Magic eDeveloper V10
コーディングサンプル
Version 2
（モデル編）

Magic Software Japan K.K.

Enabling Business with Superior Technology

初版　2007 年 10 月 10 日

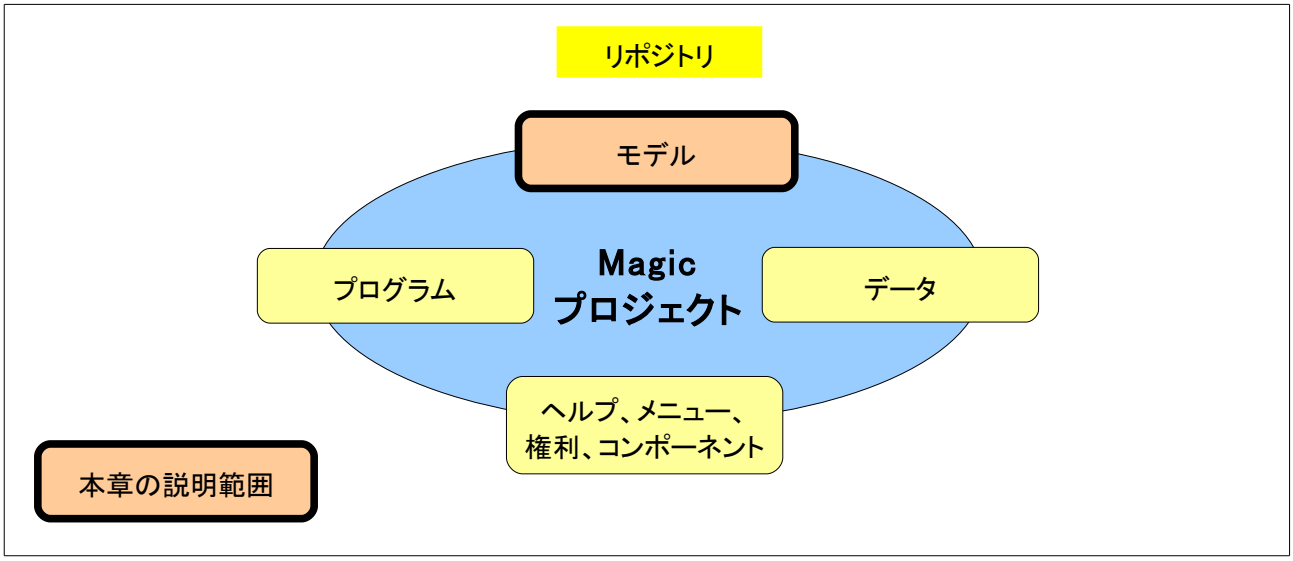
マジックソフトウェア・ジャパン株式会社

目次

4 モデル

V10 の **モデル** は、dbMAGIC V8 およびそれ以前では タイプ と呼ばれていたもので、モデルリポジトリ（旧バージョンでは、「タイプ辞書」）にまとめて定義します。モデルリポジトリは、データリポジトリ（ファイル辞書、テーブル辞書）、プログラムリポジトリ（プログラム辞書）と並んで、Magic のプロジェクト（アプリケーション）を構成する主要要素であり、Magic の生産性と保守性の大きな源泉のひとつとなっていました。

本章では、最初に V10 でのモデルのメカニズムについて簡単に説明した後、サンプルアプリケーションでのモデルリポジトリを例にとりながら、V10 でのモデルの機能を最大限生かすことにより、より生産性と保守性の高いアプリケーションを作成する方法について学んでいきます。

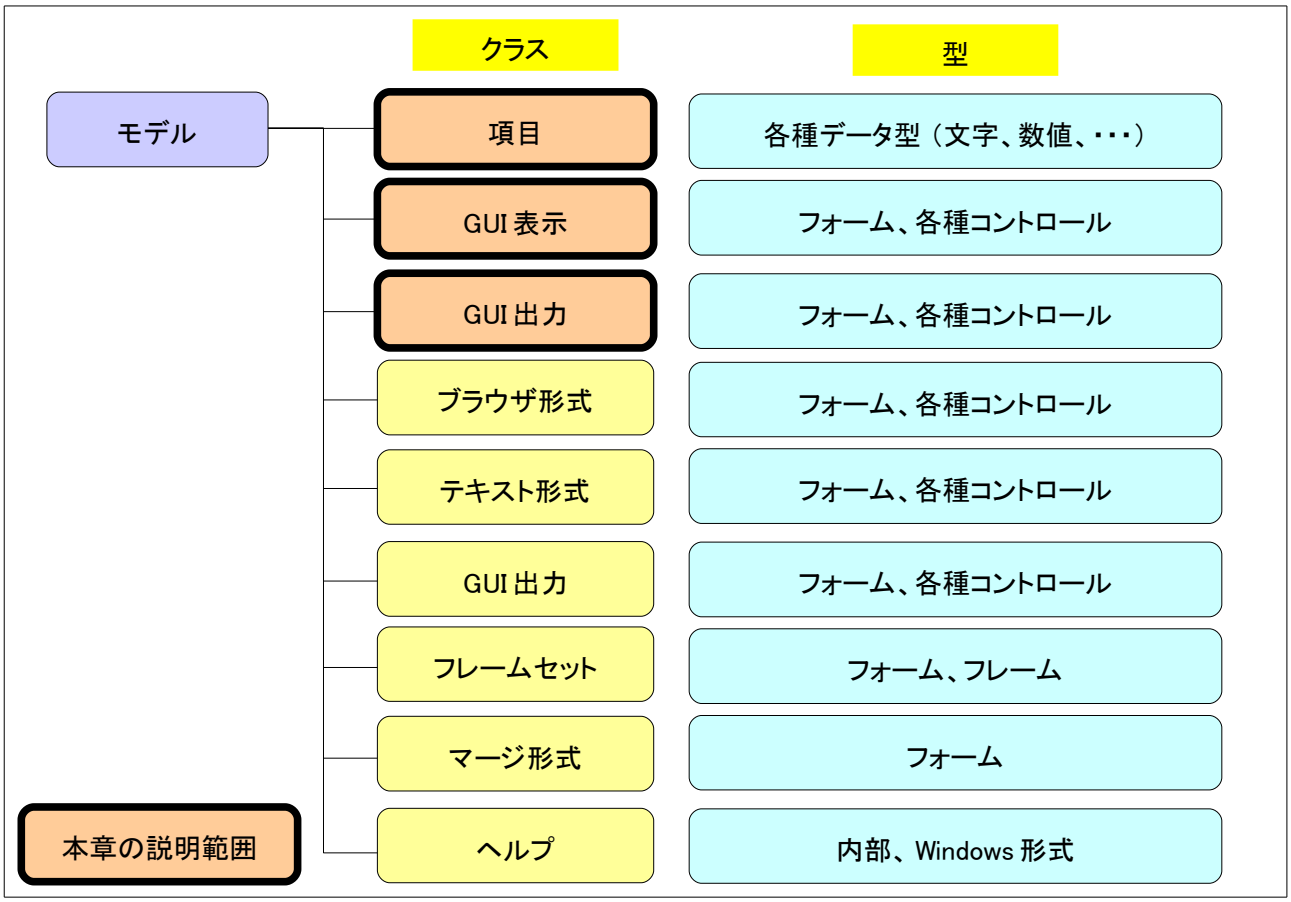


4.1 モデルとは？

モデルとは、Magic のオブジェクトが継承することのできる一連の特性のことです。

旧バージョンでは、モデルは「タイプ」と呼ばれていて、データ項目に対するタイプしか定義することができませんでした。

Magic V9 より、タイプリポジトリは **モデルリポジトリ** と呼ばれるようになり、モデルの対象も、データ項目のみならず、GUI 部品（フォーム、コントロール）、ヘルプなどに拡張されました。Magic V10 では、次のような**クラス**（種別）のモデルがあります。そして、それぞれに **型**（細分類）が細分されています。



また設定可能な項目も詳細に及ぶようになったので、モデルを活用することにより、更なる生産性の向上を図ることができます。

モデルは、**モデルリポジトリ** に定義します。一度、モデルがモデルリポジトリに定義されれば、アプリケーション中で繰り返し使うことができます。

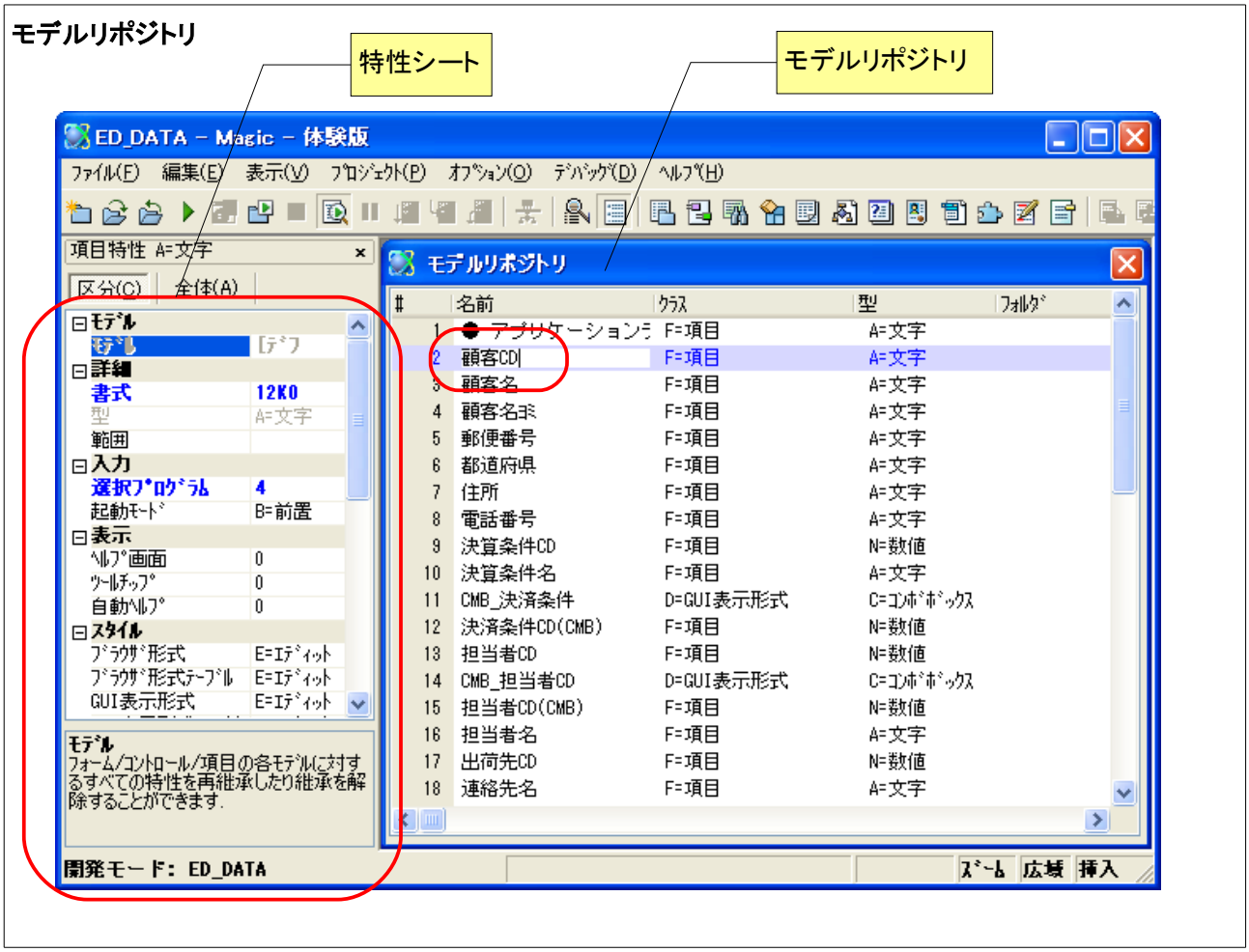
モデルを使用する主な理由は次のとおりです。

- 時間の節約 – モデルを一度定義したら、何度でも使うことができます。
- メンテナンスの簡易化と修正の集中化 – 個別のオブジェクトにではなく、一つのモデルを修正するだけですみます。
- 一貫性 – 異なったテーブルにまたがってカラムのモデルが一致していることを保証します。

更に、同じく V9 から導入された**コンポーネント**機能を使って、多くのプロジェクトで共通的に使われるモデルをコンポーネント化することにより、優れたメンテナンス性を得ることができます。

4.2 モデルの定義

モデルは、モデルリポジトリに定義します。下図は、サンプルアプリケーションのデータコンポーネント ED_DATA のモデルリポジトリです。



画面の右半分がモデルリポジトリで、このプロジェクトに定義されているモデルが一覧で表示されます。ここには、モデル名とクラス、型、その他の情報が表示されます。

画面の左半分は **特性シート** と呼ばれるもので、ここのモデルの特性を詳細に表示するものです。上図では、「顧客 CD」のモデルの特性を表示しています。

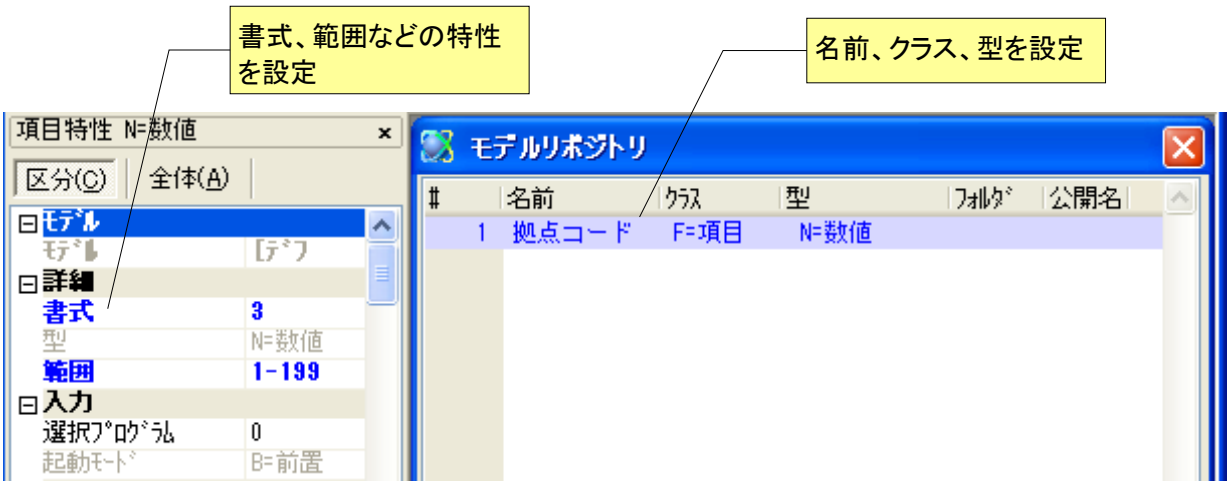
別の例として、新しい空のプロジェクトに、モデルをひとつ定義してみましょう。このモデルは、

- 名前は「拠点コード」
- クラスは「項目」
- 型は「文字型」
- 書式は「3」
- 範囲は「1-199」

です。

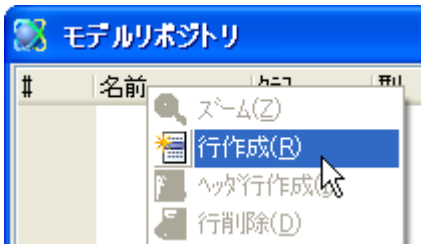
モデルリポジトリは次のようになります。

モデルの定義

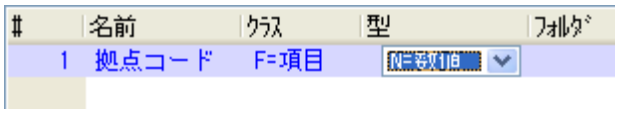


モデルの定義は、次の手順で行います。

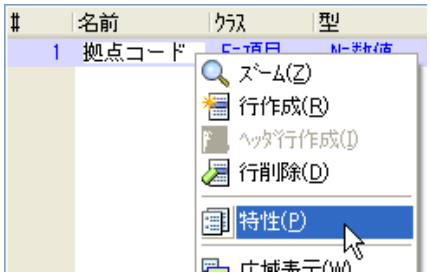
1. モデルリポジトリを開きます。(Shift+F1)
2. ポップアップメニューで、「行作成(R)」を選択します。
   (あるいは F4 キーを押します)
   → 新規行が作成されます。



