Studis 3 : Sourcecode

Version 28.3.1999

Inhalt:

```
//---------------------------------------------------------------------
// Studis3.dpr
//---------------------------------------------------------------------
```

program Studis3;

```
uses
  Forms,
  Main in 'Main.pas'                        {MainForm},
  StudiDataMod in 'StudiDataMod.pas'        {SDM: TDataModule},
  Personen in 'Personen.pas'                {PersDataForm},
  StudiDialog in 'StudiDialog.pas'             {StudiDlg},
  Taten in 'Taten.pas',
  Reglement in 'Reglement.pas'              {RegleForm},
  AbschlussForm in 'AbschlussForm.pas'      {AbschForm},
  MigrationDataMod in 'MigrationDataMod.pas' {MigDM: TDataModule},
  PropertiesDlg in 'PropertiesDlg.pas'      {PropDlg},
  Leistung in 'Leistung.pas',
  NeuDialog in 'NeuDialog.pas'              {NeuDlg},
  Node in 'Node.pas',
  ImpMainForm in 'ImpMainForm.pas'             {ImpMainForm},
  ProgrDlg in 'ProgrDlg.pas'                {ProgressDlg},
  RawImpThr in 'RawImpThr.pas',
  ProcessThr in 'ProcessThr.pas',
  Tokenizers in 'Tokenizers.pas',
  TatenImpThr in 'TatenImpThr.pas',
  Migration in 'Migration.pas'              {MigrationForm},
  Auswertung in 'Auswertung.pas'            {AuswertungsForm},
  SearchDlg in 'SearchDlg.pas'              {SuchDlg},
  Abschluss in 'Abschluss.pas',
  AuswahlDialog in 'AuswahlDialog.pas'      {AuswahlDlg},
  Studi in 'Studi.pas';

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TMainForm, MainForm);
  Application.CreateForm(TSDM, SDM);
  Application.CreateForm(TMigDM, MigDM);
  Application.CreateForm(TNeuDlg, NeuDlg);
  Application.CreateForm(TProgressDlg, ProgressDlg);
  Application.CreateForm(TAuswertungsForm, AuswertungsForm);
  Application.CreateForm(TSuchDlg, SuchDlg);
  Application.CreateForm(TPropDlg, PropDlg);
  Application.CreateForm(TAuswahlDlg, AuswahlDlg);
  Application.Run;
end.
//---------------------------------------------------------------------
```

```
//---------------------------------------------------------------------
// Studis 3 : Abschluss.pas
// einzelner Abschluss in Reglement
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit Abschluss;

```
interface

uses
  Node, StudiDataMod, MigrationDataMod,
  SysUtils, Classes, DBTables, Dialogs;

type
//---------------------------------------------------------------------
  TAbschluss = class( TObject )
  public
    RegId, AbschId, RootId: Integer;
    AbschName: String;
    StrukturL: TList;
    AbschModified: Boolean;

    constructor CreateDefault;
    constructor Make( RId, AId: Integer );
    procedure LoadStrukturList;        // DB -> StrukturL
    procedure SaveStrukturList;        // StrukturL -> DB
    procedure MigrateFromCALC_DB;
    function GetNodeFromStruktList( NId: Integer ) : TNode;
  private
    function LoadRootId: Integer;
    function LoadName: String;
    function GetLeistNodeId( LId: Integer ) : Integer;
    function GetALTNodeId( Aeq: Integer; Art: Char ) : Integer;
    function GetSameNodeId( TeilLeist, Leist: Integer ) : Integer;
    procedure AddCurrentRecAsNode( var NId: Integer );
  end;


implementation

uses AbschlussForm;

//---------------------------------------------------------------------
procedure TAbschluss.MigrateFromCALC_DB;
var
  Node, AltNode, Par, Chi: TNode;
  ToLoad: TList;
  id, strukId, k, aeq, parent, i, AltId : Integer;
  p: ^Integer;
  fertig: Boolean;
  Current, Further: Integer;
  msg, S: String;
// ---------------------------------
  function GetNewID : Integer;
  // create new (unused) NodeId ( StruktTable must be open )
  begin
    SDM.StruktTable.Last;
    if SDM.StruktTable.RecordCount > 0
      // last node has biggest id
      then Result := SDM.StruktTable['NodeId'] + 1
      else Result := 0;
  end;
// ---------------------------------
  procedure SetQueryToThisAbschluss;
  begin
    // alle Recs aus Calc.db, die zu AbschId & RegId gehören:
    with MigDM.Query2 do begin
      Close;
      SQL.Clear;
      SQL.Add( 'select * from ''' + CALC_DB
```

```
          + ''' where ReglementsNr ='   + IntToStr( RegId )
          +    ' and AbschlussNr ='     + IntToStr( AbschId )
          +   ' order by Stelle ' );
             // ^ former 'Position'
      Filtered := false;
      Open;
    end;
  end;
// ---------------------------------
begin
  StrukturL.Clear;
  with SDM do with MigDM do begin       //  CalcTable must be open!
    if TryOpentable( StruktTable ) then begin

      SetQueryToThisAbschluss;

      // insert root-node to tree ( TeilLeistNr = AbschNr )
      if Query2.Locate( 'TeilLeistungsNr', AbschId,
                        [loPartialKey] ) then begin
        // new rootId, parent = LEER
        id := GetNewID;
        RootId := id;   // to be saved in AbschTable!
        //---  AddRecAsNode( Query2, id, LEER );
        AddCurrentRecAsNode( id );
        // Liste der als Leistungen zu checkenden TeilLeistungen
        ToLoad := TList.Create;
        ToLoad.Add( @AbschId );

        // restl. Nodes laden
        while ToLoad.Count > 0 do begin
          // zu ladende LeistId holen & entfernen
          Current := Integer( ToLoad.First^ );
          ToLoad.Remove( ToLoad.First );

          // only rec's hanging @ current node (excl. root)
          Query2.Filter :=
                    '(LeistungsNr = ' + IntToStr(Current) + ') '
              + 'and (TeilLeistungsNr <> ' + IntToStr(Current) + ')';
          Query2.Filtered := true;
          Query2.First;

          while not Query2.Eof do begin
            AddCurrentRecAsNode( id );
            // zu ladende potentielle Child-LeistId nach ToLoad
            new( p );
            p^ := Query2['TeilLeistungsNr'];
            ToLoad.Add( p );
            Query2.Next;
          end;
        end;  // while ToLoad not empty
        ToLoad.Free;

        // clear tmp aequiv.nr.
        for i := 0 to StrukturL.Count-1 do begin
          Node := TNode( StrukturL.Items[i] );
          Node.Sibling := LEER;  // wird unten angepasst
          Node.Child := LEER;    // wird unten angepasst
        end;

        // hier ist StrukturL komplett inkl. Parent-Referenzen.

