【セッションNo. 2】

Delphi/400開発ノウハウお教えします
「メニュー」開発のテクニック

株式会社ミガロ.
システム事業部　プロジェクト推進室
小杉　智昭

【アジェンダ】

- 『メニュー』の種類と基本的な作成方法
- 『メニュー』開発テクニックのご紹介
  - ①メンテナンス性を考慮したメニュー押下制御
  - ②ツリー形式で動的に作成するメニュー
  - ③使い勝手を向上させるメニュー
- アプリケーション開発スタイルに応じた『メニュー』開発
- まとめ

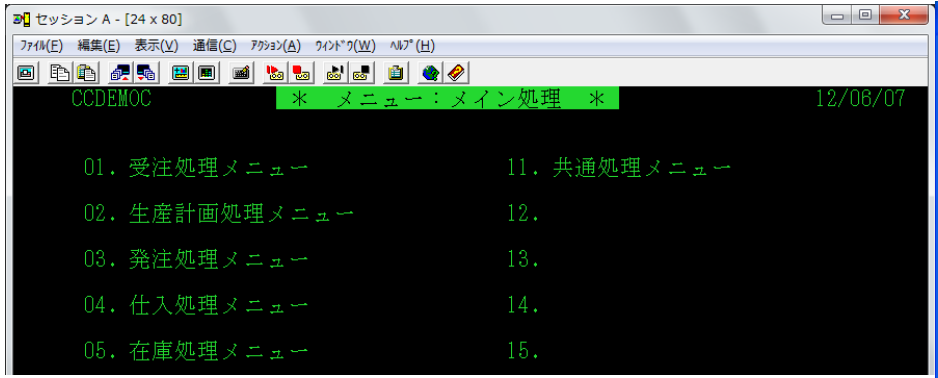
『メニュー』の種類と基本的な作成方法

『メニュー』画面

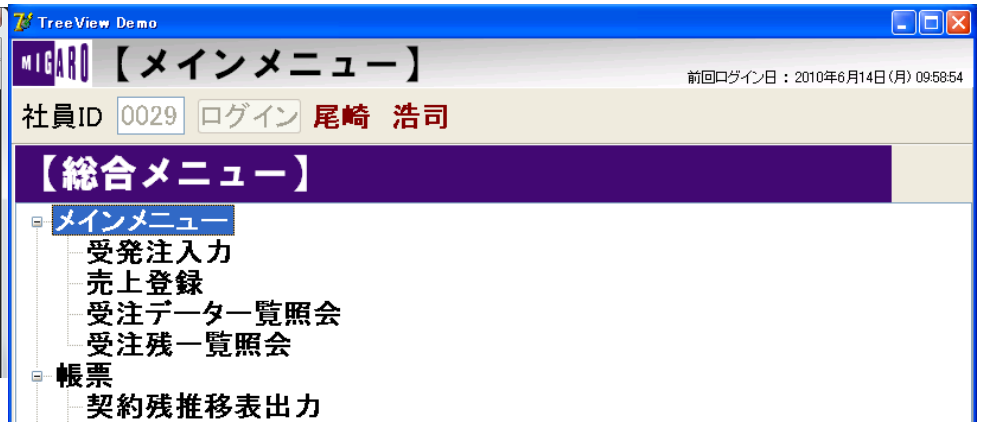
CUI と GUI

- IBM i(AS/400)の『メニュー』画面 （CUI画面）
  - 80×24の制限のため、機能番号を入力して、実行する方式が一般的
- Delphi/400の『メニュー』画面 （GUI画面）
  - ボタン配置形式、ツリー形式等GUIの特性を生かした多彩なメニューが作成可能

IBM i（AS/400）の『メニュー』

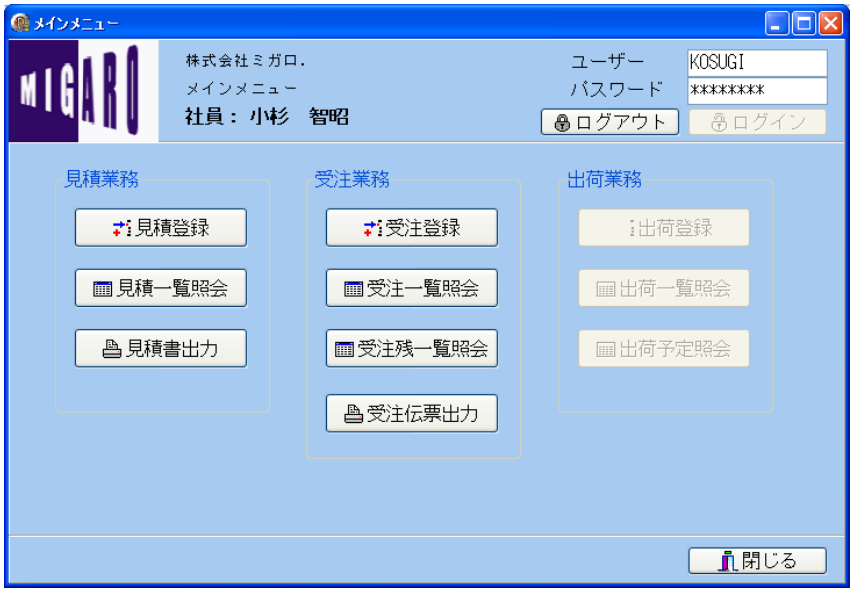


Delphi/400の『メニュー』

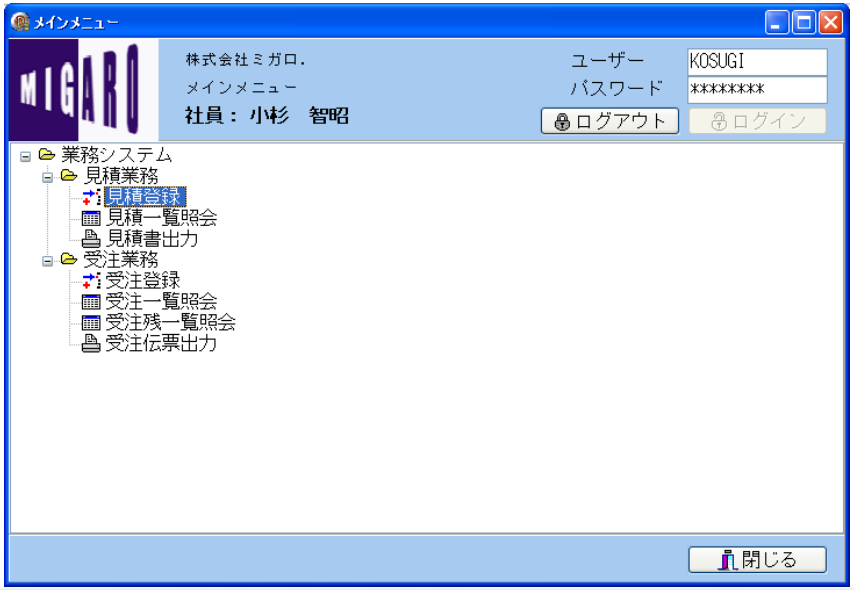


Delphi/400での代表的な2つの『メニュー』形式

- ボタン配置形式
  - 画面上に機能ボタンを配置する方式
  - レイアウト作成が容易
  - 機能数が多い場合、メインメニュー⇒サブメニューと画面展開するような構成が多い



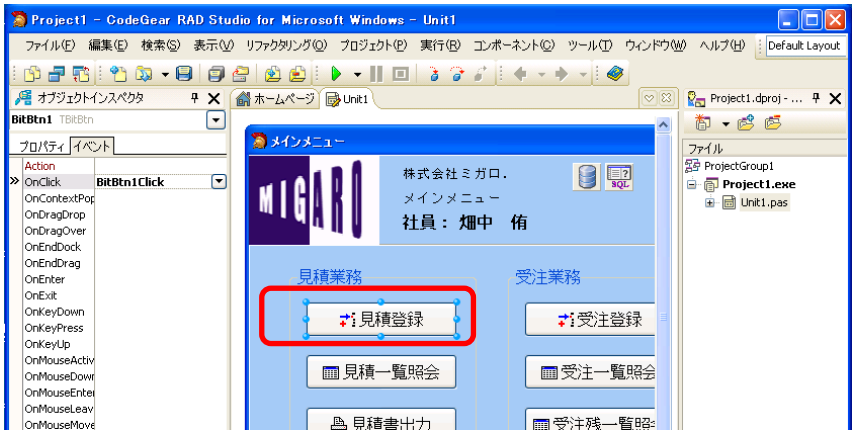
- ツリー形式
  - ツリー構造に機能（ノード）を配置する方式
  - 展開/折り畳み等が容易で、一画面で全体像が表現しやすい
  - メンテナンスが容易



ボタン配置形式メニューの作成手順

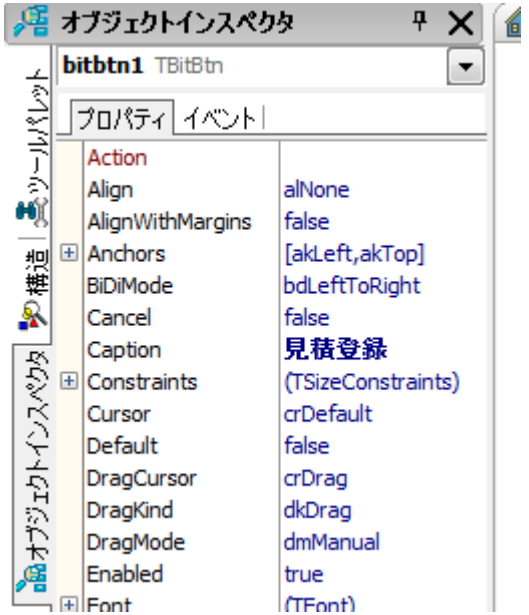
STEP1　フォームのレイアウトを作成する

STEP2　TBitBtnをフォームの任意の場所に追加し、各プロパティを設定する



＜プロパティ例＞

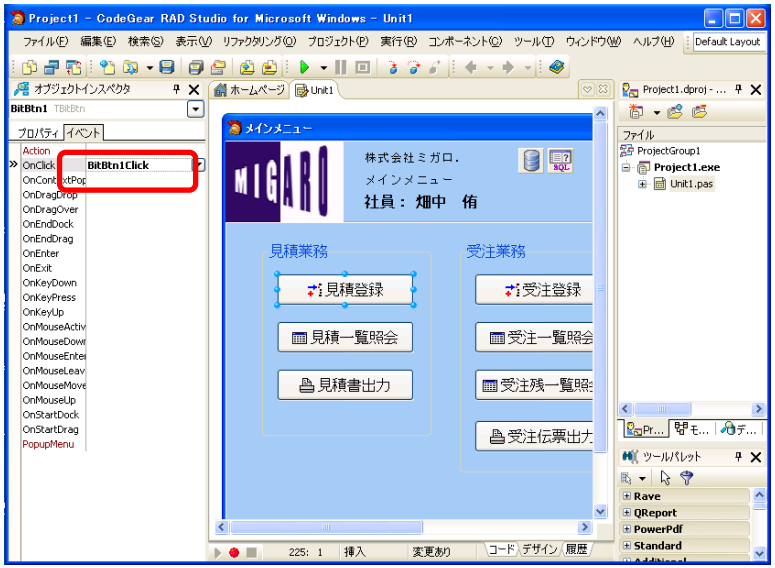
Caption　：ボタン表面の任意の説明文
Glyph　：任意のアイコン画像
Name　：命名規則にしたがった名前
TabOrder　：ボタンのタブ順