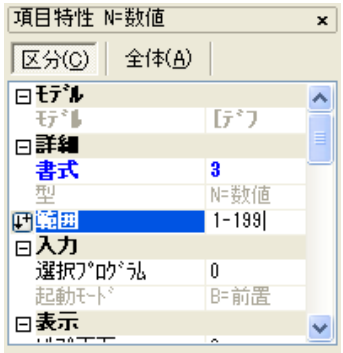
3. 名前、クラス、型を設定します。



4. ポップアップメニューで「特性(R)」を選びます。(あるいは、Alt+Enter を押します)。
   → 特性シートが開きます。



5. 特性シート上で、書式と範囲を設定します。



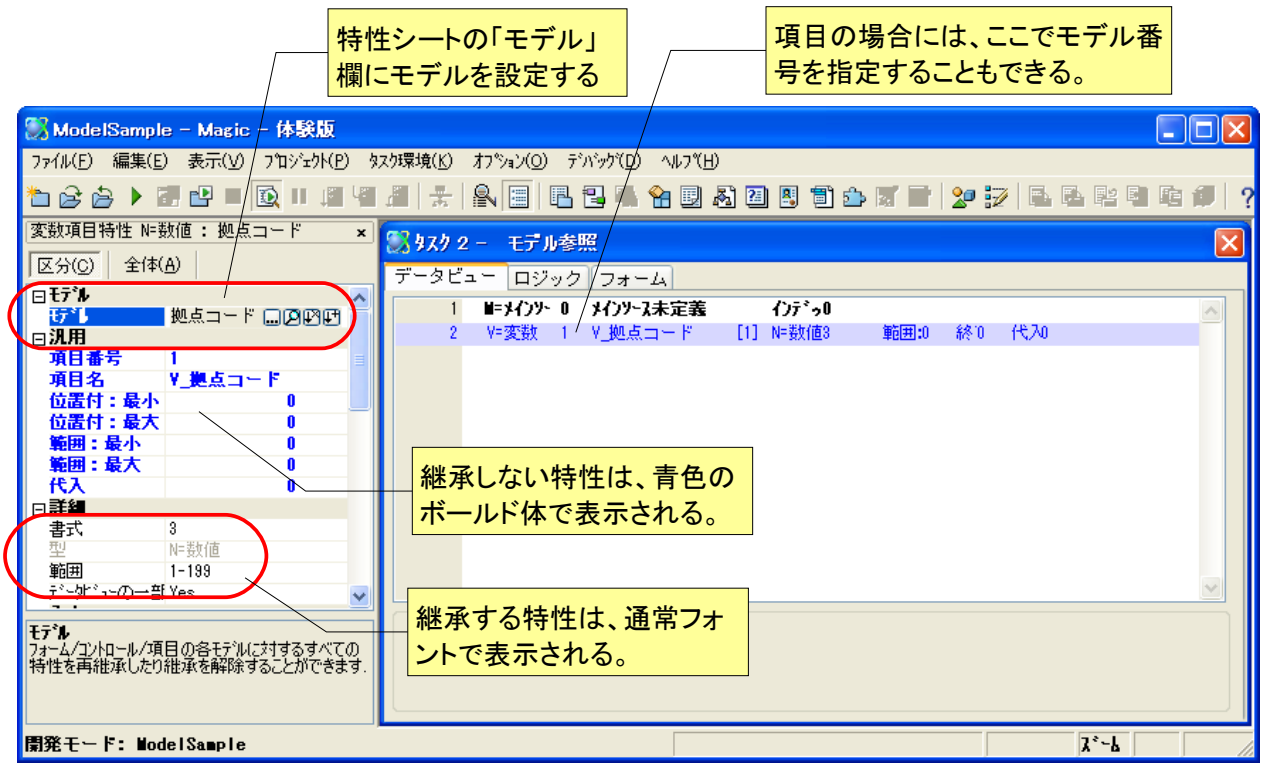
これで、「拠点コード」モデルの定義ができました。

4.3 モデルの利用

モデルの利用は、特性シートの「モデル」欄にモデルを設定することにより行います。

例えば、下図は、タスクの中で変数項目「V_拠点コード」を定義し、「拠点コード」モデルを参照している例です。

モデルの参照（タスクの変数でモデル「拠点コード」を参照）



- 上図では、変数「V_拠点コード」と、その特性シートが表示されています。
- 特性シートの「モデル」欄では、「拠点コード」モデルが設定されています。
- 特性シートの「書式」「範囲」などは、「拠点コード」モデルから特性値を継承しており、それぞれ、「書式」は「3」、「範囲」は「1-199」となっています。
- 特性シートの「汎用」カテゴリにある特性値（項目番号、位置付最小/最大など）は、タスクの中で定義される変数に特有な特性値であり、モデルから継承不可です。従って、これらの特性はモデルリポジトリ中には現れていません。

この図で見るように、項目の特性値には、モデルから継承する特性値と、継承しない特性値とがあり、それぞれ、通常フォント（黒色）と青色ボールド体で表示されます。モデルからの継承については、次節でもうすこし詳しく説明します。

項目モデルの場合には、特性シートの「モデル」欄でモデル名を設定することもできますが、タスクエディッタの「V=変数」行の第 2 パラメータ（モデル番号の欄）で、モデル番号を設定することによりモデルを参照することも可能です。実際、Magic 開発者はこちらの方法を使うのが一般的です。

4.4 モデルの継承メカニズム

V10 でのモデルのメカニズムは、特性値の **継承** という概念をもとにしています。ここでは、継承について説明します。

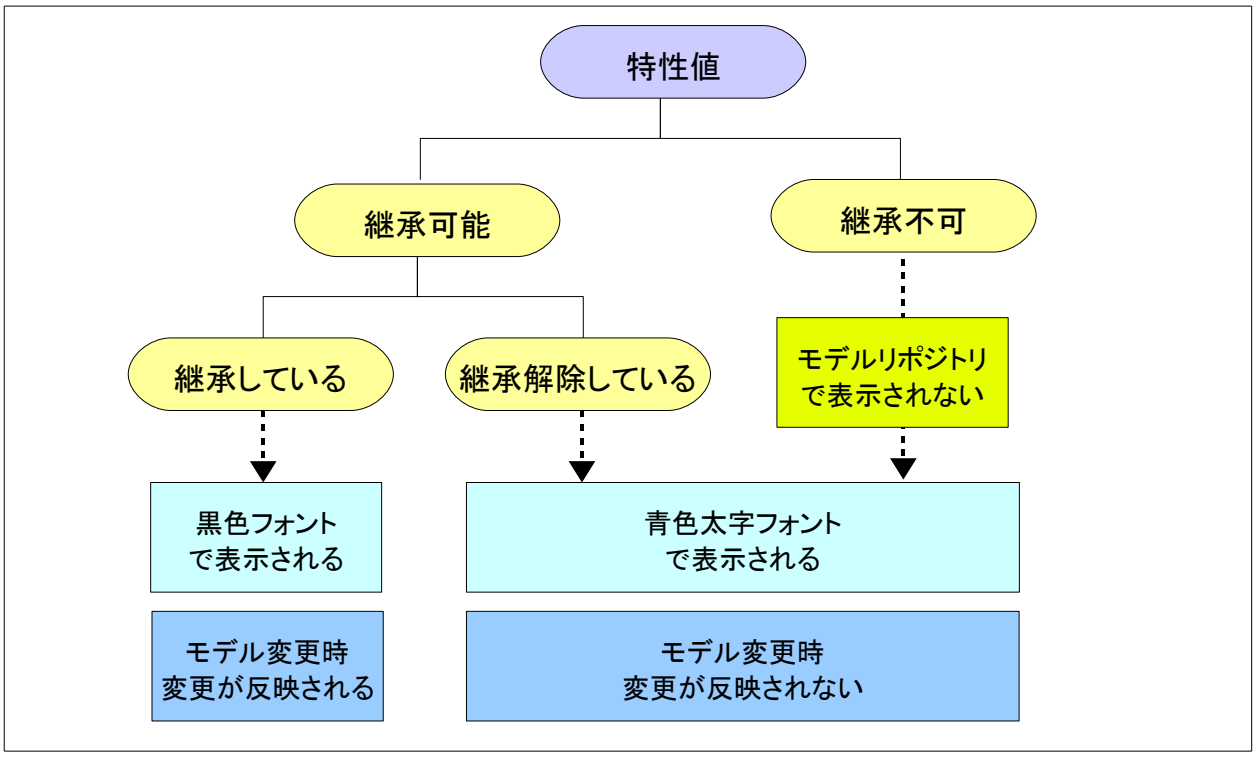
4.4.1 継承とは

継承とは、オブジェクト(データ項目、GUI コントロールなど) の特性値をモデルより引き継いで同じ値となる、ということで、基本的にモデルを参照しているオブジェクトは、継承可能なすべての特性の値をモデルから継承します。モデルでの特性値の設定が変わった場合、継承されている特性の値も自動的に変わります。これは、ちょうどガラス越しに遠方の物体を見ているようなものです。

ただし、オブジェクトのレベルで、**継承を解除** することもできます。継承を解除された特性は、モデルとは異なる値を持つことができます。この場合、モデルで特性値の設定が変わっても、オブジェクトのレベルでその変更は反映されません。同じガラスのたとえを用いれば、ガラスの上にステッカーが張ってあるようなもので、その向こう側にある物体が変わったとしても、ガラスのこちら側から見るとステッカーだけが見えて、物体が変わったことがわからない、というようなものです。

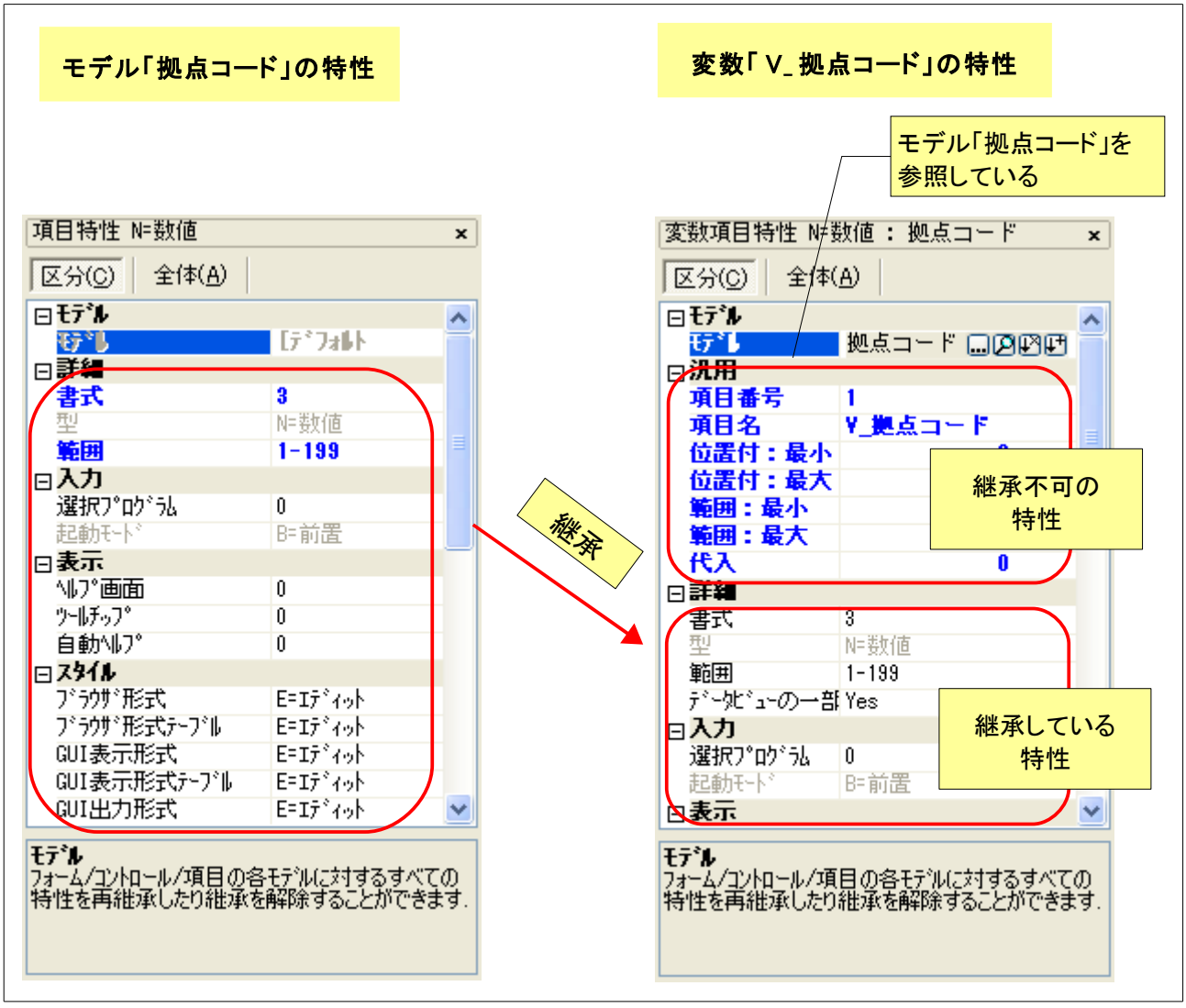
また、**継承不可**の特性もあります。これは、本質的に、継承する意味が少ないもの、あるいは他の項目とのからみがあって継承値の意味がないものなどです。例としては、項目特性の「記憶形式」や「データベース定義」などがあります。これらの特性は、格納する DBMS によって最適値が異なるので、データリポジトリのカラムに設定した時点で Magic が設定を変更しますから、DBMS の情報がわからないモデルのレベルでは設定することができない特性です。

このような継承不可の特性は、モデルリポジトリでは表示されません。



4.4.2　モデルと継承の例

例として、前述の「拠点コード」モデルの特性シートと、それを参照している変数項目「V_拠点コード」の特性シートとを並べて見比べてみたものです。



これから、次のようなことがわかります。

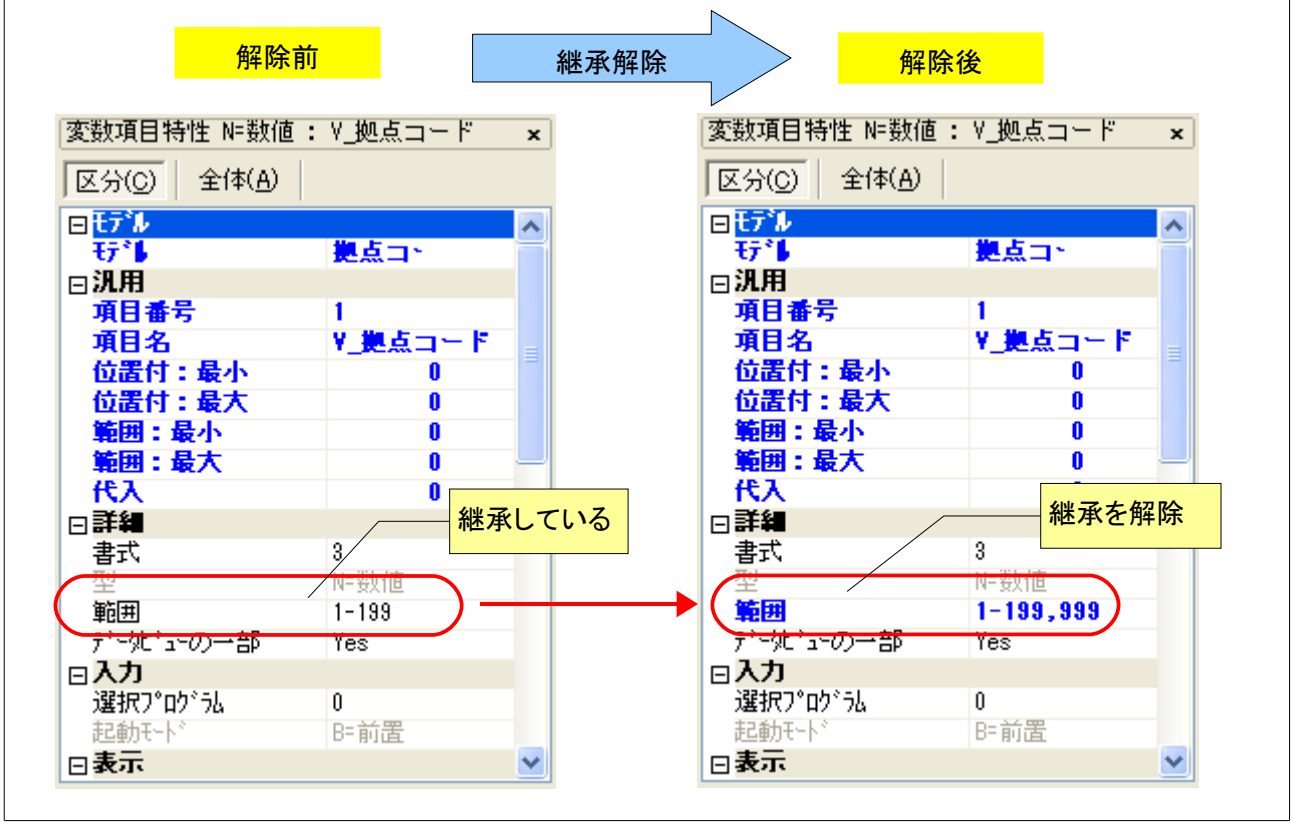
- 変数「V_拠点コード」の「モデル」特性では、「拠点コード」モデルを参照しています。
- 「書式」「範囲」などは、「拠点コード」モデルから特性値を継承しており、通常フォントで表示されています。
- 特性シートの「汎用」カテゴリにある特性値（項目番号、位置付最小/最大など）は、タスクの中で定義される変数に特有な特性値であり、モデルから継承不可の特性ですので、青色ボールド体で表示されています。また、これらの特性はモデルリポジトリ中には現れていません。

4.4.3 継承の解除

モデルから継承されている特性値のうち、モデルとは異なる値を設定したい場合には、特性シート上で新しい値を直接設定することにより、継承を解除することができます。

例えば、「拠点コード」の例で、「範囲」の値を「1-199,999」にしたいとします。この場合には、特性シートの「範囲」欄に、直接「1-199,999」とキー入力します。すると、「1-199,999」という値が設定され、継承が解除されていることを示すために、青色ボールド体のフォントで表示されます。