        // Child-, Sibling-Referenz anpassen
        for k := 0 to StrukturL.Count-1 do begin
          Node := TNode( StrukturL.Items[k] );
          if Node.Parent > LEER then begin  // leave out the root
            //---
            Par := GetNodeFromStruktList( Node.Parent );
            if Par.Child = LEER then begin
              // current Node is first child
              Par.Child := Node.NodeId;
            end
            else begin
              // get first Child of Parent
              Chi :=  GetNodeFromStruktList( Par.Child );
              // pass existing Siblings
```

```
              while Chi.Sibling > LEER do
                Chi := GetNodeFromStruktList( Chi.Sibling );
              // set current as the last's Sibling
              Chi.Sibling := Node.NodeId;
            end;
            //---
          end;
        end;

        // Struktur speichern
        for k := 0 to StrukturL.Count-1 do begin
          Node := TNode( StrukturL.Items[k] );
          Node.Save;
        end;
        AbschModified := False;

      end
      else ShowMessage('no root found');
    end
    else ShowMessage('kann Struktur.db nicht öffnen!?!?!?!') ///
  end; // with ...
end;

//---------------------------------------------------------------------
procedure TAbschluss.AddCurrentRecAsNode(var NId: Integer );
var
  ParNodeId, ALTNodeId, ExistingNodeId: Integer;
  Node, ALTNode: TNode;
  n, t     : Integer;
  wN, wT   : Boolean;
  art      : Char;
  LAusw1, LAusw2, LA1F, LA2F : Boolean;
  ZeitLimT, ZeitLimN, ErfordN, RundenA, Gewicht: Integer;

begin
  with MigDM do begin
    // collect attributes ...

    //    " not ( ... = NULL )" !!!
    if not ( Query2['AnzahlNoten'] = NULL )
      then n := Query2['AnzahlNoten']
      else n := LEER;
    if not ( Query2['AnzahlTestate'] = NULL )
      then t := Query2['AnzahlTestate']
      else t := LEER;
    if Query2['TeilLeistungsArt'] = 'N'     // zum checken
      then art := 'N'        // ***  eher TLNAnforderung ?! ***
      else art := 'T';
    if Query2['Wahl'] = 'W'         // alle ausser root sind def.
      then if art = 'N' then wN := true
                        else wT := true;
    if Query2['Wahl'] = 'O'
      then if art = 'N' then wN := false
                        else wT := false;
    // .. restl. Attr. !! ********************

    if not ( Query2['LeistungsAusweis1'] = NULL )
      then LAusw1 := Query2['LeistungsAusweis1'];
    if not ( Query2['LeistungsAusweis2'] = NULL )
      then LAusw2 := Query2['LeistungsAusweis2'];

    if not ( Query2['LA1Vollendet'] = NULL )
      then LA1F := Query2['LA1Vollendet'];
    if not ( Query2['LA1Vollendet'] = NULL )
      then LA2F := Query2['LA2Vollendet'];

    if Query2['ZeitLimite'] >= 0 then
      if art = 'T'
        then ZeitLimT := Query2['ZeitLimite']
        else ZeitLimN := Query2['ZeitLimite'];
    // -> besser unten unterscheiden !
    {
    else
      if art = 'T'                // wenn TL = L; immer 99
        then ZeitLimT := 99
```

```
        else ZeitLimN := 99
      }

      if Query2['TLNAnforderung'] >= 0
        then ErfordN := Query2['TLNAnforderung']
      else
        if Query2['LNAnforderung'] >= 0
          then ErfordN := Query2['LNAnforderung'];
        // (else Absch 80000 oder 80400)

      if Query2['Rundungsgenauigkeit'] >= 0   // alle not-leafes
        then RundenA := Query2['Rundungsgenauigkeit'];
      if Query2['Gewichtung'] >= 0            // alle not-roots
        then Gewicht := Query2['Gewichtung'];
      // else's ?!
  {
      if NId <> RootId then begin
        LAusw1 := Query2['LA1'];     // else LA1 is null ... (gdw)
        LAusw2 := Query2['LA2'];
        LA1F := False;
        LA2F := False;
      end
      else begin
        LAusw1 := False;             // ?!
        LAusw2 := False;
      end;
  }

      // it's the root
      if Query2['TeilLeistungsNr'] = Query2['LeistungsNr'] then begin
        Node := TNode.Create( NId, LEER, LEER, LEER,
                              Query2['TeilLeistungsNr'],
                              n, t, wN, wT, '',
                              LAusw1, LAusw2, LA1F, LA2F,
                              99, 99,
                              ErfordN, RundenA, Gewicht );
        StrukturL.Add( Node );       // collect it
        inc( NId );
      end
      else begin
        // check if same teilleistung already in tree at same position
        ExistingNodeId := GetSameNodeId( Query2['TeilLeistungsNr'],
                                Query2['LeistungsNr'] );
        if ExistingNodeId > -1 then begin    // * set attr.
          Node := GetNodeFromStruktList( ExistingNodeId );
          if art = 'N' then begin
            Node.WahlNote := wN;
            Node.AnzNoten := n;
            Node.AnzTestate := t;
          end
          else begin
            Node.WahlTestat := wT;
          end;   // ... restl. Attr. !!
        end

        else begin                           // * make new node
          // search par.-node in StruktL -> parent-node-id's are unique
          ParNodeId := GetLeistNodeId( Query2['LeistungsNr'] );
          if ParNodeId = -1 then begin
            // report error !
            ShowMessage( 'AddCurrentRecAsNode():no parent found' );
          end
          else begin   // parent found, append new node to it
            // check for 'Aequivalenz'
            if Query2['Aequivalenz'] > 0 then begin
              // must be an ALT-node betwwen par & current node
              ALTNodeId := GetALTNodeId( Query2['Aequivalenz'], art );

              if ALTNodeId = -1 then begin   // not found -> create one
                ALTNode := TNode.Create( NId,
                              ParNodeId, LEER, Query2['Aequivalenz'],
                              // remember 1st Aeq'nr in Child-Field
                              ALTERNATIVE, n, t,
                              wN, wT, '',
                              false, false, false, false,
```