ボタン配置形式メニューの作成手順

STEP3

TBitBtnのイベントを実装する
（OnClickイベント等）



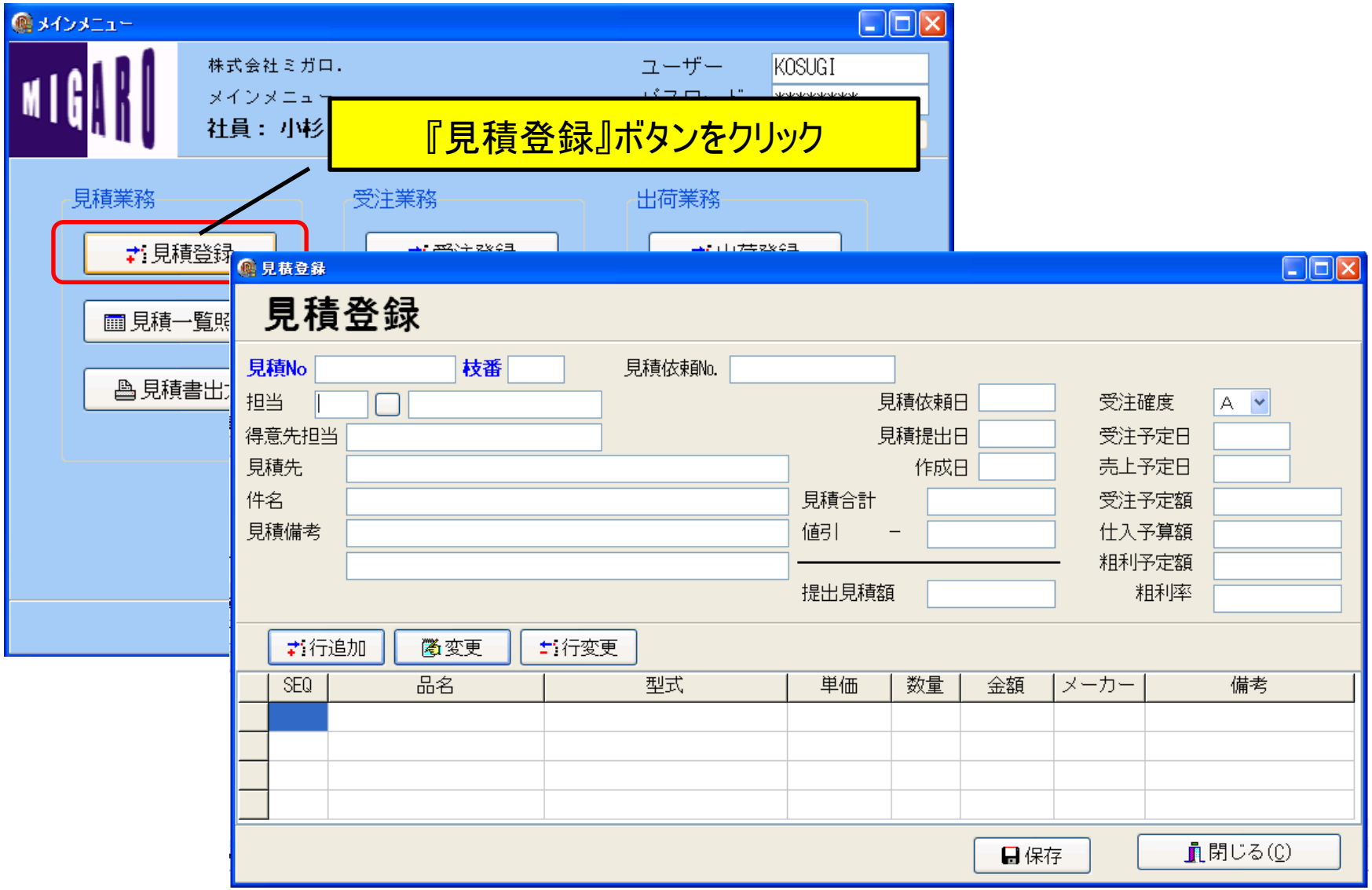

```
procedure TfrmMenu.BitBtn1Click(Sender: TObject);
begin
  frmInputEst := TfrmInputEst.Create(Self);
  try
    // 見積登録画面を表示
    frmInputEst.ShowModal;
  finally
    // 画面の終了
    frmInputEst.Release;
  end;
end;
```

①画面の生成

②画面の表示

③画面の破棄

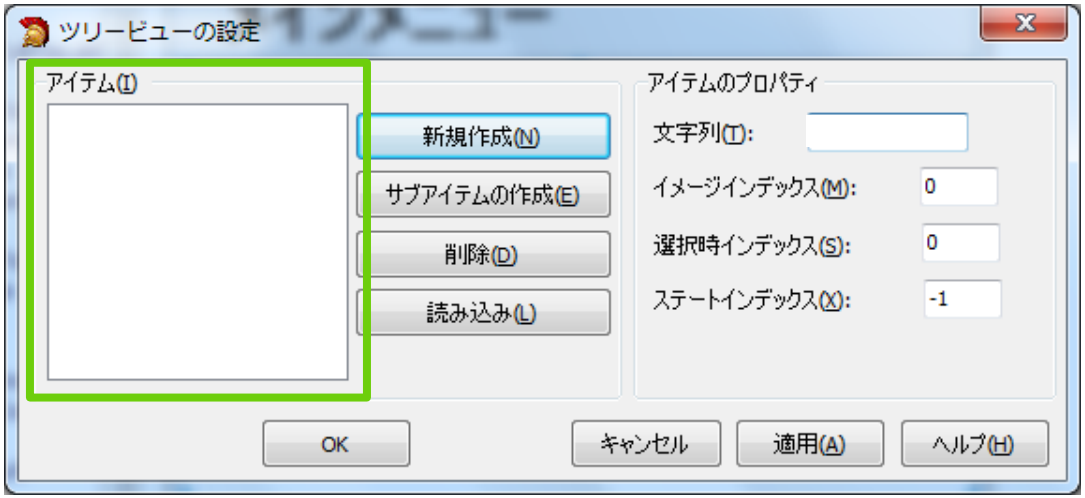
ボタン配置形式の実行イメージ



ツリー形式メニューの作成手順

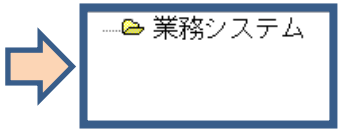
STEP1 フォームのレイアウトを作成する

STEP2 TTreeViewをフォーム上の任意の場所に配置し、項目の設定を行う



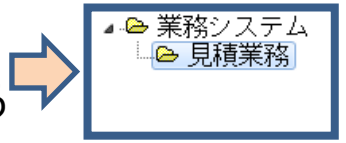
新規作成

現在のノードと同じ階層にノードを追加する



サブアイテムの作成

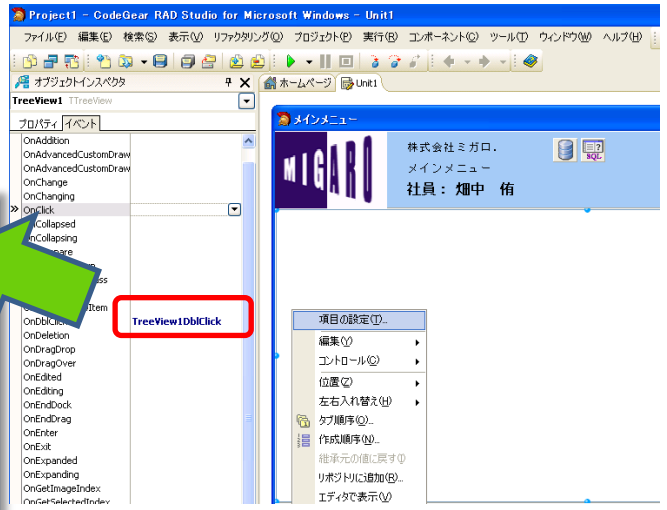
選択しているノードの下の階層にノードを追加する



ツリー形式メニューの作成手順

STEP3

OnDblClickのイベントを実装する



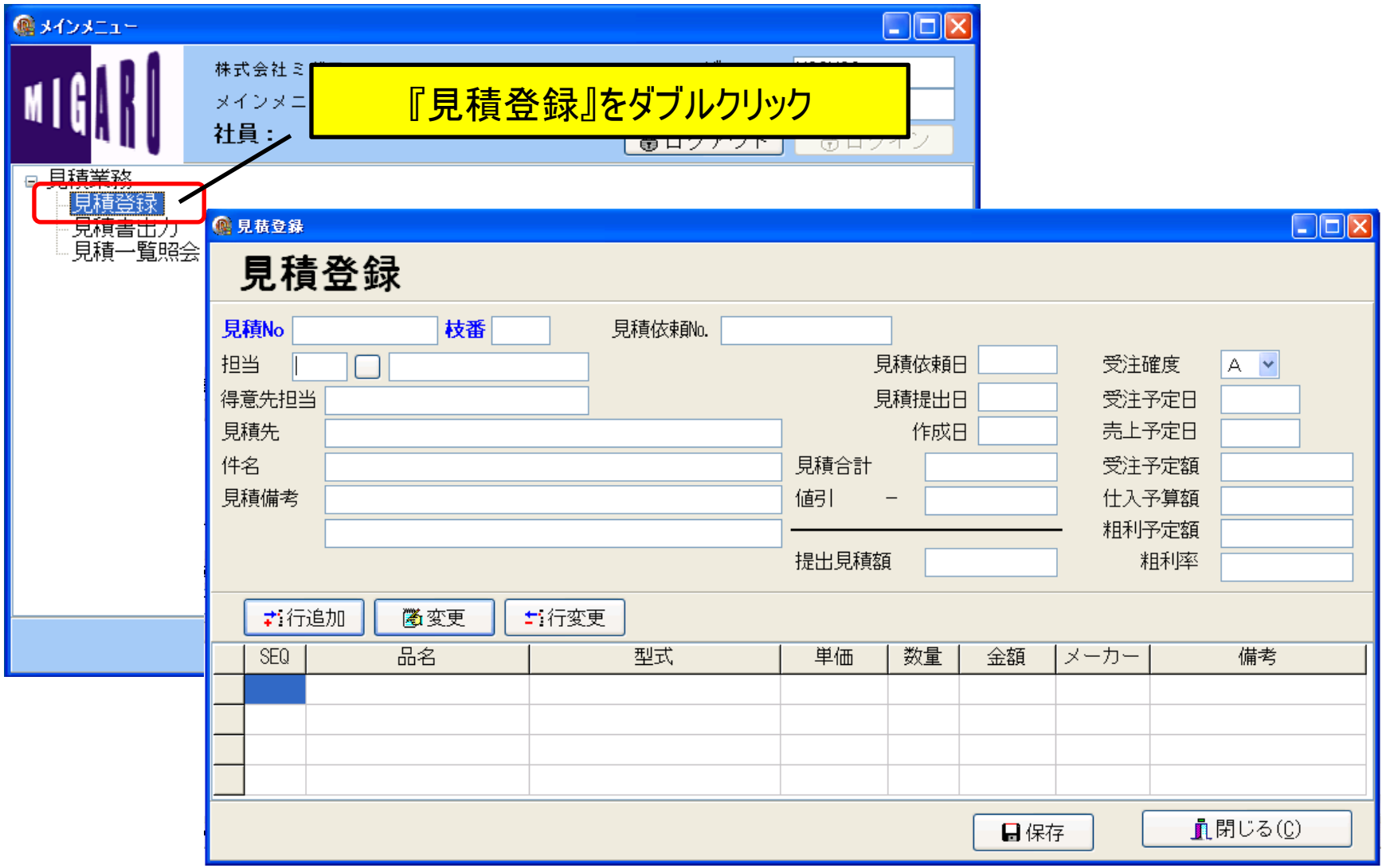

```
procedure TfrmMenu.TreeView1DblClick(Sender: TObject);
var
  sText: String;  //選択機能アイテム文字列
begin
  sText := TreeView1.Selected.Text;  //選択行文字列取得

  //画面遷移
  if sText = '見積登録' then
  begin
    frmInputEst := TfrmInputEst.Create(Self);
    try
      // 見積登録画面を表示
      frmInputEst.ShowModal;
    finally
      // 画面の終了
      frmInputEst.Release;
    end;
  end
  else if sText = '見積書出力' then
...
```