継承の解除（「範囲」特性の継承を解除した例）



4.4.4 継承とモデルの変更

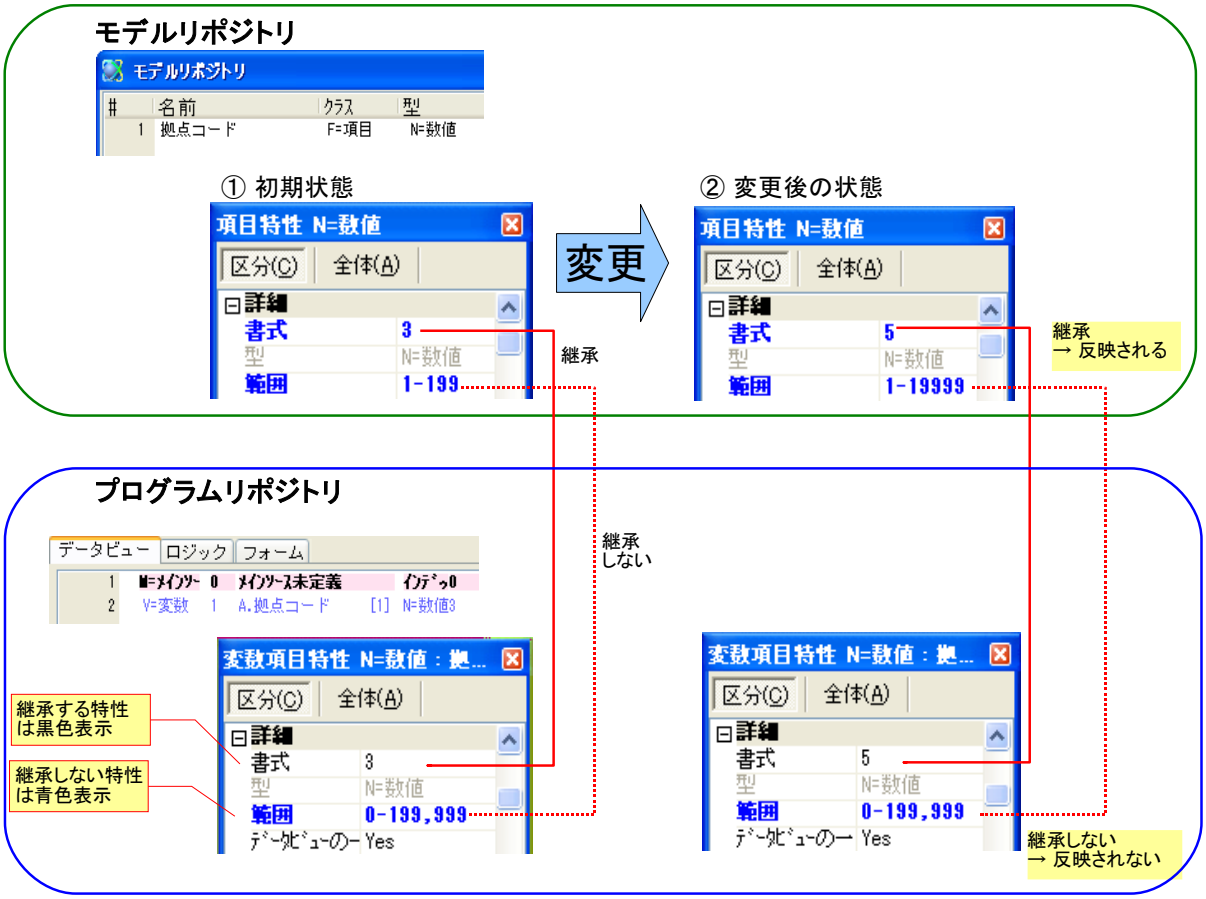
ここで、継承されている特性（先の例では「書式」など）と、継承されていない特性（「範囲」など）について、モデルリポジトリでモデルの定義を変更したばあいに、変更がどのように反映されるかというと、一般的に

> **継承されている特性にはモデルの変更が反映されるが、継承を解除されている特性には、モデルの変更が反映されない**

となります。

例えば、次の図は、「拠点コード」モデルの設定を変更し、**書式** を「3」から「5」、**範囲**を「1-199」から「1-1999」とした場合に、変更前と変更後の特性の値を表したものです。

- モデルから特性値を継承している **書式** の方は、モデルでの変更が反映されて、「5」になります。
- しかし、モデルからの継承を解除されている **範囲** の方は、モデルでの変更に関わらず、以前の値「0-199,999」のままになっています。



モデルの継承と継承解除とを自由に行えることが、V10 のモデルの便利さでもあるのですが、一方において、継承解除されている特性はモデルの変更を反映しないので、Magic の生産性と保守性の源の一つであるモデルの良さを殺してしまうことにもなります。

従って、一般的に言って、モデルの良さを最大限生かすためには、継承が解除されている特性ができるだけ少なくなるようにすること、すなわち、

> **特性シート上での青色（継承解除の値）をできるだけ減らしましょう**

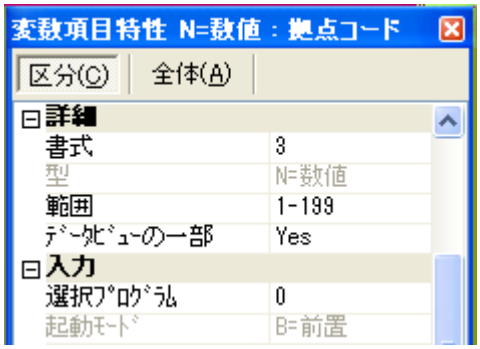
ということを推奨します。逆に言うと、継承解除が最小になるように、モデルを設計することが肝要です。

4.4.5　継承の解除と再継承の操作

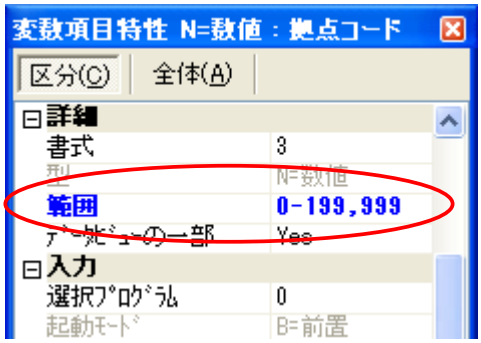
ここで、継承、継承解除、再継承などを行う操作について、変数にモデルを使う場合を例にとって、簡単に説明します。

継承の解除と再継承を行うには・・・

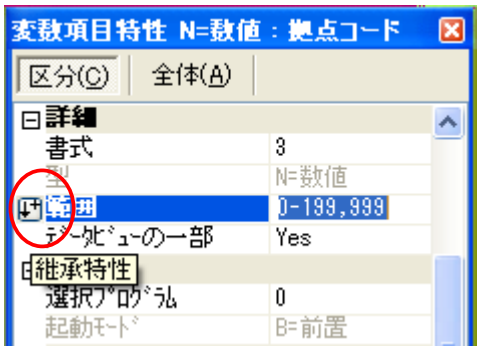
6. 最初に、変数を定義して、モデル欄にモデル番号を設定した状態では、継承可能な項目はすべて継承されています。



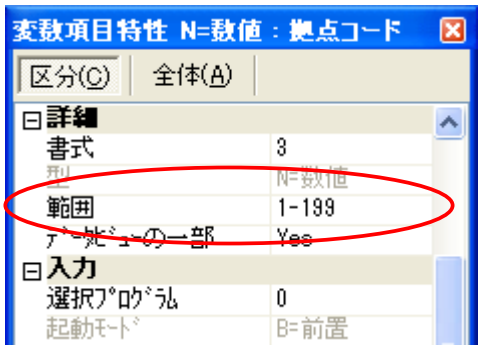
7. **継承の解除**: 特定の値の継承を解除するには、値を直接入力します。右図では、**範囲** 特性の値の継承が解除されました。



8. **再継承**: 継承が解除されている特性に、再度継承を復旧するには、その特性にカーソルを合わせて、左側に現れる  アイコンをクリックします。



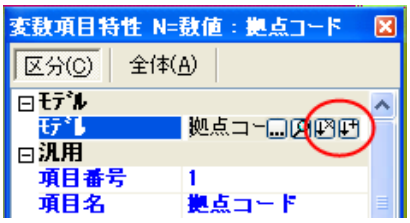
9. すると継承が復旧されて、値がモデルでの設定値に戻るとともに、黒色の通常フォントで表示され、特性が継承されていることを表すようになります。



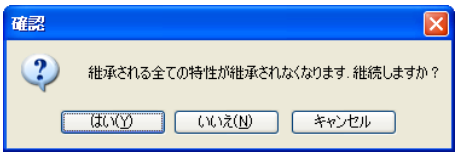
以上は、個々の特性について継承を解除・復旧する操作でしたが、すべての継承可能な特性について、一度に継承解除、継承復旧することもできます。

全特性を一度に継承解除するには・・・

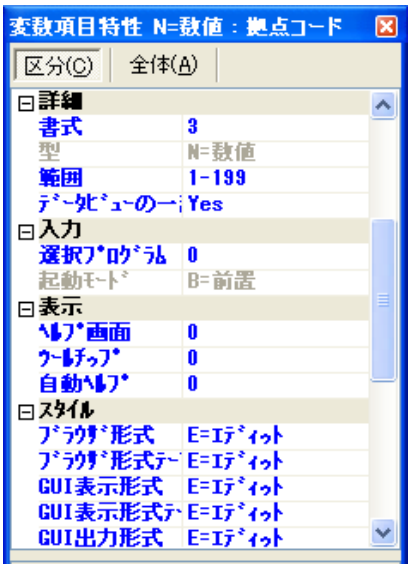
1. 特性シートの モデル 欄に 4 つのアイコンがありますが、そのうちの右の二つが、継承解除と再継承に関するものです。



2. 特性のすべてを継承解除するには、 アイコンをクリックします。右図のような確認ダイアログが出るので、**はい(Y)** を押します。

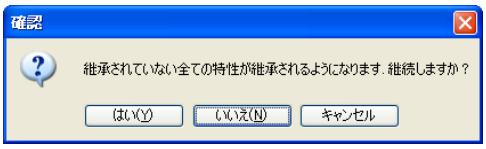


3. すると、特性シート上のすべての特性が、継承解除を表す青色ボールド体のフォントで表示されるようになります。

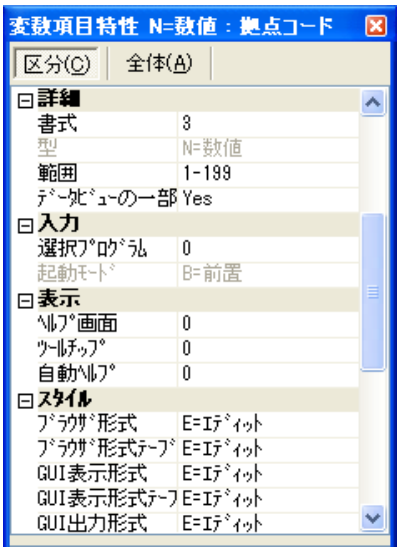


全特性を一度に再継承させるには・・・

1. 次に、継承可能な特性を一度にすべて再継承させるには、モデル 行の  のアイコンをクリックします。
   右図のような確認ダイアログが出るので、**はい(Y)** を押します。



2. すると、特性シート上の特性のうち、継承可能な特性がすべて、継承されていることを表す黒色の通常フォントで表示されるようになります。



再継承すると、値はすべてモデルリポジトリ上の値になります。特性シートで直接設定した値はなくなってしまいますので、注意してください。

4.5 モデルとコンポーネント

モデルには大別して、

- アプリケーションに固有なモデル（顧客番号、商品名など）
- アプリケーションに関係なくMagicで作ったアプリケーションであるならばどれでも非常によく使われる共通モデル（終了プッシュボタン、ファイル名など）

とがあります。

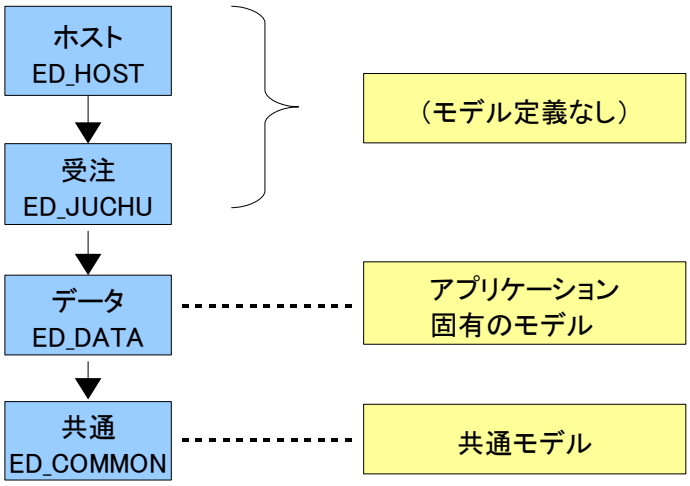
dbMAGIC V8までは、アプリケーションは一つのCTLファイルに格納され、分割することができなかったので、アプリケーション固有のモデル(タイプ)も共通モデル(タイプ)も一つのCTLに入れておく必要がありました。

V9からは、アプリケーションはコンポーネントに分割できるようになり、コンポーネントを複数のアプリケーションで共有できるようになりました。

この利点を生かして、サンプルアプリケーションでは、次のようなモデル定義をしています。

- アプリケーション固有のモデルは データコンポーネント ED_DATA で定義
- 共通モデルは共通コンポーネント ED_COMMON で定義
- 受注コンポーネント ED_JUCHU、ホスト コンポーネント ED_HOST では、モデル定義なし。

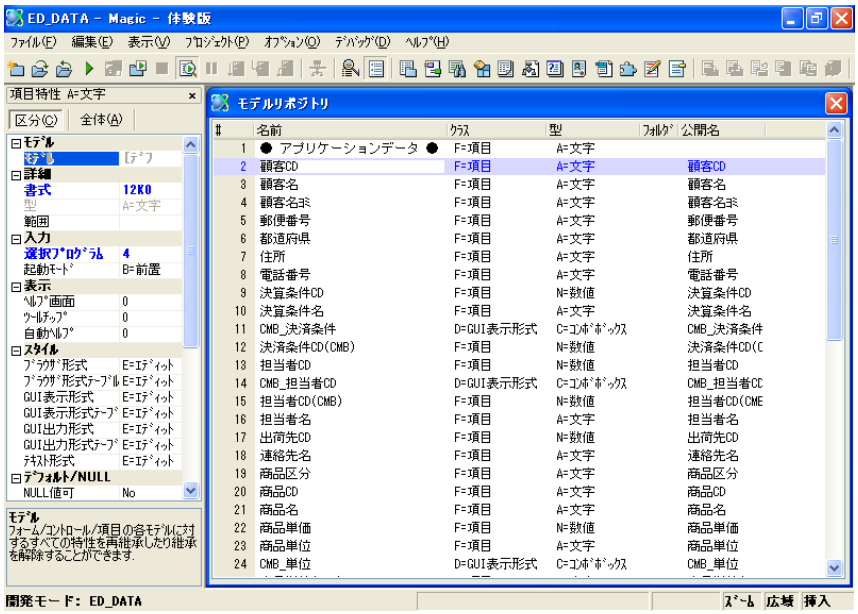
コンポーネントとモデル定義



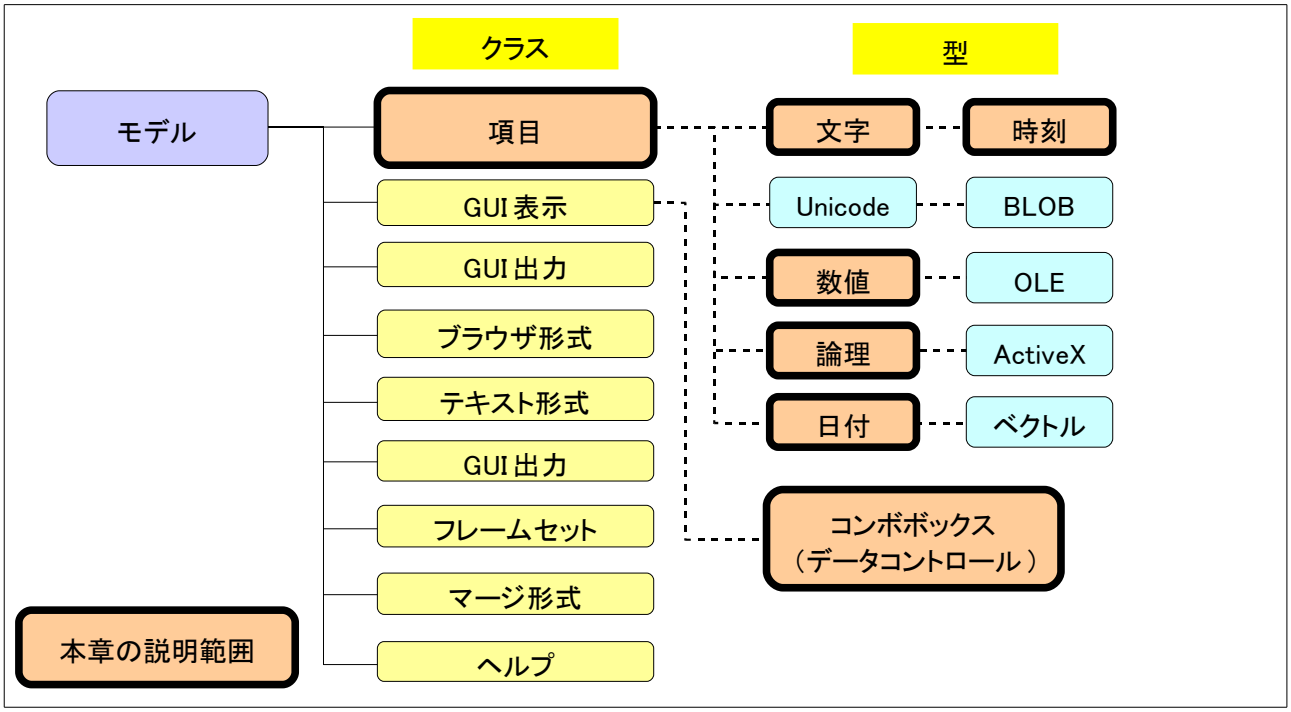
5 アプリケーションデータのモデル

モデルのうち、一番多用されるのは、アプリケーションのデータについてのモデルでしょう。このモデルは、サンプルアプリケーションの ED_DATA コンポーネントに定義されています。(下図)

