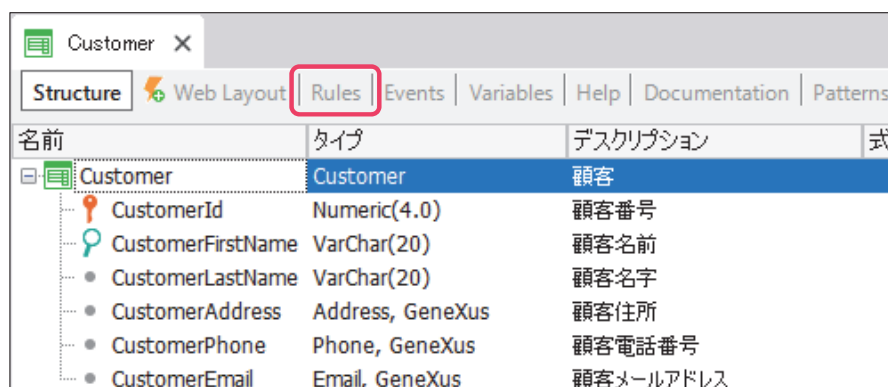


ルールの定義

GeneXus™

制御の追加



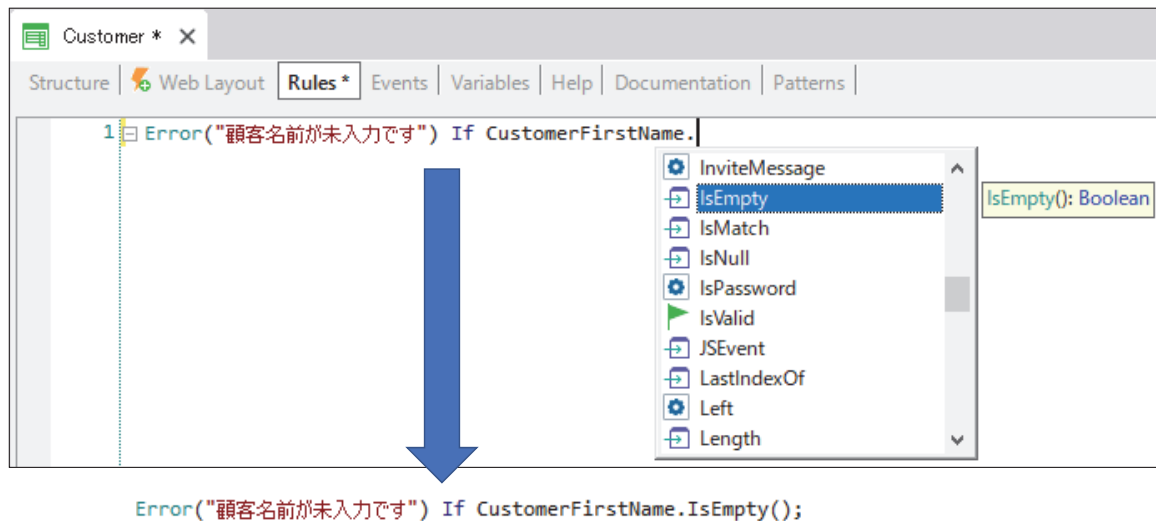
ここまでの内容で、GeneXus はアプリケーション生成時にいくつかの制御を自動的に追加していることを確認しました。

しかし、アプリケーションにおいては、ユーザーの要望に基づく制御を追加する必要があります。

このような場合、トランザクションオブジェクトでは、利用者が従うべきルールや、検証の制御を [Rules] エLEMENTで定義します。

エラー制御の追加

- Error ルール



ある条件を満たした場合、登録処理にエラーを発生させ、登録をさせないという実装が可能です。

このような制御を追加するためには、Error というルールを利用します。

「Error」と入力し、括弧を開きます。ここにダブルクォーテーションでくくり、エラーとなった際に表示するメッセージを入力します。その後、括弧を閉じます。このままでは、無条件にエラーが発生してしまうため、条件を指定します。

