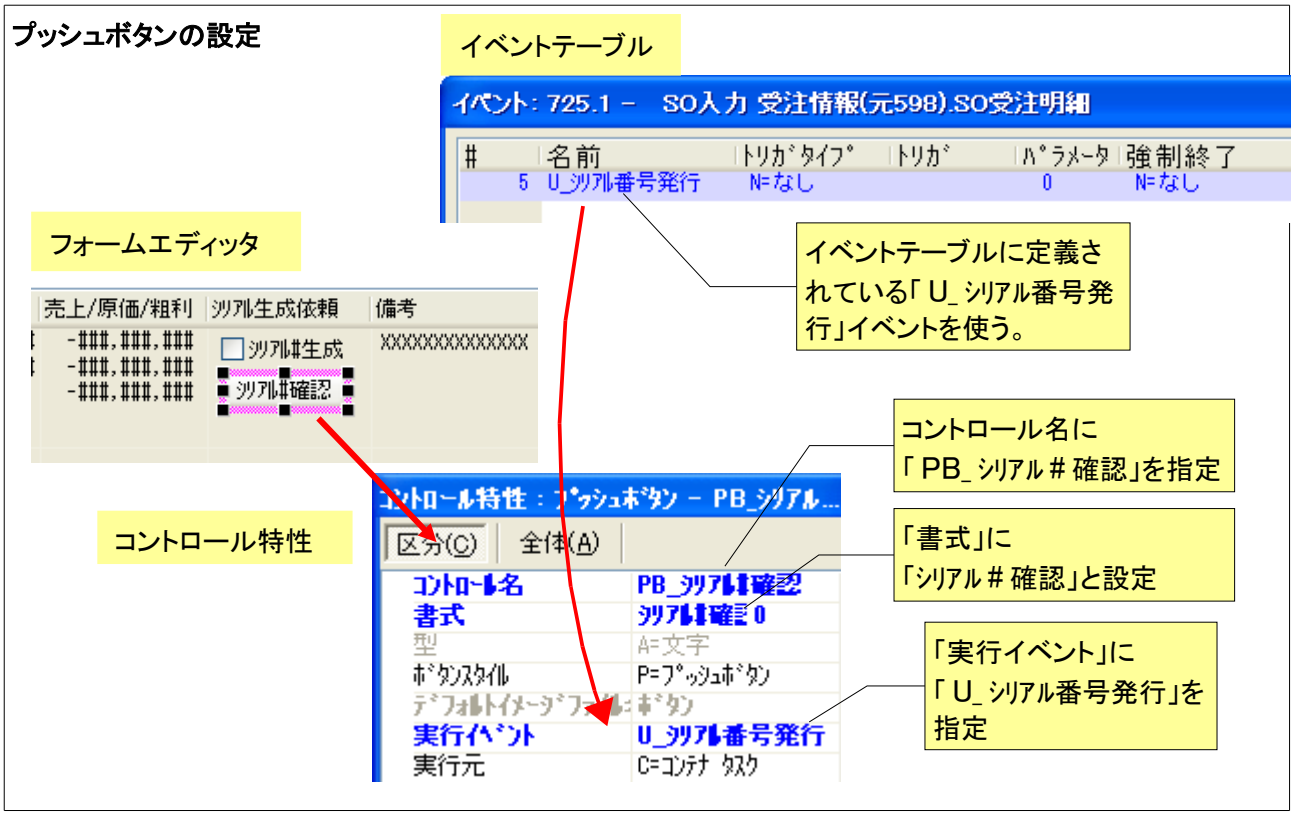
7.2.2 プッシュボタンの設定

プッシュボタンでは、実行時にユーザがプッシュボタンを押すと、「U_ｼﾘｱﾙ番号発行」イベントが発行されるように設定します。

これは、フォームのプッシュボタンの特性シートで、

- 実行時のボタンラベルとして、「書式」特性には、「ｼﾘｱﾙ#確認」と設定。
- プッシュボタンを押したときにに発行させるイベントとして、「実行イベント」特性には、上で定義したユーザ定義イベント「U_ｼﾘｱﾙ番号発行」を設定。
- イベントハンドラで、プッシュボタンを区別するため、「コントロール名」特性には「PB_ｼﾘｱﾙ#確認」と設定。

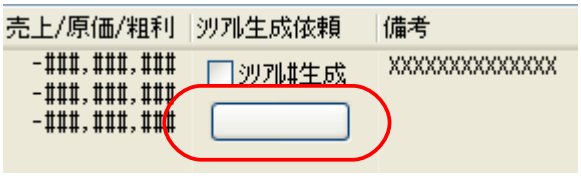
とします。



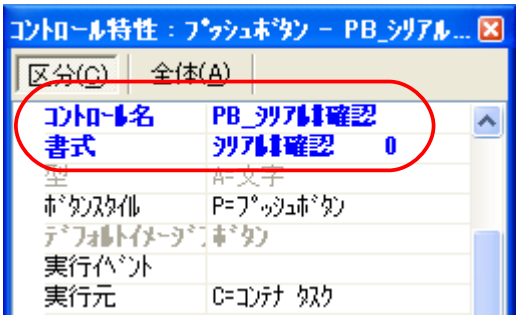
プッシュボタンの命名規約について： 上の例では、プッシュボタンのコントロール名として「PB_ｼﾘｱﾙ#確認」という名前が付けられていました。このように、プッシュボタンには「PB_」という接頭辞をつけておくと、プログラム中で「これはプッシュボタンだな」とすぐに区別できるので便利です。

プッシュボタンの設定は、次の手順で設定します。

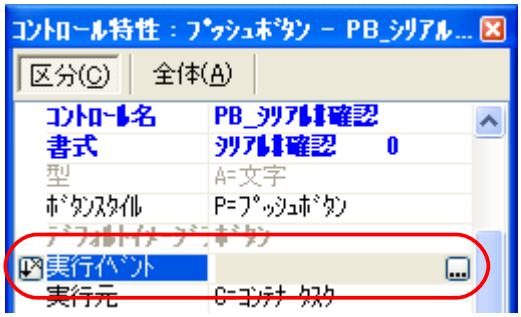
1. フォームエディッタを開きます。
2. プッシュボタンを配置します。(V8 からの移行アプリケーションであれば、すでにフォーム上に配置されているはずです)。



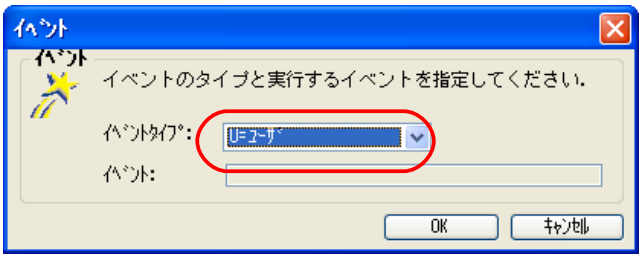
3. プッシュボタンを選択して、コントロール特性を開きます (Alt+Enter)。
4. 「コントロール名」特性に「PB_ｼﾘｱﾙ#確認」と指定します。
5. 「書式」特性に「ｼﾘｱﾙ#確認」と設定します。



6. 「実行イベント」特性にカーソルを合わせます。

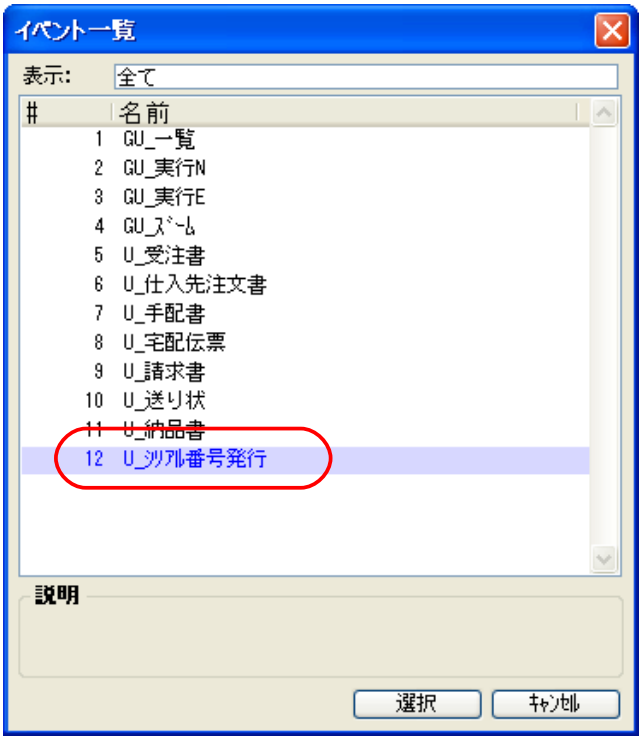


7. ズームします (F5 キーあるいはダブルクリック)
   → イベントダイアログが開きます。
8. 「イベントタイプ」は「U=ユーザ」とします。



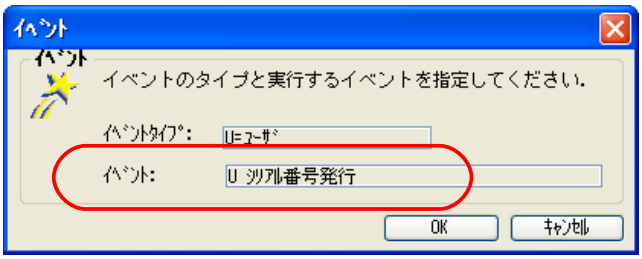
9. 「イベント」欄でズームします。(F5 キー あるいはダブルクリック)

   → 定義されているユーザ定義イベントの一覧が表示されます。



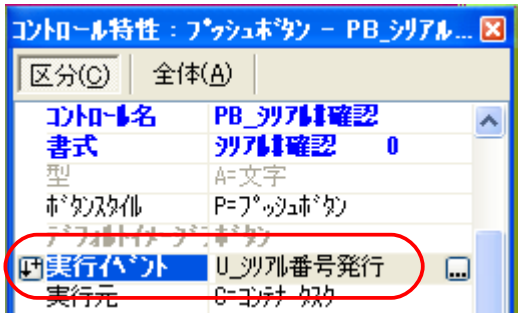
10. 「U_ｼﾘｱﾙ番号発行」を選択します。

    → イベント ダイアログに設定されます。



11. 「OK」を押します。

    → プッシュボタン特性シートの「実行イベント」特性に、「U_ｼﾘｱﾙ番号発行」が設定されます。



以上で、プッシュボタンの設定は終了です。
フォームエディッタを閉じてください。

7.2.3 イベントハンドラの設定

このプッシュボタンに対応するイベントハンドラは、次のようなものとなります。

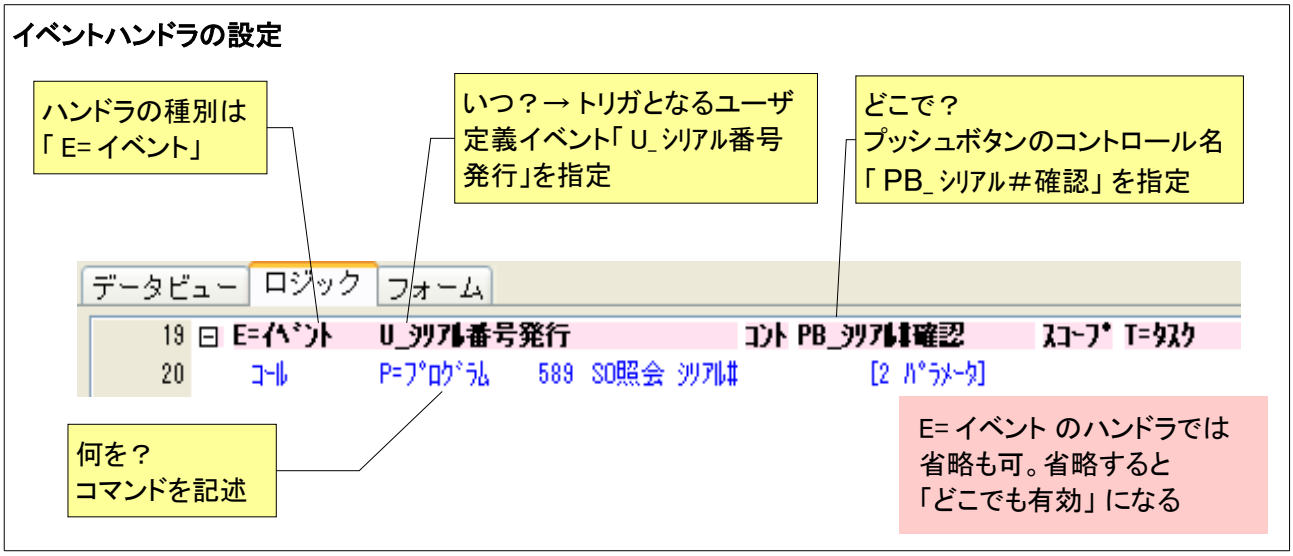
A) どこで？「PB_ｼﾘｱﾙ#確認」という名前のコントロール（プッシュボタンコントロール）において。