```
                          99, 99, 0.0, 0.0, 0 );

              StrukturL.Add( ALTNode );       // collect it
              ALTNodeId := NId;               // for node-parent-ref.
              inc( NId );                     // for next new node
            end
            else begin
              ALTNode := GetNodeFromStruktList( ALTNodeId );
            end;
            ParNodeId := AltNodeId;  // new node's parent
          end;  // Aequiv.

          // create new node
          Node := TNode.Create( NId, ParNodeId, LEER, LEER,
                          Query2['TeilleistungsNr'],
                          n, t, wN, wT, '',
                          LAusw1, LAusw2, LA1F, LA2F,
                          ZeitLimT, ZeitLimN,
                          ErfordN, RundenA, Gewicht );

          StrukturL.Add( Node );
          inc( NId );
        end;
      end;

    end;  // not root
  end;  // with MigDM ...
end;


//---------------------------------------------------------------------
function TAbschluss.GetNodeFromStruktList( NId: Integer ) : TNode;
// get Node from StrukturL with NodeId 'NId'
var
  i: Integer;
  n: TNode;
begin
  n := TNode.CreateDefault;
  n.Error := true;
  Result := n;     // default
  for i := 0 to StrukturL.Count-1 do
    if TNode( StrukturL.Items[i] ).NodeId = NId then
      Result := TNode( StrukturL.Items[i] );
end;

//---------------------------------------------------------------------
function TAbschluss.GetSameNodeId( TeilLeist, Leist: Integer )
                                                          : Integer;
var Id, ParId: Integer; Node, ParNode: TNode;
begin
  Result := -1;
  Id := GetLeistNodeId( TeilLeist );              // node itself
  if Id > -1 then begin                           // found something
    Node := GetNodeFromStruktList( Id );

    if Node.Parent > LEER then begin
      ParNode := GetNodeFromStruktList( Node.Parent );

      if ParNode.LeistId = ALTERNATIVE then       // skip ALT-nodes
        ParNode := GetNodeFromStruktList( ParNode.Parent );
      if ParNode.LeistId = Leist then begin       // really the same!
        Result := Id;
{
        ShowMessage( 'The SAME, ParId: ' + IntToStr( ParId ) + ' >> '
                + IntToStr( TeilLeist )  + ' @ ' + IntToStr( Id )  );
        // it never happens |-(
        ShowMessage( 'The SAME: ' + IntToStr( TeilLeist )
                + ' @ ' + IntToStr( Id )  );
}
      end;
    end;
  end;
end;

//---------------------------------------------------------------------
```

```
function TAbschluss.GetLeistNodeId( LId: Integer ) : Integer;
// NodeId der Node in StrukturL mit LeistId 'LId'
// ( entspricht TeilLeistungsNr in CALC.db )

// findet nur die LETZTE entspr. node !!!! -> nur für unique nodes
var i: Integer;
begin
  Result := -1;
  for i := 0 to StrukturL.Count-1 do
    if TNode( StrukturL.Items[i] ).LeistId = LId then
      Result := TNode( StrukturL.Items[i] ).NodeId;
end;

//---------------------------------------------------------------------
function TAbschluss.GetALTNodeId( Aeq: Integer; Art: Char ): Integer;
// NodeId der ALTERNATIVE-Node in StrukturL mit 'Aeq' im Child-Feld
var
  i, j: Integer;
  Node, ChiNode: TNode;
begin
  Result := -1;
  for i := 0 to StrukturL.Count-1 do begin
    Node := TNode( StrukturL.Items[i] );
    if Node.LeistId = ALTERNATIVE then begin

      // gleiche Aequ.Nr oder (Aeq - 1), wenn ...
      if ( Node.Child = Aeq ) or
       ( ( Node.Child = Aeq - 1 ) and
       ( ( (Art = 'T') and (Node.NodeText = 'N') ) or
         ( (Art = 'N') and (Node.NodeText = 'T') ) ) )
      then Result := Node.NodeId;
    end;
  end;
end;

//---------------------------------------------------------------------
procedure TAbschluss.LoadStrukturList;    // DB -> StrukturL
var
  NextId: Integer;
  ToLoad: TList;
  TmpNode: TNode;
  s: String;
    //-------------------------------------
    procedure _ShowToLoad;
    var k:Integer; m: String;
    begin  m := 'ToLoad: ';
           for k := 0 to ToLoad.Count-1 do
             m := m + IntToStr(Integer( ToLoad.Items[k]^ )) + ' , ';
           ShowMessage( m );
    end;
    //-------------------------------------
    procedure _ShowNode( N: TNode; str:String );
    var m: String;
    begin
      m := str + #10#13#10#13
         + 'NodeId: ' + IntToStr( N.NodeId )
         + '  LeistId: ' + IntToStr( N.LeistId )
         + '  AnzNoten: ' + IntToStr( N.AnzNoten )
         + '  AnzTestate: ' + IntToStr( N.AnzTestate )
         + '  NodeText: ' + N.NodeText + #10#13#10#13
         + 'GetString: ' + N.GetString;
      ShowMessage( m );
    end;
    //-------------------------------------

begin
  // Root in STRUKT_DB holen, Node generieren, in StrukturListe geben
  // & LeistTable oeffnen & LeistName -> NodeText
  // & alle Nachbar-Referenzen je in ToLoad geben zum Laden merken
  if SDM.TryOpenTable( SDM.StruktTable ) then begin
    if SDM.TryOpentable( SDM.LeistungTable ) then begin // Node.Load
      with SDM.StruktTable do begin
        StrukturL.Clear;
        ToLoad := TList.Create;
        // root
```