これには、各種コード(顧客コード、商品コード、出荷先コード、・・・)、名前類 (顧客名、商品名、・・・)、住所、電話番号、等々、アプリケーションドメインに現れるデータタイプがすべて含まれます。



これらはほとんどが「項目」クラスのモデルで、この「型」としては、文字、Unicode、数値、論理、日付、時刻、その他があります。その他には、「GUI 表示」クラスのコンボボックスも、「データコントロール」として定義されています。データコントロールについては、5.3 データコントロール で説明します。



5.1 命名規則

ここでは、モデル名はモデルの意味するものそのままとし(「顧客 CD」「顧客名」など)、特別なプレフィックスなどはつけていません。これは、データリポジトリのカラムテーブルでモデルを参照して設定した場合、カラム名にデフォルトでモデル名が設定されることを考慮したものです。

例外としているのは、データコントロールの設定されているコンボボックス (CMB_担当者 CD など)です。データコントロールはデータリポジトリでカラム定義に使われることがまずないことと、データコントロールでない、通常のモデル(担当者 CD)を区別する必要があるために、データコントロール(この場合 コンボボックス) であることがすぐわかるように、CMB_ というプリフィックスをつけてあります。

流儀によっては、モデル名の先頭に、モデルの種類(データ型や用途)をプレフィックスとして付けたり、数値の場合には桁数をつけたりするやりかたもあります。こうしておくと一覧からモデルを選択する場合に区別しやすくなりますが、カラムテーブルで設定した場合に名前以外の情報を取り除く必要があります。

5.2 「項目」クラスのモデルで便利な設定

5.2.1 書式

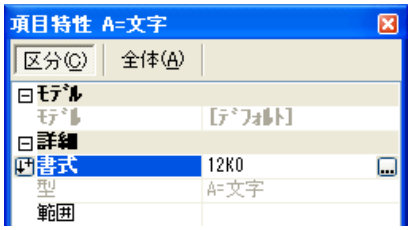
書式については、V8 までに慣れている開発者にとってはよく慣れたものなので、特記することもないと思いますが、TIPS として、次のような書式を活用すると便利かもしれません。

- **A 書式** (全データ型に利用可能): A 書式は **オートスキップ** を行う書式であり、書式で決められた桁数のデータが入力された場合に、自動的に次項目に進みます。例えば、書式「5」の項目があった場合、ユーザが 5 文字を入力したら、Tab キーや Enter キー、[→] 矢印キーなどを押さなくとも、自動的に次項目に移動します。
- **K 書式** (文字型およびユニコード型で利用可能): K 書式は、IME (仮名漢字変換フロントエンド)を制御する書式で、エディットコントロールの 漢字入力 欄に設定するのと同じ効果があります。書き方は、K の直後に数字の 0 ～ 9 の値を設定します。この数字の意味は、以下の通りです。

値	入力モード	文字サイズ	入力方法	入力例
0	OFF	–	–	–
1	全角ローマ字かな	全角	ローマ字	かな
2	全角かな	全角	直接	のちみち(KANA)
3	全角ローマ字カナ	全角	ローマ字	カナ
4	全角カナ	全角	直接	ノチミチ(KANA)
5	半角ローマ字カナ	半角	ローマ字	ｶﾅ
6	半角カナ	半角	直接	ﾉﾁﾐﾁ(KANA)
7	全角英数	全角	直接	KANA
8	半角英数	半角	直接	KANA

例えば、10K5 と書くと、10 桁で IME は半角ローマ字カナ入力モードとなります。また、10K0 と書くと、IME は OFF となります。

サンプルでは、文字型のコード類(顧客番号、商品番号など)の書式に K0 書式が使われています。右図は顧客コードの書式です。



IME の制限により、上のモードのすべてが有効にならない場合があります。

5.2.2 選択プログラム

選択プログラム特性は、その項目でズームした場合に呼び出される一覧選択プログラムです。各種のコードデータ (顧客コード、商品コード)などには、それぞれのマスタから選択するための一覧選択プログラムがあると思いますが、そのプログラムを設定しておきましょう。

5.2.3 DB カラム名

DB カラム名は、SQL データベースを使った場合に、実際に DBMS 上に作成されるテーブルのカラム名を指定するものです。モデルがデータリポジトリで利用されることを考慮して、DB カラム名も設定しておきましょう。

なお、DB カラム名は DBMS 上に作成されるテーブルで使われる名前であるので、利用を想定している DBMS での命名規則に則っていることが必要です。一般的に言って、あまり長い名前は使わないこと、特殊文字は使わないこと、半角カナ文字は避けることなどに留意しましょう。

5.2.4 範囲

範囲特性は、この項目に入力可能なデータ範囲を定義するものです。例えば、取引タイプ CD には、「I:在庫,D:直販」という範囲が設定されています。

範囲を指定した場合には、できるだけデフォルト値も設定しておきましょう。デフォルト値を設定していない場合には、文字型の場合は空文字列、数値型の場合には 0 がデフォルトになりますが、このデフォルト値が、範囲指定と矛盾していると、デフォルト値が不正な値、ということになり、データモデルの観点から好ましくありません。

5.3 データコントロール

5.3.1 データコントロールとは

データコントロール というのは、オンライン画面上に配置されるコントロールのうち、多者択一型のコントロール（コンボボックス、リストボックス、及びラジオボタン）であり、その選択肢をデータソース（テーブル）から取ってくるものを言います。

データコントロールは、Magic V9 からサポートされるようになった機能で、V10 でもそのままサポートされています。

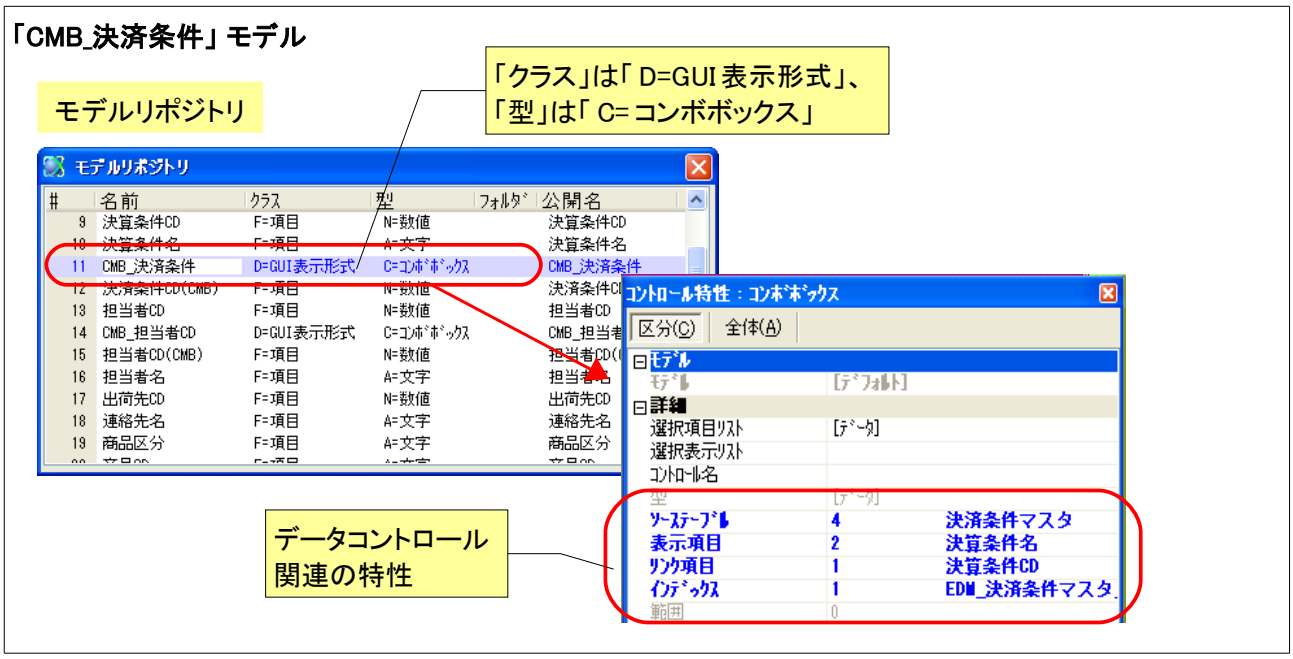
V8 およびそれ以前のバージョンでは、他者択一型のコントロールの選択肢は、「範囲」に指定していました。そのため、選択肢はアプリケーション開発時に固定的な値になってしまい、実行時に動的に変更しようとすると、「ラベル」特性に文字型の式を設定し、選択肢をカンマで区切った文字列として連結する、というようなプログラミング技法が必要でした。

V9 以降でも、この手法は引き続きサポートされていますが、それよりもっと簡単なやりかたとして、以下のような特性を指定するだけで、実行時に動的に選択肢となるデータを、データソース（テーブル）から取ってくることができるようになりました。

- ソーステーブル: データリポジトリ中のデータソースの番号を指定します。このデータソースからデータを取ってきます。
- 表示項目: データソースのカラム番号を指定します。このカラムのデータが、選択肢の一覧として画面上に表示されます。
- リンク項目: データソースのカラム番号を指定します。このカラムのデータが、選択肢の実際のデータとなります。
- インデックス: データソースのインデックスの番号を指定します。画面上での選択肢の並び順を決めます。
- 範囲: ズームすると、項目範囲を指定するダイアログが出てきます。これには、項目（データソースのカラム番号を指定)と、開始、終了（それぞれ式番号を指定します）を指定します。複数の項目を AND 条件で結合することもできます。この特性は、式を参照するものなのでタスクの中でだけ意味がありますから、モデルリポジトリでは設定できないようになっています。

5.3.2 データコントロールの例

例えば、下図は「CMB_決済条件」というモデル（データコンポーネント ED_DATA のモデルリポジトリの 11 番に定義）です。



これに見るように、「クラス」は「D=GUI 表示形式」（オンラインフォームとして使うもの）、「型」は「C=コンボボックス」となっています。

その特性として、前述の「ソーステーブル」「表示項目」「リンク項目」「インデックス」などが設定されています。

- 「ソーステーブル」として、「決済条件マスタ」が指定されているので、このテーブルのデータをもとにして選択肢が表示されます。
- 「表示項目」として、「決済条件名」（決済条件マスタ の第 2 カラム）が指定されているので、画面上（コンボボックスの内容)としては、決済条件名 カラムのデータ内容（「15 締め翌月末払い」など）が選択肢として表示されます。
- 「リンク項目」として、「決済条件 CD」（決済条件マスタの第 1 カラム)が指定されているので、ユーザがコンボボックスでデータを選択した場合、実際の内部データとしては決済条件コードの数値型データ（「6」など）が選択されます。
- 「インデックス」はデータソースの第 1 インデックス「EDM_決済条件マスタ_決済条件 CD」となっているので、コンボボックスの内容は決済条件 CD 順に表示されます。

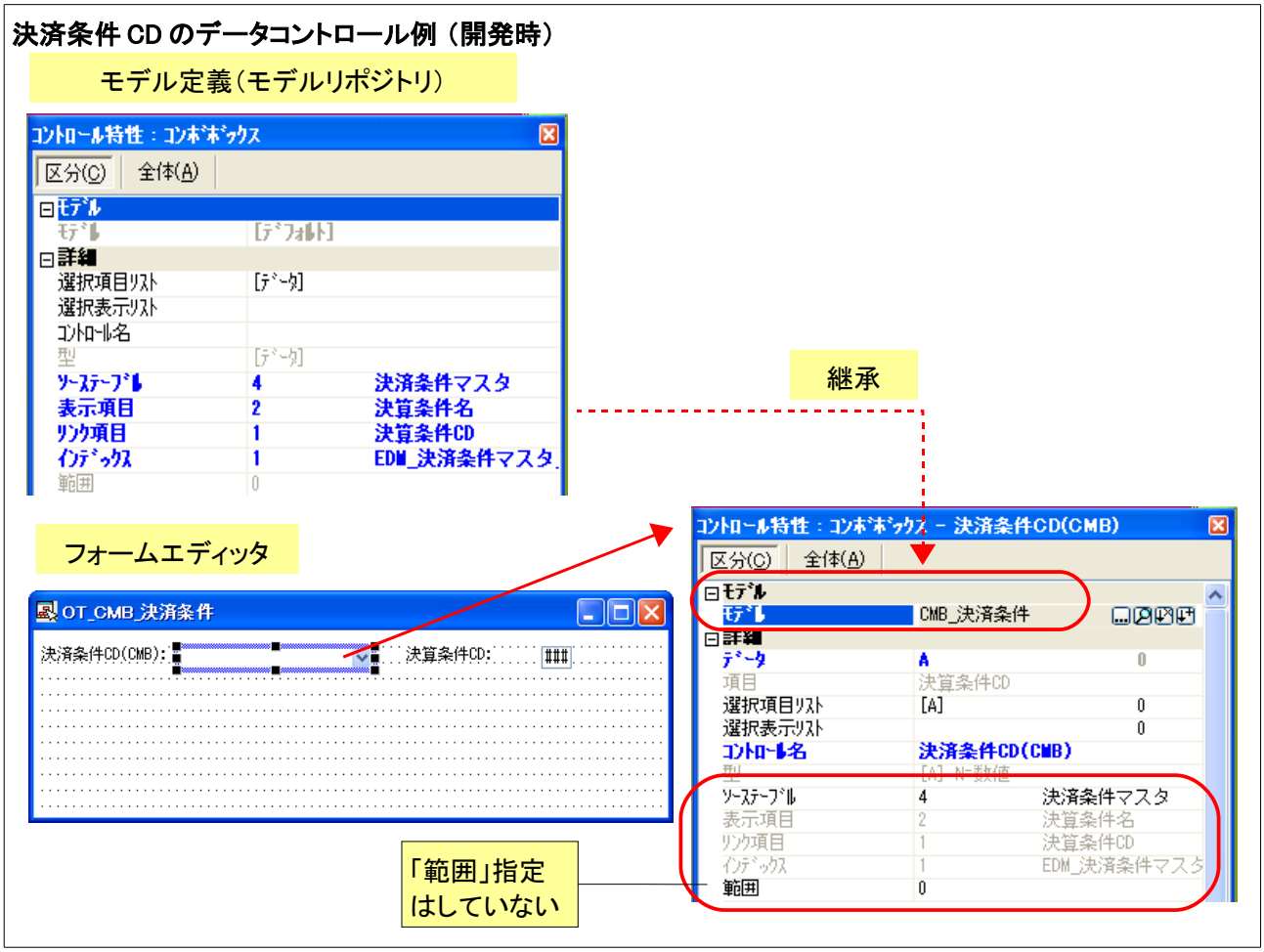
「CMB_決済条件」のほかに、データコンポーネントの中には、「CMB_」で始まる名前のデータコントロールのモデルがいくつか定義されています。

- CMB_担当者 CD
- CMB_単位
- CMB_課税区分 CD
- CMB_申告区分 CD

5.3.3　データコントロールの利用（開発時）

次の図は、このモデルを参照しているオンラインタスクのフォームの例です。ここでは、フォームにコンボボックスが配置されており、このコンボボックスではモデルとして、モデルリポジトリの「CMB_決済条件」を参照しています。そうすると、「ソーステーブル」などの特性が継承されてきます。ここでは「範囲」は指定していません（「0」のままになっています）