[Rules] エlementにおける条件分岐は、記述したルールの末尾に「If」と続け、Boolean 型の結果となる条件式を記述します。
例えば、「ある項目の値が空の場合」という条件を指定する場合、項目属性名を記述し、続けてピリオド、表示される選択肢から「IsEmpty」メソッドを選択します。
ピリオドを入力した際に表示される候補は、対象の項目属性に対するプロパティやメソッドが含まれています。
また、「IsEmpty」メソッドを利用することで、入力が空の場合、True、何かしらの値が入力されている場合、False が取得できます。

[Rules] Elementにおける制御は、1 つの制御に関する定義の終了を明記する必要があります。
明記するためには「;」（セミコロン）を入力します。

エラー制御の挙動



顧客

|< < > >> 選択

顧客番号

顧客名前 ① 顧客名前が未入力です

顧客名字

顧客住所

顧客電話番号

顧客メールアドレス

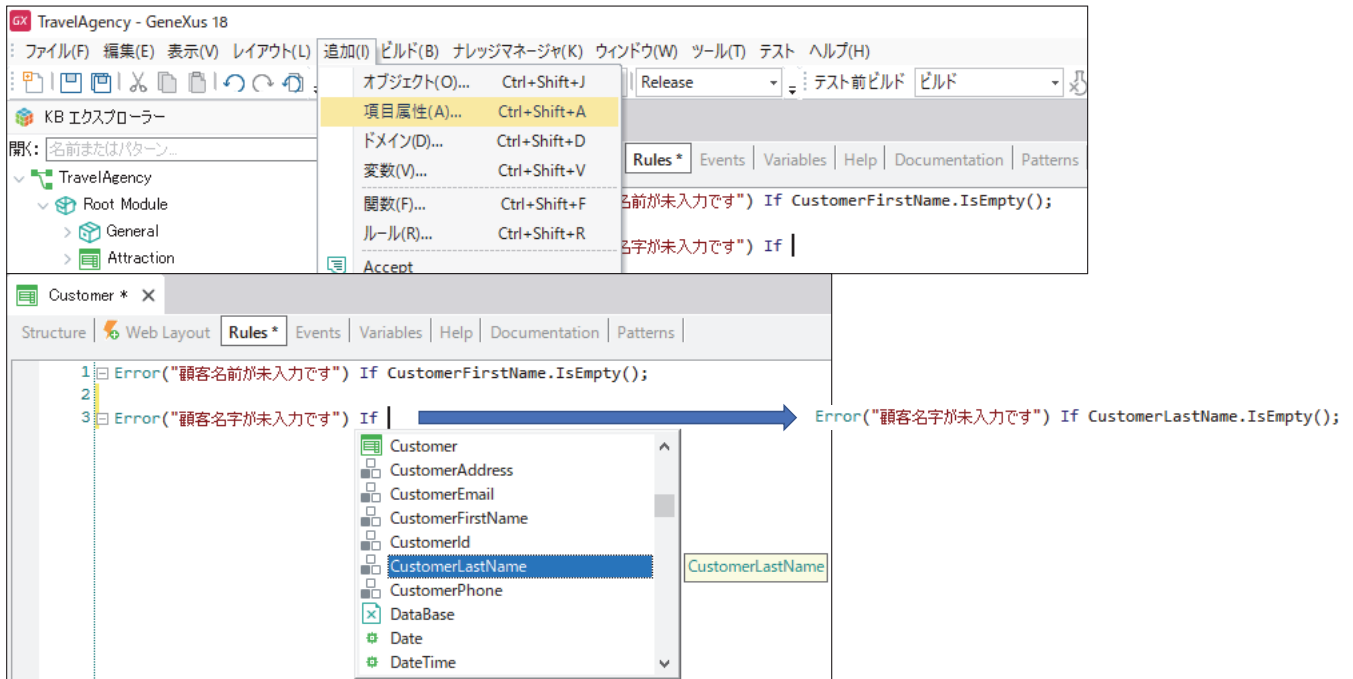
削除 終了 実行

アプリケーションを実行し、追加したエラー制御の挙動を確認します。

Customer トランザクションにより生成された顧客登録画面を実行します。
顧客名前の項目に何も入れずにフォーカスを先へ進めると、ルールに定義したメッセージが、条件に利用した項目に対し表示されます。
Error ルールの条件を満たしている限り、このメッセージは表示され続けます。