```
        if Locate( 'NodeId', RootId, [loPartialKey] ) then begin
          TmpNode := TNode.CreateDefault;
          TmpNode.Load( RootId );
          if TmpNode.Error then
            _ShowNode( TmpNode, 'Error @ root '+IntToStr( RootId ) );
          // else ...
          StrukturL.Add( TmpNode );
          if TmpNode.Child <> LEER then
            ToLoad.Add( @TmpNode.Child );
          // each node
          while ToLoad.Count > 0 do begin
            //  _ShowToLoad;

            // take 1st Id from ToLoad-List
            NextId := Integer( ToLoad.First^ );
            ToLoad.Remove( ToLoad.First );
            if Locate( 'NodeId', NextId, [loPartialKey] ) then
            begin
              TmpNode := TNode.CreateDefault;
              TmpNode.Load( NextId );
              if TmpNode.Error then
                _ShowNode(TmpNode,'Error @ node: '+IntToStr(NextId));
              // else ...
              StrukturL.Add( TmpNode );
              if TmpNode.Sibling <> LEER then
                ToLoad.Add( @TmpNode.Sibling );
              if TmpNode.Child <> LEER then
                ToLoad.Add( @TmpNode.Child );
            end
            else begin
              ShowMessage( 'ERROR: finde Node '
                          +  IntToStr( NextId ) + ' nicht!' );
            end;
          end;
          ToLoad.Free;
          AbschModified := False;
        end
        else begin
          // ShowMessage( 'ERROR: finde RootNode nicht!' );
        end;
        Active := false;
      end;
      SDM.LeistungTable.Active := false;
    end
    else begin  //  LeistT ...
    end;
  end
  else begin  //  StruktT ...
  end;

end;

//---------------------------------------------------------------------
procedure TAbschluss.SaveStrukturList;    // StrukturL -> DB
//  Id's der bestehenden Liste merken  ->  IdList
//  Liste -> DB,  mit neuer Id je Node
//                Absch.rootId in AbschTable korrigieren
//  wenn ok: alte Nodes löschen
//  -> benutzt AbschForm für: NewIds(...)
//  schöner wär's mit Node.Save ! ( vgl. MigrateFromCALC )
var
  TmpNode: TNode;
  OldIdList: TList;
  i, OldId: Integer;
  S: String;
  p: ^Integer;

begin
  if AbschModified then begin  // save only if modified

    // collect old Id's (new Nodes have 'LEER')
    OldIdList := TList.Create;
    for i := 0 to StrukturL.Count-1 do begin
      new( p );
      p^ := TNode( StrukturL.Items[i] ).NodeId;
```

```
      if p^ <> LEER then OldIdList.Add( p );
    end;
    {
    // test
    S := 'OldList: ';
    for i := 0 to OldIdList.Count-1 do
      S := S + IntToStr( Integer( OldIdList.Items[i]^ ) ) + ' , ';
    ShowMessage( S );
    }
    // open StruktTable
    with SDM do begin
      if TryOpenTable( StruktTable ) then begin
        // neue Id's berechnen: (last rec).NodeId + 1 **
        StruktTable.Last;
        if StruktTable.RecordCount > 0 then
          AbschForm.NewIds( AbschForm.TV, StruktTable['NodeId']+1 )
        else
          AbschForm.NewIds( AbschForm.TV, 0 );

        for i := 0 to StrukturL.Count-1 do begin
          TmpNode := TNode( StrukturL.Items[i] );
          TmpNode.Save;
        end;

        // wenn's geklappt hat: RootId koorigieren
        RootId := TNode( StrukturL.Items[0] ).NodeId;

        // & alte Nodes aus DB löschen
        for i := 0 to OldIdList.Count-1 do begin
          OldId := Integer( OldIdList.Items[i]^ );
          if StruktTable.Locate( 'NodeId', OldId, [loPartialKey] )
          then begin
            StruktTable.Delete;  // entfernt den aktuellen Datensatz
          end
          else ShowMessage( 'ERROR! kann alten Index: '
                          + IntToStr( OldId ) + ' nicht finden!' );
        end;
        AbschModified := False;
        StruktTable.Active := False;
      end;
    end;
    OldIdList.Free;
  end;
end;

//------------------------------------------------- TAbschluss ---
constructor TAbschluss.CreateDefault;
begin
  Create;
  AbschName := '';
  RegId   := LEER;
  AbschId := LEER;
  RootId := LEER;
  StrukturL := TList.Create;
  AbschModified := False;
end;

//---------------------------------------------------------------------
constructor TAbschluss.Make(RId, AId: Integer);
begin
  CreateDefault;
  RegId     := RId;
  AbschId   := AId;
  AbschName := LoadName;
  RootId    := LoadRootId;
end;

//---------------------------------------------------------------------
function TAbschluss.LoadRootId: Integer;
begin
  Result := -1;
  with SDM do
    if TryOpentable( AbschTable ) then
      if AbschTable.Locate( 'AbschId', AbschId, [loPartialKey] ) then
        Result := AbschTable['StruktId'];
```

```
     // else's...
end;


//---------------------------------------------------------------------
function TAbschluss.LoadName: String;
begin
  Result := '***';
  with SDM do
    if TryOpentable( LeistungTable ) then
      if LeistungTable.Locate( 'LeistId', AbschId,
                               [loPartialKey] ) then
        Result := LeistungTable['LeistText']
  //      else ShowMessage('?!? Locate : ' + IntToStr( AbschId ) )
  //    else ShowMessage('?!? Open AbschTable');
  // ...
end;