①画面の生成

②画面の表示

③画面の破棄

ツリー形式の実行イメージ

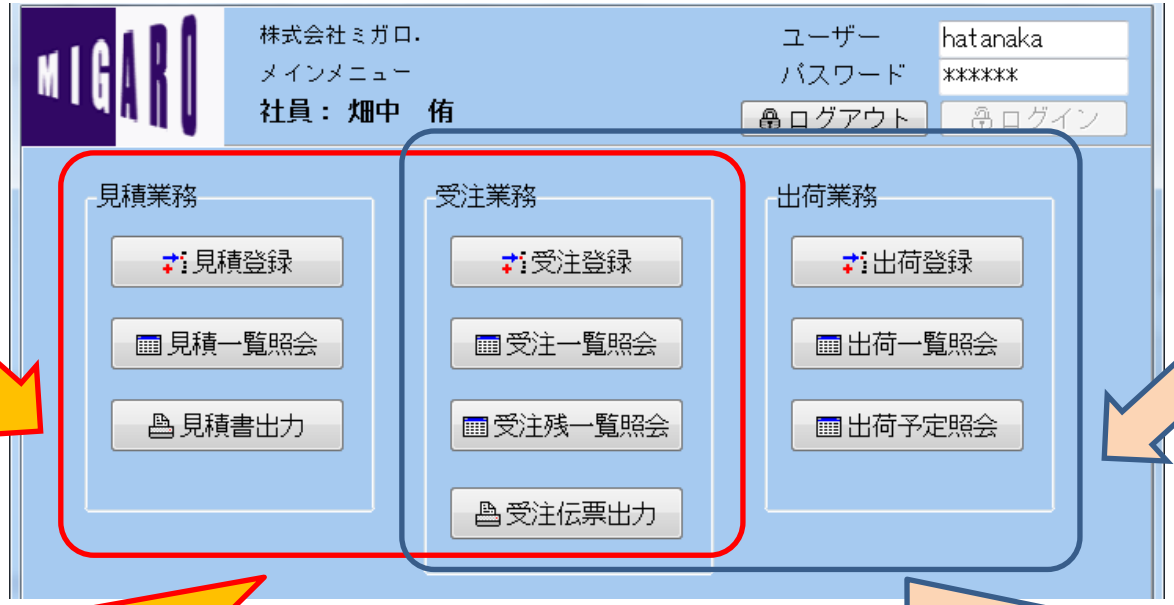


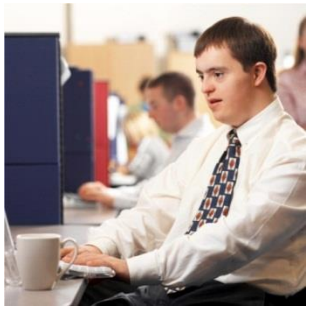
開発テクニック①
メンテナンス性を考慮したメニュー押下制御

担当者の権限によるメニューの制御

- 各業務担当者の権限により、使用できるメニューを制限するにはどうすればよいか？

営業担当者





出荷担当者

営業担当者が使用可能

出荷担当者が使用可能

- 各担当者毎に個別に割り当てられた権限をもとに、機能ボタンの制御を行えばよい（使用できないボタンは、グレイアウト（押下不可）にする。）

担当者の権限取得方法

- 「営業担当者」「出荷担当者」等、それぞれの役割に応じて使用できるボタンを制御する

  - 担当者マスタ

ユーザーID	パスワード	ユーザータイプ
KOSUGI	PASSWORD	“1”
OZAKI	MIGARO	“1”
YOSHIWARA	RAD	“2”
HATANAKA	SUSUMU	“2”

ユーザータイプ
“1”： 営業担当者
“2”： 出荷担当者

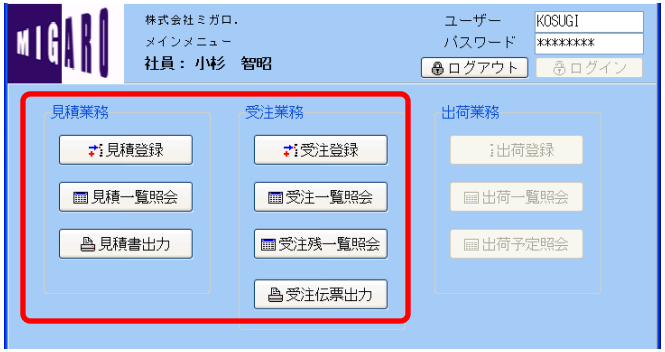
【実現方針】
指定したユーザーIDよりユーザータイプを取得して、
各機能ボタンの押下可否を動的に指定する。

3つの処理タイミング

処理1 画面起動時

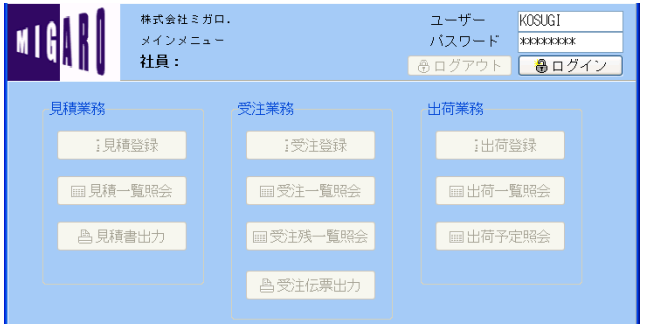
ログイン前の為
全てのボタンを押下不可とする

処理2 ログイン処理時



ユーザータイプに応じて
使用可能なボタンのみ押下可能とする

処理3 ログアウト処理時



再び
全てのボタンを押下不可とする

TBitBtnのみで押下制御を実現する場合

- ボタン押下制御の実装例

処理1 画面起動時

処理3 ログアウト処理時

```
procedure TfrmMenu.FormShow(Sender: TObject);
begin
  //全機能ボタンを使用不可とする
  btnEntryEst.Enabled := False;
  btnReferEst.Enabled := False;
  btnPrintEst.Enabled := False;
            …
  btnEntryShip.Enabled := False;
  btnReferShip.Enabled := False;
  btnReferPlan.Enabled := False;
end;
```

処理2 ログイン処理時

```
procedure TfrmMenu.btnLoginClick(Sender: TObject);
begin
      ～～（ログイン処理）～～
  //営業担当者　使用可能ボタン
  btnEntryEst.Enabled := (FUSERTYP = '1');
  btnReferEst.Enabled := (FUSERTYP = '1');
  btnPrintEst.Enabled := (FUSERTYP = '1');
                …
  //出荷担当者　使用可能ボタン
  btnEntryShip.Enabled := (FUSERTYP = '2');
  btnReferShip.Enabled := (FUSERTYP = '2');
  btnReferPlan.Enabled := (FUSERTYP = '2');
end;
```

FUSERTYP ： ログインユーザーのユーザータイプを保持
'1'： 営業担当者
'2'： 出荷担当者

- それぞれの処理タイミング毎に、個別にEnabledプロパティを設定するロジックを記述しなければならない。

機能ボタンの押下制御をシンプルに処理できないか？

画面の状態を一元管理することができる『TAction』を使用すると
押下制御を一括制御することができる！

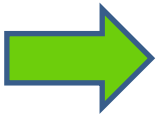
- アクション（TAction）コンポーネント
  - 画面上で使用する処理（アクション）を一元管理するためのコンポーネント
  - OnUpdateイベントを使うことで、どのイベント発生時も適用されるルールを記述できる

例）ログインボタンの制御

＜従来の考え方＞

ログインボタンのOnClickイベント
　ログインボタンを**使用不可**

ログアウトボタンのOnClickイベント
　ログインボタンを**使用可**

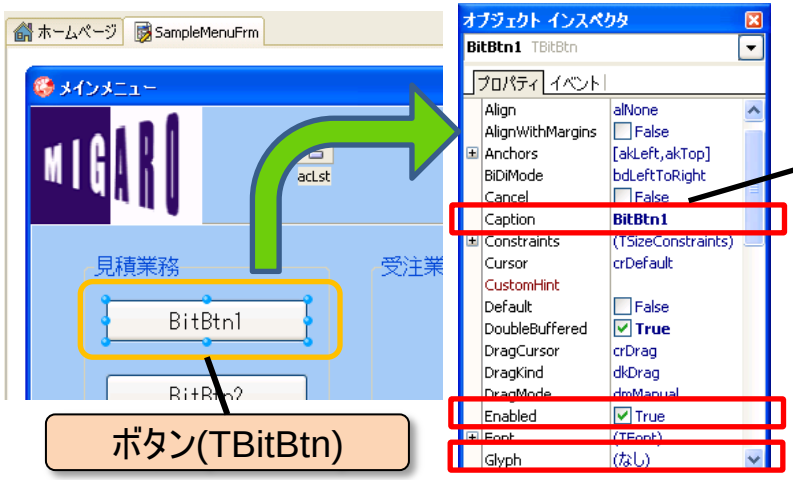


＜アクションを使った考え方＞

アクションのOnUpdateイベント
　ログインされていれば**使用不可**

アクション（TAction）概念図

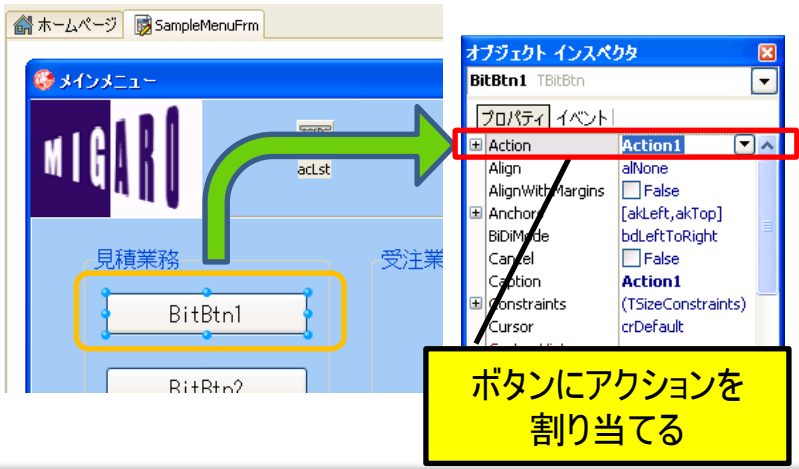
ボタンに直接設定する場合



ボタン(TBitBtn)

ボタンに表題や画像など
プロパティを直接セット

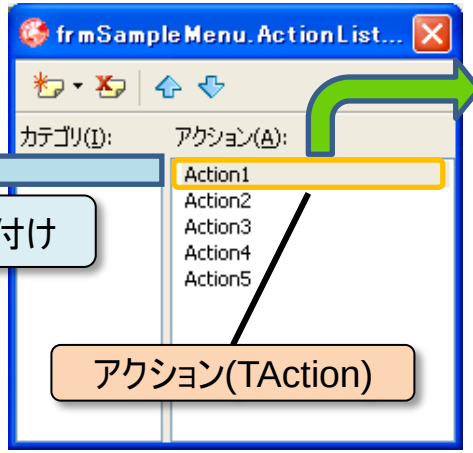
アクションを使用する場合



ボタンにアクションを
割り当てる

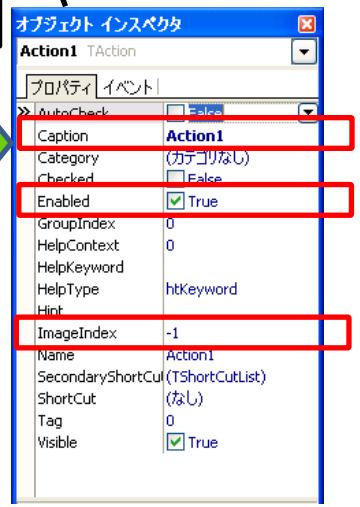
関連付け

「アクションリストの設定」画面



アクション(TAction)

アクションに表題や画像など
プロパティをセット



アクション（TAction）の作成手順

STEP1

TActionListをフォームに貼り付ける

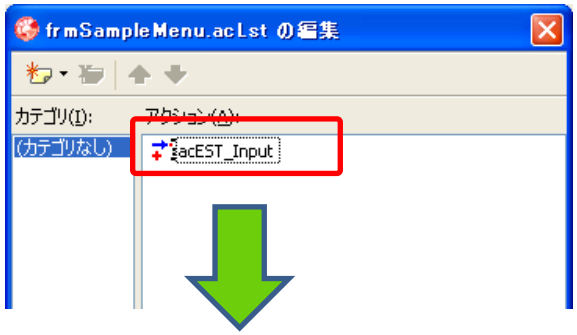
・[ツールパレット] → [Standard]カテゴリ → [TActionList]を選択