B) いつ？「U_ｼﾘｱﾙ番号発行」というユーザ定義イベントが発行されたときに。

C) 何を？ プログラム 589 番「SO 照会 ｼﾘｱﾙ#」を呼び出します。

これを実現するために、イベントハンドラは次のように設定します。

イベントハンドラの設定



1. イベント発生時に実行すべきハンドラなので、ヘッダ行は「E=イベント」とします。
2. 「いつ？」を指定するものを、「トリガ」と呼びます。この例では、ユーザ定義イベント「U_ｼﾘｱﾙ番号発行」がトリガです。トリガは、ヘッダ行の第2パラメータとして指定します。
3. 「どこで？」は、ヘッダ行の「コントロール」欄に、コントロール名を指定します。ここでは、プッシュボタンのコントロール名「PB_ｼﾘｱﾙ#確認」を指定します。
4. 「何を？」は、詳細行に記述します。プッシュボタンを押したときに実行すべきプログラムを、コールコマンドで呼び出します。

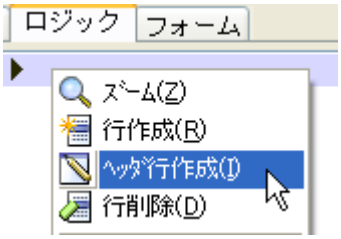
コントロールレベルハンドラと異なり、イベントハンドラでは「コントロール名」欄の指定は任意です。「コントロール名」欄を指定した場合には、カーソルがそのコントロールにある場合にだけイベントハンドラが有効になります。一方、「コントロール名」欄を指定しない場合には、カーソルがどこにあっても常に有効となります。

今の例では、イベント「U_ｼﾘｱﾙ番号発行」は、プッシュボタン「PB_ｼﾘｱﾙ#確認」を押した場合にしか発行されないので、「コントロール名」欄を空白にしたままでも、動作は同じになります。

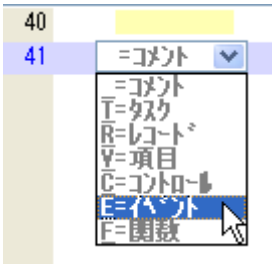
同じイベントが複数のプッシュボタンに割り当てられている場合など、コントロールを区別しなければならない場合には「コントロール名」を設定します。

イベントハンドラを作成するための操作は、次の通りです。

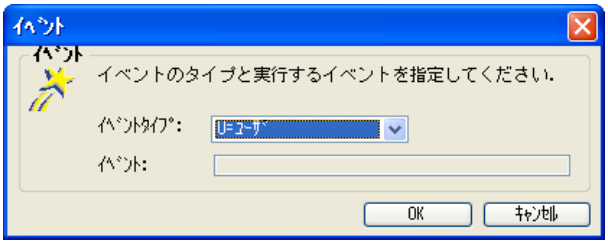
1. ロジック タブをクリックします。
   → ロジック画面が開きます。
2. ポップアップメニューで、「ヘッダ行作成(I)」を選びます。
   → 空のヘッダ行が作成されます。



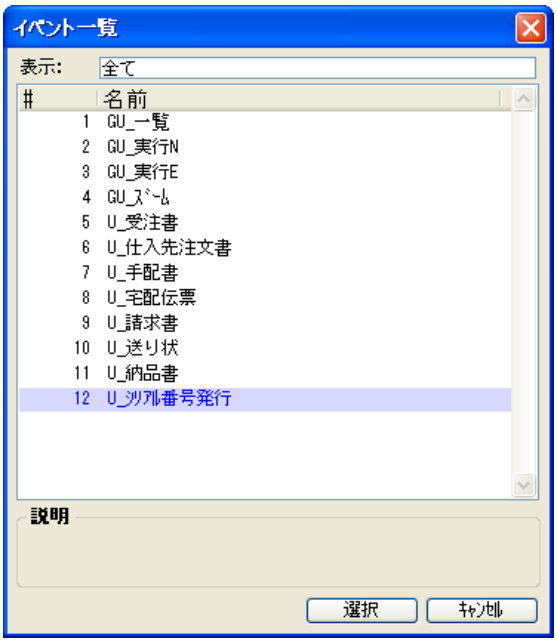
3. ヘッダ行の先頭で、ハンドラの種別を、コンボボックスから選びます。ここでは、「E=イベント」を選択します。



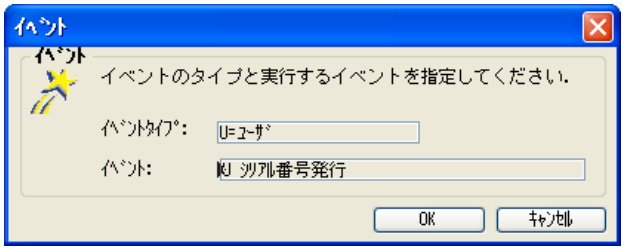
   → イベントハンドラのヘッダ行が作成されるとともに、トリガを設定するための イベント ダイアログが表示されます。



   プッシュボタンの設定と同様に、トリガとなるイベントを設定します。
4. 「イベントタイプ」は「U=ユーザ」とします。
5. 「イベント」欄でズームします。(F5 キー あるいは ダブルクリック)
   → イベント一覧が表示されます。



6. 「U_シリアル番号発行」を選択します。
→「イベント」欄に、選択したイベント名が表示されます。



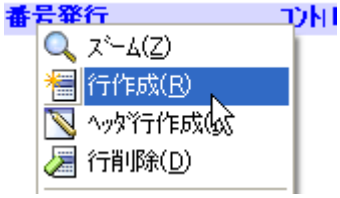
7. OK ボタンを押します。
→ イベントハンドラのヘッダ行に戻り、トリガ欄には選択したイベント名が表示されます。

8. コントロール欄に移り、ズームします。(F5 キー あるいは ダブルクリック)
→ このタスクのフォームにあるコントロールの一覧が表示されます。(コントロール名が設定されているもののみ)

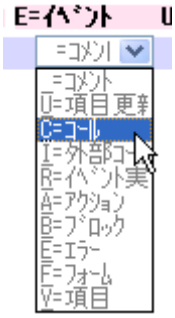
9. 「PB_シリアル#確認」を選択します。
→ ヘッダ行に戻り、「コントロール」欄には選択したコントロール名が表示されます。

以上で、ヘッダ欄の設定は終了です。
次に、詳細行の設定にはいります。詳細行は、処理コマンドを記述します。操作方法としては、旧バージョンの場合とほとんど同じです。この例では、コールコマンドで、プログラム 589 番を呼び出すようにします。