メッセージを表示したまま次の項目の入力を継続することができますが、「実行」ボタンをクリックしてもデータは登録できません。
つまり、顧客の名前を入力し、エラー制御の条件を満たさない状態にしなければデータは登録できない状態となりました。

「項目属性名」の入力



[Rules] エlementにおける制御の定義では、項目属性の名前を入力しなければならないケースが多数あります。

この時、項目属性名を思い出しながら入力することは、誤った名前の入力を発生させる可能性があり、オブジェクト保存時のエラーにつながります。

これを予防するためには、[Structure] エlementで項目属性を定義する際にナレッジベース内のすべての項目属性が候補として表示される挙動と同じ挙動を利用できることが望ましいです。

この機能は、GeneXus によってあらかじめ準備され、利用可能となっています。

手段は 2 つあり、状況に合わせて利用することができます。

1 つ目の方法は、「項目属性を挿入」ダイアログの利用です。

メニューバーから [追加] → [項目属性] を選択することで表示されるダイアログで、ナレッジベース内で定義された項目属性すべてが表示され、フィルタリングを行い、対象の項目属性を絞り込むことができます。

そして、目的の項目属性を選択し、OK をクリックすることで、入力画面のフォーカスが合った場所へ選択した項目属性名が入力されます。

2 つ目の方法は、入力候補からの選択です。

[Rules] Elementで、項目属性名を入力したい位置にフォーカスがある状態で、

[Ctrl] + [スペース] キーを押すことで、入力候補を表示できます。

この一覧から対象の項目属性を選択します。

もし、表示内容が多数あり、対象の項目属性までスクロールが必要な場合、この状態で文字を追加で入力することで、一致する文字列から始まるものがある場所まで一覧の表示が移動します。

この状態で確定することで、項目属性名が入力されます。

複数のエラー制御

顧客

|< < > >| 選択

顧客番号	<input type="text" value="0"/>
顧客名前	<input type="text"/>
顧客名字	<input type="text"/> ① 顧客名字が未入力です
顧客住所	<input type="text"/>
顧客電話番号	<input type="text"/>
顧客メールアドレス	<input type="text"/>

削除 終了 実行

複数のエラー制御が定義されている場合、常に表示されるメッセージは 1 つだけです。そのため、最後に発生したメッセージが画面上に表示されつづけます。ただし、エラー制御が実装され、その条件を満たしたままの項目属性は、入力欄の枠線が赤色のまま表示されます。この赤枠の入力項目にマウスオーバーすると、メッセージが確認できます。

警告メッセージの追加

- Msg ルール



条件を満たしている場合、警告メッセージを表示させ、ユーザーに注意を促す実装が可能です。

このような制御を追加するためには、Msg（メッセージ）というルールを利用します。

構文は、前述の「Error」ルールとよく似ていて、「Msg」と入力し、括弧を開きます。ダブルクォーテーションでくくった警告メッセージを入力し、括弧を閉じます。そして、どのような条件でメッセージを表示するか条件を指定します。

警告メッセージの挙動



顧客

|< < > >| 選択

顧客番号

顧客名前

顧客名字

顧客住所

顧客電話番号 ⚠ 顧客電話番号が未入力です

顧客メールアドレス

削除 終了 実行

警告メッセージが発生すると、条件の対象となったフィールドはオレンジ色の枠線で表示されます。

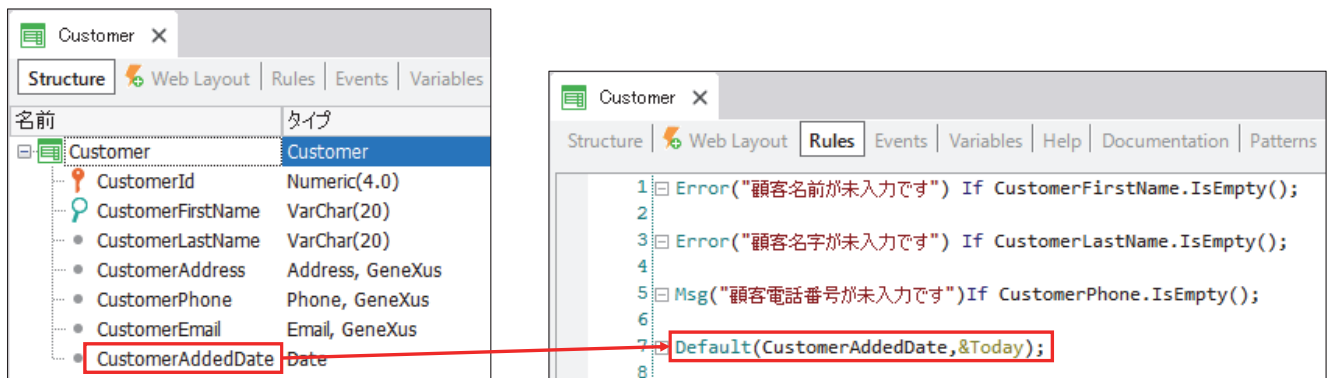
また、メッセージについても、先頭に表示されるアイコンはオレンジの三角形に白抜き「!」マークで表示されます。