//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : AbschlussForm.pas
// Abschluss-Formular
//   Darstellung der Baumstruktur mit TreeView
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit AbschlussForm;

```
interface

uses
  Abschluss, Node, PropertiesDlg, StudiDataMod, MigrationDataMod,
  DBTables, Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, ComCtrls, StdCtrls, Buttons;

type
//---------------------------------------------------------------------
  TAbschForm = class( TForm )
    TV: TTreeView;
    PropertyBtn: TBitBtn;
    NewNodeBtn: TButton;
    DeleteNodeBtn: TButton;
    CutNodeBtn: TButton;
    PasteNodeBtn: TButton;
    OKBtn: TBitBtn;
    Label1: TLabel;
    LoadBtn: TButton;
    SaveBtn: TButton;
    procedure PropertyBtnClick( Sender: TObject );
    procedure OKBtnClick( Sender: TObject );
    procedure NewNodeBtnClick(Sender: TObject);
    procedure DeleteNodeBtnClick(Sender: TObject);
    procedure CutNodeBtnClick(Sender: TObject);
    procedure PasteNodeBtnClick(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure TVChange(Sender: TObject; Node: TTreeNode);
    procedure TVDblClick(Sender: TObject);
    procedure LoadBtnClick(Sender: TObject);
    procedure SaveBtnClick(Sender: TObject);
  public
    constructor Make( Owner: TComponent; Absch: TAbschluss );
    procedure NewIds( TV: TTreeView; Root: Integer );
  private
    TheAbsch: TAbschluss;
    CopyNode: TNode;     // für copy / paste
    procedure SetTreeView;
    function AddANode( Node: TNode ): Integer;
    procedure UpdateNode( i: Integer );    // nach PropDlg
    procedure ErrMsg( Msg: String );
  end;
//---------------------------------------------------------------------
var
  AbschForm: TAbschForm;

implementation
{$R *.DFM}

//------------------------------------------------- TAbschForm ----
constructor TAbschForm.Make( Owner: TComponent; Absch: TAbschluss );
begin
  Create( Owner );
  TheAbsch := Absch;
end;

//---------------------------------------------------------------------
procedure TAbschForm.LoadBtnClick(Sender: TObject);
begin
  TheAbsch.LoadStrukturList;
  SetTreeView;
end;
```

```
//---------------------------------------------------------------------
procedure TAbschForm.SetTreeView;
// stellt StrukturL in TreeView dar
var
  i, StrParent, TVParent: Integer;
  TreeN: TTreeNode;
  TmpNode: TNode;
    //-----------------------------------------------
    function GetTVIndex( Id: Integer ) : Integer;
    var g: Integer;
    begin
      Result := -1;
      for g := 0 to TV.Items.Count-1 do
        if TNode( TV.Items[g].Data ).NodeId = Id then
        begin
          Result := g;
          Exit;
        end;
    end;
    //-----------------------------------------------
begin
  TV.Items.Clear;
  if TheAbsch.StrukturL.Count > 0 then begin
    TmpNode := TheAbsch.StrukturL.Items[0]; // root
    TreeN := TV.Items.Add( TV.Selected, TmpNode.GetString ); ///
    TreeN.Data := TmpNode;
    with TheAbsch.StrukturL do begin
      for i := 1 to Count-1 do begin
        TmpNode := Items[i];
        TVParent := GetTVIndex( TmpNode.Parent );
        if TVParent <> LEER then begin
          TreeN := TV.Items.AddChild( TV.Items[TVParent],
                                              TmpNode.GetString );
          TreeN.Data := TmpNode;
          // updateNode ?!  **
        end;
      end;
    end;
  end;
  TV.FullExpand;
end;

//---------------------------------------------------------------------
procedure TAbschForm.PropertyBtnClick( Sender: TObject );
var ANode: TNode;
begin
  if TV.Selected = nil then begin
    ErrMsg( 'select a node first!' );
  end
  else begin
    ANode := TV.Selected.Data;
    PropDlg.TheNode := ANode;
    PropDlg.ShowModal;
    if PropDlg.NodeModified
      then TheAbsch.AbschModified := true;
    UpdateNode( TV.Selected.AbsoluteIndex );
  end;
end;

//---------------------------------------------------------------------
// synchronize TreeNode & Data.Node
procedure TAbschForm.UpdateNode( i: Integer );
var
  TreeNode: TTreeNode;
  Node: TNode;
begin
  TreeNode := TV.Items[i];
  Node := TreeNode.Data;
  TreeNode.Text := Node.GetString;
  if TreeNode.Parent = nil then
    Node.Parent := LEER
  else
    Node.Parent := TNode( TreeNode.Parent.Data ).NodeId;
  if TreeNode.GetNextSibling = nil then
```

```
    Node.Sibling := LEER
  else
    Node.Sibling := TNode( TreeNode.GetNextSibling.Data ).NodeId;
  if TreeNode.GetFirstChild = nil then
    Node.Child := LEER
  else
    Node.Child := TNode( TreeNode.GetFirstChild.Data ).NodeId;
  // TV.FullExpand;
end;

//---------------------------------------------------------------------
procedure TAbschForm.NewIds( TV: TTreeView; Root: Integer );
// bereinigte Nummerierung d. Baums mithilfe d. TV
// Ptr's in SrukturL zeigen auf dieselben Nodes
var
  TreeN: TTreeNode;
  TmpNode: TNode;
  i: Integer;
begin
  for i := 0 to TV.Items.Count-1 do begin
    TreeN := TV.Items[i];
    TmpNode := TNode( TreeN.Data );
    TmpNode.NodeId := Root + i;
    if TreeN.Parent = nil then
      TmpNode.Parent := LEER
    else
      TmpNode.Parent := Root + TreeN.Parent.AbsoluteIndex;
    if TreeN.GetNextSibling = nil then
      TmpNode.Sibling := LEER
    else
      TmpNode.Sibling := Root + TreeN.GetNextSibling.AbsoluteIndex;
    if TreeN.GetFirstChild = nil then
      TmpNode.Child := LEER
    else
      TmpNode.Child := Root + TreeN.GetFirstChild.AbsoluteIndex;
  end;
end;

//---------------------------------------------------------------------
function TAbschForm.AddANode( Node: TNode ): Integer;
var TN: TTreeNode;
begin
  TheAbsch.StrukturL.Add( Node );
  // nur 1 root
  if TV.Items.Count > 0 then
    if TV.Selected = nil then TV.Selected := TV.Items[0];
  TN := TV.Items.AddChild( TV.Selected, Node.GetString );
  TN.Data := Node;
  Result := TN.AbsoluteIndex;
end;

//---------------------------------------------------------------------
procedure TAbschForm.NewNodeBtnClick(Sender: TObject);
var
  i: Integer;
  NewNode: TNode;
  TreeNode: TTreeNode;
begin
  NewNode := TNode.CreateDefault;
  PropDlg.TheNode := NewNode;
  PropDlg.ShowModal;
  if PropDlg.ModalResult = mrOK then begin
    i := AddANode( NewNode );
    UpdateNode( i );
    SetTreeView;
    TV.Selected := TV.Items[i];
    TheAbsch.AbschModified := true;
  end
  else begin
    NewNode.Free;
  end;
  TV.SetFocus;

end;
```

```
//---------------------------------------------------------------------
procedure TAbschForm.FormActivate(Sender: TObject);
begin
  Label1.Caption := TheAbsch.AbschName;
  TheAbsch.LoadStrukturList;
  SetTreeView;
  if TV.Items.Count > 0 then begin
    TV.Selected := TV.Items[0];
    PropertyBtn.Enabled := true;
  end
  else
    PropertyBtn.Enabled := false;
end;

//---------------------------------------------------------------------
// Vor.: all nodes updated ! ( neue nodes.NodeId <> -1 ! )
procedure TAbschForm.DeleteNodeBtnClick(Sender: TObject);
begin
  if TV.Selected = nil then begin
    ErrMsg( 'select a node first!' );
  end
  else
  if TV.Selected.GetFirstChild <> nil then begin
    ErrMsg( 'a node with children cannot be deleted!' );
  end
  else begin
    TNode( TV.Selected.Parent.Data ).Child := LEER;
    TheAbsch.StrukturL.Remove( TNode( TV.Selected.Data ) );
    SetTreeView;
    TheAbsch.AbschModified := true;
  end;
end;

//---------------------------------------------------------------------
procedure TAbschForm.CutNodeBtnClick(Sender: TObject);
begin
  // CopyNode := TNode( TV.Selected.Data )
  // DeleteNode( )
  //  TheAbsch.Modified := true;
end;

//---------------------------------------------------------------------
procedure TAbschForm.PasteNodeBtnClick(Sender: TObject);
begin
  // if CopyNode <> nil then
  // an selektierte Node als child einfuegen
  //  TheAbsch.Modified := true;
end;

//---------------------------------------------------------------------
procedure TAbschForm.ErrMsg( Msg: String );
begin
  MessageDlg( Msg, mtInformation, [mbOk], 0 );
end;

//---------------------------------------------------------------------
procedure TAbschForm.TVChange(Sender: TObject; Node: TTreeNode);
begin
  PropertyBtn.Enabled := true;
end;

//---------------------------------------------------------------------
procedure TAbschForm.TVDblClick(Sender: TObject);
begin
  PropertyBtnClick( Sender );
end;

//---------------------------------------------------------------------
procedure TAbschForm.OKBtnClick( Sender: TObject );
var Back: Word;
begin
 if TheAbsch.AbschModified then begin
   Back := MessageDlg('Der Abschluss ist geändert worden! '
             + 'Soll er jetzt gespeichert werden?',
             mtWarning, mbYesNoCancel, 0);
```

```
   if Back = mrYes then begin
     TheAbsch.SaveStrukturList;
     Close;
   end
   else if Back = mrNo then begin
     Close;
   end;
   // else do nothing
 end
 else Close;  // Free?
end;

//---------------------------------------------------------------------
procedure TAbschForm.SaveBtnClick(Sender: TObject);
begin
  TheAbsch.SaveStrukturList;
  SetTreeView;
  TheAbsch.AbschModified := false;
end;

//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : AuswahlDialog.pas
// Auswahl-Dialog zum Key-Value-Paare aus einem Table auswählen
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit AuswahlDialog;