決済条件 CD のデータコントロール例（開発時）



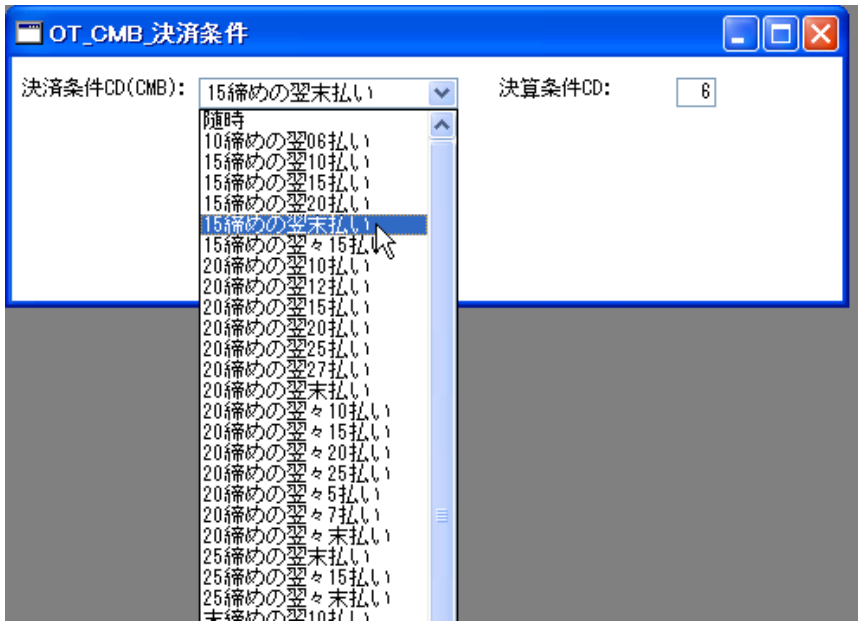
5.3.4　データコントロールの利用（実行時）

ここで、ソーステーブル「決済条件マスタ」の内容が、右図のようになっているとします。

「決済条件マスタ」のデータ内容

この状態で、タスクを実行すると、右図のようになり、決済条件CD のコンボボックスの選択肢が、決済条件マスタのデータからとられていることがわかります。

タスクの実行結果

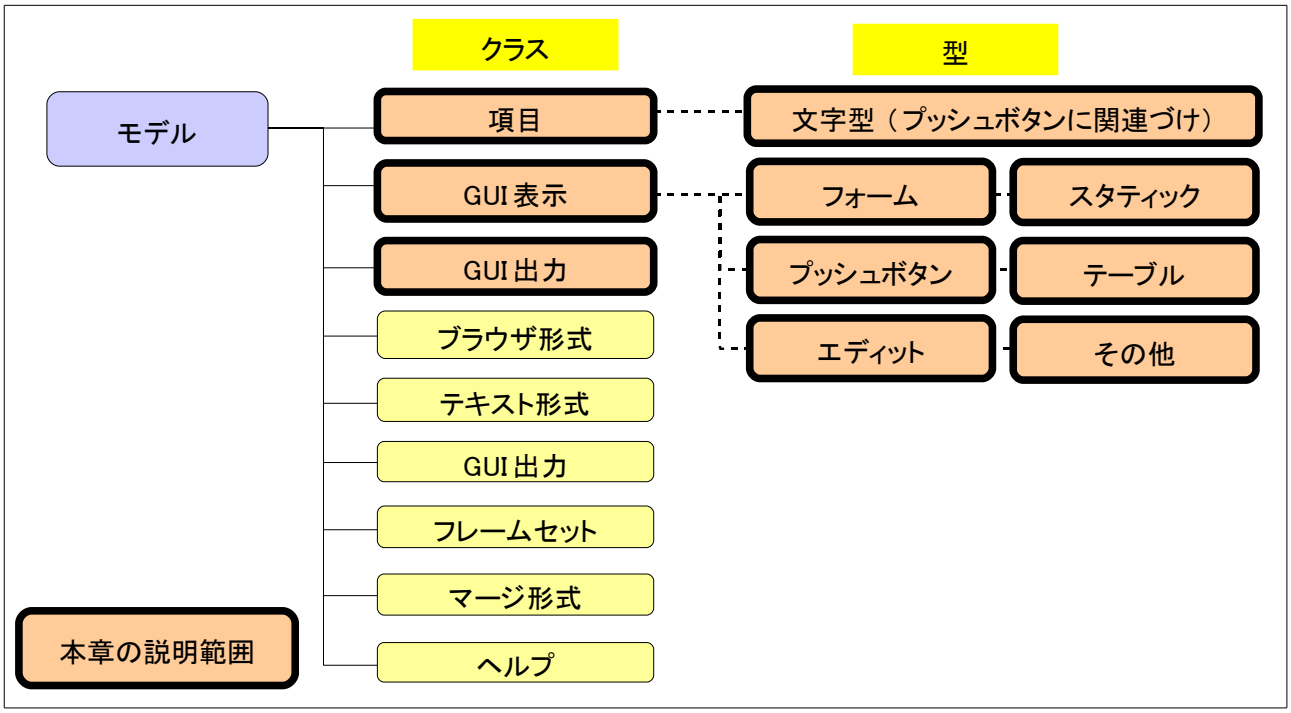


6 共通モデル

共通コンポーネント ED_COMMON プロジェクトでは、次のようなモデルが定義されています。

モデル種類	説明	例
プッシュボタン	よく使う内部イベントやユーザ定義イベントを**実行イベント**として設定してあり、よく使われるボタンラベルを**書式**として設定してあるものです。	PB_終了、 PB_U 実行 E
プッシュボタン項目	上記プッシュボタンモデルに関連付けを行った項目モデルです。	TB_終了、 TB_U 実行 E
GUI 表示形式	よく使われる GUI フォーム、GUI コントロールをモデル化したものです。	FRM_基本、 TBL_交互色
GUI 出力形式	印刷によく使われる GUI フォーム(出力形式)や、そのコントロールをモデル化したものです。	FRM_GUI 印刷基本 LBL_印刷タイトル (14pt、中央)
その他	一般的な項目モデル	ファイル名、 プログラム名、 インデックス番号

次節以下に、それぞれについて掘り下げて見ていきます。



6.1 命名規則

共通コンポーネントで定義しているモデルには、GUI 表示形式（オンラインフォーム）、GUI 出力形式（GUI 印刷）で使うフォームやコントロールのモデルが多くあります。これらは次のような簡単な命名規約に従って名前をつけています。

種別	命名規約（接頭子）	例
プッシュボタン	PB_	PB_取消
テーブル	TBL_	TBL_交互色
フォーム	FRM_	FRM_選択画面フォーム
スタティックテキスト	LBL_	EDT_中ｻｲｽﾞ(12pt)
エディット	EDT_	EDT_表示専用
プッシュボタンに関連づけられた項目	TB_	TB_取消
その他の項目	（なし）	ファイル名

6.2 内部イベントのプッシュボタンモデル

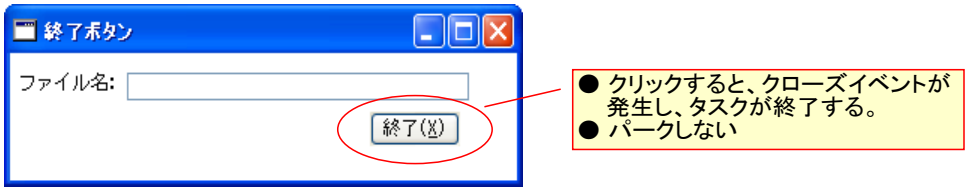
内部イベントというのは、Magic エンジンに対して何らかの動作を行うことを指示する機能を持ったイベントで、例えば「クローズ(C)」、「キャンセル(C)」「OK」「選択」「ズーム(Z)」などがあります。

内部イベントをプッシュボタンに割り当てることにより、機能を持ったボタンをフォームに配置することができます。さらに、これをモデル化することによって、フォームへのボタンの配置が非常に簡単になります。

例えば、「終了(X)」という名前で、クリックすると「クローズ(C)」イベントを発行する(タスクが終了する) というプッシュボタンは大半のオンラインフォームで使われる非常に頻度の高いプッシュボタンですが、こういうプッシュボタンはモデル化することにより開発時の手間が省けるようになります。

下図は、このプッシュボタンモデルを使った簡単な例です。

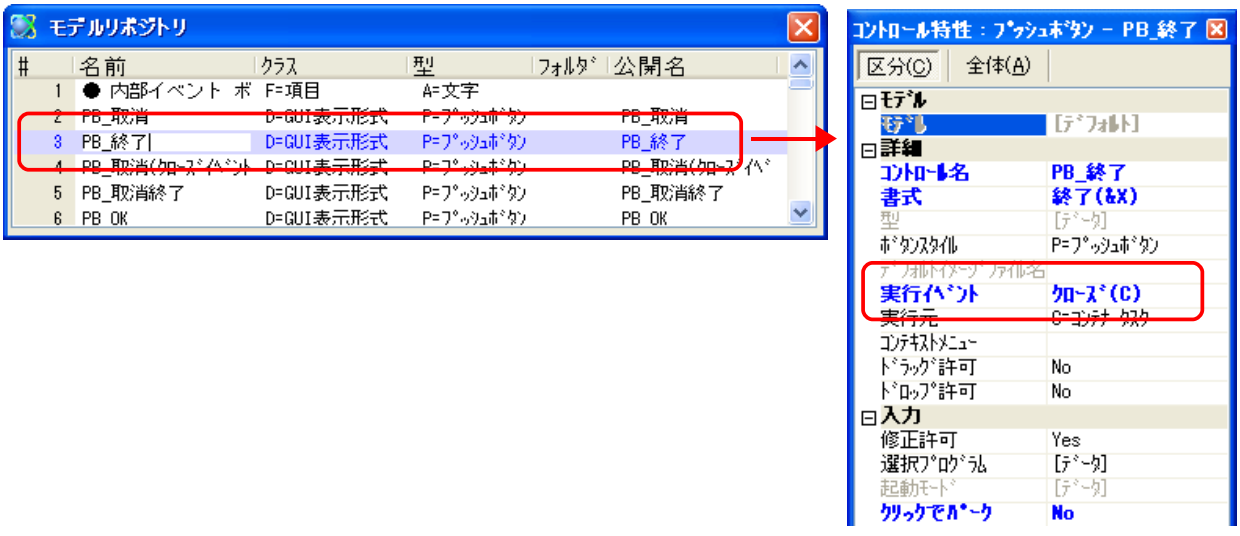
「クローズ(C)」内部イベントのプッシュボタンモデル
実行時のイメージ



6.2.1 プッシュボタンモデルの設定

このプッシュボタンモデルは、共通コンポーネント ED_COMMON のモデルリポジトリに「PB_終了」という名前で登録されています。

プッシュボタンモデル「PB_終了」



このプッシュボタンモデルは、次のような設定になっています。

- オンラインフォームで使うものなので、「クラス」は「D=GUI 表示形式」
- プッシュボタンコントロールとしたいので、「型」は「P=プッシュボタン」

- ボタンラベルを「終了(X)」としたいので、「書式」は「終了(&X)」
- ユーザがボタンを押したときにタスクを終了させたいので、ボタンを押したときに発行すべきイベントとして、「実行イベント」に内部イベント「クローズ(C)」を設定

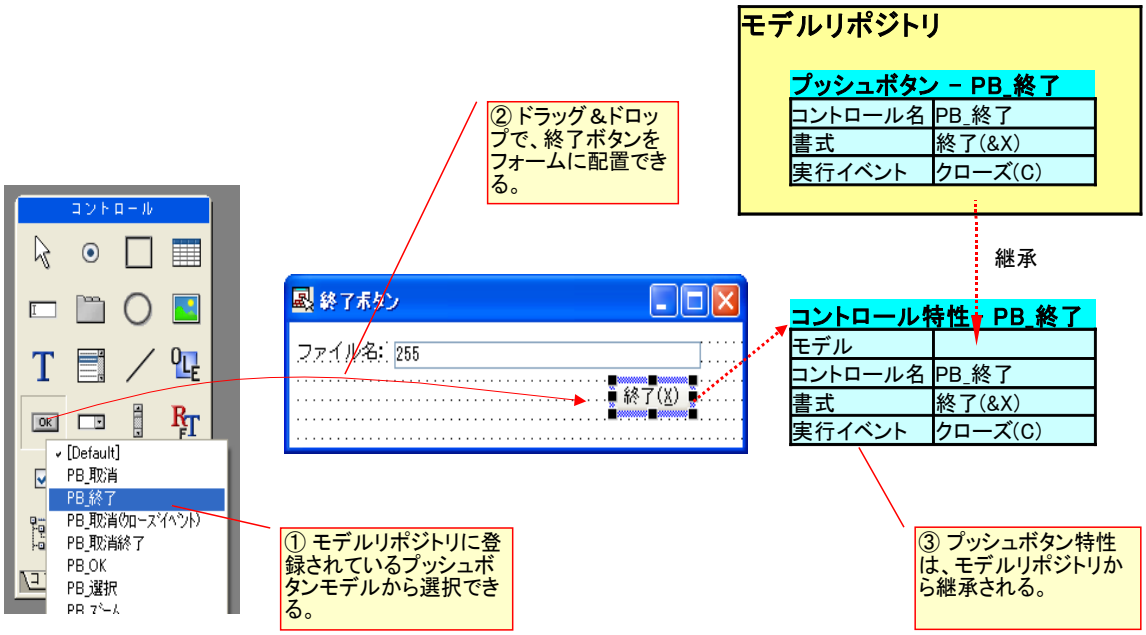
6.2.2 プッシュボタンモデルのフォームへの配置

フォームエディタでこのモデルを利用して、「終了」プッシュボタンを配置するには、次のようにします。

① コントロールパレット上で、プッシュボタンのところで、マウスで右クリックします。→ モデルリポジトリに登録されているプッシュボタンモデルの一覧が表示されますので、この中から「PB_終了」を選択します。

② ドラッグ &ドロップで、フォーム上にこのプッシュボタンを配置します。

③ プッシュボタンコントロールのコントロール特性を見てみると、「モデル」でモデルリポジトリの PB_終了モデルを参照しており、特性がすべてここから継承されていることがわかります。これにより、書式 (ボタンラベル)、実行イベントなどが自動的に設定されます。



6.3 プッシュボタンモデルを参照する項目モデル

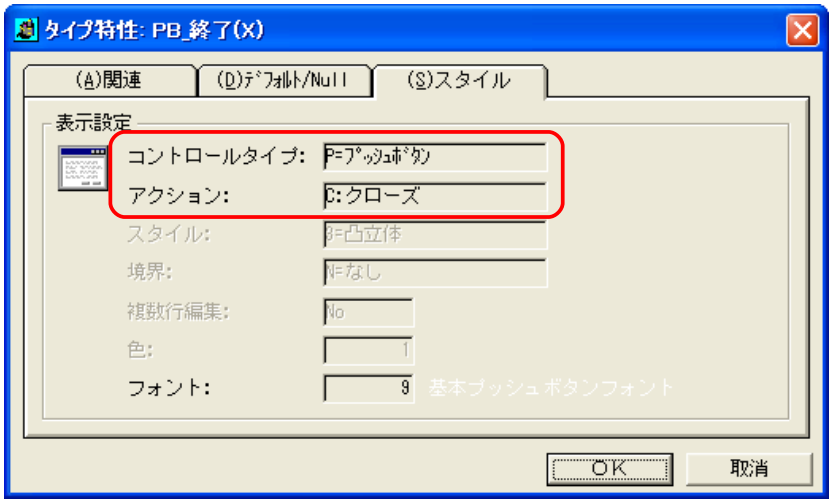
旧バージョンでもそうであったように、前節の例の終了ボタンのように、プッシュボタンのみをフォームに配置した場合には、マウスでプッシュボタンを押すことはできますが、キーボード操作でカーソルがプッシュボタンにパークしません。

キーボード操作でカーソルがボタン上にパークするようにしたい場合には、プッシュボタンに変数項目と組み合わせなければなりません。

この場合、タスクの「データビュー」画面で変数項目を定義し、フォーム上でプッシュボタンコントロールを定義して、関連付けを行っても良いのですが、変数項目の定義とプッシュボタンコントロールの定義をワンセットにしてモデルで定義しておくことができれば、もっと便利です。特に、「終了(X)」ボタンなどは、ほとんどすべてのオンライン画面につけるものなので、モデル化できれば大変有効です。

V8 でも、タイプリポジトリの「スタイル」タブで、オンラインフォームでの表示形式を指定することができましたが（下図）、V9 および V10 ではもっときめ細かに設定できるようになっています。

V8 のタイプリポジトリの「スタイル」タブ



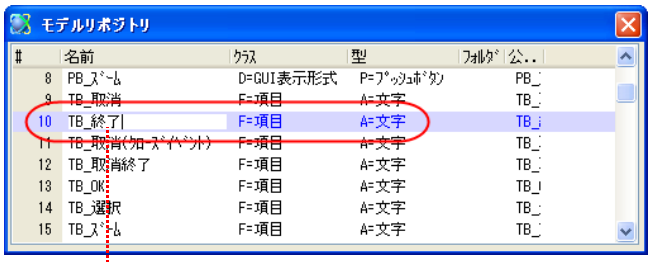
本節では、プッシュボタンを関連付けた項目モデルについて説明します。

6.3.1 項目モデルの定義

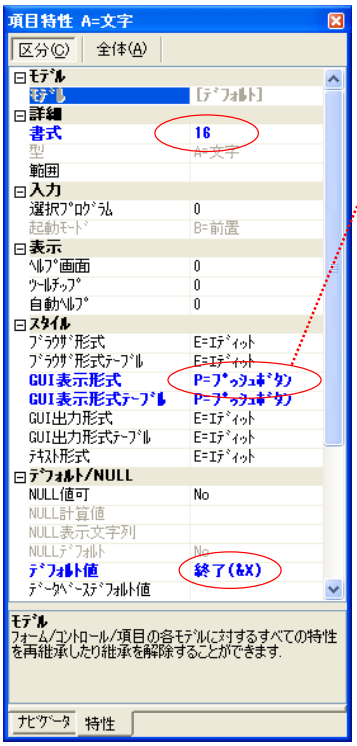
V10 のモデルリポジトリで「クラス」が「F=項目」であるモデルを定義した場合、その項目をフォーム上に配置した場合、どのような形式のコントロールで表示されるかは、項目特性の「スタイル」カテゴリの中にある各特性に設定します。