警告メッセージが表示されていてもユーザーはデータを登録することができます。そのため、このルールを利用することで、登録の制御に影響を与えず、ユーザーへのメッセージを実装することができます。

また、このルールによるメッセージも Error ルールと同等に扱われ、すべてのメッセージを表示するルールのうち、画面上に常に表示されるルールは1つだけとなります。

既定値を設定

- Default ルール



ルールで定義可能な制御は、メッセージを表示するものだけではありません。
例えば、データを新規登録する際の既定値を設定したいという要望があった場合、
この要望を容易に実装するルールが用意されています。

このルールは、Default ルールと言います。
[Rules] エlementで、「Default」と入力し、括弧を開きます。
括弧内には、2 つの引数を必要とし、1 つ目の引数は既定値を設定した項目属性を
記述し、カンマ区切りで、2 つ目の引数として、代入する既定値を指定します。
例えば、日付型の項目に既定値として、今日の日付を指定したい場合、
「&Today」という変数（※）を利用できます。


このルールは、データの新規登録時のみ実行され、既存データを更新したり、
削除する場合には実行されないルールです。

※ GeneXus における変数については、後述

既定値を設定する制御の挙動

顧客

K < > >I 選択

顧客番号	<input type="text" value="0"/>
顧客名前	<input type="text"/>
顧客名字	<input type="text"/>
顧客住所	<input type="text"/>
顧客電話番号	<input type="text"/>
顧客メールアドレス	<input type="text"/>
顧客追加日	<div><input type="text" value="24/08/06"/> </div>

削除

終了

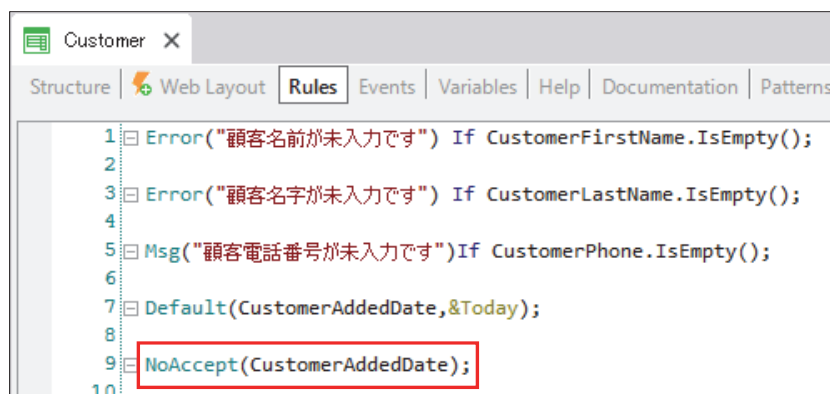
実行

Default ルールが定義された場合、データを新規登録する場合、既定値が設定されます。また、設定される値は、「既定値」であるため、必要に応じてユーザーが変更可能な状態で生成されます。

このルールは、前述の通り、新規登録時のみ動作し、既定値を入力する制御です。そのため、既存のデータがある場合、これらには Default ルールによる既定値の設定は発生しません。

フィールドの入力不可制御

- NoAccept ルール



トランザクションオブジェクトによって生成される登録画面において、ユーザーによる入力を許可しない項目が必要な場合があります。
このような要望があった場合、NoAccept ルールを利用することで、実現することができます。