```
interface

uses
  StudiDataMod,
  DBTables,
  Windows, SysUtils, Classes, Graphics, Forms, Controls, StdCtrls,
  Buttons, ExtCtrls;

//---------------------------------------------------------------------
type
  TAuswahlDlg = class(TForm)
    OKBtn: TButton;
    CancelBtn: TButton;
    Bevel1: TBevel;
    ComboBox: TComboBox;
    Label1: TLabel;
    procedure OKBtnClick(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  public
    Id: Integer;
    Txt: String;
    procedure LoadList( ATable: TTable; AField, AnIndex: String );
    procedure LoadKeyList( ATable: TTable; AnIndex: String );
    procedure SetLabel( Header: String );
  private
    Ids: TList;
  end;

var
  AuswahlDlg: TAuswahlDlg;

implementation

{$R *.DFM}

//---------------------------------------------------------------------
// lädt 'AField' in ComboBox und 'AnIndex' in Ids
procedure TAuswahlDlg.LoadList( ATable: TTable;
                                AField, AnIndex: String );
var p: ^Integer;  // Vor.: Index ist Integer
    Absch: Boolean;
begin
  Ids := TList.Create;
  ComboBox.Items.Clear;
  // um nur Abschluss-Leistungen zu zeigen:
  // AnIndex markiert mit 'Abschluss'
  Absch := AnIndex = 'Abschluss';
  if Absch then AnIndex := 'LeistId';
  if SDM.TryOpenTable( ATable ) then begin
    while not ATable.Eof do begin
      if ( not Absch ) or ( ATable[AnIndex] >= 80000 ) then begin
        ComboBox.Items.Add( ATable[AField] );
        new( p );
        p^ := ATable[AnIndex];
        Ids.Add( p );
      end;
      ATable.Next;
    end;
  end;
end;