10. ポップアップメニューで、「行作成(R)」を選択します。(あるいは、F4 キーを押します)
→ 詳細行が 1 行作成されます。

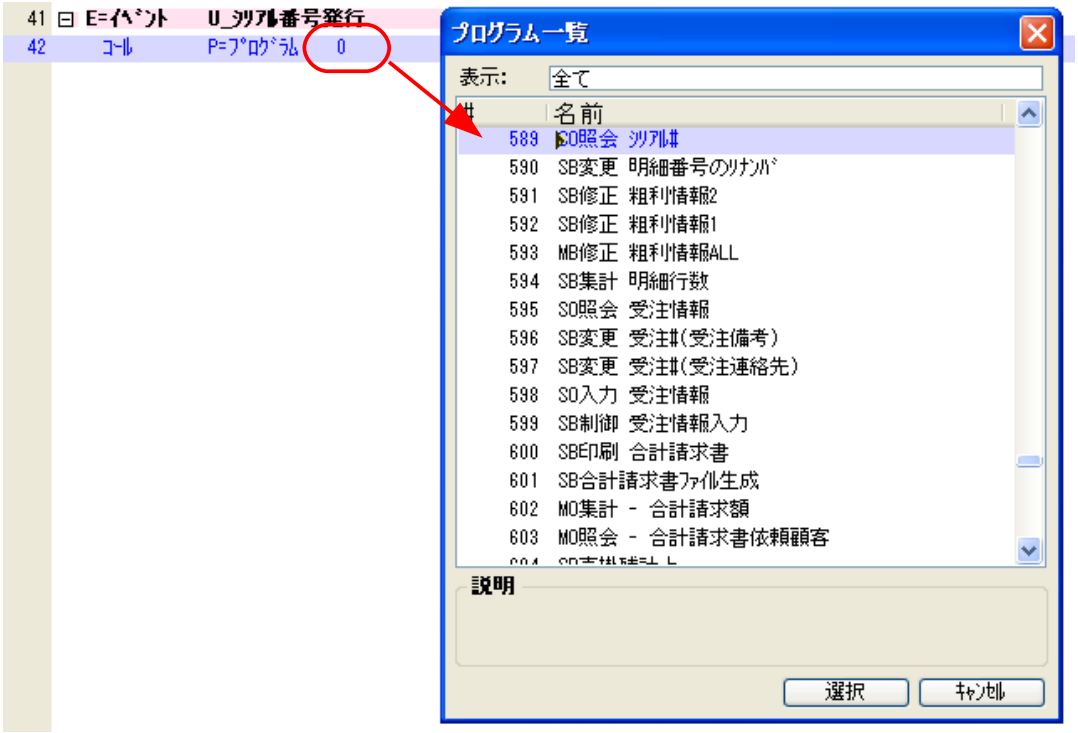


11. 詳細行の先頭でズーム (F5 キー、あるいはダブルクリック)をします。
→ コマンドの一覧がコンボボックスで表示されます。

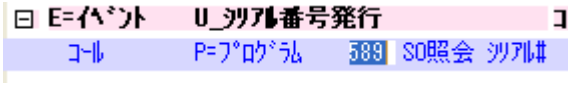


12.「C=コール」を選択します。
→ コールコマンドが表示され、パラメータ類が表示されます。

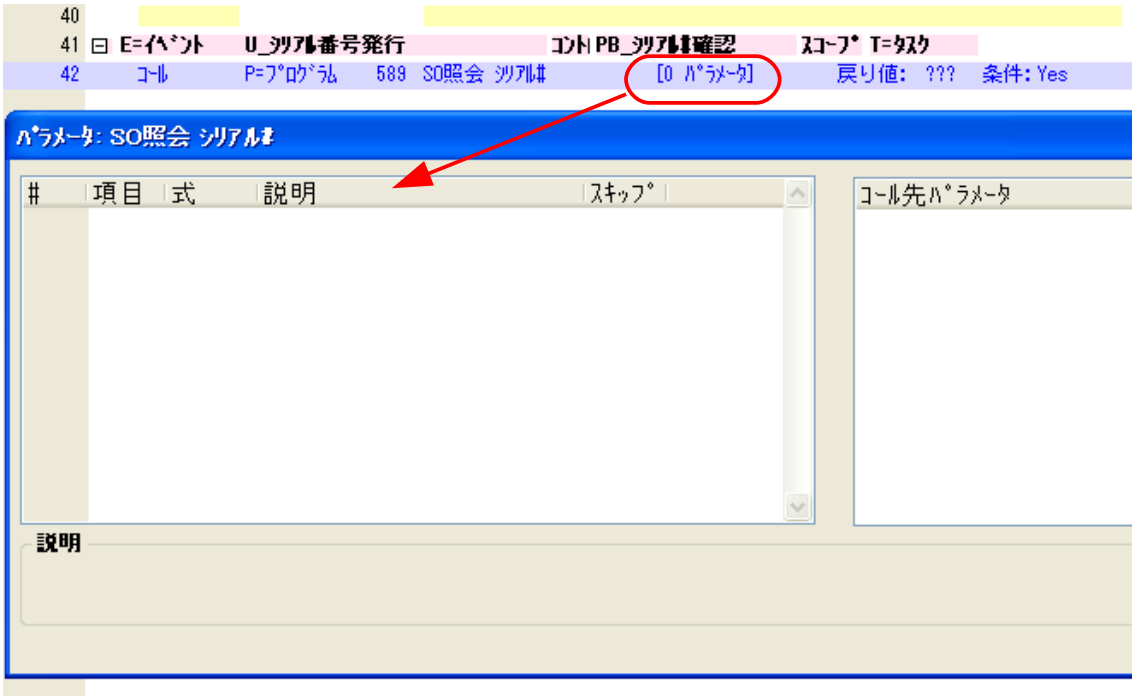
13. コールのタイプは、「P=プログラム」とします。
14. プログラム番号欄でズームします。→ プログラム一覧が表示されます。



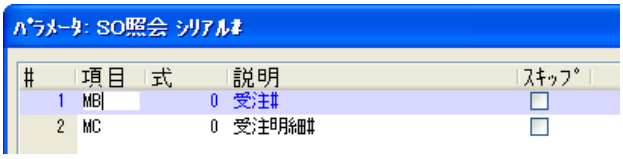
15. プログラム 589 番「SO 照会 ｼﾘｱﾙ#」を選択します。→ 選択したプログラム番号とプログラム名とが表示されます。



16. パラメータに移動し、ズームします。→ パラメータテーブルが表示されます。



17. 必要なパラメータを設定します。



18. OK ボタンを押します。
→ 詳細行に戻ります。パラメータ欄には、パラメータの個数が表示されています。

以上で、できあがりです。

7.3 イベントハンドラ方式の利点

レコードメインを使った方式に比べ、イベントハンドラを使った方式は、次のような利点があります。

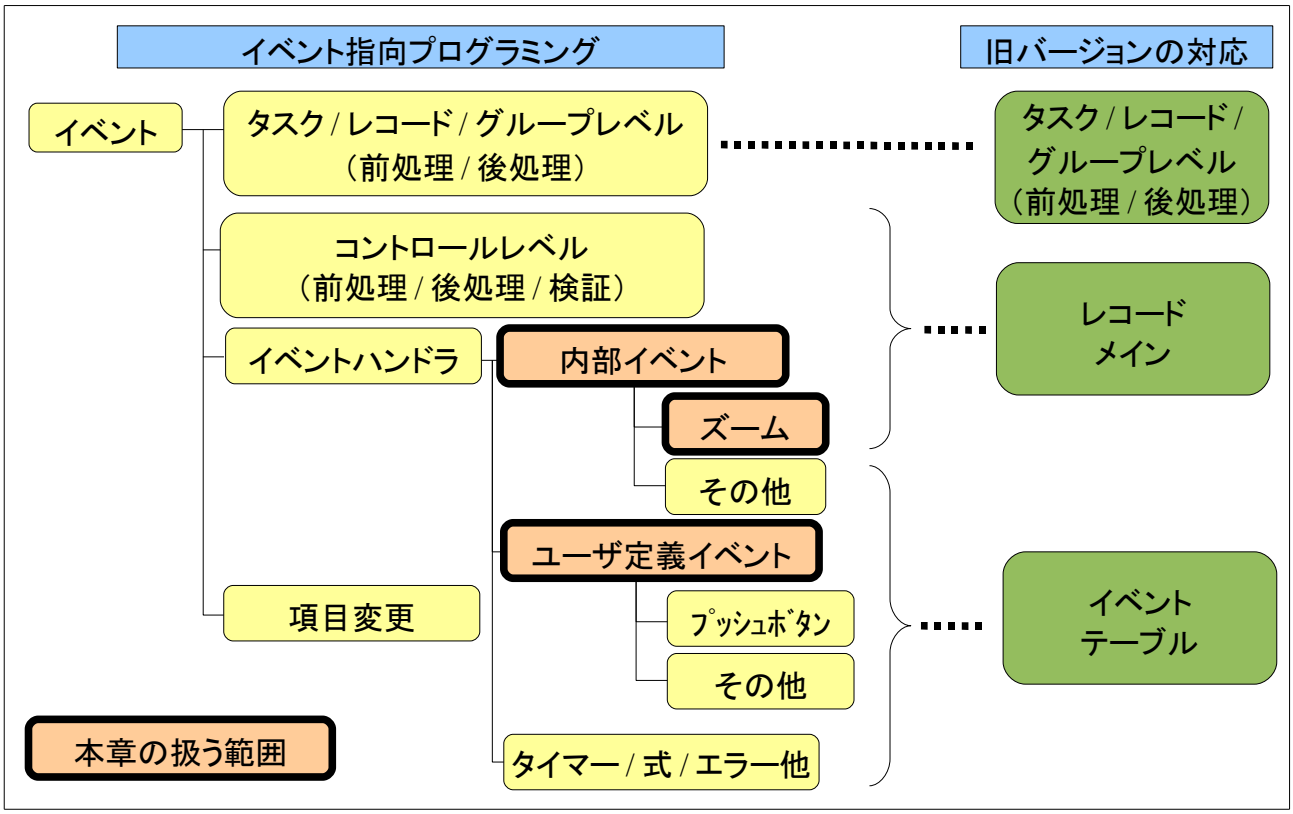
- ボタンとイベントの関連がわかりやすい。V8 では「ユーザアクション *n*」しかなく、どのボタンにどのユーザアクションが割り当てられているのかがわからなかったのですが、V10 では、アプリケーションにとって意味のある名前のユーザ定義イベントを使って記述できるので、意図がすぐにわかるようになりました。
- プログラムの見通しが良い。V8 では、ズームや画面の操作はレコードメインに、プッシュボタン等の操作はタスクイベントテーブルにそれぞれ記述していましたが、V10 ではすべて「ロジック」タブに記述するようになりました。
- 処理内容がわかりやすい。V8 のタスクイベントテーブルは、プログラム番号のみしか見ることができず、プログラム名はプログラム番号欄からズームしなければなりませんでした。このため、タスクイベントテーブルだけを見ても、どのボタンで何が呼び出されるのかわかりませんでした。V10 では処理内容は詳細行に記述されるので、プログラム名、パラメータ、条件などがわかりやすくなっています。
- イベント数に制限がない。V8 での「ユーザアクション *n*」は、「ユーザアクション 20」までの 20 個しかありませんでした。それ以上のボタンを使いたい場合には、同じユーザアクションを複数のプッシュボタンに割り当て、CTRLNAME()で条件付けして区別していました。V10 ではユーザ定義イベントの数に上限がないので、必要なだけ定義して利用することができます。
- 別途プログラムを作る必要がない。V8 のタスクイベントテーブルでは、必ずプログラムあるいはサブタスクを呼び出すようになっており、実際の処理内容がコマンド1行だったとしても、別途にタスクを作る必要がありました。V10 のイベントハンドラでは、詳細行にさまざまな処理コマンドを記述できるので、例えば、項目更新コマンドひとつで済む場合には、わざわざ別タスクを作成する必要はありません。

第8章 ズーム

ズーム機能は、ある項目に選択肢がある場合に、F5 キー、あるいはダブルクリックによって、一覧選択プログラムを表示させて、その中から欲しいものを選択する、というもので、Magic アプリケーションでは非常に多く使われます。

V8 では、ズームはレコードメインのフローモード「B=前置」あるいは「A=後置」を使って実現していました。V10 では、イベントハンドラを用いて実現します。

本章では、ズーム機能の実現方法と注意点などについて説明します。



ズームを実現するには、データ項目の「選択プログラム」特性にプログラム番号を指定する方法もあります。この方法は簡便で使いやすく、お勧めなのですが、プログラムに渡すことのできるパラメータ数が 1 個に限定される、という制限があります。
「選択プログラム」特性を使ってズームを実現する方法は、旧バージョンでも V10 でも全く同じように使うことができますので、本書では説明を省略します。

旧バージョンでのズーム機能は、一覧選択プログラムの呼び出しだけでなく、

- プッシュボタンに「ズーム(Z)」イベントを関連づけて、ボタンを押した場合の処理をレコードメインに記述する。
- タブコントロールで、ユーザがタブをクリックした場合に「ズーム(Z)」イベントが発生することを利用して、タブ選択変更時の処理をレコードメインに記述する

などの使いかたもしていました。
V10 では、プッシュボタンについては第 7 章「プッシュボタン」に説明したようにユーザ定義イベントを利用することがお勧めです。また、タブ選択変更時の処理は、第 6 章「項目変更ハンドラ」に説明した、項目変更ハンドラを利用することをお勧めします。

8.1 レコードメインでのやりかた

旧バージョンでは、ズームを実現するのに、レコードメインのフローモードを使って実現していました。これには大きく分けて二つの方法があります。

8.1.1 「B=前置」を利用

一つは、フローモード「B=前置」を利用する方法です。

- ズームを実装したい項目（ズーム可能項目）のセレクトコマンドの直後に、ズーム時に実行させたい処理コマンド(通例、選択プログラムを呼び出すコールコマンド)を記述します。
- 処理コマンドのフローモードは「B=前置」とします。

例えば、次の図は、ズーム可能項目として「商品＃」があり、そのセレクトコマンド（第 12 行目)の直前（第 11 行目)に、商品選択プログラムを呼び出すコールコマンドがあります。そして、コールコマンドのフローモードは B（B=前置）となっています。

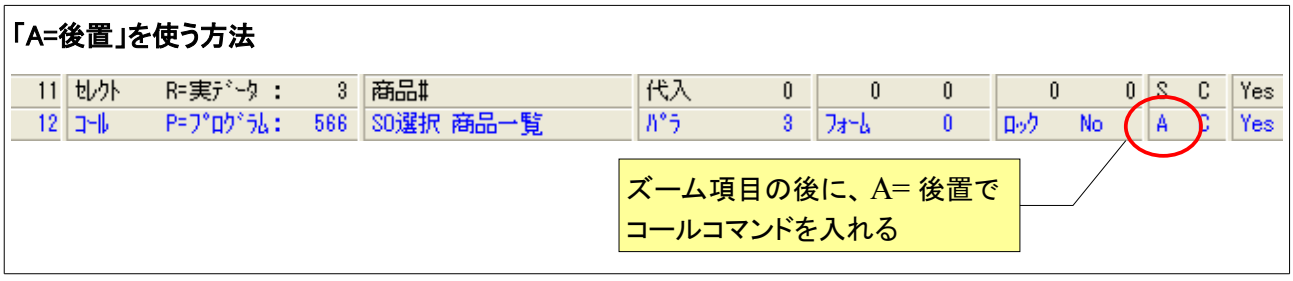
このように設定されている場合には、次のような動作になります。

- ズーム可能項目「商品#」にカーソルがパークしたときに、ズーム可能となります。
- ユーザが F5 キーを押すか、マウスのダブルクリックを行うと、直前のコールプログラムが実行され、プログラム 566 番「SO 選択 商品一覧」が呼び出されます。
- 呼び出されたプログラムが終了したときには、カーソルは再びズーム可能項目「商品#」でパークします。