このルールでは、括弧内に入力を許可しない項目属性名の入力を行います。

入力不可制御の挙動

顧客

K < > >I 選択

顧客番号	<input type="text" value="0"/>
顧客名前	<input type="text"/>
顧客名字	<input type="text"/>
顧客住所	<input type="text"/>
顧客電話番号	<input type="text"/>
顧客メールアドレス	<input type="text"/>
顧客追加日	<input type="text" value="24/08/06"/>

削除

終了

実行

アプリケーションを実行すると、対象とした項目属性は、入力不可となっていることが確認できます。

この制御は、新規登録、更新、削除すべての状態で有効となっています。
そのため、ユーザーによる一切の入力または変更ができない状態となりました。

Default ルールと組み合わせることで、登録時に値を設定し、この値をユーザーは変更できないという実装が可能となります。

値の代入

- Assignment ルール

```
CustomerAddedDate = &Today;
```

データの操作内容に関係なく常に実行

```
CustomerAddedDate = &Today If Insert;
```

データの新規登録時のみに実行

既定値を割り当てるために、Default ルールの定義を行いました。項目属性に値を割り当てる場合、「=」を利用した値の代入も実装可能です。一般的な値の代入式と同様に、「=」の左辺に値を受け取るもの、右辺に代入する値を記述します。

このような記述を GeneXus では、Assignment ルールと総称しています。

Default ルールと異なる点としては、条件を指定しない場合、データの新規登録、更新、削除、どのデータ操作でも実施されるルールになります。

もし、Default ルール同様に、新規登録時のみ動作させたい場合、条件に「Insert」と指定することで実装可能です。

この「Insert」については、GeneXus が生成したトランザクションオブジェクトに基づき、生成された画面で、データ操作のモードが該当する場合に True、しない場合には False を返す関数です。

他に、「Update」、「Delete」が用意され、それぞれのモードのみで利用したいルールを定義する際に活用できます。

では、「Insert」を条件にし、値を代入した場合、挙動は Default ルールと同じでしょうか？

この点について、後ほど詳細を解説します。

ルールのトリガー順序



ルールは定義した順序で実行されるわけではない

GeneXus において、ルールを定義する際に留意すべき重要な点は、

「ルールの定義は宣言型である」

ということです。

つまり、ルールが定義されている順序は、必ずしも実行される順序ではないということです。

例えば、スライドのようなルールが定義されていた場合、定義順を変えた場合も、アプリケーション上で実行される順序は常に同じになります。

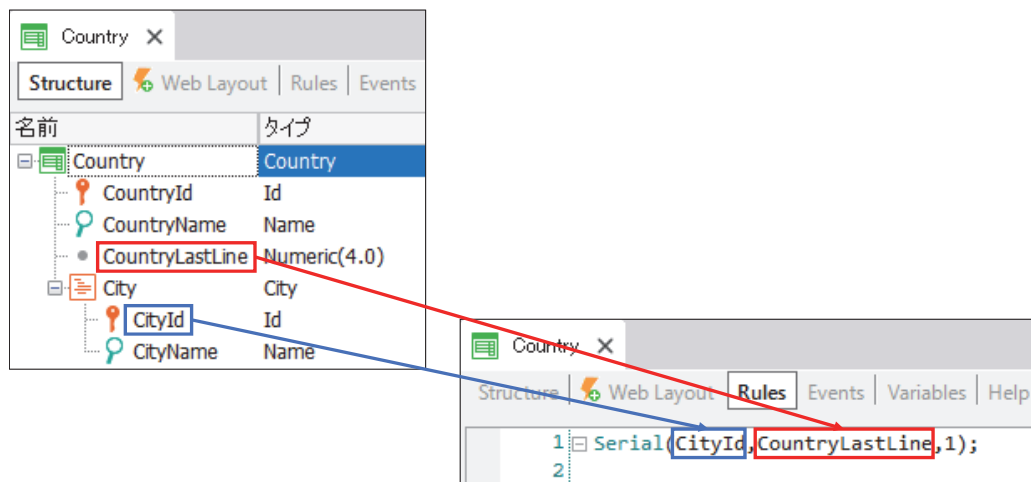
前述の通り、ルールは宣言型となるため、宣言されたルールをどのように実行するかは GeneXus によって決定します。