//---------------------------------------------------------------------
```

```
// 'AnIndex' in ComboBox und in Ids
procedure TAuswahlDlg.LoadKeyList( ATable: TTable; AnIndex: String );
var p: ^Integer;  // Vor.: Index ist Integer
begin
  Ids := TList.Create;
  ComboBox.Items.Clear;
  if SDM.TryOpenTable( ATable ) then begin
    while not ATable.Eof do begin
      ComboBox.Items.Add( IntToStr( ATable[AnIndex] ) );
      new( p );
      p^ := ATable[AnIndex];
      Ids.Add( p );
      ATable.Next;
    end;
  end;
end;

//---------------------------------------------------------------------
procedure TAuswahlDlg.SetLabel( Header: String );
begin
  Label1.Caption := Header;
end;

//---------------------------------------------------------------------
procedure TAuswahlDlg.OKBtnClick(Sender: TObject);
begin
  if ComboBox.ItemIndex > -1 then begin
    Id := Integer( Ids.Items[ComboBox.ItemIndex]^ );
    Txt := ComboBox.Items[ComboBox.ItemIndex];
  end
  else ModalResult := mrCancel;
end;

//---------------------------------------------------------------------
procedure TAuswahlDlg.FormShow(Sender: TObject);
begin
  ComboBox.Text := 'aus der Liste auswählen';
end;

//---------------------------------------------------------------------
procedure TAuswahlDlg.FormClose(Sender: TObject;
                                var Action: TCloseAction);
begin
    Ids.Free;
    ComboBox.Items.Clear;
end;

//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : Auswertung.pas
// Traversierung der Abschluss-Bäume & Vergl. mit den Taten je Studi
// Prototyp zur Berechnung der LA's
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit Auswertung;

```
interface

uses
  Abschluss, Node, Studi, Taten, StudiDataMod, AuswahlDialog,
  Reglement,
  StdCtrls, Classes, Controls, Windows, Messages, SysUtils, Graphics,
  Forms, Dialogs, Quickrep;

type
  //---------------------------------------------------------------------
  TAuswertung = class( TObject )
  public
    TheAbsch: TAbschluss;
    procedure CheckStudi( MatrNr: Integer; Lines: TStrings );
  private
    Done: TList;  // schon bearbeitete 'Taten'
    procedure CheckNode( NodeId: Integer;
                         Lines: TStrings;
                         Indent: String;
                         Taten: TList;
                         var N, T: Integer );
    procedure AddOptionals( Taten: TList; Lines: TStrings );
  end;

  //---------------------------------------------------------------------
  TAuswertungsForm = class(TForm)
    Memo1: TMemo;
    TraverseBtn: TButton;
    QuitBtn: TButton;
    WhoButton: TButton;
    Memo2: TMemo;
    Label1: TLabel;
    procedure TraverseBtnClick(Sender: TObject);
    procedure QuitBtnClick(Sender: TObject);
    procedure WhoButtonClick(Sender: TObject);
    procedure FormShow(Sender: TObject);
  public
    Ausw: TAuswertung;
    constructor Create( Owner: TComponent ); override;
  end;


var
  AuswertungsForm: TAuswertungsForm;
  KandidatMatrNr: Integer;

implementation
{$R *.DFM}

//---------------------------------------------------------------------
constructor TAuswertungsForm.Create( Owner: TComponent );
begin
  inherited Create( Owner );
  Ausw := TAuswertung.Create;
end;

//---------------------------------------------------------------------
procedure TAuswertungsForm.TraverseBtnClick(Sender: TObject);
var
  i: Integer;
  Str: TStringList;
begin
```

```
  Memo1.Lines.Clear;
  // Memo1.Lines.Add('checking: ' + IntToStr( KandidatMatrNr ) );
  Ausw.CheckStudi( KandidatMatrNr, Memo1.Lines );
end;

//---------------------------------------------------------------------
procedure TAuswertung.CheckNode( NodeId:    Integer;
                                 Lines:     TStrings;
                                 Indent:    String;
                                 Taten:     TList;
                                 var N, T:  Integer );
var Node: TNode;
    i: integer;
    no,            // local counters of accomplished t's / n's
    te: Integer;   // below this node
    S: String;
    Tat: TTat;
    Found: Boolean;
    MaxNote: Real;
begin
  // current node
  Node := TheAbsch.GetNodeFromStruktList( NodeId );
  S := '   ---   ';

  // ist zur Node gehörende Leistung in TatenListe?
  Found := False;
  MaxNote := 0;
  for i := 0 to Taten.Count-1 do begin   // search Tat
    Tat := TTat( Taten.Items[i] );
    if Tat.LeistId = Node.LeistId then begin
      Found := True;
      Done.Add( Tat );
      if Tat.Note = 0.0 then begin    // Testat
        inc( te );
        inc( T );
        S := S + 'T: '
               + IntToStr(Tat.Jahr) + '.' + IntToStr(Tat.Termin);
      end
      else begin                      // Note
        if MaxNote = 0 then begin     // count only once
          inc( no );
          inc( N );
        end;
        if Tat.Note > MaxNote then begin
          MaxNote := Tat.Note;
          S := S + '  N: '
             + IntToStr(Tat.Jahr) + '.' + IntToStr(Tat.Termin)
             + '  ('
             + FloatToStrF(Tat.Note, ffFixed, 15, 1)
             + ')';
        end;
      end;
    end;
  end;
  if Found and (Node.Child = LEER) then begin     // Leafes only
    Lines.Add( Indent + IntToStr( Node.LeistId )
                 + ' - ' + Node.NodeText
                 + S );
  end;

  // count children
  if Node.Child > LEER then begin
    no := 0;
    te := 0;
    CheckNode( Node.Child, Lines, Indent + '.', Taten, no, te );

    // show # of T / N for current node
    Lines.Add( Indent + '( > '
               + IntToStr( Node.LeistId )     + ' - '
               + Node.NodeText                + ' : '
               + IntToStr( te )               + ' / '
               + IntToStr( no )               + ' von '
               + IntToStr( Node.AnzTestate )  + ' / '
               + IntToStr( Node.AnzNoten )    + ' )'
    );
```

```
    if no >= Node.AnzNoten   then N := 1 else N := 0;
    if te >= Node.AnzTestate then T := 1 else T := 0;
  end;

  // count siblings
  if Node.Sibling > LEER then begin
    no := 0;
    te := 0;
    CheckNode( Node.Sibling, Lines, Indent, Taten, no, te );
    N := N + no;      // add to current level
    T := T + te;
  end;
end;

//---------------------------------------------------------------------
procedure TAuswertung.CheckStudi( MatrNr: Integer; Lines: TStrings );
var Studi: TStudi; No, Te: Integer;
begin
  Done.Free;
  Done := TList.Create;
  Studi := TStudi.Load( MatrNr );
  AuswertungsForm.Label1.Caption := 'Checking '
               + IntToStr( Studi.MatrikelNr )
               + ': ' + Studi.Vorname + ' ' + Studi.Name
               + ' - Studienziel in Informatik: '
               + IntToStr( Studi.ZielAbschId )
               + ' ( ' + Studi.ZielAbschluss
               + ' ) nach Reglement '
               + IntToStr( Studi.RegId )
  ;
  if Studi.RegId <> DEFAULT_REGLE then begin
    AuswertungsForm.Memo1.Lines.Add( 'Achtung: falsches Reglement' );
  end;
  Studi.LoadTatenListe;
  // passenden Absch laden
  TheAbsch := TAbschluss.Make( Studi.RegId, Studi.ZielAbschId );
  TheAbsch.LoadStrukturList;
  // checken
  No := 0;
  Te := 0;
  CheckNode( TheAbsch.RootId, Lines, '', Studi.TatenListe, No, Te );
  AddOptionals( Studi.TatenListe, Lines );
end;

//---------------------------------------------------------------------
procedure TAuswertung.AddOptionals( Taten: TList; Lines: TStrings );
var
  i:         Integer;
  S:         String;
  Tat:       TTat;
  TitleDone: Boolean;
begin
  Lines.Add('');
  TitleDone := False;
  // Lines.Add('Optionale Leistungen:');
  with SDM do begin
    if TryOpenTable( LeistungTable ) then begin
      for i := 0 to Taten.Count-1 do begin
        Tat := TTat( Taten.Items[i] );
        if Done.IndexOf( Tat ) = -1 then begin // noch nicht in Lines
          if not TitleDone then begin
            Lines.Add('Optionale Leistungen:');
            TitleDone := True;
          end;
          if LeistungTable.Locate( 'LeistId',
                          Tat.LeistId, [loPartialKey] )
            then S := LeistungTable['LeistText']
            else S := 'unbekannt';
          Lines.Add( IntToStr( Tat.LeistId ) + ' - ' + S );
        end;
      end;
    end;
  end;
end;
```

```
//---------------------------------------------------------------------
procedure TAuswertungsForm.QuitBtnClick(Sender: TObject);
begin
  Close;
end;

//---------------------------------------------------------------------
procedure TAuswertungsForm.WhoButtonClick(Sender: TObject);
var Str: String;
begin
  with AuswahlDlg do begin
    LoadKeyList( SDM.StudiTable, 'MatrikelNr' );
    SetLabel( 'MatrikelNr auswählen' );
    ShowModal;
    if ModalResult = mrOk then begin
      KandidatMatrNr := Id;
    end;
  end;
end;

//---------------------------------------------------------------------
procedure TAuswertungsForm.FormShow(Sender: TObject);
begin
  Memo1.Lines.Clear;
  Label1.Caption := '';
end;

initialization
  KandidatMatrNr := 84904218;   // default is me :)

//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : ImpMainForm.pas
// ImportHauptFormular: Personendaten, Leistungsdaten importieren
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit ImpMainForm;