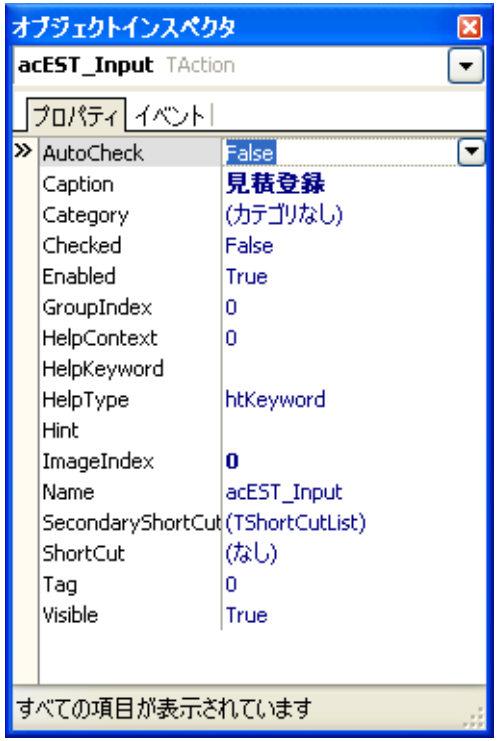
STEP2

TActionListに新しいアクション(TAction)を追加する

・アクションリストコンポーネント右クリックより[アクションリストの設定]を選択
・[アクションリストの設定]画面で、「新規アクション」ボタンを押下

アクション(TAction)の作成手順





STEP3 プロパティ(設定)の定義

- Caption :表題となる文字列
- ImageIndex :アイコン画像のインデックス
- Name :アクションに付与する名前

STEP4 イベント(動作)の実装

- OnExecute :実行した時の処理を記述
  (ButtonのOnClickに相当)

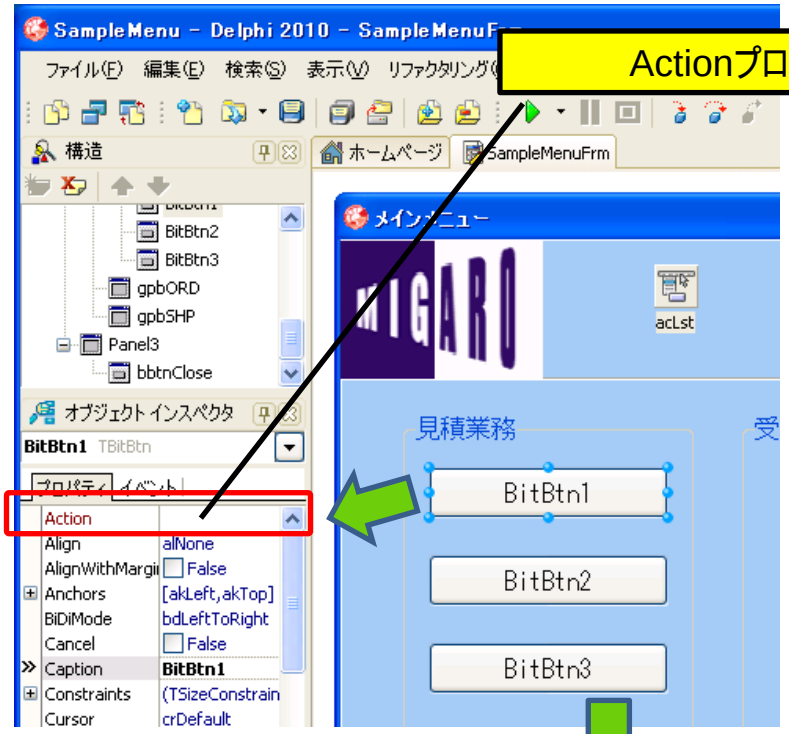
```
procedure TfrmMenu.acEST_InputExecute(Sender: TObject);
begin
  frmInputEst:= TfrmInputEst.Create(Self);
  try
    // 見積登録画面を表示
    frmInputEst.ShowModal;
  finally
    // 画面の終了
    frmInputEst.Release;
  end;
end;
```

アクション（TAction）の作成手順

STEP5

ボタンコンポーネントにアクションを割当

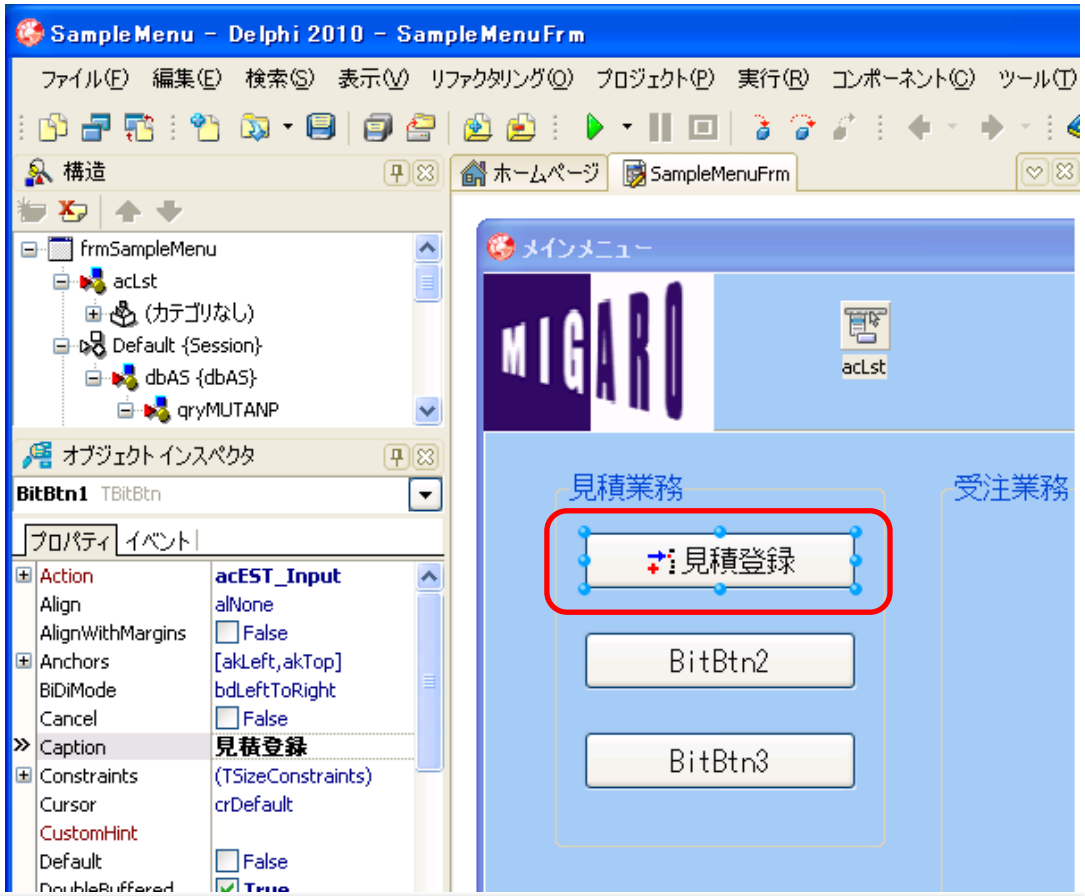
・ボタンコンポーネントのActionプロパティに値を設定



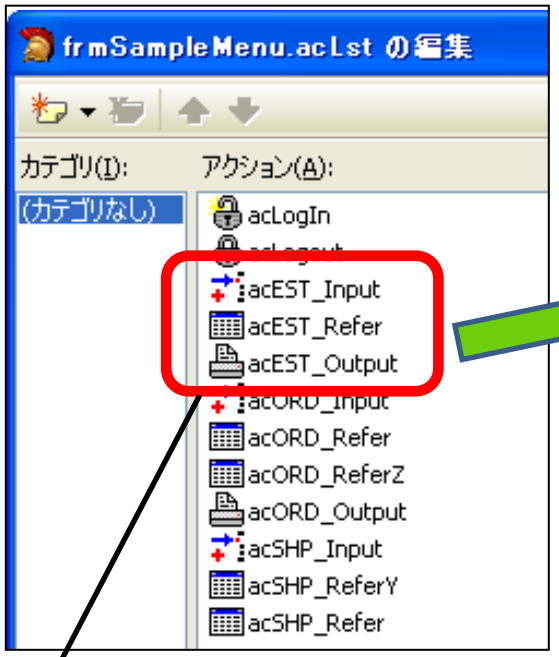
Actionプロパティ

初期状態のボタンに
アクションを割り当てる

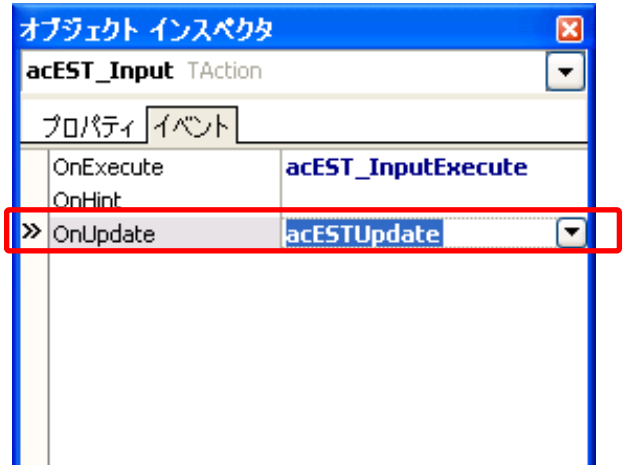
アクションで定義したプロパティ、
イベントがTBitBtnに適用される



アクション(TAction)を使用した押下制御



アクションのイベント



- OnUpdate　　：**待ち受け時の処理を記述**
  （アクションの有効/無効等を定義）

見積業務の
各アクションを複数選択

```
procedure TfrmSampleMenu.acESTUpdate(Sender: TObject);
begin
  //ユーザータイプ＝１（営業）のみ使用可能
  (Sender as TAction).Enabled := (FUSERTYP = '1');
end;
```

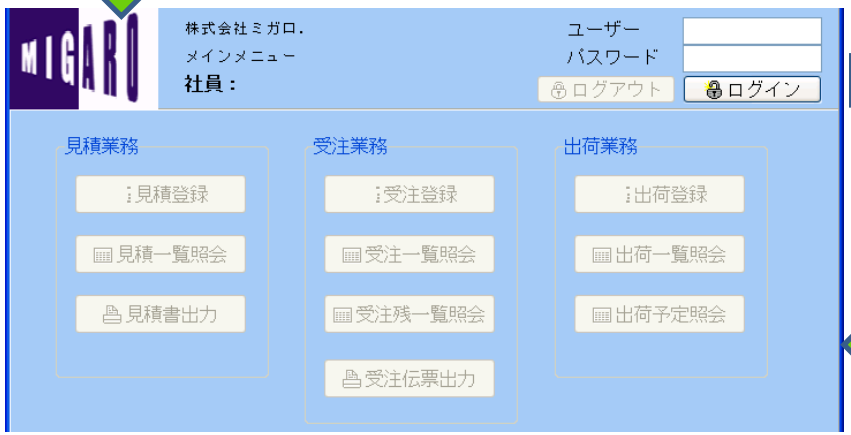
FUSERTYP (ユーザータイプ) が ’1’ となっている間は、アクションが有効
そうでない場合は、アクションが無効

アクション(TAction)を使用した押下制御

OnUpdateイベントの動作イメージ

処理1 画面起動時

FUSERTYP = '' (アクション無効)



処理2 ログイン時

FUSERTYP = '1' (アクション有効)

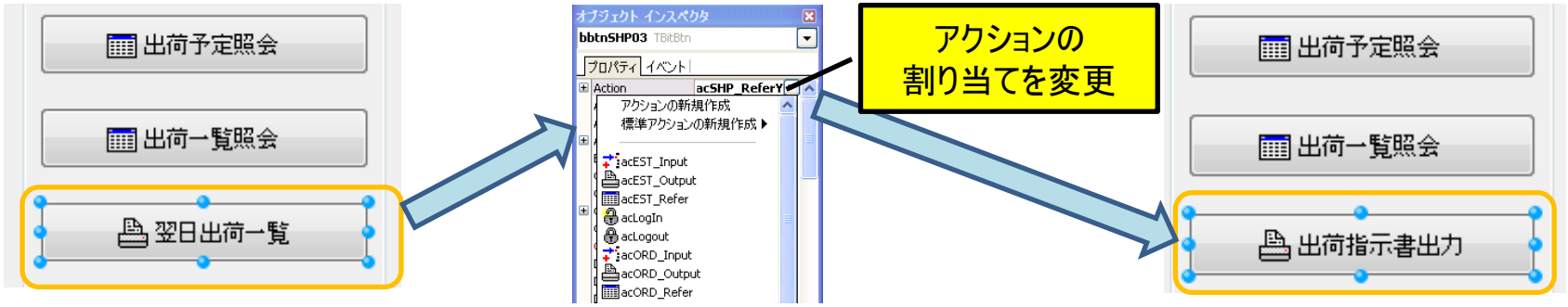
処理3 ログアウト時

FUSERTYP = '' (アクション無効)