8.1.2 「A=後置」を利用

もうひとつの方法は、これと似ていますが、フローモードは「A=後置」を使うものです。

- ズーム可能項目のセレクトコマンドの直前ではなく、直後に、ズーム時の処理コマンドを置きます。
- 処理コマンドのフローモードは「A=後置」とします。



このように設定されている場合には、ズーム可能項目「商品#」にカーソルがパークしたときに、ズーム可能となることは同じですが、選択プログラムが終了したときには、カーソルはズーム可能項目「商品#」の次の項目に進んでパークします。

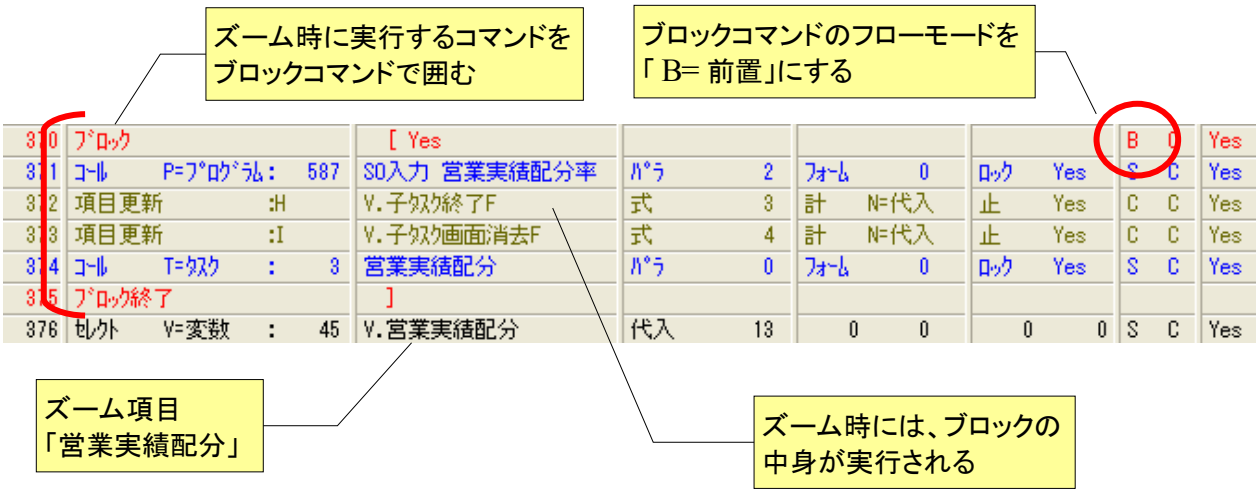
8.1.3　ズーム時に複数のコマンドを実行

場合によっては、ズーム時に実行したいコマンドが一つではなく、複数のコマンドからなる場合があります。このような場合には、

- 実行したい一連の処理コマンドをブロックコマンドで囲みます。
- ブロックコマンドのフローモードとして、「B=前置」あるいは「A=後置」を設定します。

下図は、フローモード「B=前置」を指定したブロックコマンドを使って複数のコマンドをズーム時に実行させるようにした例です。「営業実績配分」項目がズーム可能項目であり、ここでズームすると、ブロックコマンド内の一連の処理コマンドが実行されます。

ズーム時に複数のコマンドを実行させる方法



8.2 イベントハンドラを使うと・・・

これと同じズーム機能を、レコードメインを使わずにイベントハンドラで実現する方法を以下に説明します。

イベントの考え方で整理すると、ズームの扱いは次のようになります。

A) どこで？ ズーム項目で。

B) いつ？ ズームが発生したときに。

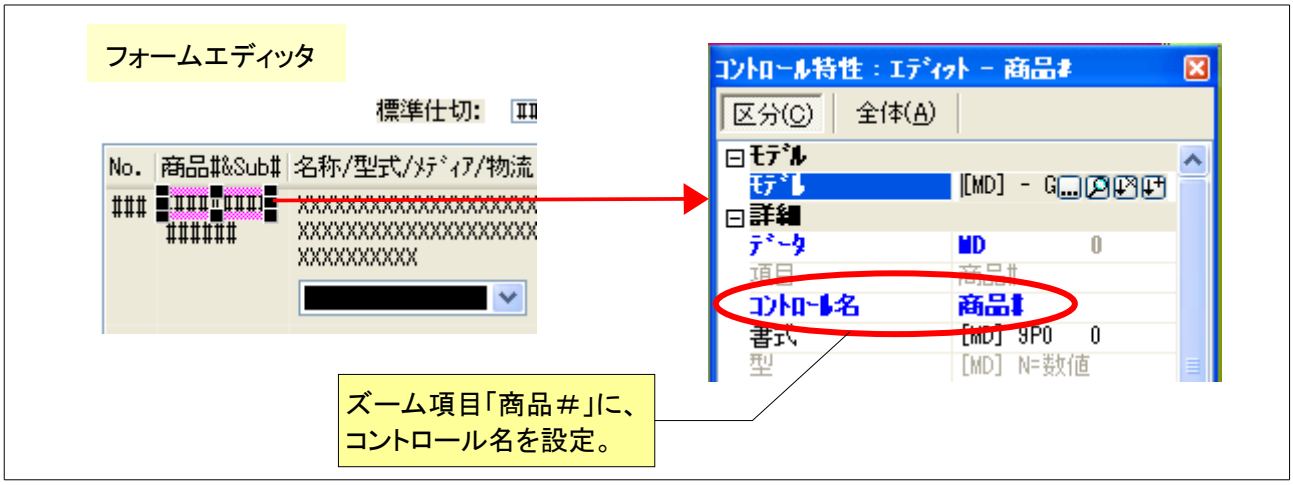
C) 何を？ 選択プログラムなどをコールする。

この考え方で、プログラムを作ってみましょう。

8.2.1 ズーム項目

「どこで？」の指定は、イベントハンドラの「コントロール」欄に、ズーム項目のコントロール名を指定することによって行います。そのため、まずズーム項目にコントロール名を設定する必要があります。

下図の例では、「商品＃」がズーム項目となっている例です。「商品＃」を表示しているエディットコントロールのコントロール特性で、「コントロール名」が「商品＃」に設定されています。



8.2.2 イベントハンドラ（その1）

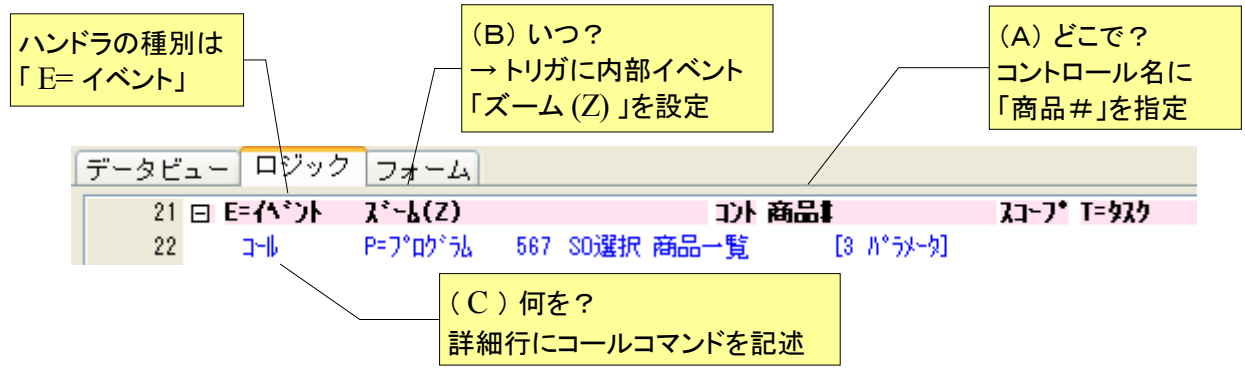
「いつ？」は、「ズームが発生したとき」です。もうすこし正確に言うと、F5 キーを押したり、ダブルクリックしたりすると、内部イベント「ズーム(Z)」が発生しますので、それをそのままイベントハンドラにすると、イベントハンドラのヘッダ行のトリガとして内部イベント「ズーム(Z)」を指定することになります。

実際のところは、内部イベント「ズーム(Z)」を直接ハンドルすると、後述するような問題が出てきますので、ユーザ定義イベントをひとつ介して行わなければならないのですが、まずは、直接ハンドルすることの問題点を実地に見ていくために、内部イベント「ズーム(Z)」をトリガとするハンドラを作ってみます。その後、正しい動作をさせるためにどうするかを解説します。

内部イベント「ズーム(Z)」をトリガとするハンドラは、次のようになります。

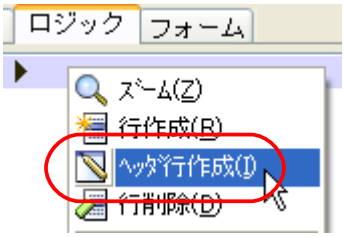
- 内部イベントに対するハンドラなので、ヘッダ行は「E=イベント」になります。
- ズーム時に実行させたいので、トリガは、内部イベント「ズーム(Z)」になります。
- ズーム項目「商品#」でだけ有効にしたいので、コントロール名に「商品#」を指定します。
- 詳細行では、選択プログラムを呼び出すコール コマンドを実行します。

内部イベント「ズーム(Z)」を直接ハンドルする方法（問題あり）

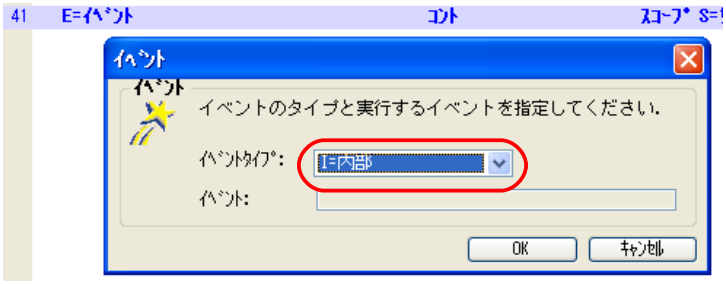


上記のイベントハンドラは、次のような操作で作成します。

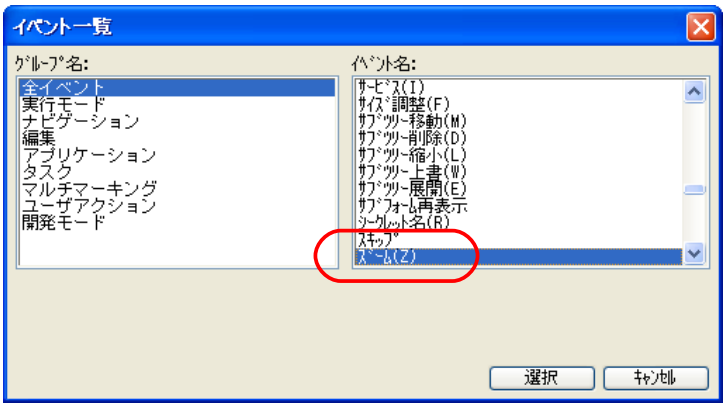
1. ロジック画面を開きます。
2. ポップアップメニューで、「ヘッダ行作成(I)」を選択します。（あるいは Ctrl+H キー）
   → 新規ヘッダ行が作成されます。