プッシュボタンモデル PB_終了 を参照する項目モデル TB_終了

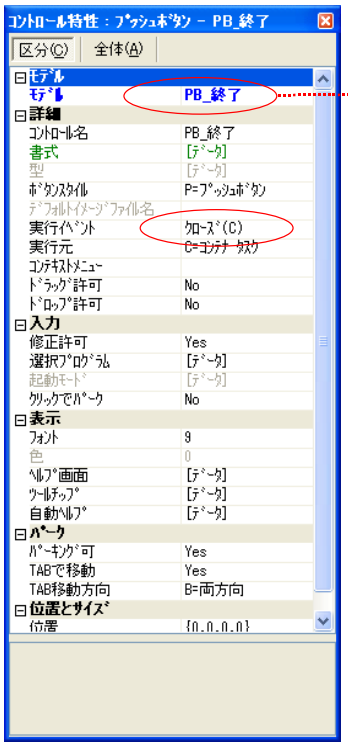
① モデルリポジトリ



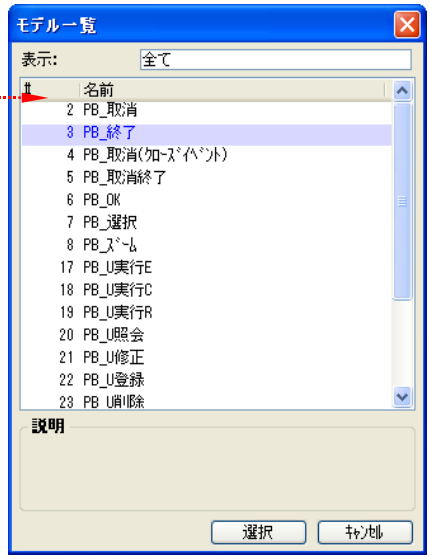
② 項目モデル **TB_終了** の特性



③ **GUI表示形式** の特性



④ モデル一覧



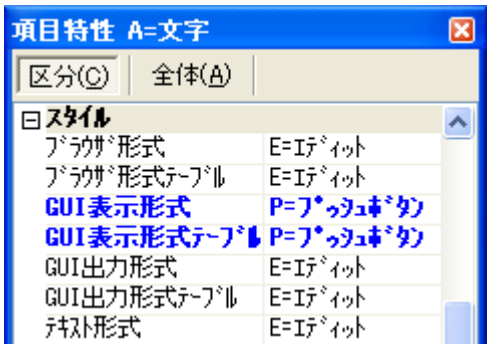
例として、終了ボタンに関連づけられた項目モデル **TB_終了** の定義を見てみます。

① ED_COMMON プロジェクトのモデルリポジトリを開いてください。10 番目に **TB_終了** という名前のモデルがあります。「クラス」は「F=項目」で、「型」は「A=文字」です。

② 特性シートを開いて、特性を見てください。次のような特性が設定されています。

- データサイズは最大 16 バイトとしたいので、「書式」は「16」にしています。
- オンラインフォームに配置した場合、デフォルトではプッシュボタンとして配置したいので、「GUI 表示形式」および「GUI 表示形式テーブル」特性では、いずれも「P=プッシュボタン」に設定しています。
- ボタンの表示ラベルのデフォルトは「終了(X)」としたいので、「デフォルト値」に「終了(&X)」と設定し

ています。



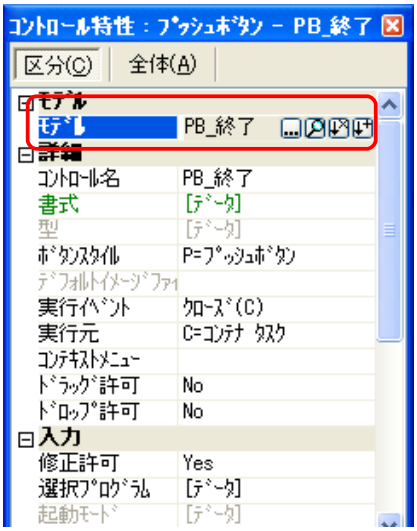
項目特性の「スタイル」カテゴリにある設定項目は、それぞれの形式のフォーム上に配置した場合に、デフォルトでどの形式のコントロールとして配置されるかを定義するものです。

- **ブラウザ形式**：ブラウザクライアントフォームに配置する場合のコントロール形式。
- **GUI 表示形式**：オンラインフォームに配置する場合のコントロール形式。
- **GUI 出力形式**：GUI 印刷 のフォームに配置する場合のコントロール形式。
- **テキスト形式**：テキスト形式のフォームに配置する場合のコントロール形式。

「～テーブル」という名前が付いているものは、テーブルコントロール中に配置される場合（ラインモードの場合）であり、付いていないものは、テーブル外に配置される場合（スクリーンモードの場合）の設定です。

このように、V8 ではオンラインフォームの場合のコントロールタイプだけを 1 種類設定できましたが、V9/V10 では各フォームタイプ毎、表示形式（スクリーン/ラインモード）ごとに設定できるようになりました。

③ 「GUI 表示形式」の欄からズームしてください。プッシュボタンについての特性が表示されます。この設定はそのまま、オンラインフォームに配置された場合のプッシュボタンの特性のデフォルト値となります。



④ この特性シートで、「モデル」特性がありますが、ここでモデルリポジトリ中に定義されているプッシュボタンモデルを参照することもできます。

モデルを参照すると、そのモデルの特性がすべて継承されます。

ここでは**モデル** 特性にプッシュボタンのモデル **PB_終了** が参照されています。このため、**PB_終了** に定義されている特性が継承されて、**実行イベント** が 内部イベントの **クローズ(C)** になっています。

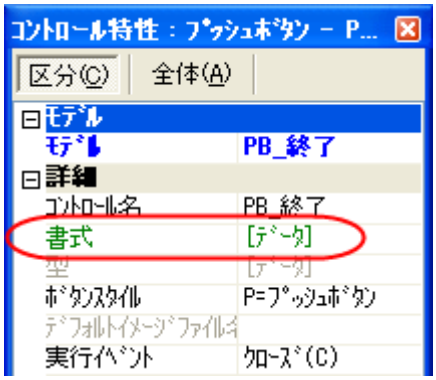
継承されている特性は、黒色の通常フォントで表示されますが、一箇所だけ、緑色のフォントとなっている特性「書式」があります。これは「データからの継承」を意味するものですが、これについては次に説明します。

6.3.2 データからの継承

GUI 表示形式の特性は、ほとんどがモデル **PB_終了** から継承しますが、一つだけ継承を解除している特性があります。それは **書式** 特性で、継承ではなく [データ] として緑色で表示されています。

これは「データからの継承」を意味するもので、「書式」特性については「モデル」に設定されている「PB_終了」から継承するのではなく、項目特性から継承することを意味します。

ここの例では、項目モデル「TB_終了」の書式は 16 なので、フォームに配置した場合のプッシュボタンの書式は 16 となります。



継承を [データ] にしておくと、実行時には次のようになります。

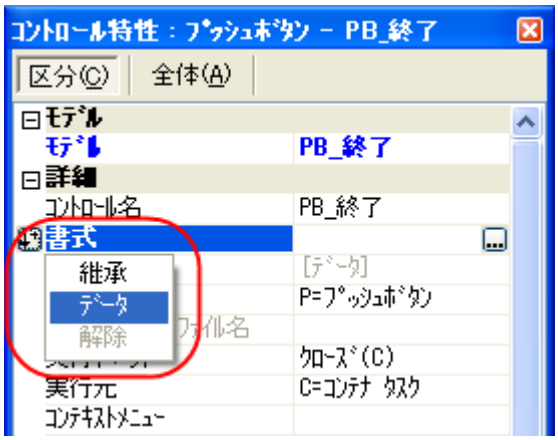
- プッシュボタンの書式は、固定的な「終了(X)」ではなく、項目の書式「16」となります。
- 開発時、フォームエディッタに配置する場合、書式 16 を元にしてプッシュボタンの幅を計算します。
- 実行時、プッシュボタンに表示される文字列は、変数項目に格納されている文字列データとなります。

これにより、実行時ダイナミックにプッシュボタンの表示文字を変更することができるようになります。

ただし、固定的に「終了(X)」でもよい場合にでも変数にデータを設定する手間を省くため、項目モデルの **デフォルト値** 特性に「終了(X)」を設定しておきます。

書式 特性は、デフォルトでは継承になっています。これを [データ] に変えたい場合には、次のようにします。(右図参照)

1. プロパティシートの **書式** 特性にカーソルをおきます。
2. 左にある  ボタンをクリックします。
3. メニューが出てくるので、**データ** を選択します。



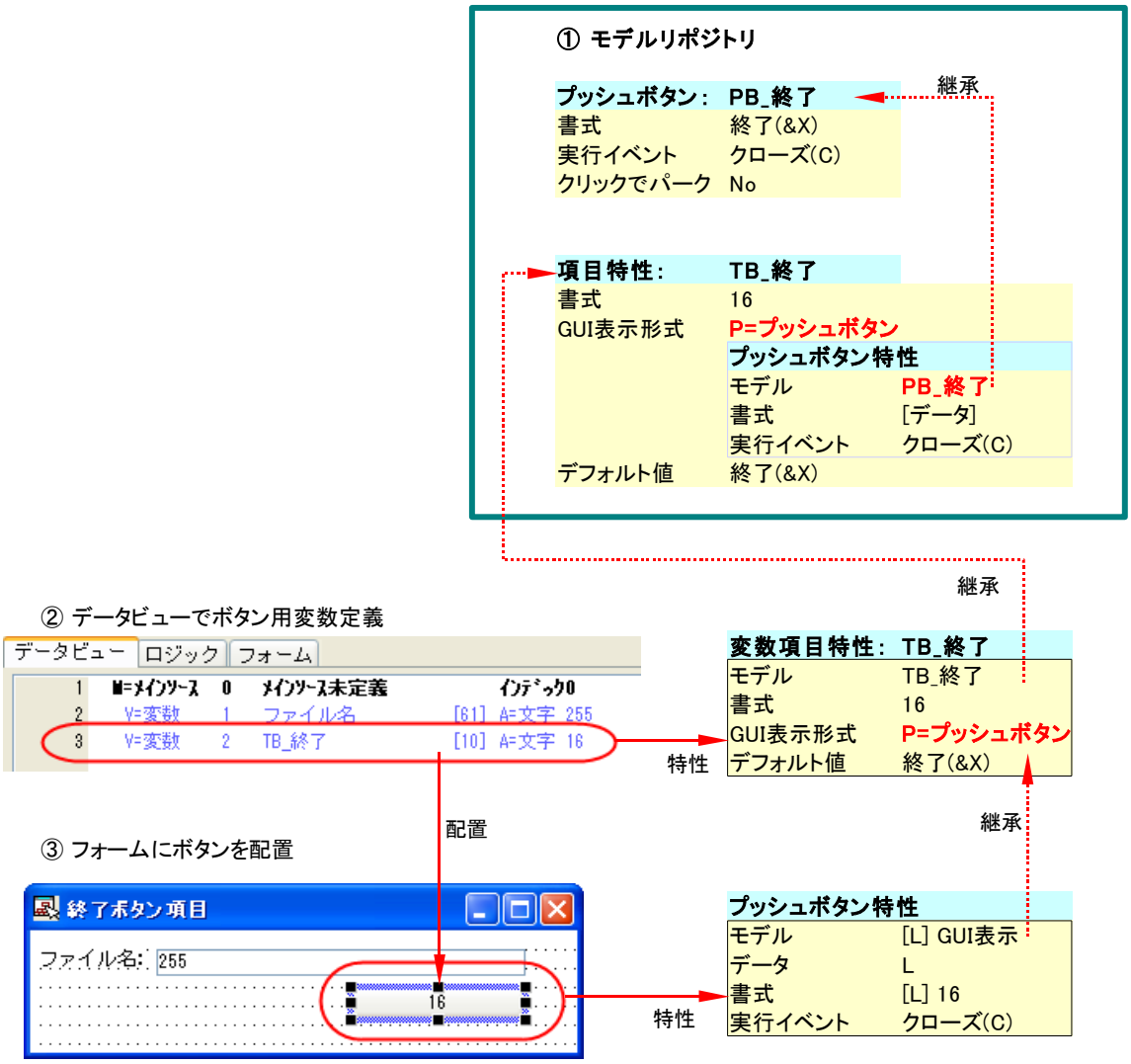
6.3.3 プログラムでの利用

このように定義されている項目モデルをプログラムで利用するときは、次のようになります。

① モデルリポジトリには、プッシュボタンモデル **PB_終了** と、これを GUI 表示形式で参照している項目モデル **TB_終了** があります。

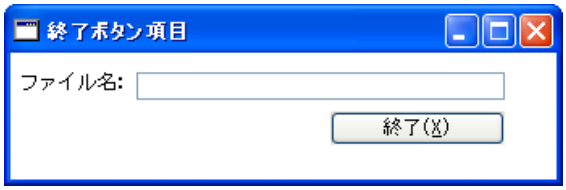
② プログラムのデータビューで、変数項目を作成し、モデル **TB_終了** を参照します。この変数項目の特性はすべてモデルリポジトリから継承します。

③ この変数項目をフォームに配置します。このとき、項目の **GUI 表示形式** 特性が **P=プッシュボタン** になっているので、フォーム上ではプッシュボタンとして配置されます。そして、このプッシュボタンの特性については、書式はデータから継承して「16」になり、実行イベントはモデルリポジトリの **PB_終了** から継承して「クローズ(C)」になります。



6.3.4 実行時の動作

このようにしてフォームにプッシュボタンを配置すると、実行時には次のような動作になります。

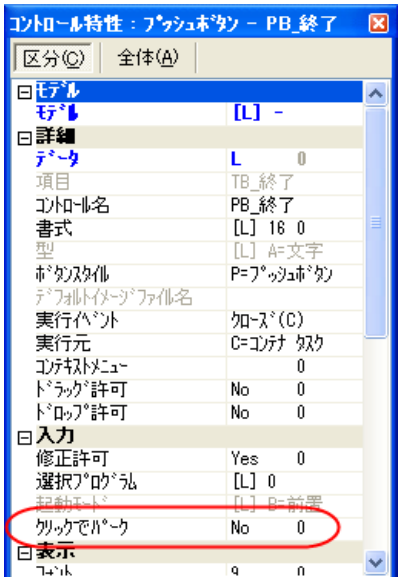


1. カーソルがプッシュボタンにパークします。
2. マウスでプッシュボタンをクリックすると、クローズイベントが発生し、タスクが終了します。

カーソルがパークするところが、ボタンのみを配置した場合と動作が異なるところです。

6.3.5 クリックでパーク 特性

V10 のプッシュボタンには、**クリックでパーク** という特性があります。この特性のデフォルト値は Yes ですが、ED_COMMON コンポーネントに登録されているプッシュボタンモデルではすべて No に設定してあります。



この特性は、コントロールレベルのイベントのサイクル（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）の実行を制御するもので、これを No にしておくことにより、プッシュボタンをクリックした時のフォーカスの移動を抑えて、コントロール検証の処理をスキップすることができるようになります。

詳しいことはここでは省略しますが、リファレンスヘルプの 表示フォーム > GUI コントロール > GUI 表示コントロール特性 > プッシュボタンコントロール特性 を参照してください。

6.4 ユーザイベントのプッシュボタンモデル

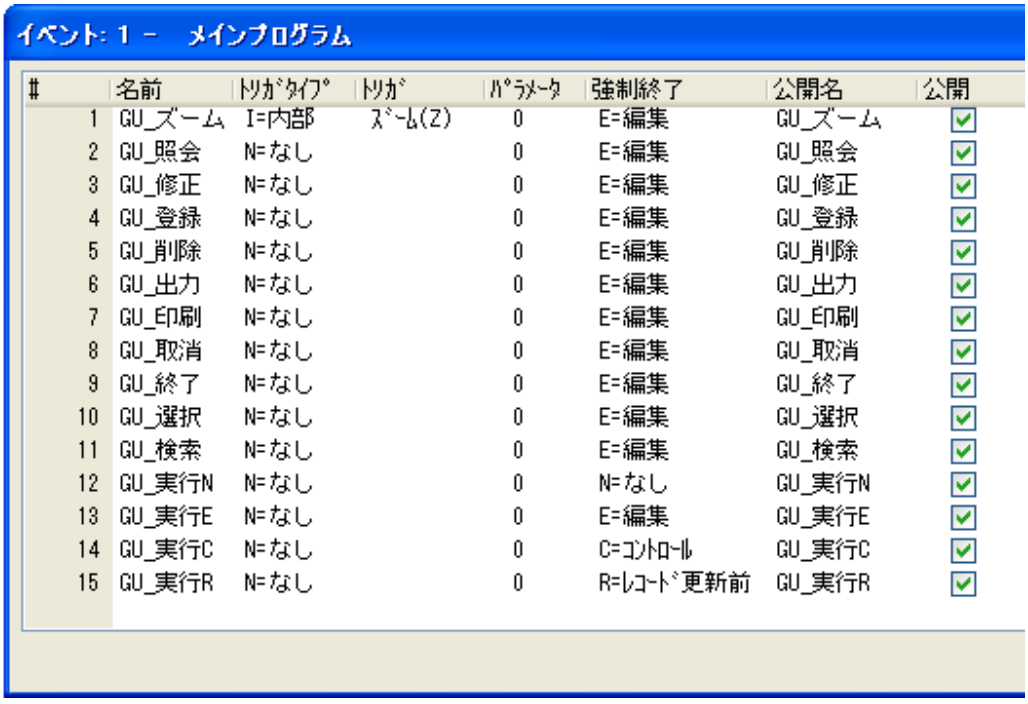
タスクを終了するための「クローズ」イベント、入力内容を取り消すための「キャンセル」イベント、選択プログラムで特定のレコードを選択するための「選択」イベントなどは、内部イベントとして予め Magic エンジンに実装されているので、プッシュボタンの「実行イベント」特性にはそれぞれの内部イベントを設定しました。

一方、「印刷」「出力」「検索」などといったボタンについては、Magic エンジンでそれに対応する内部イベントはないので、ユーザイベントを発行させて、対応するイベントハンドラをプログラムに定義する、という形になります。従って、プッシュボタンモデルの実行イベントには、適当なユーザ定義イベントを設定することになります。