また、定義されたルールが実行される画面については、定義されたトランザクションに基づき、生成された画面（機能）のみです。

例えば、旅行代理店が利用するアプリケーションに顧客を管理する画面以外に、観光名所や国を管理する画面がある場合、それぞれのトランザクションオブジェクトで、そのオブジェクトから生成される画面に必要なルールを定義する必要があります。

第 2 レベルの自動採番

- Serial ルール



ここまでの章の中で、Country トランザクションへ、第 2 レベル City を追加したところ、第 2 レベルの主キー CityId に対しては、[Autonumber] プロパティの設定が無視されることが影響分析において表示されていました。

このような第 2 レベルの主キーを対象に自動採番する機能が必要となる場合、Serial ルールを定義することで、要望をかなえることができます。

Serial ルールを定義するにあたり、初めに項目属性を 1 つ追加する必要があります。この項目属性は、自動採番する項目属性の最後の値を格納するために利用します。そのため、自動採番の対象となる項目属性よりも 1 つ上のレベルで定義する必要があります。

では、実際に Serial ルールを定義していきます。このルールには、引数が 3 つ必要となります。

- 1 つ目の引数は、自動採番を行いたい項目属性、つまり第 2 レベルの主キーです。
- 2 つ目の引数は、自動採番された最後の値を格納する項目属性、つまり直前に定義した項目属性です。
- 3 つ目の引数は、自動採番する際に、加算する値を指定します。通常は 1 を指定し、値が 1 ずつ加算されるように定義します。

ルールの最後にセミコロンを入力し、ルールの定義ができました。

第 2 レベルの自動採番

The screenshots illustrate the automatic numbering process in a GeneXus application. The interface consists of a '国' (Country) section and a '都市' (City) section.

国 (Country) Section:

- 国番号 (Country Number): 0
- 国名 (Country Name): ドイツ (Germany)
- 都市最終番号 (City Final Number): 0 (highlighted with a red box)

都市 (City) Section:

- 都市番号 (City Number): 0 (highlighted with a blue box)
- 都市名 (City Name):

Process Flow:

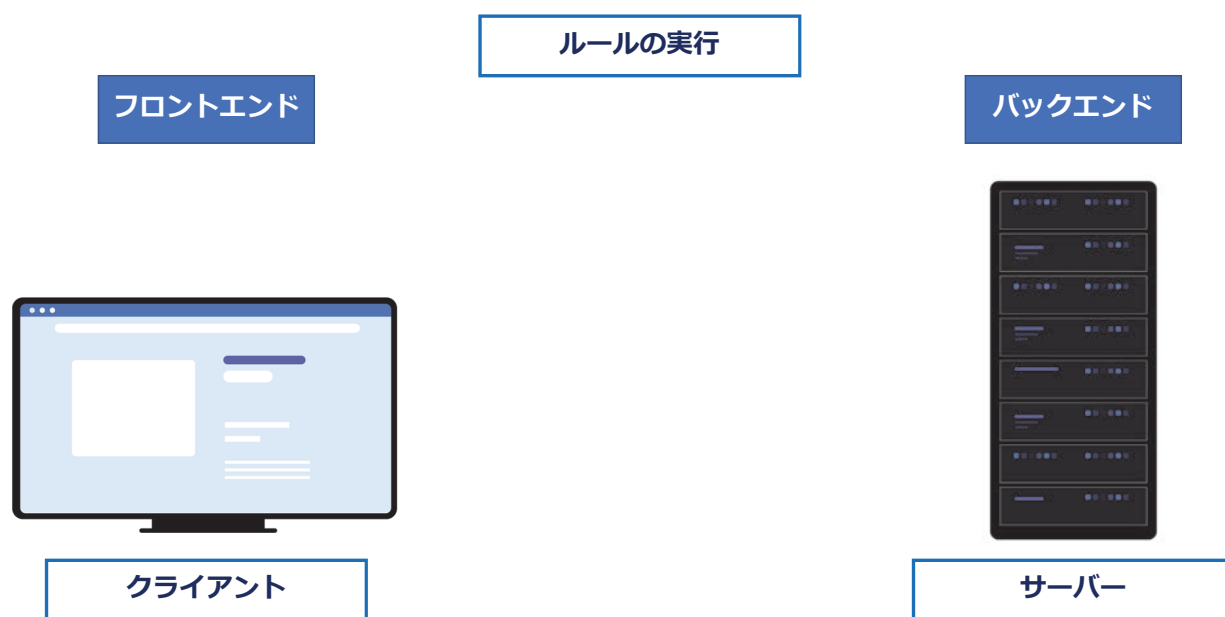
- Initial State:** The '都市番号' field is 0, and the '都市最終番号' field is 0.
- First Transition:** After moving focus to the next item, the '都市番号' field becomes 1 and the '都市名' field is populated with 'ベルリン' (Berlin). The '都市最終番号' field remains 0.
- Second Transition:** After adding more items, the '都市番号' field becomes 3 and the '都市名' field is populated with 'ミュンヘン' (Munich). The '都市最終番号' field remains 0.