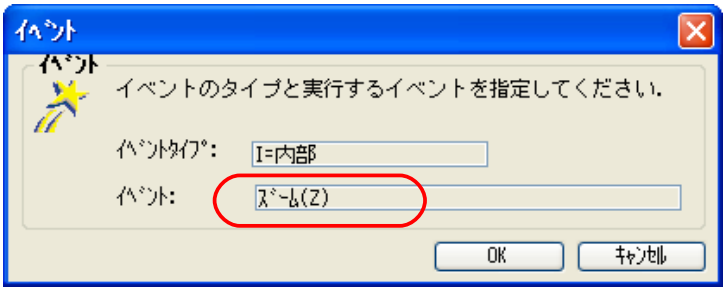
3. イベント種別のコンボボックスから、「E=イベント」を選択します。
   → ヘッダ行に「E=イベント」と表示され、トリガを設定するイベントダイアログが表示されます。



4. 「イベントタイプ」としては、「I=内部」を選びます。
5. 「イベント」欄でズームすると、内部イベント一覧が表示されます。

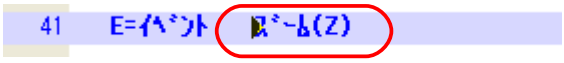


6. 「ｽﾞｰﾑ(Z)」を選択します。
   → イベントダイアログに選択したイベント名が表示されます。



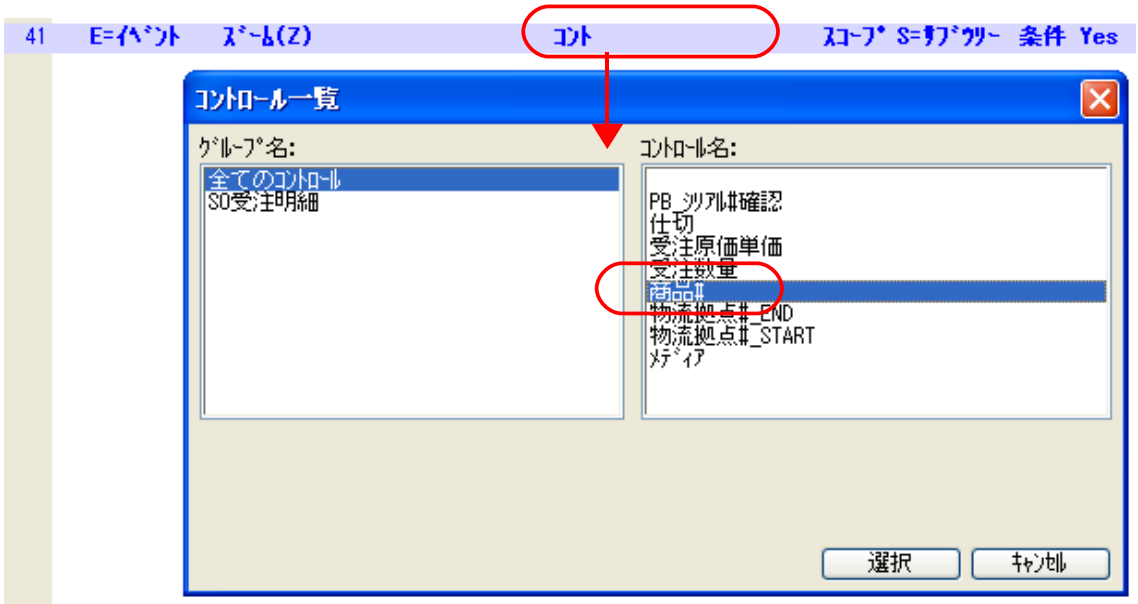
7. OK ボタンを押します。

   → トリガ欄に、選択したイベント名が表示されます。



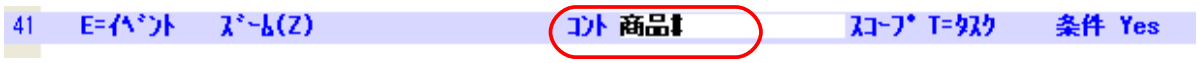
8. コントロール欄 に移って、ズームします。

   → このタスクのフォーム上のコントロール一覧（コントロール名が設定されているものだけ）が表示されます。



9. 「商品＃」を選択します。

   → 選択したコントロール名が設定されます。

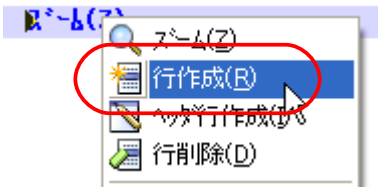


   以上で、ヘッダ行の設定は完了です。

   詳細行の作成方法は、旧バージョンの場合とほとんど同じです。

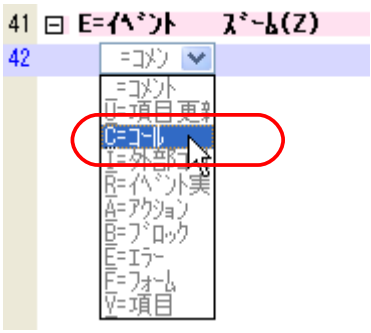
10. ポップアップメニューで、「行作成(R)」を選びます。（あるいは、F4 キー）

    → 詳細行が新規作成されます。

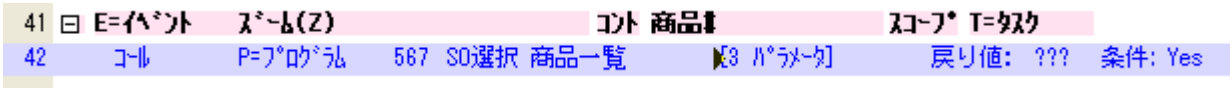


11. 詳細行の先頭でズームして、コンボボックスから「C=コール」コマンドを選択します。

    → 詳細行がコールコマンドとなり、パラメータが表示されます。



12. コールコマンドのパラメータ類を設定します。

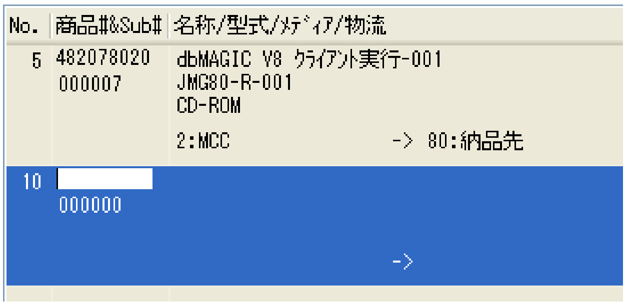


以上で、イベントハンドラができあがりました。

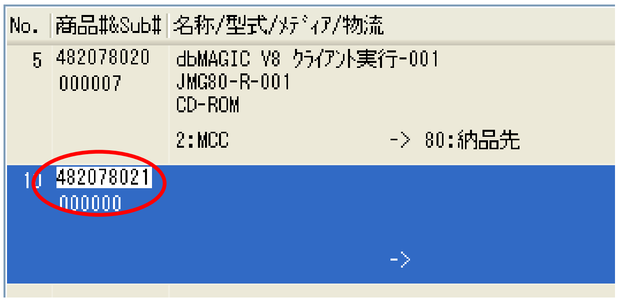
8.2.3　内部イベント「ズーム」を直接ハンドルするときの問題

このようにして直接内部イベント「ズーム(Z)」をハンドルすると、実行時に予想しない動作となってしまいます。実際に実行させて見てみましょう。

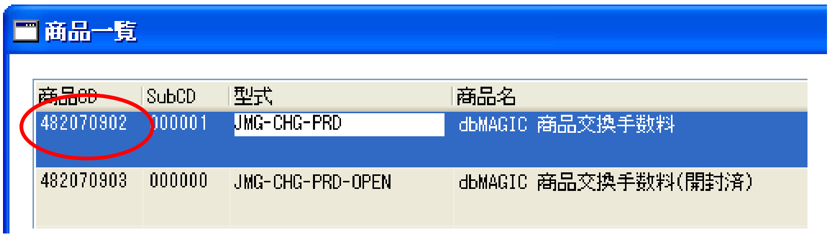
1. 受注明細の画面に入り、新規に明細行を作成します。
   この時点では、「商品＃」もその他の項目もすべて空になっています。



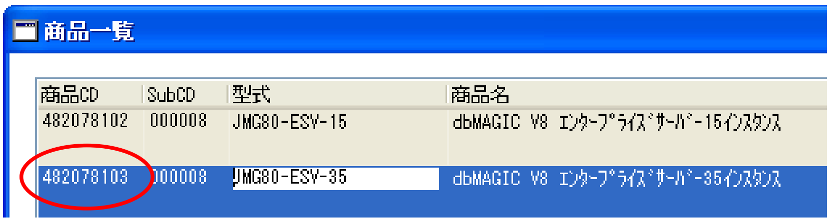
2. 手で「商品#」欄に 482078021 と入力します。
   ※ この商品は、「dbMAGIC V8 ｸﾗｲｱﾝﾄ実行 5 ユーザ版」です。



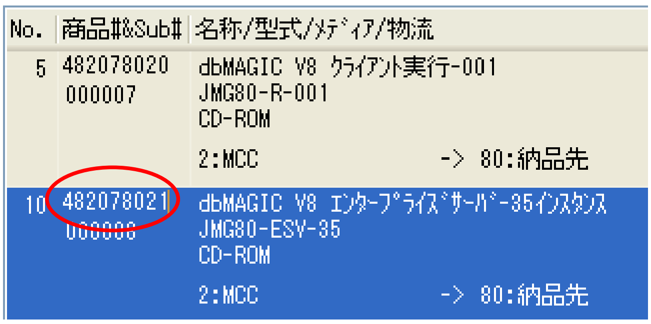
3. F5 キーでズームします。
   商品一覧が表示されます。
   入力した番号　482078021 で位置付されるはずなのですが、位置付されておらず、先頭レコードに位置付されています。



4. スクロールして、商品番号 482078103 を選択します。
   ※　この商品は、「dbMAGIC V8 ｴﾝﾀｰﾌﾟﾗｲｽﾞ ｻｰﾊﾞ 35 ｲﾝｽﾀﾝｽ」です。



5. 受注明細画面に戻ると、「商品＃」は、商品一覧で選択した 482078103 になるはずですが、実際には 482078021 のままです。
   しかし、「商品名称」の方は、商品一覧で選択した「dbMAGIC V8 ｴﾝﾀｰﾌﾟﾗｲｽﾞ ｻｰﾊﾞ 35 ｲﾝｽﾀﾝｽ」になっています。
   つまり、表示上矛盾した状態になっています。