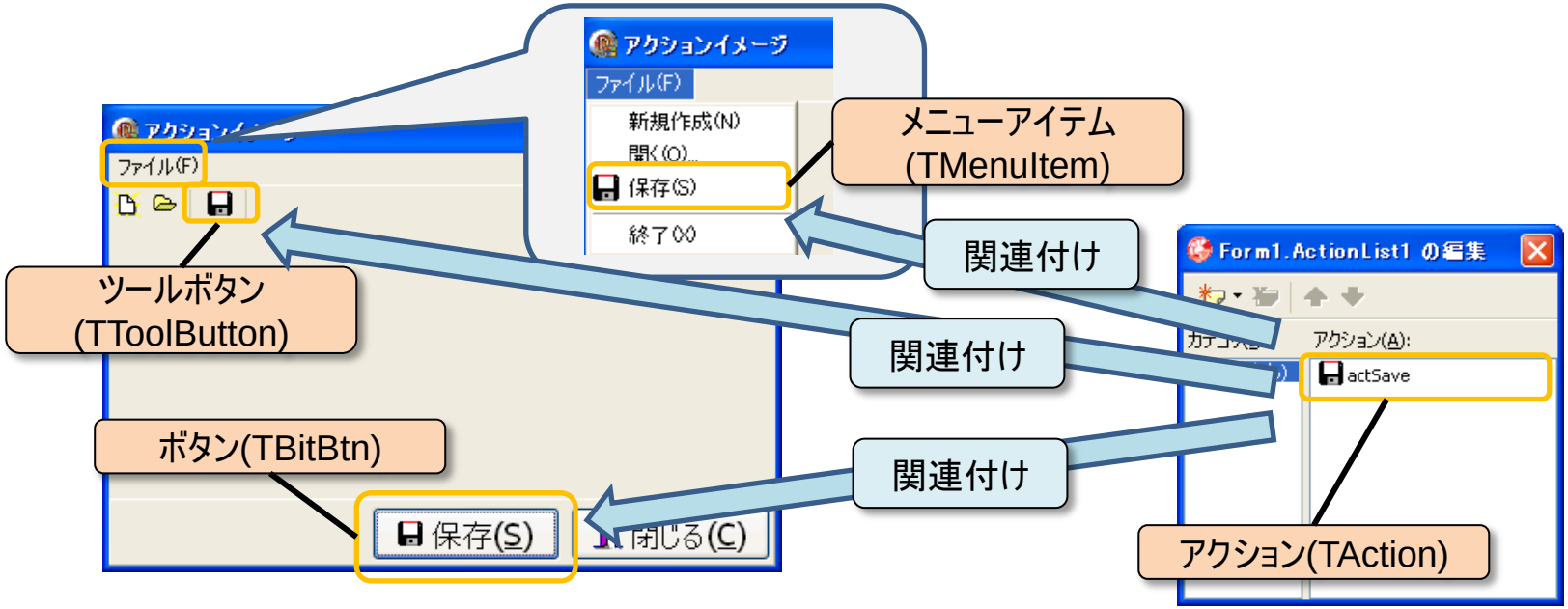
- OnUpdateイベントに制御を記述することで、FUSERTYP(ユーザータイプ)の値変化に連動したボタンの押下制御が可能になる
（処理タイミング毎の個別ボタン制御が不要となる。）

【参考】OnUpdateイベント以外にアクション（TAction）を使うメリット

- ボタンの割り当てを変えるだけで機能の変更が可能



- ひとつのアクションを複数のボタンに割り当てることが可能



開発テクニック②
ツリー形式で動的に作成するメニュー

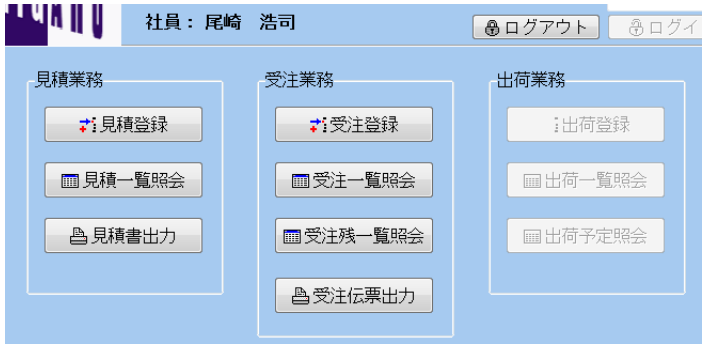
『動的』に作成するメニュー

- ツリー形式では、メニューを『動的』に作成することができる。

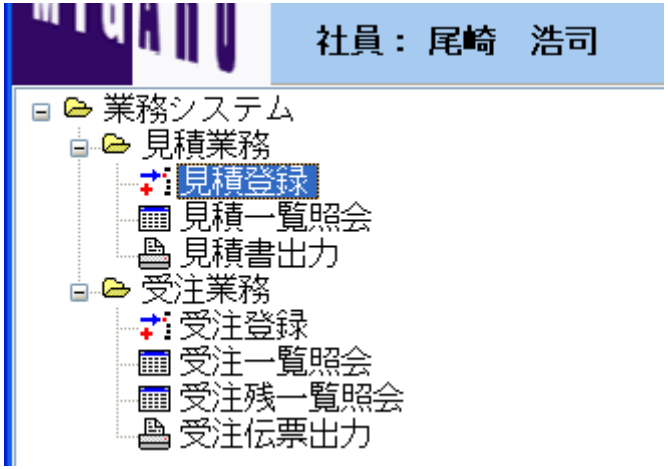
営業担当者の場合

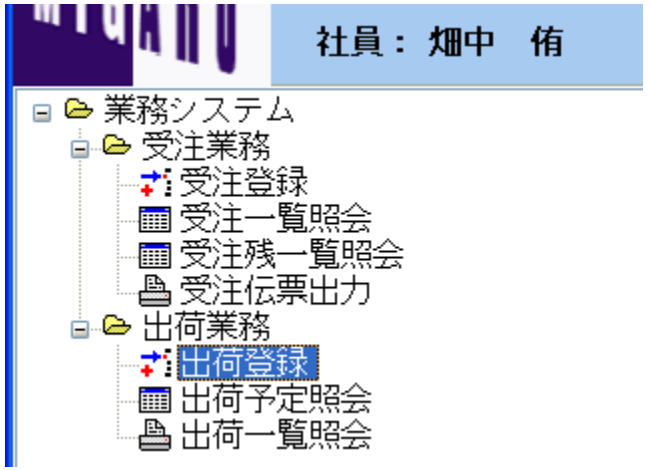
出荷担当者の場合

ボタン形式の場合



ツリー形式の場合

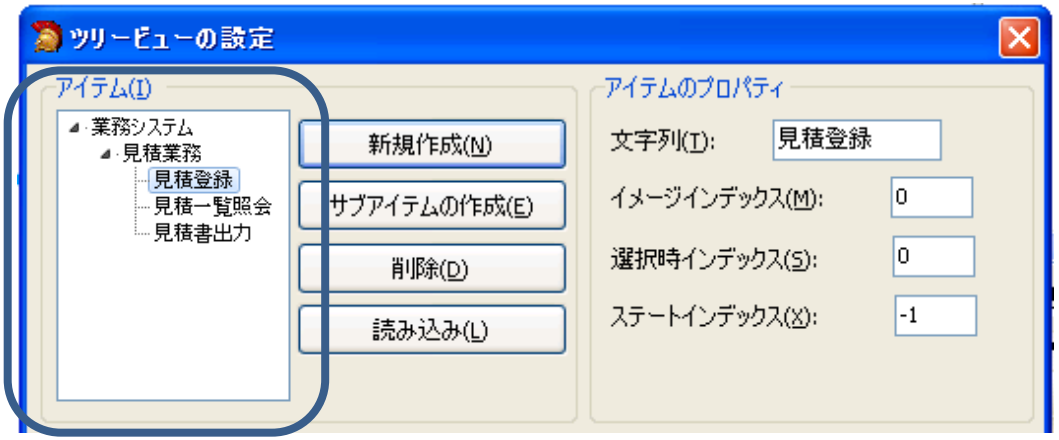




ツリー形式メニューの動的な作成手順

静的に作成するメニュー（P.9でご紹介）

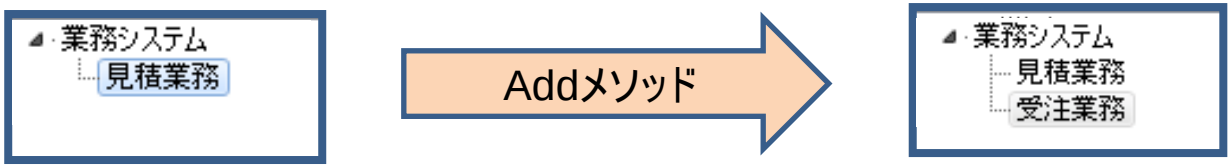
アイテム（ツリー全体）＝
Itemsプロパティ



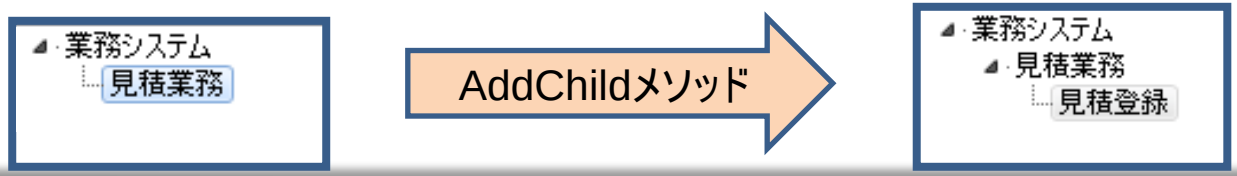
- 「新規作成」より
  ノードを追加、
  「サブアイテムの作成」より
  ノード(子)を追加する

動的に作成するメニュー

- 新規作成　＝　Addメソッドによってノードを追加



- サブアイテムの作成　＝　AddChildメソッドにて選択ノードにノード(子)を追加



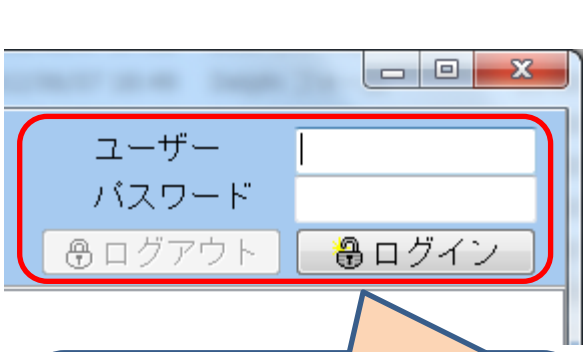
ツリー形式メニューの動的な作成手順

- ログオン時にユーザータイプを取得し、メニュー項目を作成する実装例

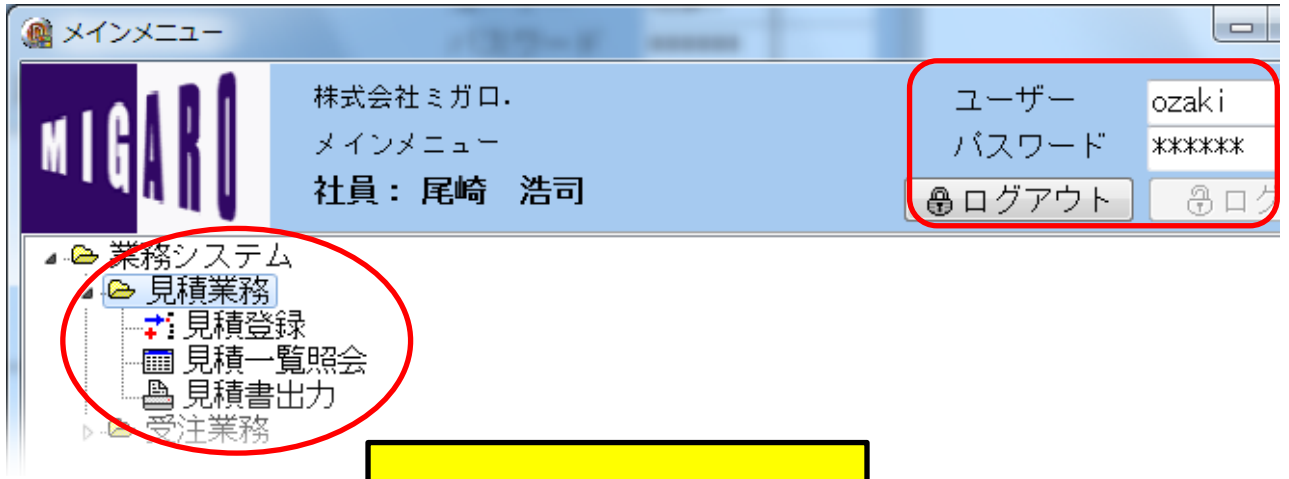
```
procedure TForm1.bbtnLogInClick(Sender: TObject);
var
  tRootNode: TTreeNode;          // 最上位のノード
  tParentNode: TTreeNode;        // メニューカテゴリーのノード
  tChildNode: TTreeNode;         // メニューのノード
Begin
  …
  //担当者マスター存在チェックし、 ユーザータイプを取得
  FUSERTYP := FieldByName('UTUTYP').AsString;
  with tvMenu.Items do
  begin
    // 基本のメニューを設定
    tRootNode := Add(nil, '業務メニュー');
    tRootNode.ImageIndex := 5;
    tRootNode.SelectedIndex := 5;
    //ユーザータイプ＝１（営業）のみ見積業務メニューを追加
    if FUSERTYP = '1' then
    begin
      tParentNode := AddChild(tRootNode, '見積業務');
      tParentNode.ImageIndex := 5;
      tParentNode.SelectedIndex := 5;
      tChildNode := AddChild(tParentNode, '見積登録');
    …
```