Blue arrows indicate the flow between the screenshots. The '都市最終番号' field is highlighted with a red box in all three screenshots, indicating it is the target of the automatic numbering process.

アプリケーションを実行すると、Serial ルールの第 1 引数に指定した項目は、値を入力しない状態で、フォーカスを次の項目へ移動すると、自動的に採番された最新の値が入力されることが確認できます。

また、第 2 引数に指定した 1 レベル上の項目属性は、画面上に配置されますが、入力を許可しない項目として配置されることが確認できます。

クライアントサイドとサーバーサイド



ここまでで説明してきたルールと、その実行時の動作から、ルールはクライアントサイドで実行されるもののように見えます。

例えば、Error ルールの場合、条件を満たすとすぐにメッセージが画面上に表示されました。

ですが、Error ルールの条件を満たしたまま他の項目の入力を続けることができ、最終的に実行ボタンをクリックすることができます。

この実行ボタンをクリックした場合、トランザクションに関連するサーバーサイドのプログラムも実行され、このプログラムの中には、ルールに基づく処理も含まれます。つまり、ルールに基づく処理はサーバーサイドでも再度実行され、最終的な処理を実施します。

前述の例の通り、Error ルールの条件を満たしていた場合、サーバーサイドでエラーの条件を満たしていると判断し、テーブルへの保存処理は実行されません。

クライアントとなるブラウザは、アプリケーションへの攻撃対象となることがあり、改ざんされた情報をサーバーに送信する可能性があります。

そのため、最終的なデータ処理の制御をサーバーが担うことが必要となり、GeneXus は、このようにプログラムを生成します。

繰り返しとなりますが、優れた UX を提供するためのフロントエンドプログラムと、ビジネスロジックの最終的な制御を行うバックエンドプログラム両方に、ルールによる制御が生成されています。

この点を理解しておくことが、今後の説明でルールによって制御可能な対象が複雑になったときに重要となります。

また、フロントエンドでのルールの実行については、必要に応じて無効にする実装も可能です。

ただし、バックエンドでのルールの実行は必ず生成されます。

クライアントサイドとサーバーサイド（補足）

ルールの実行

フロントエンド

```
Default(CustomerAddedDate,&Today);
CustomerAddedDate = &Today If Insert;
```



クライアント

バックエンド

```
Default(CustomerAddedDate,&Today);
CustomerAddedDate = &Today If Insert;
```



サーバー

フロントエンドと、バックエンド両方で同じルールが実行されるという点で重要となるのが定義されたルールの内容です。

Default ルールと、Assignment ルールによる結果の比較を行います。

Default ルールの場合、新規登録時に既定値を設定するルールとなるため、フロントエンドでルールを実行すると、既定値が設定されます。そして、実行ボタンが押された場合のバックエンドでもルールが実行されますが、もしユーザーによって既定値から別の値へ変更があった場合、この変更された値が優先されます。これは、Default ルールがあくまでも「既定値」を設定するルールのためです。

では、Assignment ルールの場合はどうでしょうか。

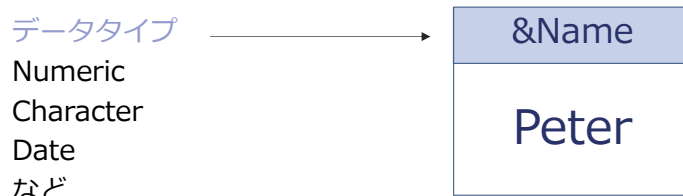
Default ルールと同様に新規登録時のみ実行するように条件として「If Insert」が記述されている場合、フロントエンドでルールを実行すると、対象の項目属性に値が代入されます。