6.4.1 ユーザ定義イベント

印刷、出力、検索などをトリガーするユーザ定義イベントは、多くのアプリケーションでよく使うものなので、共通コンポーネント ED_COMMON で定義しておきます。グローバルなイベントにする必要があるため、メインプログラムで定義し、公開名をつけて、コンポーネントの外部に公開しておくようにします。

ED_COMMON では、下図のようなユーザ定義イベントが定義されています。

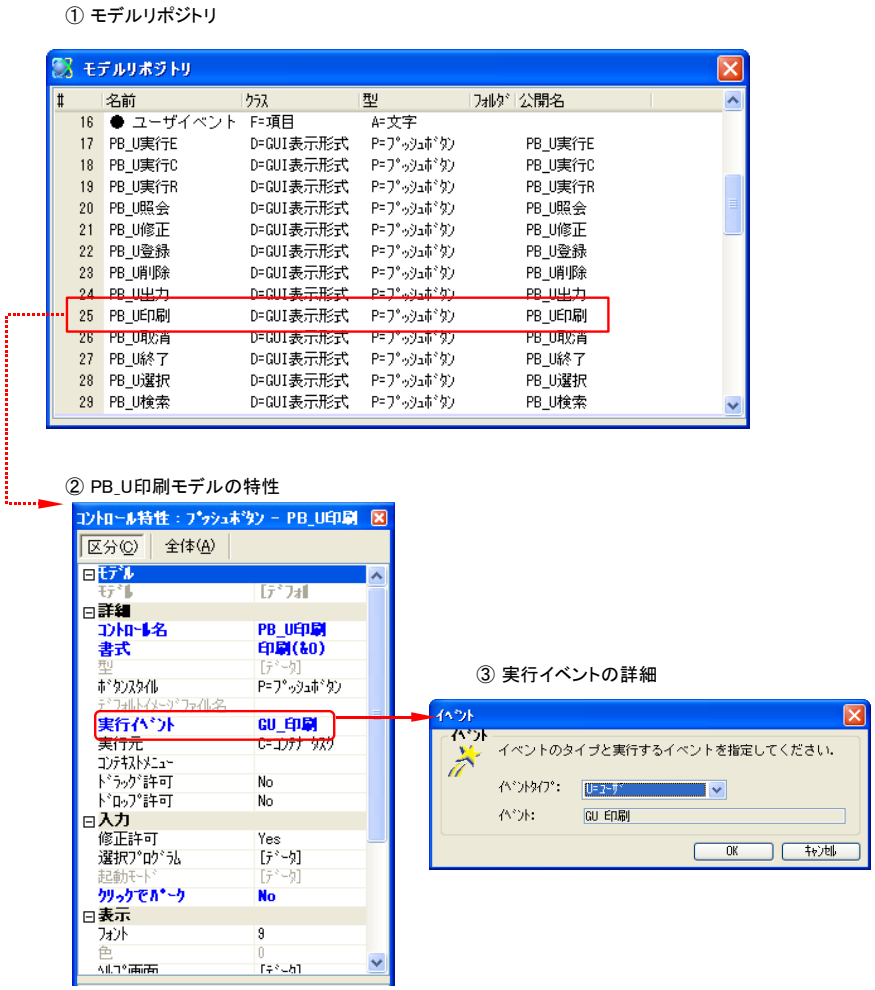


#	名前	ﾄﾘｶﾞﾀｲﾌﾟ	ﾄﾘｶﾞ	ﾊﾟﾗﾒｰﾀ	強制終了	公開名	公開
1	GU_ズーム	I=内部	ｽﾞｰﾑ(Z)	0	E=編集	GU_ズーム	☑
2	GU_照会	N=なし		0	E=編集	GU_照会	☑
3	GU_修正	N=なし		0	E=編集	GU_修正	☑
4	GU_登録	N=なし		0	E=編集	GU_登録	☑
5	GU_削除	N=なし		0	E=編集	GU_削除	☑
6	GU_出力	N=なし		0	E=編集	GU_出力	☑
7	GU_印刷	N=なし		0	E=編集	GU_印刷	☑
8	GU_取消	N=なし		0	E=編集	GU_取消	☑
9	GU_終了	N=なし		0	E=編集	GU_終了	☑
10	GU_選択	N=なし		0	E=編集	GU_選択	☑
11	GU_検索	N=なし		0	E=編集	GU_検索	☑
12	GU_実行N	N=なし		0	N=なし	GU_実行N	☑
13	GU_実行E	N=なし		0	E=編集	GU_実行E	☑
14	GU_実行C	N=なし		0	C=ｺﾝﾄﾛｰﾙ	GU_実行C	☑
15	GU_実行R	N=なし		0	R=ﾚｺｰﾄﾞ更新前	GU_実行R	☑

6.4.2 ユーザ定義イベントを参照するプッシュボタンモデル

モデルリポジトリでは、ユーザ定義イベントを使うプッシュボタンのモデルが、モデル 17 番以降に定義されています。

例として、ユーザ定義イベント **GU 印刷** を発行するプッシュボタンイベント **PB_U 印刷** の特性を見てみると、次の図のようになっています。



① モデルリポジトリの PB_U 印刷 行から、プロパティシートを開きます。

② プロパティシートでは、コントロール名、書式、実行イベント などの特性が設定されています。

③ 実行イベントからズームすると、ユーザ定義イベント **GU_印刷** が設定されています。

これを見てわかるように、6.2 内部イベントのプッシュボタンモデル で説明した、プッシュボタンモデルとほとんど同じで、ただ、実行イベントとしてユーザ定義イベントが設定されている、ということだけが異なります。

6.4.3 ユーザ定義イベントのプッシュボタンモデルを参照する項目モデル

前述のように定義されている一連のプッシュボタンモデルを参照して、項目モデルがモデル 30 番以降に定義されています。これらの項目モデルについては設定内容は内部イベントの場合と同様なので、説明は省略します。

6.4.4 どんなユーザイベントを用意すべきか？

サンプルアプリケーションの ED_COMMON コンポーネントでは、ユーザ定義イベントが 15 個定義されています。そして、それに合わせて、プッシュボタンモデルも定義されています。ユーザ定義イベントは、文字通りユーザが自由に定義することのできるイベントなので、定義できる数に制限はありません。それでは、どんなユーザ定義イベントを何個くらい用意しておけば良いでしょうか？

この答えは、もちろんアプリケーションに依存します。しかし、モデル利用の目的である、共有による開発・保守効率の向上を考慮すれば、基本的な指針は出すことができます。

具体的な例を使って考えてみると、「印刷」「編集」などといった、多くの画面で出てくるような機能のボタンに対しては、モデル化する利点が高いといえます。同じ設定を何回も繰り返して行う必要がなくなるからです。

一方で、「ログイン」というようなボタンは、アプリケーション中で「ログイン」画面にしか出てこないかもしれません。このように一箇所だけでしか現れないプッシュボタンに対しても、ユーザ定義イベント（例えば、**GU_ログイン** とか)を定義し、それに対応するプッシュボタンモデルを定義し、それを参照する項目モデルを定義し、タスクのデータビューでその項目モデルを参照する、というのは、いかにも迂遠で、返って開発効率が悪くなります。

このような出現頻度の非常に低いプッシュボタンに対しては、それ専用のユーザ定義イベント/プッシュボタンモデル/項目モデルを作らずに、汎用のユーザ定義イベント/プッシュボタンモデル/項目モデルを定義しておいて、ボタンラベルだけをプログラムで変える、というようにするのが簡便です。

このような目的の汎用定義が、ユーザ定義イベント **GU_実行 E**、プッシュボタンモデル **PB_U 実行 E**、項目モデル **TB_U 実行 E** です。次ページの図は、受注入力プログラムで「確定」ボタンを定義している例です。

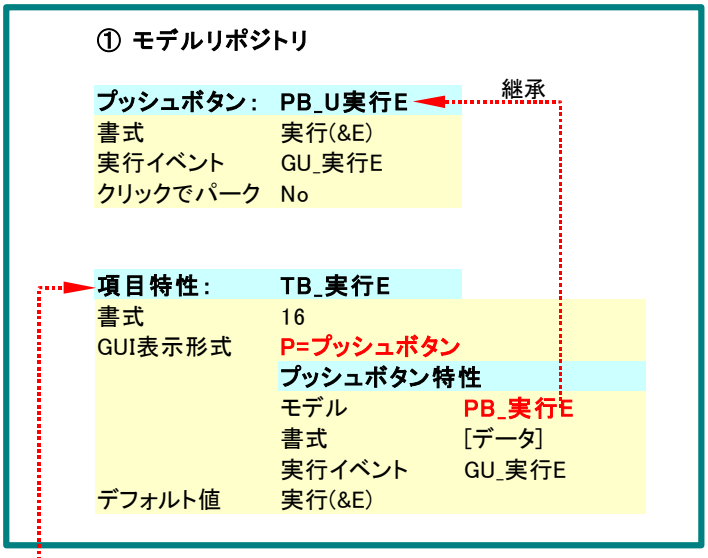
① モデルリポジトリでは、ユーザ定義イベント **GU_実行 E** を使った プッシュボタンモデル **PB_U 実行 E**、項目モデル **TB_U 実行 E** が登録されています。

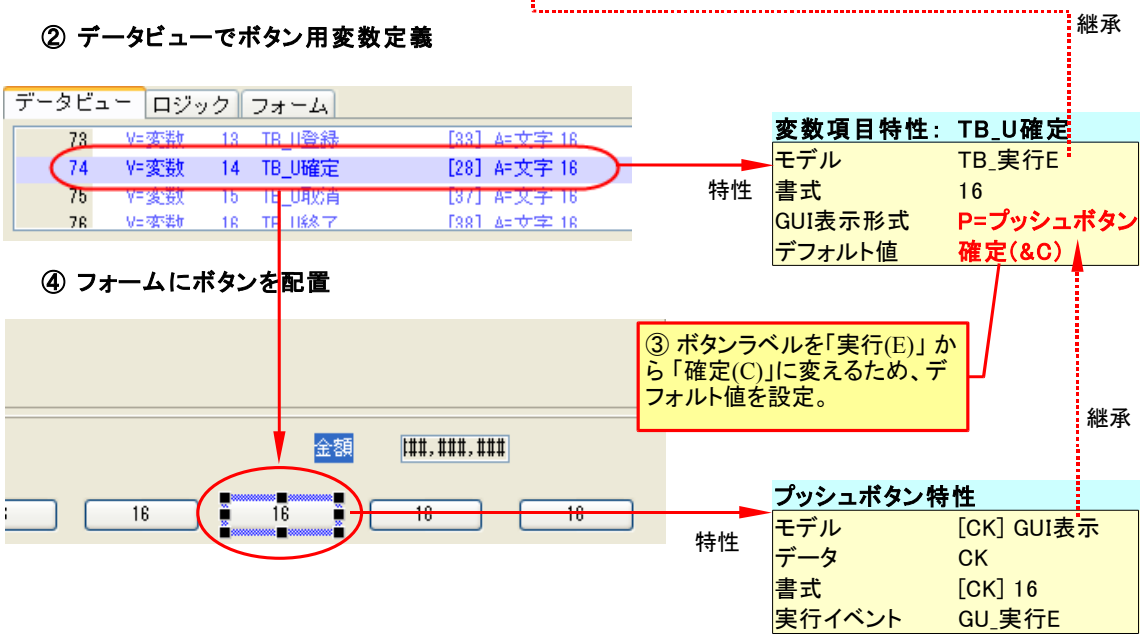
② プログラムでは、データビューで変数項目 **TB_U 確定** を定義します。この変数は、モデル **TB_U 実行** を参照します。

③ 変数特性はほとんどがモデルより継承しますが、ボタンラベルを変更するために、**デフォルト値** を「確定(&C)」にしています。

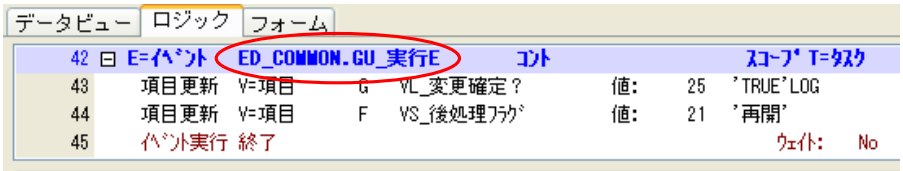
④ この変数をフォームに配置します。変数の **GUI 表示形式** 特性が **プッシュボタン** になっているので、フォーム上ではプッシュボタンとして配置されます。
このプッシュボタンの特性は、モデル **PB_U 実行** から継承しますが、書式だけはデータから継承して、「16」となります。

⑤ 実行時にこのプッシュボタンをクリックすると、**実行イベント** として設定されている **GU_実行 E** イベントが発生するので、プログラムではこのイベントに対するハンドラを定義しておきます。
この例では、内部フラグをいくつか設定して、タスクを終了するために **終了** イベント(内部イベント)を指定して **イベント実行** コマンドを実行しています。





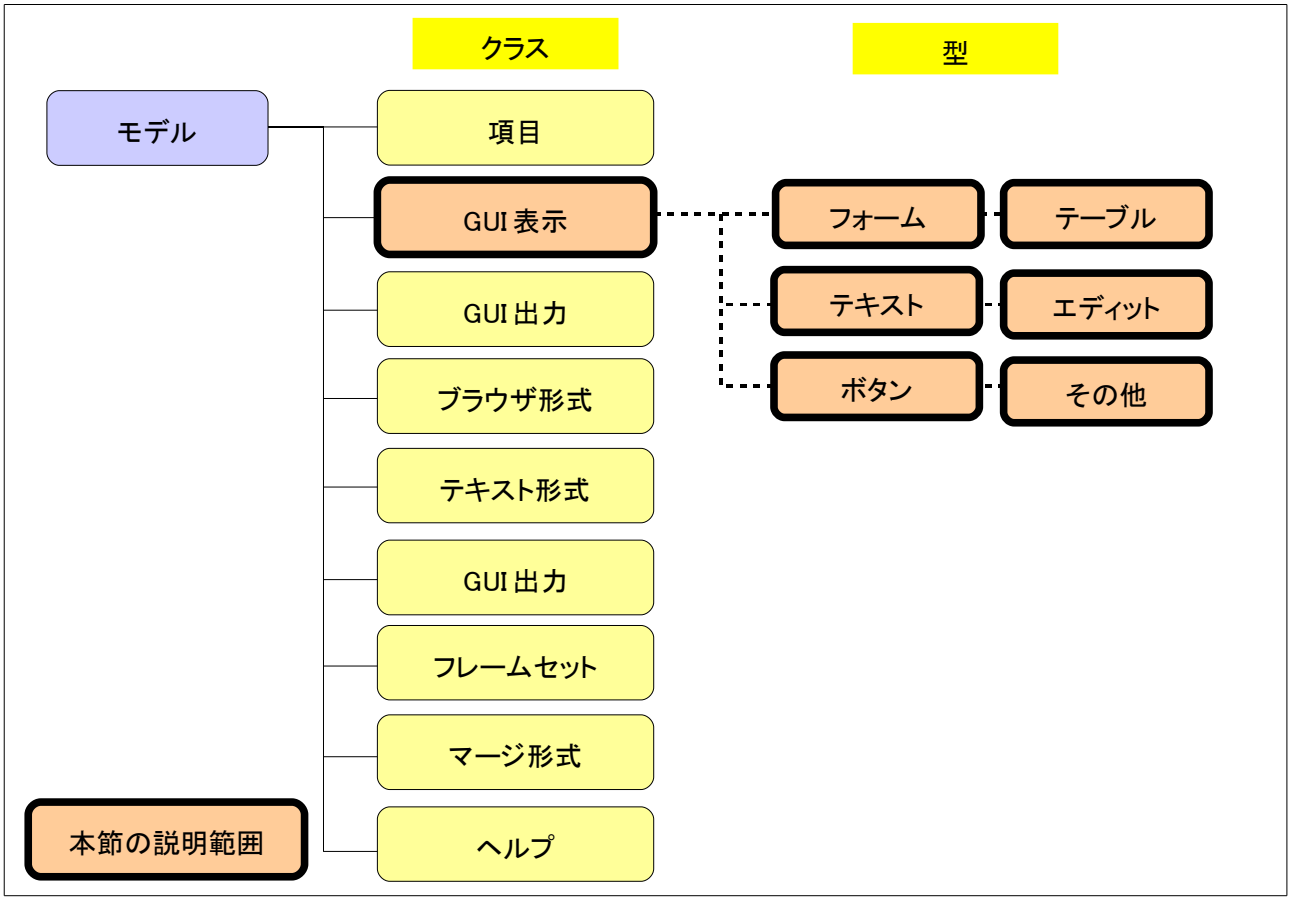
⑤ タスクのロジックで、GU_実行E に対するハンドラを定義



この例での「確定」ボタンのように、汎用実行イベント（**GU_実行 E** イベント）を使うボタンを、一つのフォーム上に複数配置することも可能です。この場合には、どのボタンでクリックされても **GU_実行 E** イベントが発生するので、どのボタンでクリックされたかを区別するために、イベントハンドラのヘッダ行で **コントロール名** パラメータを設定します。