【ノード追加ロジック】
条件に応じてノードを追加する、追加しないをコントロールする

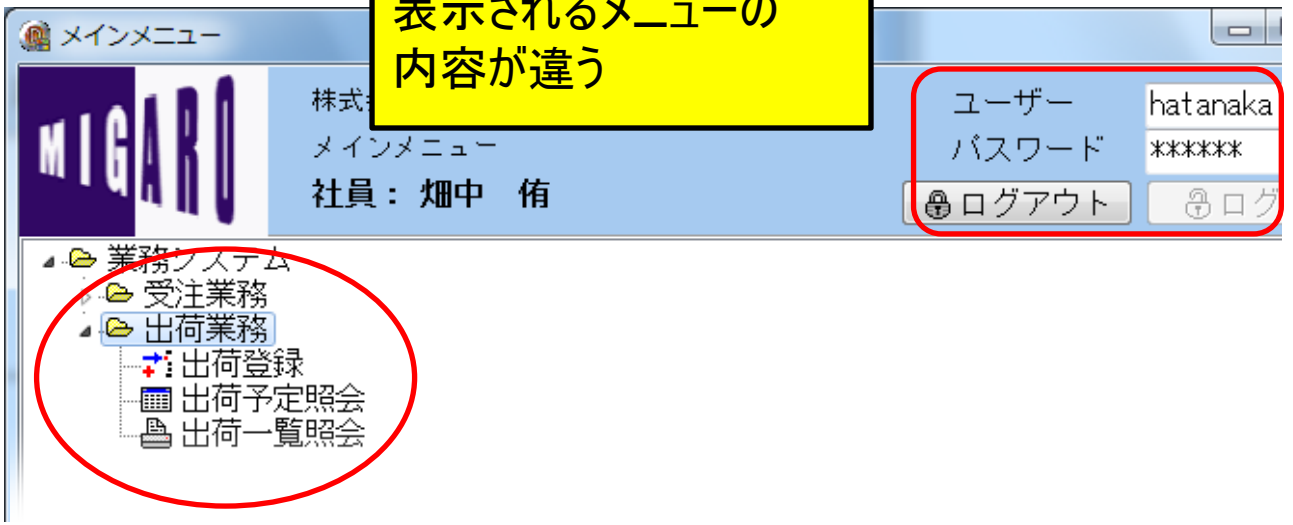
ツリー形式メニューの実行



ログインするユーザータイプによって．．．



表示されるメニューの内容が違う



開発テクニック③
使い勝手を向上させるメニュー

使い勝手を向上させるメニュー

- 使い勝手を向上させるために、
  個人毎にメニューをカスタマイズできないか？

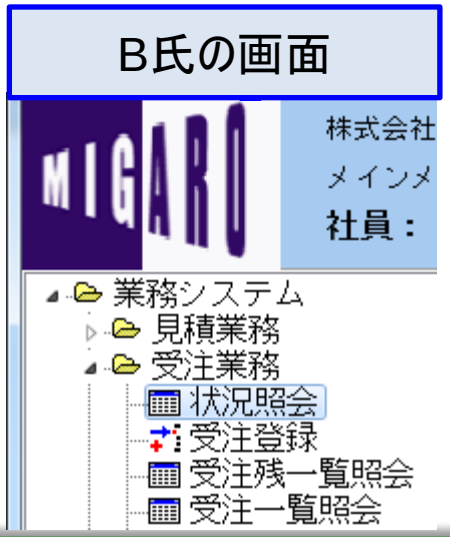
＜営業担当者A氏の要望＞
敏腕営業の私は
受注伝票出力をよく使う。

＜営業部門管理者B氏の要望＞
受注登録なんて、めったに使わない。
状況分析が私の仕事だ。

共通メニュー

A氏の画面

B氏の画面



使い勝手を向上させるメニュー

ユーザー毎のメニューの並び順をCSVファイルで管理する

①画面を起動する

個人別メニューファイル

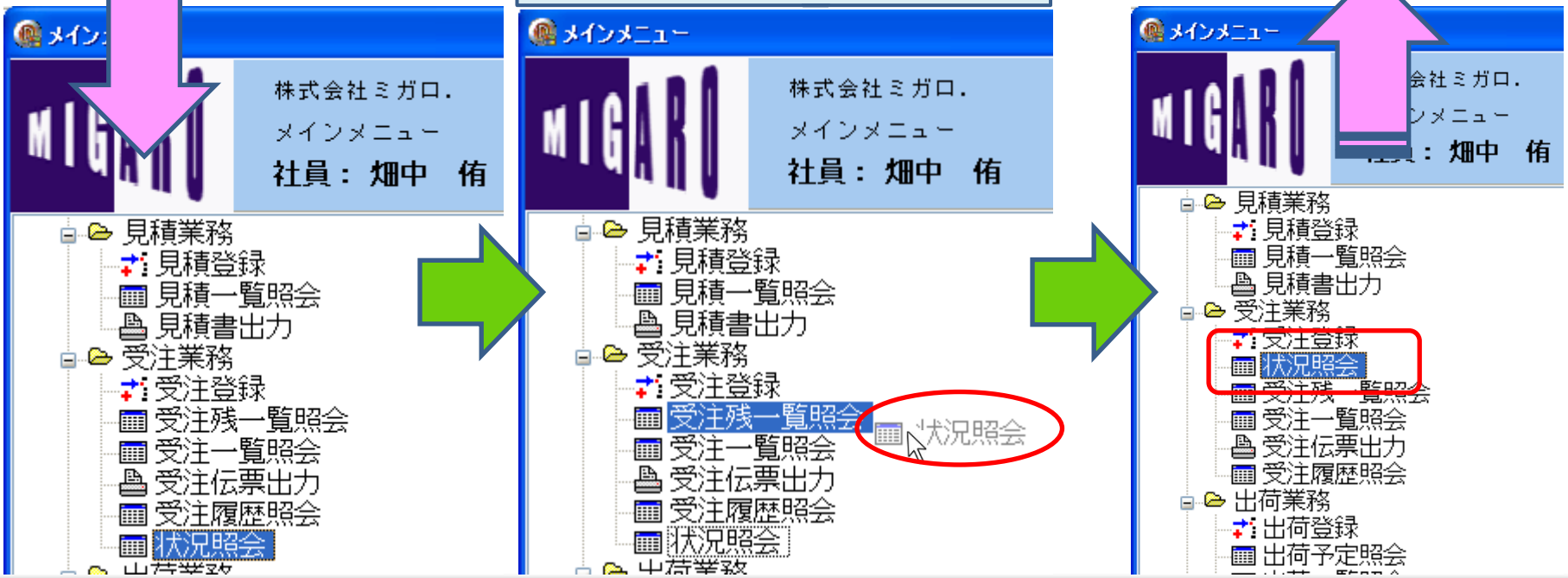
ログインユーザー用のCSVファイルを読み込みメニュー内容を作成する

②並び替える

③画面を終了する

個人別メニューファイル

メニュー内容をCSVファイルに書き出し、保存する



使い勝手を向上させるメニュー

TTreeViewでメニュー項目を並び替える(②)

【1】DragModeプロパティ
・ドラッグを開始できるようにする

【2】OnDragOverイベント
・ドロップ可能か判定する

【3】OnDragDropイベント
・選択したメニュー項目を挿入
・選択したメニュー項目を削除

使い勝手を向上させるメニュー

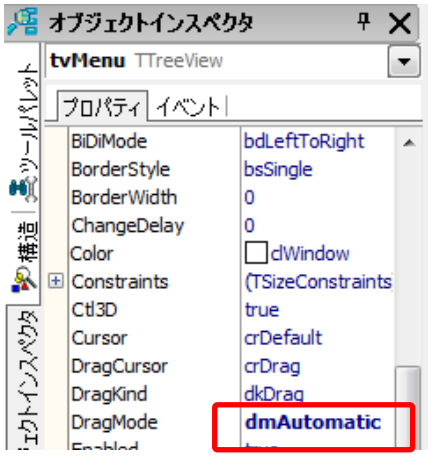
ツリー形式のドラッグ＆ドロップに必要なプロパティ・イベント(②)

OnDragOverイベント

```
procedure TfrmSampleMenu.tvMenuDragOver(Sender, Source: TObject;
  X, Y: Integer; State: TDragState; var Accept: Boolean);
begin
  if (tvMenu.Selected.Text = cMAIN)
     or (tvMenu.Selected.Text = cParent01)
 ...
  then
  begin
    // メニューカテゴリーの場合、ドロップ不可
    Accept := False;
  end
  else
  with tvMenu do
  begin
    // 移動前のノードの位置の場合、ドロップ不可
    Accept := (Source is TTreeView) and (Selected <> GetNodeAt(X, Y));
  end;
end;
```

【2】ドロップ可能判定ロジック
OnDragOverイベント内にて
Accept＝Trueとすることでドロップ可能になる

【1】プロパティ設定
DragMode＝
dmAutomatic



使い勝手を向上させるメニュー

ツリー形式のドラッグ＆ドロップに必要なプロパティ・イベント(②)

OnDragDropイベント

```
procedure TfrmSampleMenu.tvMenuDragDrop(Sender, Source: TObject; X, Y: Integer);
var
  tSelectNode: TTreeNode;       // 選択ノード
  tInsertNode: TTreeNode;       // 追加先ノード
  tAddNode: TTreeNode;
begin
  ....
  with TreeView1 do
  begin
    Items.BeginUpdate;
    try
      // 選択ノードを挿入
      tAddNode := Items.Insert(tInsertNode, tSelectNode.Text);
      tAddNode.Selected := True; // 追加ノードを選択状態にする
      // 選択ノードを削除
      tSelectNode.Delete;
    finally
      Items.EndUpdate;
    end;
  end;
end;
```