6. Enter キーを押して、次項目に移ると、商品名称が「dbMAGIC V8 ｸﾗｲｱﾝﾄ実行 5 ユーザ版」に変わってしまいます。
   結局、最初に手で入力した番号が有効になり、商品一覧で選択したものは無視されてしまっています。

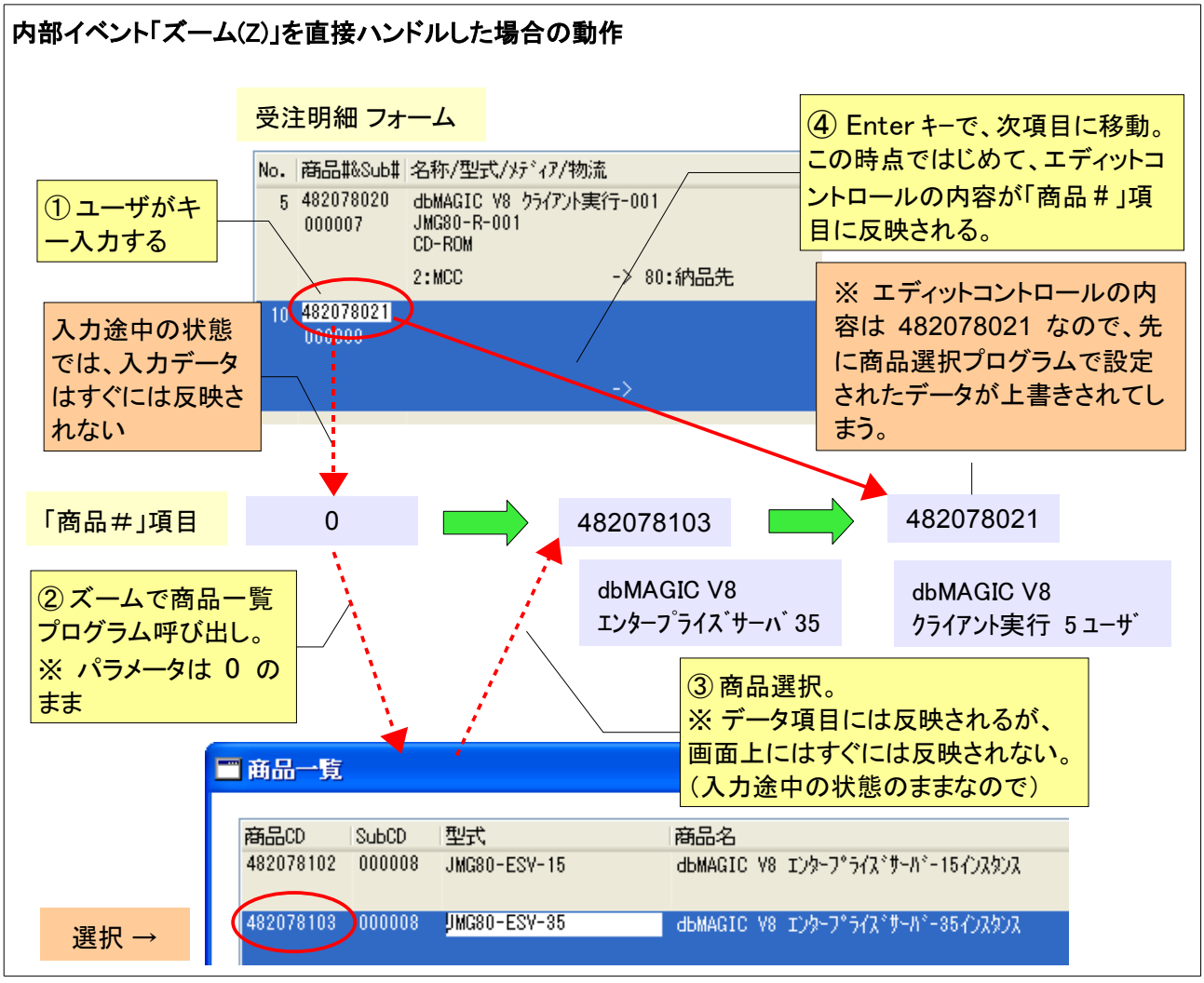
このように、内部イベント「ズーム(Z)」を直接ハンドルすると、アプリケーションの動作として極めて不都合なことになってしまいます。

このような現象が起こる理由は、表示フォーム上のエディットコントロールのデータと、内部のデータ項目のデータとが、ユーザ入力途中の状態においては一時的に異なっているためです。

一般的に、オンラインのフォームにおいて、入力途中の状態ではユーザがキー入力はエディットコントロールに蓄えられるだけで、内部のデータ項目にすぐに反映されません。ユーザが入力を終わり、Enter キーなどで別の項目に移った時点で、はじめてエディットコントロールの内容が、内部のデータ項目に反映されます。

このことを念頭において、内部でどういうことが起こっているかを示したのが、次の図です。

内部イベント「ズーム(Z)」を直接ハンドルした場合の動作



1. 受注明細画面において、新規行を作成した時点では、「商品＃」項目の値はデフォルト値 0 です。
   ここで、ユーザがキー入力し、482078021 と入れます。
   ユーザのキー入力中は「入力途中の状態」であり、フォーム上のエディットコントロールに入力データが蓄えられていきますが、「商品＃」項目にはすぐには反映されません。従って、「商品＃」項目の値は 0 のままです。

2. ここで、F5 キーを押すと、「入力途中の状態」のまま、「ズーム(Z)」アクションが発生します。
   「ズーム(Z)」ハンドラが定義されているので、ハンドラが実行され、商品一覧プログラムがコールコマンドで呼び出されます。

   この間、エディットコントロールは「入力途中の状態」のままですので、ユーザ入力はまだ項目に反映されていません。従って、商品一覧プログラムに渡されるパラメータは 0 のままです。このために、商品一覧プログラムの初期画面で、最初のレコードに位置付けられることになります。

3. 商品一覧プログラムで、482078103 を選択します。

   すると、「商品＃」項目のデータが、この番号（482078103）になります。これにともなってリンク再計算がなされ、「商品名称」が「dbMAGIC V8 ｴﾝﾀｰﾌﾟﾗｲｽﾞｻｰﾊﾞ 35 ｽﾚｯﾄﾞ」となります。

   ここでもまだ、「商品＃」エディットコントロールは「入力途中の状態」のままで、項目の値の変更はすぐにはエディットコントロールに反映されません。

4. 受注明細プログラムに戻った後、ユーザが Enter キーを押します。

   ここで初めて「商品＃」エディットコントロールの「入力途中の状態」が終了し、エディットコントロールのデータ（482078021） が「商品＃」項目に反映されます。

   これに伴いリンクの再計算が行われ、商品名称は「dbMAGIC V8 ｸﾗｲｱﾝﾄ実行 5 ユーザ」となります。

このような内部動作となっているので、結果として、商品一覧プログラムでは正しく位置付されないし、また選択した商品＃が上書きされ無視されてしまう、という現象となります。

8.2.4 ユーザ定義イベント「GU_ズーム」

この問題に対応するには、ズーム発生の時点で、エディットコントロールの「入力途中の状態」を解除してやる必要があります。そのためには、内部イベント「ズーム(Z)」を直接ハンドリングするのではなく、一段、ユーザ定義イベントを介して処理するようにします。

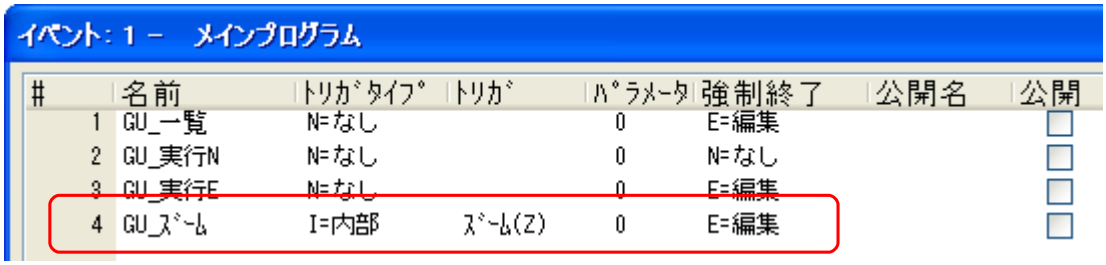
ここでは、その目的のためのユーザ定義イベントを以下のように定義します。

- 「名前」: 任意の名前でいいのですが、ここでは「GU_ズーム」とします。
- 「トリガタイプ」および「トリガ」: ここで指定されたイベントをトリガとして、このユーザ定義イベントが発行されます。

  この例では、内部イベント「ズーム(Z)」が発生したら、このユーザ定義イベント「GU_ズーム」を発生させたいので、「トリガタイプ」が「I=内部」、「トリガ」が「ズーム(Z)」とします。
- 「パラメータ」は必要ないのでそのまま (0 のまま) にします。
- 「強制終了」: イベントが発生するに当たって、エンジンのサイクルを実行することを指示します。

  この例では、「E=編集」となっていますが、これは、「入力途中の状態」を抜けて、エディットコントロールなどのフォーム上のコントロールの内容を、対応する項目に反映させることを意味します。

ユーザ定義イベント「GU_ズーム」の定義



「強制終了」には、「E=編集」のほかに、次のような設定ができます。

- N=なし: イベントハンドラが実行される前に、エンジンサイクルは何も実行されません。この場合には、エディットコントロールの入力途中の状態はそのままでイベントハンドラが実行されます。これがデフォルトの動作です。
- C=コントロール: イベントハンドラが実行される前に、コントロールレベルのサイクル(コントロール検証、コントロール後処理 → コントロール前処理) が実行されます。
- R=レコード更新の前 あるいは P=レコード更新の後: イベントハンドラが実行される前に、レコードレベルのサイクル (レコード後処理、レコード更新、レコードフェッチ、レコード前処理) を実行します。

これらの設定については、本書では説明を省略します。

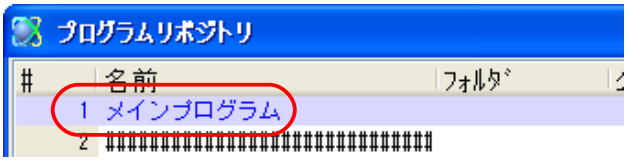
このユーザ定義イベントは、アプリケーションで共通に使えるグローバルイベントとして定義すると、ズームを使うプログラムごとにいちいち定義せずにすむので便利です。