そして、実行ボタンが押され、バックエンドでもルールが実行された場合、ユーザーによって値の変更があった場合も、ルールによる代入が実行されます。つまり、画面上で値を変更しても内部的な処理で元の値に修正されるという挙動の実装となります。

もし、該当の項目属性が NoAccept ルールにより、入力不可となっている場合、このようなシナリオは発生しませんが、注意が必要です。

変数の利用

- メモリー領域へ値を一時的に格納する
- 変数を参照する場合、「&」記号を付け、変数名を指定する



例: 変数 &Name で、メモリーに「Peter」という値を格納

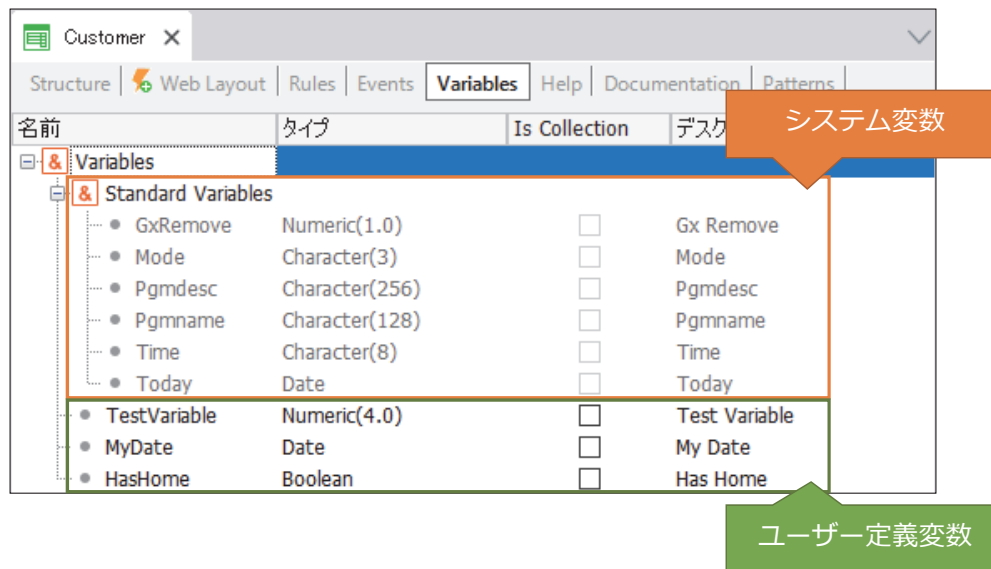
トランザクションの [Structure] エlementでは、テーブルに値を格納するために、項目属性を定義することが出来ました。

しかし、アプリケーションでは、テーブルに値を格納する必要はなく、一時的に値を格納したいという場合もあります。

このような場合に、GeneXus では、「変数」を利用することができます。

変数は、オブジェクト内で一意な名前を定義し、データタイプを指定します。指定したデータタイプに基づき、メモリー領域に値を格納可能となります。

変数の定義



GeneXus で作成可能なほとんどのオブジェクトには、変数を定義するための [Variables] エlementがあります。

このElementで定義することができる変数は、オブジェクトにローカルであり、オブジェクト内のみで利用可能です。

定義した変数を [Rules] Elementなどで利用する場合、[Variables] Elementで定義された名前の頭に「&」が必要となります。

[Variables] Elementを開き、「Standard Variables」というノードを展開すると、読み取り専用でいくつかの変数が定義済みです。

これは、システム変数と呼ばれ、GeneXus によってあらかじめ定義された変数です。Default ルールの際に利用した &Today もこのシステム変数の一つであり、現在の日付を常に取得しています。

システムに必要な変数がある場合、[Structure] Element同様に Enter キーで、新しい変数の定義を追加できます。

変数を定義する場合、変数名のキャメルケース最後の単語がデータタイプやドメインと一致する場合、変数のデータタイプも自動で選択されます。

その他、「Is」や「Has」から始まる変数名とした場合、Boolean 型が自動で選択されます。

GeneXus[™]

training.genexus.com
wiki.genexus.com