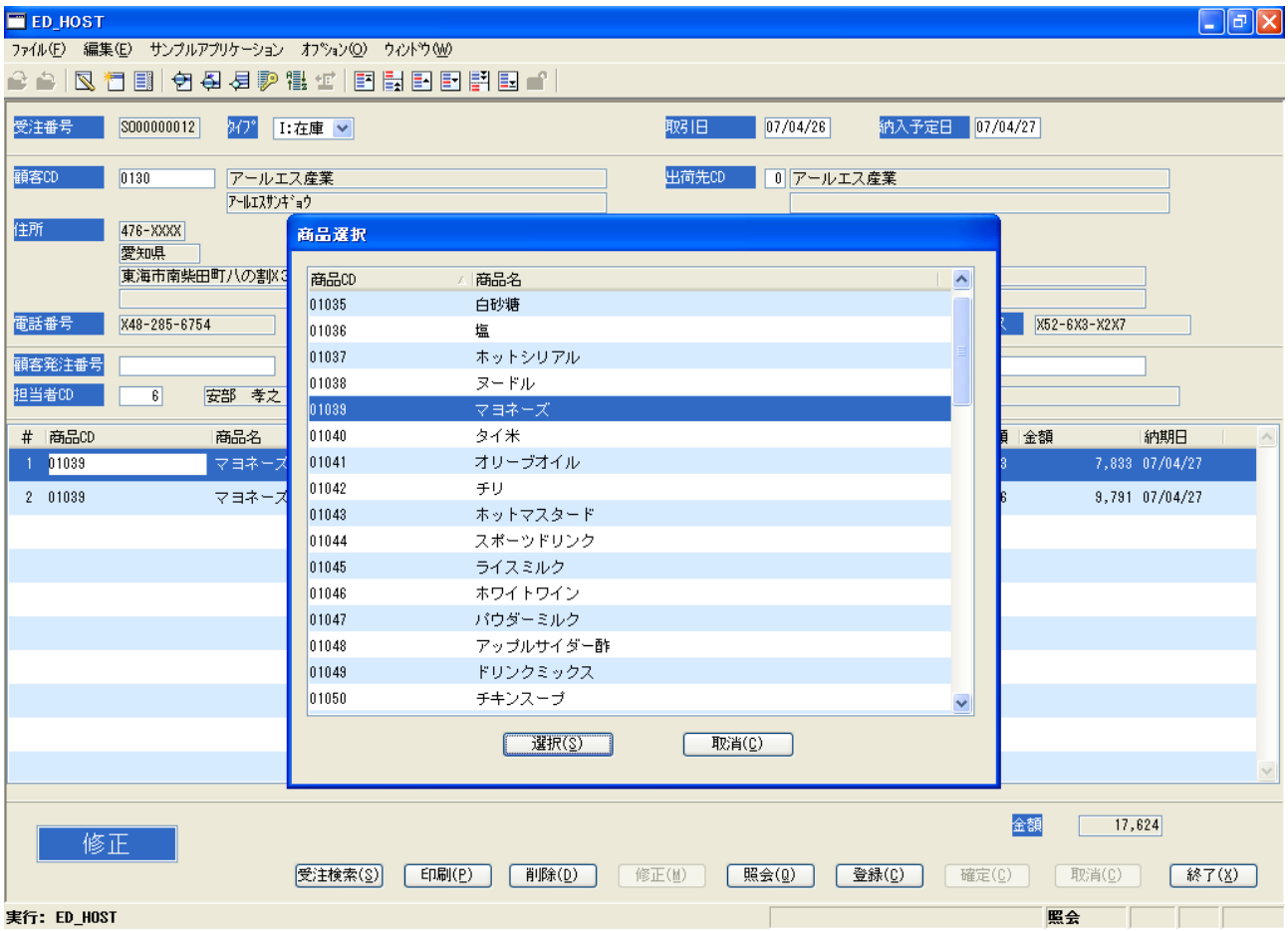
6.5 GUI 表示形式のモデル

V9 から、モデルリポジトリにフォームや GUI コントロールなどの GUI オブジェクトもモデルとして登録できるようになりましたが、まだ実際にはそれほど活用されていないようです。ここでは、オンラインフォームでの GUI オブジェクトモデルである「GUI 表示形式」のモデルについて説明します。



それでは、GUI オブジェクトのモデル化の可能性はいかなるものがあるでしょうか？サンプルアプリで見ていくことにしましょう。

下図は、受注入力画面のイメージ(修正モード時)で、商品 CD 欄からズームして商品一覧を表示しているところです。



この画面と、その他の画面とを総合して検討してみると、次のような GUI オブジェクトが共通した属性を持つことに気が付きます。

種別	内容	主要設定属性
フォーム	受注入力や受注取引画面参照プログラムでは、Magic のウィンドウー杯にフォームが広がって、灰色の背景色で表示されます。また、フォームエディッタでのグリッド間隔は X が 0.25、Y が 0.125 です。	ウィンドウタイプ: MDI 調整 タイトルバー: No 色: 4 グリッド間隔(X): 0.25 グリッド間隔(Y): 0.125
	商品選択、顧客選択、その他の選択画面は、画面中央に表示され、タイトルバーにはシステムメニューや最大化/最小化/閉じるアイコンがありません。色やグリッド間隔も上記と同様です。	システムメニュー: No 開始時の位置: MDI の中央 色: 4 グリッド間隔(X): 0.25 グリッド間隔(Y): 0.125
テーブル	テーブルコントロールは交互色表示で表示されており、フォーカスのある行は青地に白色で表示されます。	テーブル色の指定: テーブルに依存 交互表示色: 101 ハイライト行の色: 3
テキスト	表示データのテキストラベルは、青地に白色の文字で表示されています。	色: 3
エディット	表示専用の項目は、灰色の背景色をしています。また、パークしません。	修正許可: No 色: 4 パーキング可: No

ボタン	終了、取消、選択、修正、印刷、検索などのボタンは、いろいろなプログラムに使われています。	書式：(それぞれのボタン表示) 実行イベント：(それぞれの機能に応じたイベント)

このような GUI オブジェクトをモデル化することにより、一貫した GUI インターフェースを作成するのが容易になり、個々に設定するときの設定し忘れなどがなくなります。また、変更する場合もモデルのみの変更で対応することができます。

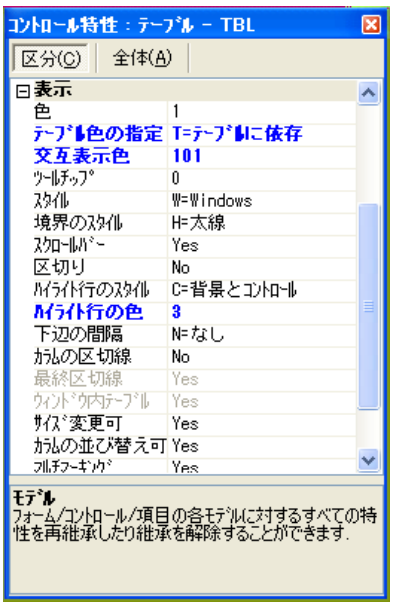
サンプルでは、これらの GUI オブジェクトのモデルは、共通コンポーネント ED_COMMON のモデルリポジトリ 43 番以降に登録しました。

これらのモデルでは、「**クラス**」欄はすべて「**D=GUI 表示形式**」となっており、「**型**」欄はフォームまたはコントロールの種別を表しています。

例えば、テーブルコントロールのモデル「TBL_交互色」の特性は右図のようになっています。これは交互色テーブルを実現するためのもので、次のように設定されています。

- 交互色テーブルを実現するため、「テーブル色の指定」は「T=テーブルに依存」としています。
- 交互色の色は、薄い空色を背景とした黒色文字としたいので、「交互表示色」として 101 番 を設定しています。
- ハイライト行は、青色を背景とした白色文字としたいので、「ハイライト行の色」は 3 番としています。

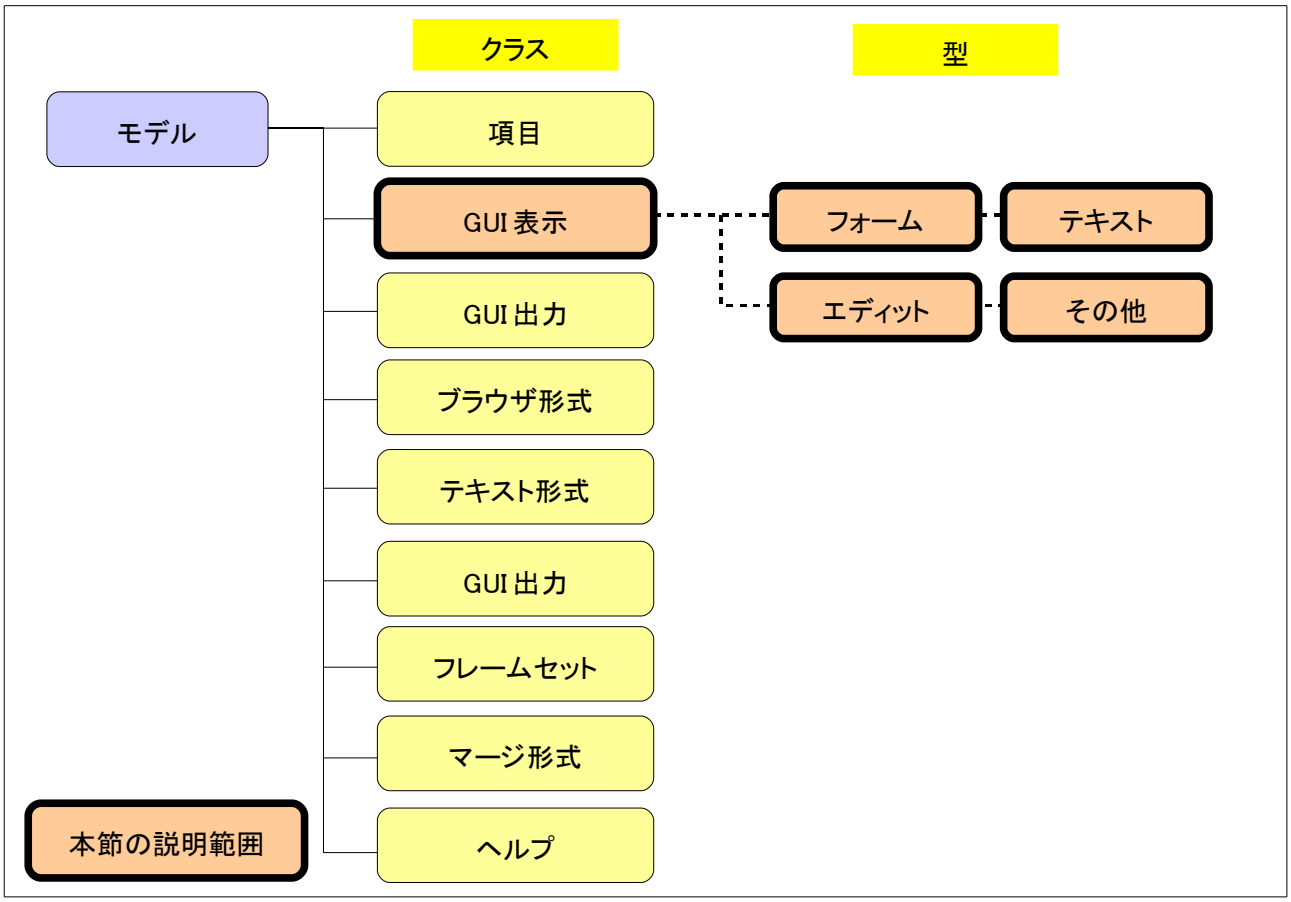
交互色テーブルはアプリケーション中で多用していますが、モデルに定義しておけば、フォームエディッタ上でテーブルコントロールの「モデル」特性でこのモデルを参照することにより、簡単に交互色テーブルとすることができます。



ここでは、各モデルの詳細についての説明は省略します。GUI インターフェースはアプリケーション（エンドユーザや開発者の標準や嗜好など）により千差万別であり、サンプルに登録したものはごくごく基本的なものだけなので、読者の皆さんが自分の財産として開発されるとよいと思います。

6.6 GUI 出力形式のモデル

GUI 出力形式は、GUI 印刷用のフォームで使うコントロールに関してモデル化を行うものです。



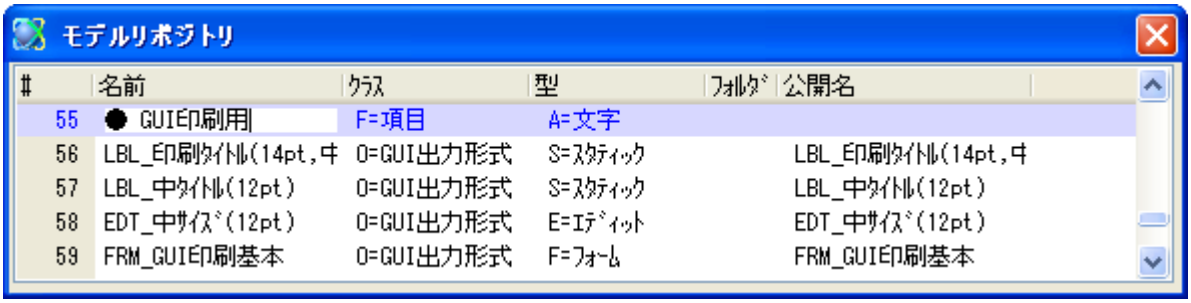
オンラインフォーム上の GUI オブジェクトを定義する GUI 表示形式のモデルに比べ、印刷用の GUI 出力形式のモデルはバリエーションが少ないと思いますが、それでもいくつかの候補が挙げられるでしょう。

下図は、サンプルの受注報告書印刷フォームをフォームエディッタ上で開いたものです。

ここでのモデル化候補としては、次のようなものがありましょう。

種別	内容	主要設定属性
フォーム	グリッド間隔がデフォルトより細かい。	グリッド間隔(X): 0.100 グリッド間隔(Y): 0.100
テキスト	レポートのタイトル「受注報告書」が大きいフォントで印刷される。	フォント: 5
	「受注番号」が中くらいのフォントで印刷される。	フォント: 7
エディット	受注番号が中くらいのフォントで印刷される。	フォント: 7

これらをモデル化したものが、共通コンポーネント ED_COMMON のモデルリポジトリ 55 番以降に登録されています。

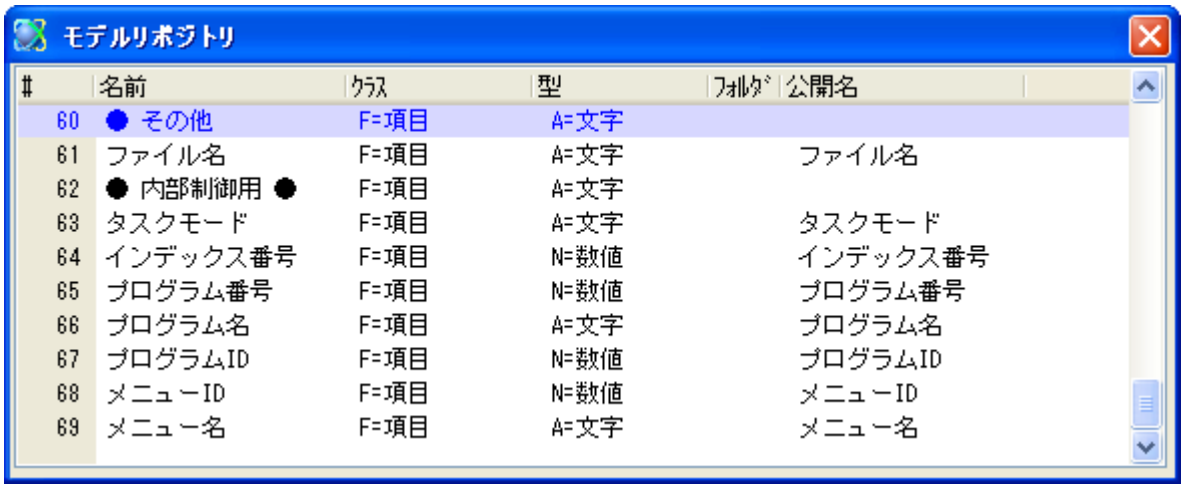


GUI 出力形式も、GUI 表示形式と同様、アプリケーションによってもっとバラエティの多いものとなろうと思います。帳票設計にも、モデルを活用するようにしましょう。

6.7 その他のモデル

その他に共通コンポーネント ED_COMMON に登録されているモデルとしては、60 番以降に次のようなものがあります。

種別	内容	主要設定属性
項目	ファイル名	書式：255 選択プログラム：(ファイルダイアログを表示)
	プログラム管理用のデータのモデル。タスクモード、プログラム番号、プログラム名など。	書式



これらはアプリケーションに依存するデータではなく、共通的に使えるデータですので、モデル化しておきました。

6.8 まとめ

本章では、サンプルアプリケーションに定義されているモデルについて、特に共通コンポーネントのモデルに焦点を当てて説明してきました。

プッシュボタンのモデルは、イベント指向プログラミングにおいて非常に役にたつモデルです。本章ではイベント指向プログラミングについて詳しく説明しませんでしたが、今後公開予定の「イベント編」でもっと掘り下げた内容を説明する予定です。

GUI 表示形式および GUI 出力形式のモデルは、V8 までにはなかったものですので、Magic のバージョンが V9、V9Plus、V10 に移行してもなかなか使われていないようです。しかしこのタイプのモデルもうまく活用すれば、項目モデル(V8 までのタイプ辞書)と同様、生産性と保守性を大きく向上させる可能性を持っています。ぜひ、GUI 表示/出力形式のモデル化も進めていってください。

Magic eDeveloper V10

Magic eDeveloper V10

コーディングサンプル Version 2 (モデル編)

第 1 版　2007 年 10 月 10 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/