グローバルイベントとするには、プログラムリポジトリの最初のプログラム「メインプログラム」のイベントテーブルで定義します。

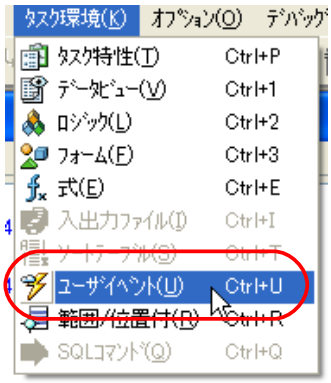
ユーザ定義イベント「GU_ズーム」は、次のような操作で作成します。

これはアプリケーション全体で有効なグローバル ユーザ定義イベントとしたいので、「メインプログラム」に定義します。

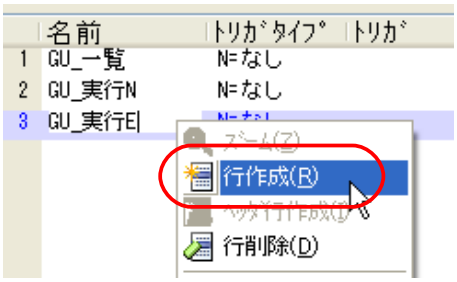
1. プログラムリポジトリの先頭にある「メインプログラム」を開きます。



2. メニュー「タスク環境(K) → ユーザイベント(U)」を選択します（あるいは Ctrl+U キーを押します）。
→ イベントテーブルが開きます。

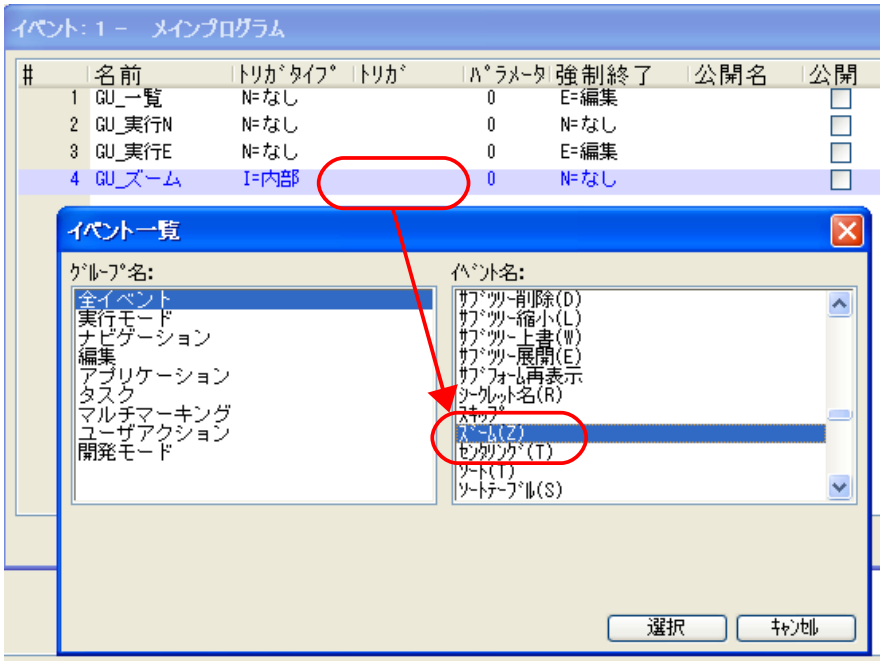


3. ポップアップメニューから「行作成(R)」を選びます（あるいは、F4 キーを押します）
→ 新しい行が作成されます。



4. 「名前」欄は「GU_ズーム」、「トリガタイプ」は「I=内部」とします。

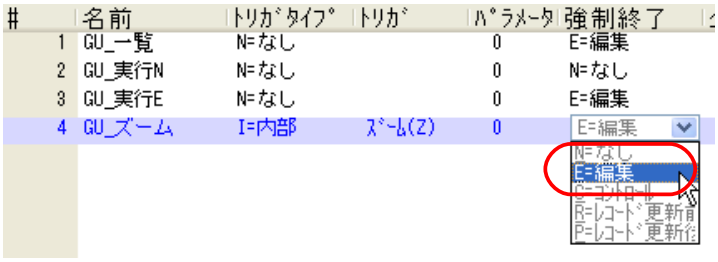
5. 「トリガ」欄に移り、ズームします → 内部イベント一覧が表示されます。



6. 「ズーム(Z)」を選択します。

   → 「トリガ」欄にイベント名が表示されます。

7. 「強制終了」欄で、「E=編集」を選択します。



以上で、ユーザ定義イベントの作成ができました。

8.2.5 イベントハンドラ（その2）

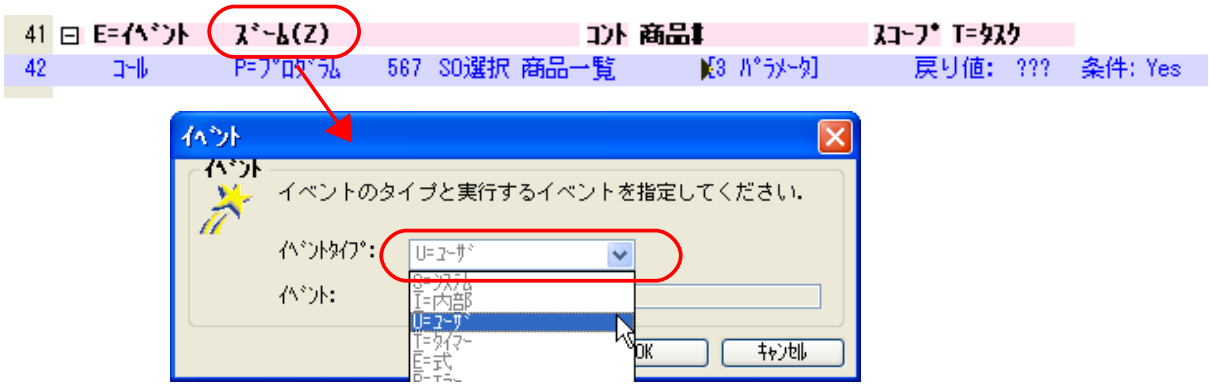
このユーザ定義イベント「GU_ズーム」を使って、イベントハンドラを書き直してみましょう。変更するのは、ハンドラのトリガの設定だけです：

- ハンドラの「トリガ」欄の設定を、内部イベント「ズーム(Z)」ではなく、ユーザ定義イベント「GU_ズーム」とします。

トリガの設定は、次の操作によって変更します。

1. イベントハンドラのヘッダ行のトリガの欄（第 2 番目のパラメータ）でズームします。

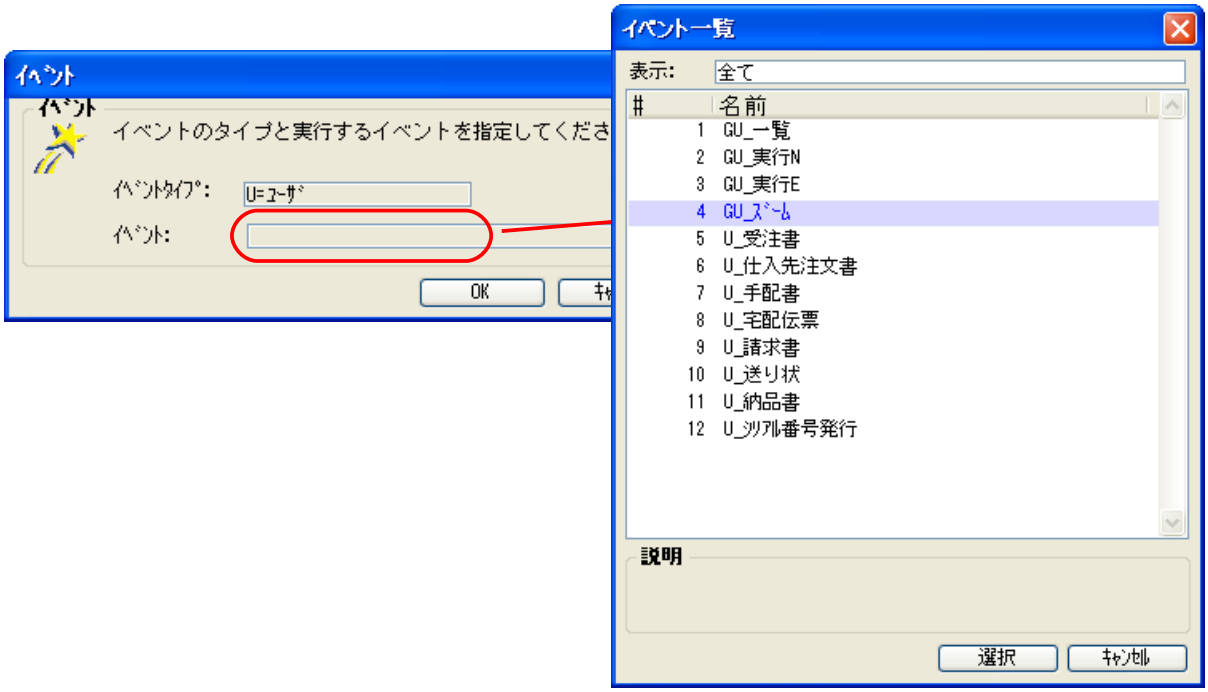
   → イベント ダイアログが表示されます。



2. 「イベントタイプ」として、「U=ユーザ」を選びます。

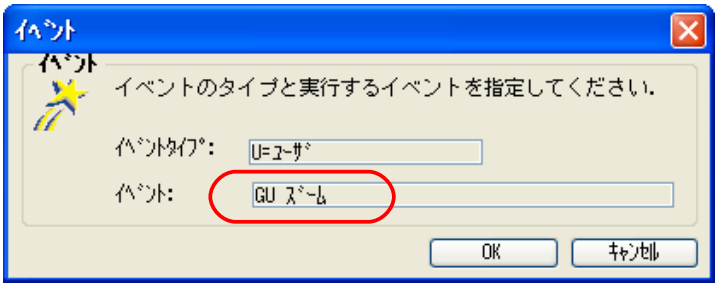
3. 「イベント」欄でズームします。

   ユーザ定義イベントの一覧が表示されます。



4. 「GU_ズーム」を選択します。

   「イベント」ダイアログに戻り、選択したイベント名が表示されます。



5. OK ボタンを押します。

   → トリガ欄に、選択したイベント名が表示されます。

以上で、トリガの設定変更は完了です。

8.2.6 実行時の動作

このようにイベントハンドラを定義すれば、次のような動作になり、意図したとおりのズーム動作が実現できます。

1. ユーザがデータをキー入力します。

   この時点では、まだ「入力途中の状態」なので、項目に反映されていません。

2. ユーザが F5 キーを押します。

   ① 内部イベント「ズーム(Z)」が発生します。

   ② これをトリガとして、ユーザ定義イベント「GU_ズーム」が発生します。

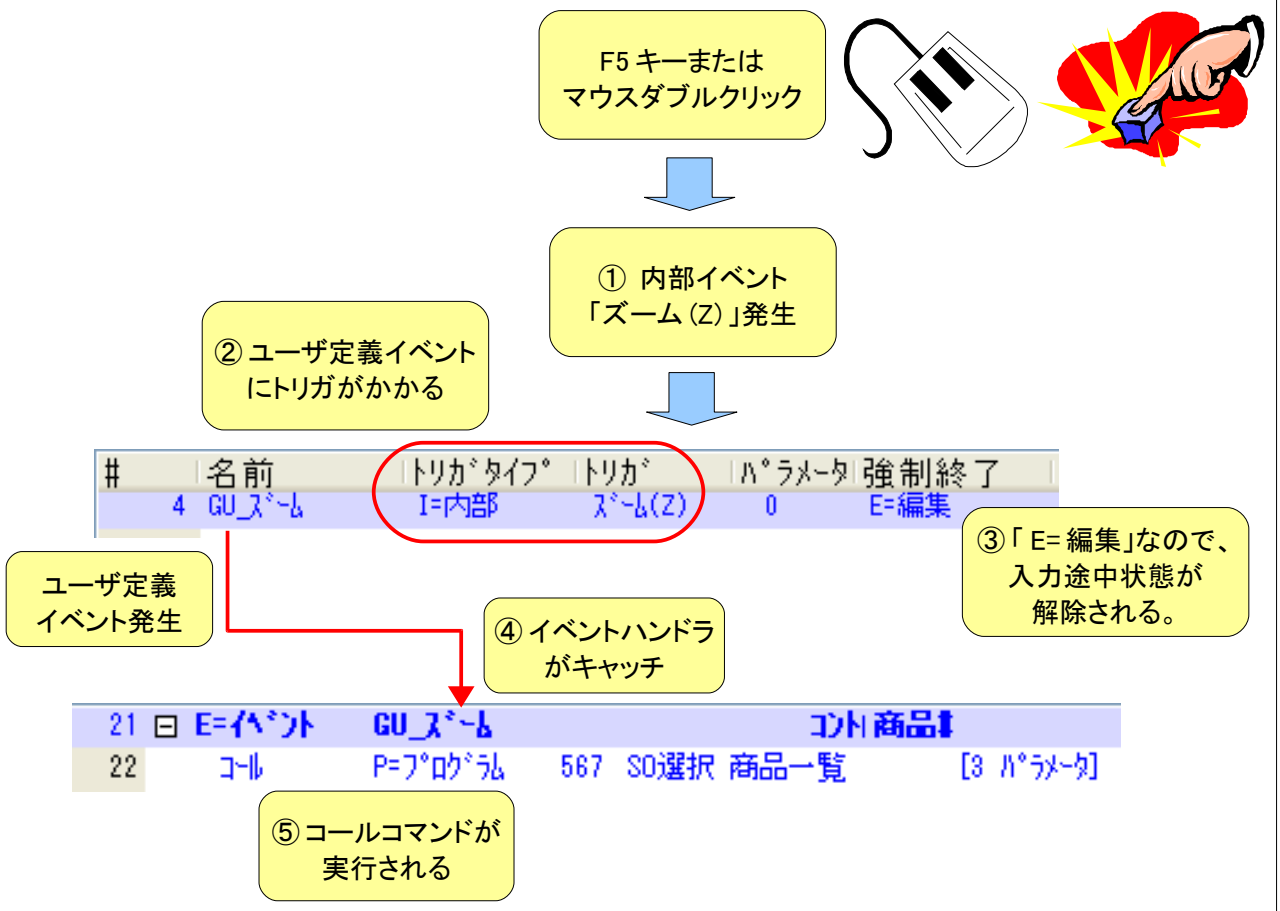
   ③ 「強制終了」が「E=編集」なので、入力途中の状態が解除され、ユーザが入力したデータが項目に反映されます。