Itemsプロパティの設定時は**BeginUpdate～EndUpdate**で描画を止める

【3】ドロップ時のロジック
選択していたノードを挿入してから、移動前のノードを削除することで入れ替えを実現する

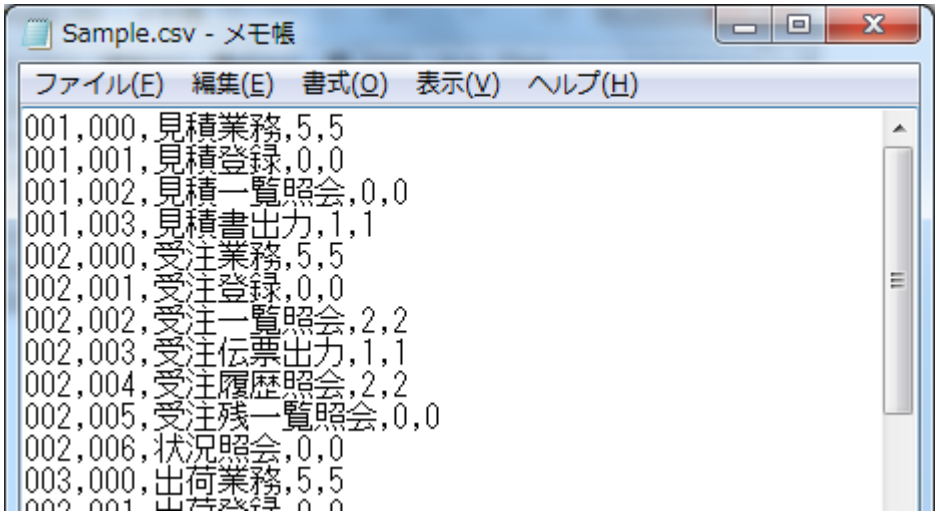
使い勝手を向上させるメニュー

【参考】ツリー形式の内容をCSVファイルより設定する方法

＜CSVの設定例＞

カンマ区切り、改行で1レコードを定義

- 1列目　　:ノード(親)の順序
- 2列目　　:ノード(子)の順序
  （※2行目が”000”であればカテゴリーを表す）
- 3列目　　:機能タイトル
- 4列目　　:アイコンの画像インデックス
- 5列目　　:選択時の画像インデックス



使い勝手を向上させるメニュー

【参考】ツリー形式の内容をCSVファイルより設定する方法(①)

```
var
 …
  sDirPath, sFileName: String; // CSVファイルのファイル情報
  slCSVFile: TStringList;      // CSVファイル読込用
  slItem: TStringList;         // メニュー項目取得用
  iCSVRow: Integer;
begin
  // CSVファイルよりメニュー内容を取得
  slCSVFile := TStringList.Create;
  try
    // CSVファイルのパス情報
    sDirPath  := ExtractFilePath(Application.ExeName);
    sFileName := cCSVFile;
    // CSVファイル設定
    slCSVFile.LoadFromFile(sDirPath + sFileName);
～～～次ページ「メニュー追加処理」として解説～～～
    end;
  finally
    // TStringList開放
    slItem.Free;
    slCSVFile.Free;
  end;
end;
```

TStringListを利用している為LoadFromFileメソッドで、容易にCSVファイルを文字列情報として取り込める

使い勝手を向上させるメニュー

【参考】ツリー形式の内容をCSVファイルより設定する方法(①)

```
{メニュー追加処理}
      // CSVファイルを最後まで読み込む
      for iCSVRow := 0 to slCSVFile.Count -1 do
      begin
        // メニュー項目取得用TStringListをクリア
        slItem.Clear;
        // 1行読み込み
        slItem.CommaText := slCSVFile.Strings[iCSVRow];
        // メニューID2＝'000'の場合、メニューカテゴリーとする
        if slItem[cID2] = '000' then
        begin
          // 追加：AddChildメソッド（P.22でご紹介）
          tParentNode := AddChild(tRootNode, slItem[cPGNM]);
          tParentNode.ImageIndex := StrToIntDef(slItem[cIIDX], -1);
…    …
        end;
```

CSVファイルの行数分のループ処理

該当行をカンマ区切りとして保持する

使い勝手を向上させるメニュー

【参考】ツリー形式の内容をCSVファイルに保存する方法(③)

```
var
  sDirPath, sFileName: String; // CSVファイルの保存先ファイル情報
  slCSVFile: TStringList;      // CSVファイル出力用
  sID1, sID2: String;          // CSVファイル列用変数
  i, k: Integer;
begin
 …   …
  try
〜〜〜次ページ「メニュー内容取得処理」として解説〜〜〜
    // CSVファイルの保存先ファイル情報
    sDirPath := ExtractFilePath(Application.ExeName);
    sFileName := cCSVFile;
    // CSVファイル出力
    if slCSVFile.Text <> '' then
      slCSVFile.SaveToFile(sDirPath + sFileName);
  finally
    // TStringList開放
    slCSVFile.Free;
  end;
end;
```

TStringListを利用している為SaveToFileメソッドで、CSVファイルとして保存できる

使い勝手を向上させるメニュー

【参考】ツリー形式の内容をCSVファイルに保存する方法(③)

```
{メニュー内容取得処理}
  with tvMenu do
  begin
    // CSVファイルにメニュー内容を出力
    for i := 0 to Items[0].Count - 1 do
    begin
      // メニューカテゴリーの場合
      sID1 := FormatCurr('000', i + 1);
      sID2 := '000';
      slCSVFile.Add(sID1 + ',' + sID2
                    + ',' + Items[0][i].Text
                    + ',' + IntToStr(Items[0][0].ImageIndex));
      // メニューカテゴリーに紐付く、ノードの情報
      for k := 0 to Items[0][i].Count - 1 do
      begin
        sID2 := FormatCurr('000', k + 1);
        slCSVFile.Add(sID1 + ',' + sID2
                      + ',' + Items[0][i][k].Text
                      + ',' + IntToStr(Items[0][i][k].ImageIndex));
      end;
    end;
  end;
```

ツリー内容のノード(親)数分のループ処理

ツリー内容のノード(親)に紐付くノード(子)数分のループ処理

アプリケーション開発スタイルに応じた『メニュー』開発

アプリケーション開発スタイルに応じた『メニュー』開発

アプリケーション開発スタイルの種類

① 全ての機能を1つのEXEにする

A） 処理毎に起動するプログラムを指定

② メニューと各機能を分割する

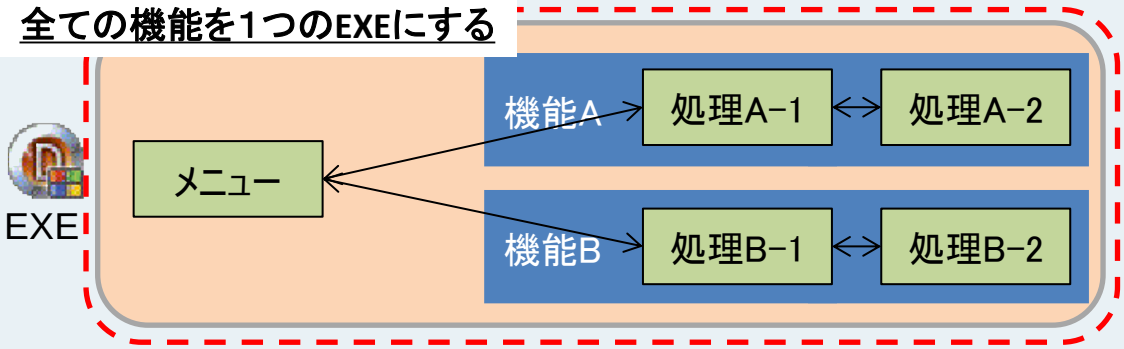
B） 各機能をEXEにし、機能毎にEXEを呼び出す

C） 各機能をDLL化し、機能毎にDLLを呼び出す

各開発スタイルの構成イメージ

・・・プロセス
・・・プログラム

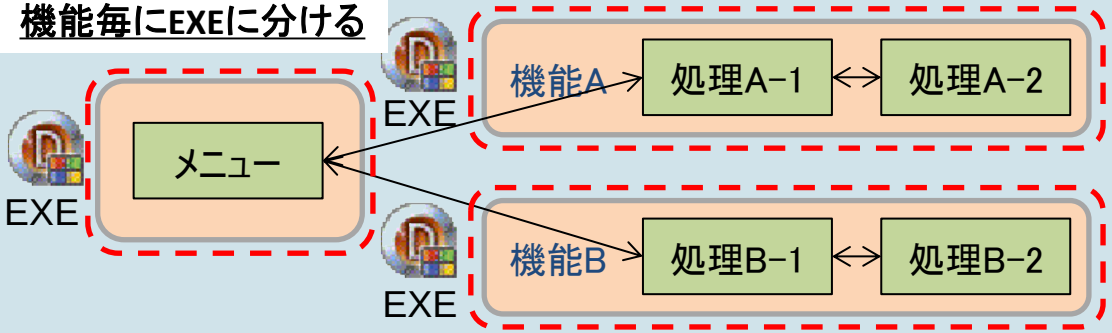
(A) 全ての機能を1つのEXEにする



○同一プロセス、同一セッションで全ての機能を処理できる。

△軽微な修正や機能追加でも、関連しない全ての機能を含めてEXEの再作成が必要になる。

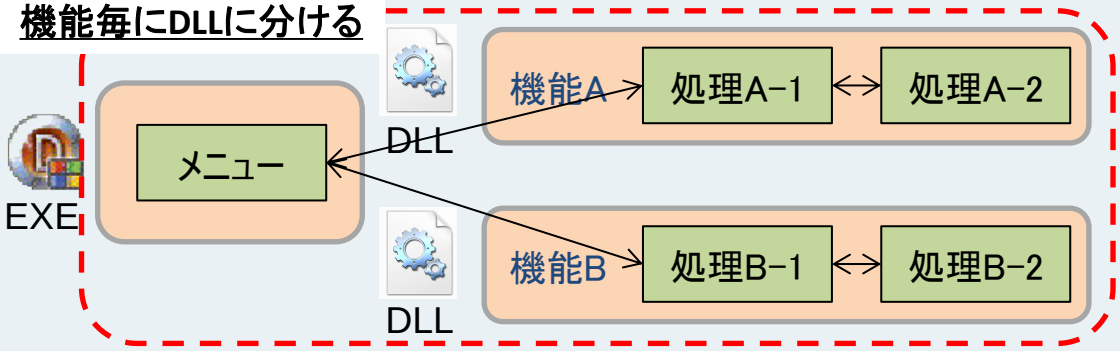
(B) 機能毎にEXEに分ける



△各機能ごとプロセス及びセッションが分かれてしまう。

○軽微な修正や機能追加に対して、関連する機能のみEXEの再作成をすればよい。

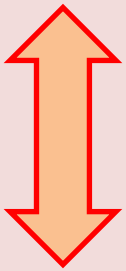
(C) 機能毎にDLLに分ける



○同一プロセス、同一セッションで全ての機能を処理できる。

○軽微な修正や機能追加に対して、関連する機能のみDLLの再作成をすればよい。

各開発スタイルの特徴のまとめ

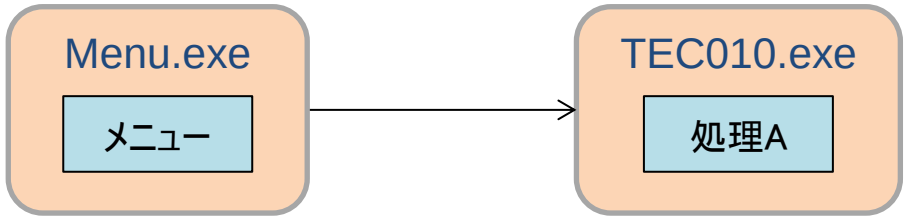
項目	開発難易度	メリット	デメリット
(A)全ての機能を1つのEXEにする	易しい	・同一EXEに全ての機能が含まれるため、画面展開等の開発がしやすい ・同一プロセス・同一セッションで全ての機能を処理できる	・軽微な修正や機能追加でも、全ての機能を含むEXEの再作成が必要になる ・機能が増えてくると、EXEのサイズが増える
(B)機能毎にEXEに分ける		・軽微な修正や機能追加には、関連する機能のみEXEを再作成すれば良い ・機能が増えても、個別のEXEサイズは抑えられる （EXE全体のサイズは増えてしまう）	・画面展開時に、ログイン情報等を実行時引数で渡す等の考慮が必要になる ・プロセスが分かれるため、セッションが分かれてしまう
(C)機能毎にDLLに分ける	難しい	・同一プロセス・同一セッションで全ての機能を処理できる ・軽微な修正や機能追加には、関連する機能のみDLLを再作成すれば良い ・機能が増えても、個別のDLLサイズは抑えられる （DLL全体のサイズは増えてしまう）	・DLL独自の開発手法の考慮が必要になる →開発時のデバッグ方法がEXEとは異なる →EXEと異なり、DLL単独での実行ができない