   ④ イベントハンドラが起動されます。

   ⑤ イベントハンドラの中で、コールコマンドが実行され、「商品一覧」プログラムが呼び出されます。

   このとき、パラメータとしては、すでにユーザ入力が反映されたデータが渡されるので、初期画面の位置付けは正しく行われます。

ユーザ定義イベントのトリガ



3. 商品一覧から適当な商品を選択して、受注明細画面に戻ります。

   ① パラメータを介して、受注明細の「商品＃」項目が更新されます。

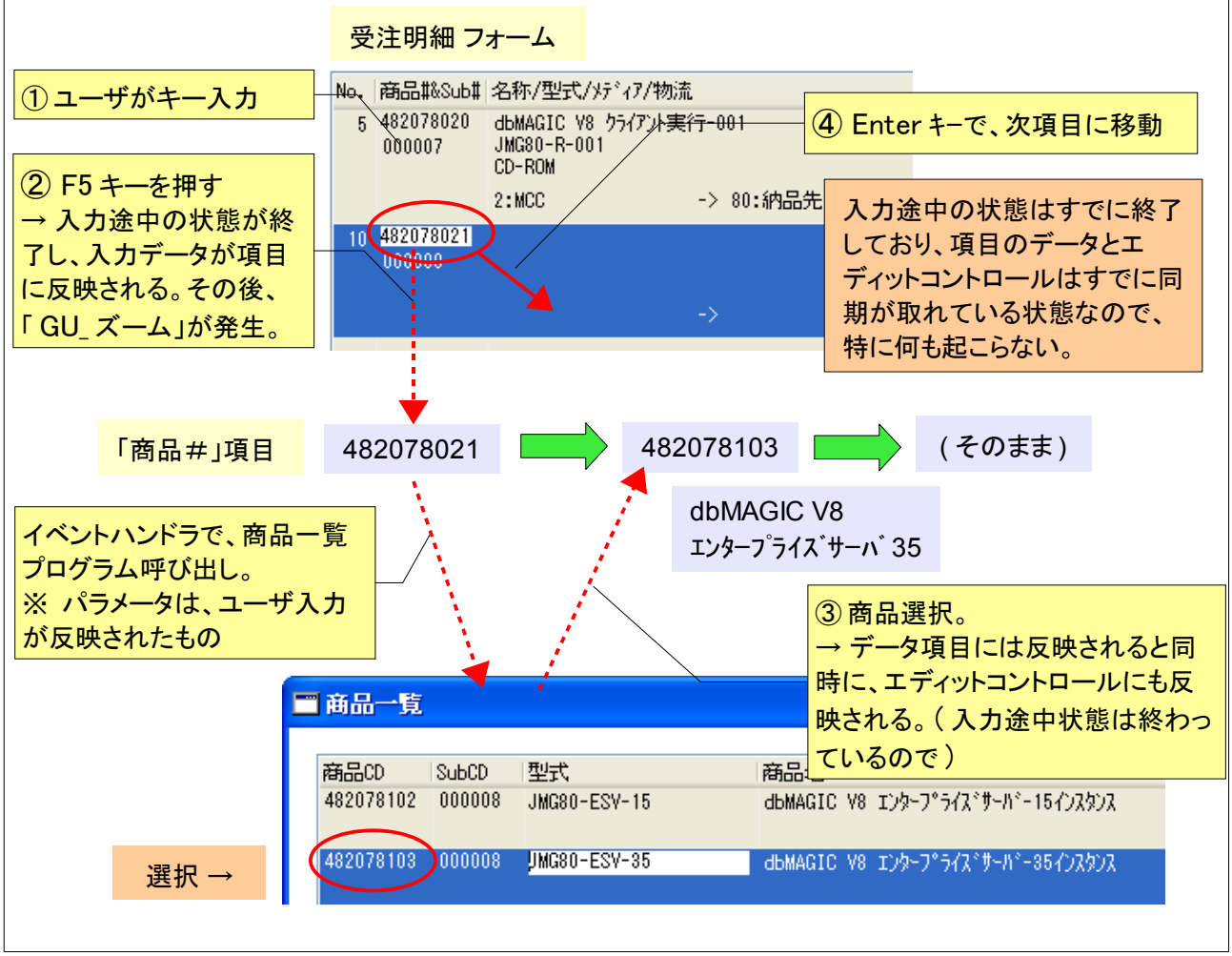
   ② それに伴い、リンク再計算が行われます。

   ③ 入力途中の状態はすでに終了しているので、項目の変更はすぐにエディットコントロールに反映されます。

4. その後、Enter キーを押して、次の項目にカーソルを移動させます。

   項目のデータとエディットコントロールの内容とは同期がとれているので、特に何も起こりません。

ユーザ定義イベント（「強制終了」＝「E=編集」）を使った場合の動作



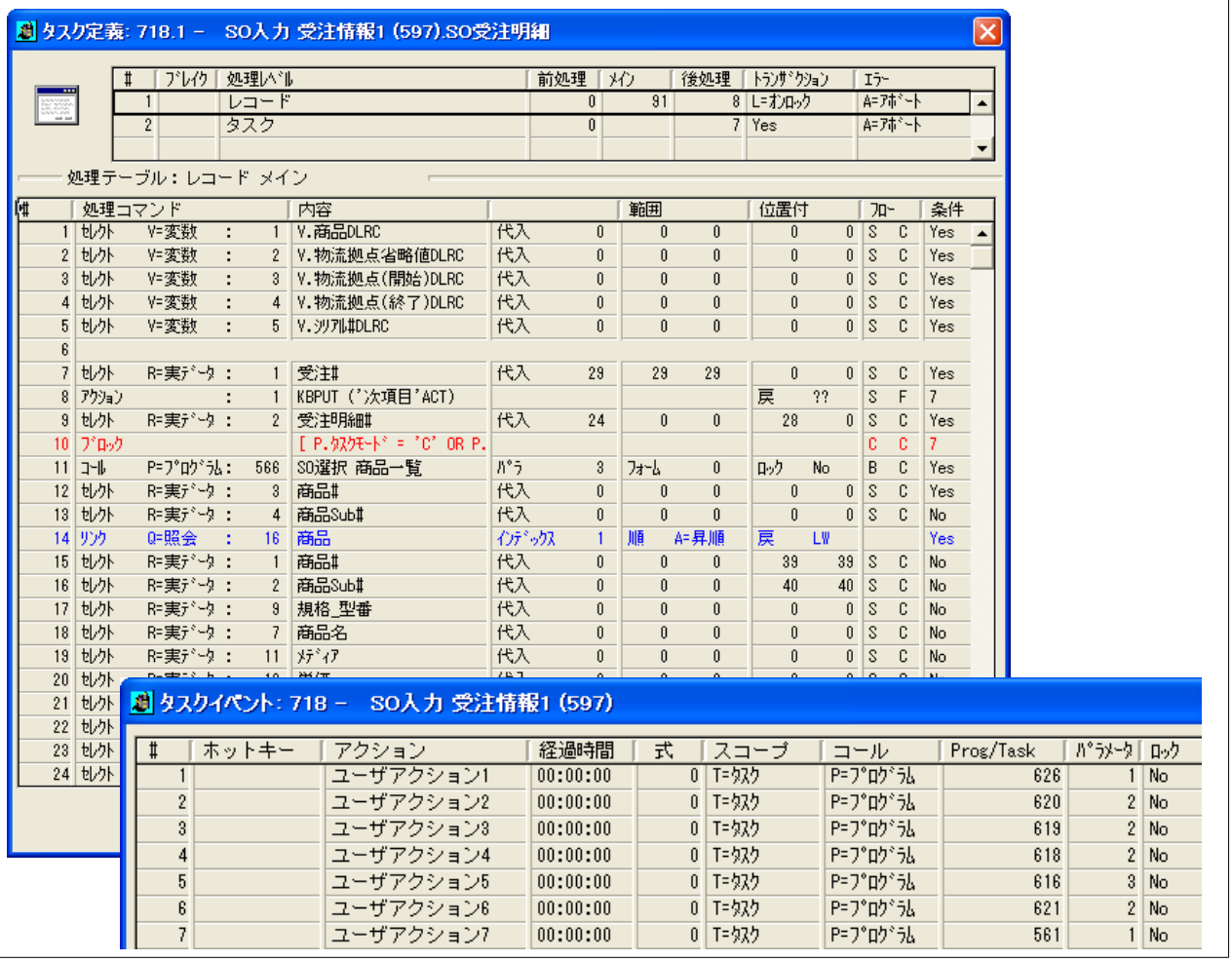
このように、ユーザイベントをひとつ間にいれてやることにより、期待していた通りの動作となることがわかります。

第9章　おわりに

最後にまとめとして、V8 のレコードメインで作った場合と、V10 のイベント指向で書き直した場合とを、プログラムリポジトリで見比べてみることにします。

下図は、V8 のレコードメインとタスクイベントテーブルの例です。

V8 のレコードメインとタスクイベントテーブル



これを見てわかるように、プログラムの内容を理解するのが難しいものとなっています。

- どの項目において、何をすると何が起こるのかが、一見しただけではわからない。レコードメインの行の前後をフローモードや条件(パークの有無)に注意しながら追ってみて、始めて理解することができる。
- プッシュボタンとその処理との関係がタスクイベントテーブルだけからはわからない。プッシュボタンとユーザアクション n の関係、および、プログラム番号とその内容の関係がわかって、始めて、タスクイベントテーブルの内容も理解できるようになる。

これに対し、V10 のイベント指向のやりかたで書き直したのが、次のページの図です。

V10 のイベント指向で書き直したもの

V8 のものと比較すれば明らかなように、V10 のイベント指向の記述は、プログラムの意図がずっと明瞭になっていて、どこでいつ何が起こるかが一見して理解することができます。

例えば、上から見ていくと、

- 「PB_ｼﾘｱﾙ#確認」という名前のコントロール（プッシュボタン）で、「U_ｼﾘｱﾙ番号発行」というイベントが発生すると、プログラム 589 番「SO 照会 ｼﾘｱﾙ#」が呼び出される。
- 「商品#」でズームすると、プログラム 567 番「SO 選択 商品一覧」が呼び出される。
- 「商品#」を通りすぎるときに(パークの有無に関わらず)、条件 34 に合致すればエラーが表示される。

などのことがすぐにわかります。

このように、イベント指向プログラムは、Magic プログラムの作成・メンテナンスを容易にしてくれる大きな可能性を持っています。ぜひとも、イベントハンドラに熟達してください。

Magic eDeveloper V10

Magic eDeveloper V10

レコードメインからイベントへ

第 1 版　2007 年 10 月 12 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
マジック ソフトウェア・ジャパン (株)
http://www.magicsoftware.co.jp/