【参考】メニュー画面からEXEを呼び出す方法

EXE間の情報受け渡しをどうするか？

実行時引数を使用する方法

例）ユーザーID＝”TESTUSER”を受け渡す → **“C:¥Projects¥TEC010.exe” TESTUSER**



【呼出元】

ボタン押下時に、edtUserの入力値を「TEC010.exe」に実行時引数として呼び出す場合

```
procedure TfrmSampleMenu.btnCallExeClick(Sender: TObject);
begin
  // 実行ファイルとして機能分割されたプログラムを呼び出します。
  ShellExecute(Handle,                                          // 自身のハンドル
               'open',                                          // 関連付けで開く
               PChar('TEC010.exe'),                             // 実行ファイル名
               PChar(edtUser.Text),                             // 実行時引数(ユーザID)
               PChar(ExtractFileDir(Application.ExeName)),      // 作業ディレクトリ
               SW_SHOWNORMAL);                                  // 位置・サイズ
end;
```

※複数のパラメータを付与する場合は、空白で区切る

【参考】メニュー画面からEXEを呼び出す方法

EXE間の情報受け渡しをどうするか？

実行時引数を使用する方法

例）ユーザーID＝”TESTUSER”を受け渡す → **“C:¥Projects¥TEC010.exe” TESTUSER**

【呼出先】

画面起動時に、実行時引数をstUSERにセットする場合

```
procedure TfrmDividedSampleExe.FormCreate(Sender: TObject);
begin
  // ユーザＩＤをクリアする
  stUSER.Caption := '';

  // 実行時引数が１つ以上あるか
  if ParamCount >= 1 then
  begin
    // １つ目の実行時引数をユーザＩＤとして表示します。
    stUSER.Caption := ParamStr(1);
  end;
end;
```

ParamCount ： 実行時引数がセットされた数

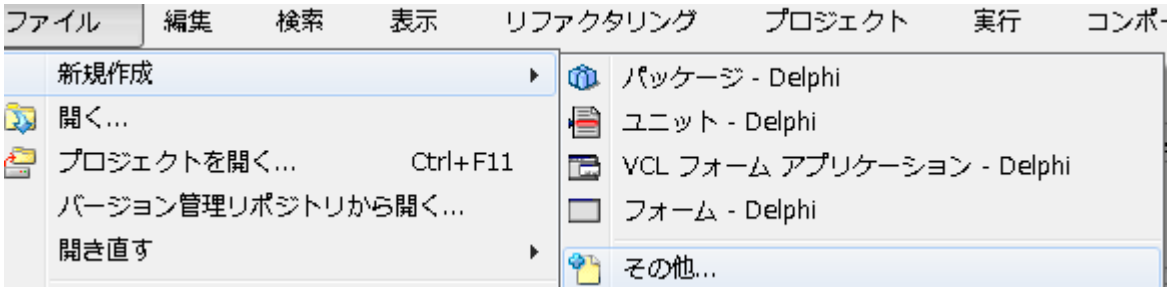
ParamStr ： 実行時引数にセットされた文字列を取得

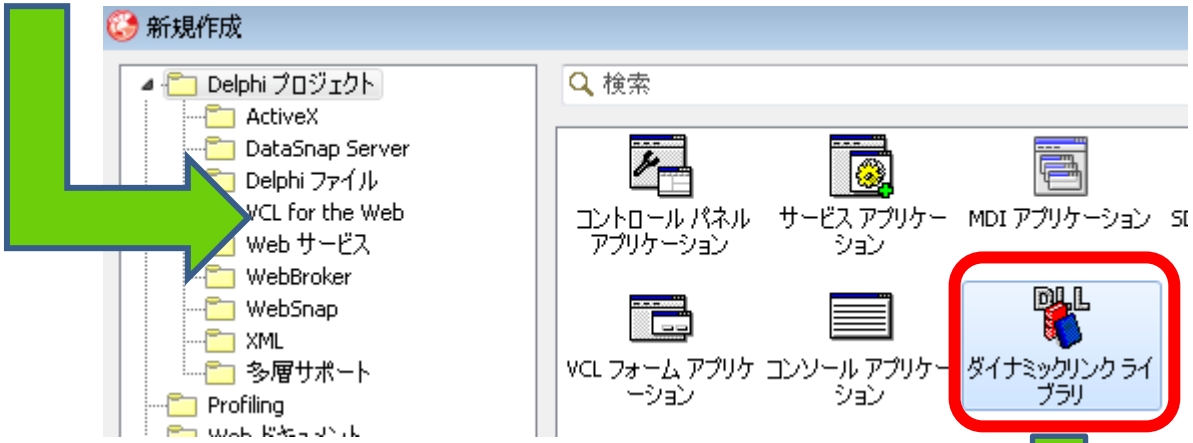
※複数のパラメータを受け取る場合は、ParamStr(2)、ParamStr(3)、・・・・とする

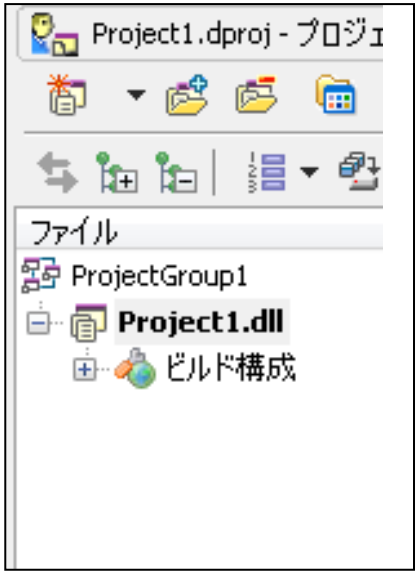
【参考】メニュー画面からDLLを呼び出す方法

DLL プロジェクトの作成方法

［ファイル］→［新規作成］→［その他］→［ダイナミックリンクライブラリ］を選択







【参考】メニュー画面からDLLを呼び出す方法

【呼出先】DLL側にあらかじめ呼び出すための手続き・関数を準備

```
library DividedSampleDLL;

uses
  SysUtils,
  Classes,
  FDividedSampleDLL in 'FDividedSampleDLL.pas' {frmDividedSampleDLL};

exports
  ShowForm;

...
```

↓フォームユニットのソース

```
var
  frmDividedSampleDLL: TfrmDividedSampleDLL;

procedure ShowForm(pcUSER: PChar); stdcall;

implementation

{$R *.dfm}

procedure ShowForm(pcUSER: PChar);
begin
  // 機能分割サンプルフォームを作成・表示します。
  frmDividedSampleDLL := TfrmDividedSampleDLL.Create(nil);
  frmDividedSampleDLL.edtUSER.Text := pcUSER;
  frmDividedSampleDLL.ShowModal;
  frmDividedSampleDLL.Free;
end;
```

↑プロジェクトソース

外部から呼び出したい関数、手続きはexportsの下に全て宣言する

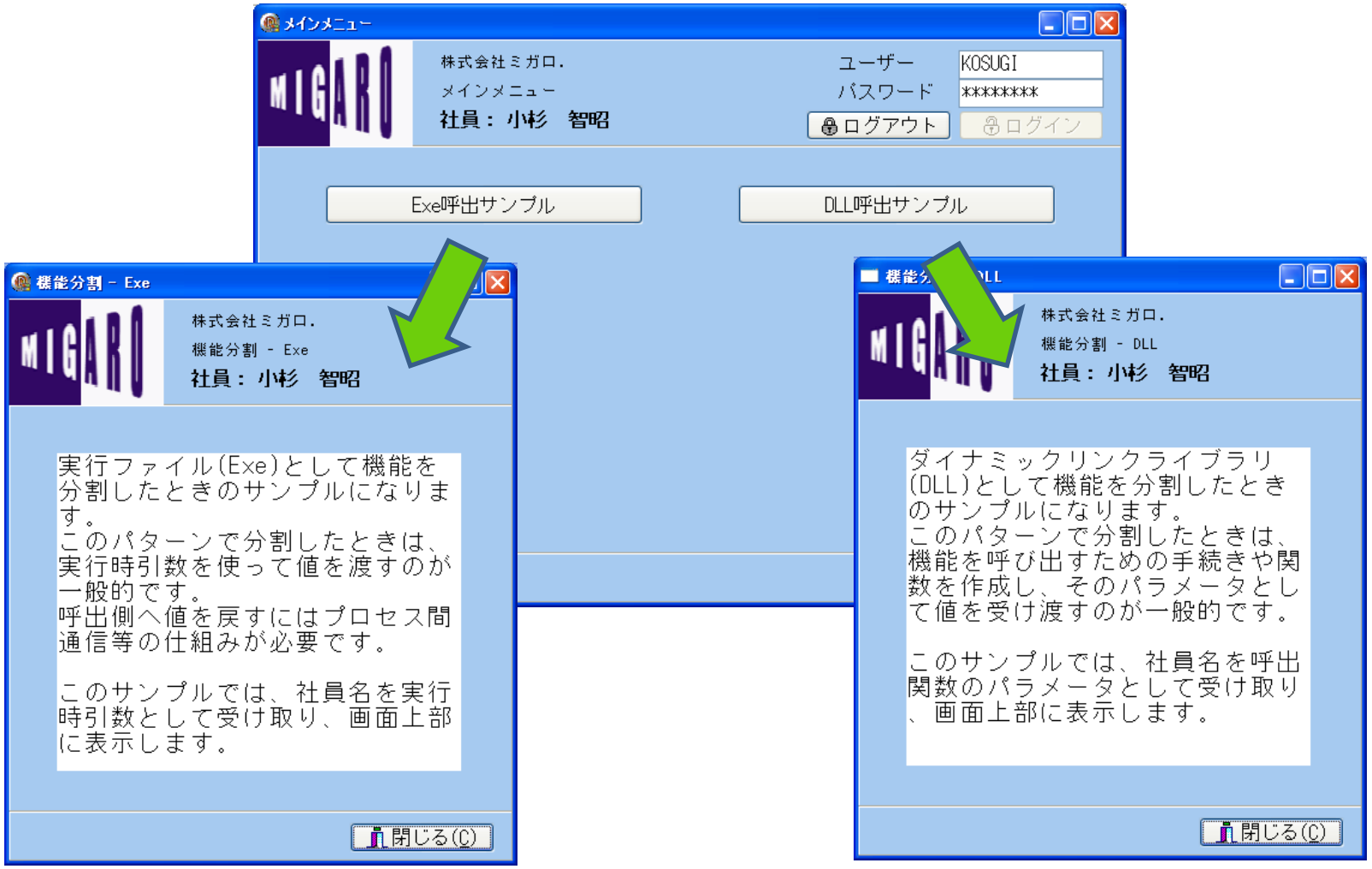
【参考】メニュー画面からDLLを呼び出す方法

【呼出元】DLLの手続き・関数を実行時に取得・実行

```
procedure TfrmSampleMenu.btnCallDLLClick(Sender: TObject);
var
  DLLHandle: THandle;
  DLLFnc:    procedure (pcUSER: PChar); stdcall;
begin
  // DLLを動的にロードする
  DLLHandle := LoadLibrary('DividedSampleDLL.dll');
  // ロード失敗時は処理を行わない
  if DLLHandle = 0 then Exit;

  // DLLのShowForm手続きのアドレスを取得する
  DLLFnc := GetProcAddress(DLLHandle, 'ShowForm');
  // 手続きのアドレスが取得できたら手続きを呼び出す
  if Assigned(DLLFnc) then DLLFnc(PChar(edtUSER.Text));

  // 使い終わったDLLをアンロードする
  FreeLibrary(DLLHandle);
end;
```

【参考】メニュー画面からEXEやDLLを呼び出す

実行イメージ



【参考】メニュー画面からDLLを呼び出す方法

- DLLプログラムの作り方・呼び出し方の詳しい説明は、2011年の第9回テクニカルセミナー「知って得する！現役ヘルプデスクが答えるDelphiテクニカルエッセンス 9.0」をご参照ください。

ミガロ. ホームページ「メンテナンス専用」ページより、資料がダウンロードができます。
URL＝http://www.migaro.co.jp/

まとめ

まとめ

- 『メニュー』の種類と基本的な作成方法
  - ボタン配置形式の作成手順
  - ツリー形式の作成手順
- 開発テクニックのご紹介
  - TActionを使用した権限制御
  - ツリー形式での動的なメニュー作成手順
  - ツリー形式でのカスタマイズ方法
- アプリケーション開発スタイルに応じた『メニュー』開発
  - アプリケーション開発スタイル毎の特徴
  - 機能毎のEXEを作成し、メニュー画面からEXEを呼び出す方法
  - 機能毎のDLLを作成し、メニュー画面からDLLを呼び出す方法

ご清聴ありがとうございました。