```
interface

uses
  ProgrDlg, Studi, StudiDataMod, Migration,
  Windows, Messages, Classes, Graphics, Controls, Forms, Dialogs,
  SysUtils, StdCtrls, DBTables, DB, Grids, DBGrids;

type
  //---------------------------------------------------------------------
  TImpMainForm = class(TForm)
    QuitBtn: TButton;
    PersImpBtn: TButton;
    ProcessBtn: TButton;
    LeistImpBtn: TButton;
    DBGrid1: TDBGrid;
    Label1: TLabel;
    OpenRawTableBtn: TButton;
    OpenStudiTableBtn: TButton;
    OpenPersLeistTableBtn: TButton;
    Label2: TLabel;
    procedure PersImpBtnClick(Sender: TObject);
    procedure ProcessBtnClick(Sender: TObject);
    procedure LeistImpBtnClick(Sender: TObject);
    procedure QuitBtnClick(Sender: TObject);
    procedure OpenRawTableBtnClick(Sender: TObject);
    procedure OpenStudiTableBtnClick(Sender: TObject);
    procedure OpenPersLeistTableBtnClick(Sender: TObject);
  private
    function OpenFile( TitleLine: String ): TStringList;
  end;
//---------------------------------------------------------------------

implementation
{$R *.DFM}

//---------------------------------------------------------------------
function TImpMainForm.OpenFile( TitleLine: String ): TStringList;
var
  Selector: TOpenDialog;
  Lines: TStringList;
begin
  Lines := TStringList.Create;
  Selector := TOpenDialog.Create( self );
  Selector.Title := 'Öffnen: ' + TitleLine;
  if Selector.Execute then  // true nach OKBtClick, false nach Cancel
  begin
    if FileExists( Selector.FileName ) then begin
      Lines.LoadFromFile( Selector.FileName );
    end
    else begin
      MessageDlg( 'Datei existiert nicht!', mtError, [mbOk], 0 );
    end;
  end;
  Selector.Free;
  Result := Lines;
end;

//---------------------------------------------------------------------
procedure TImpMainForm.PersImpBtnClick(Sender: TObject);
var
  InputLines: TStringList;
begin
  InputLines := OpenFile( '[...]PhilNat.txt' );
```

```
  if InputLines.Count > 0 then begin
    ProgressDlg := TProgressDlg.Create( self );
    ProgressDlg.StartPersImport( InputLines );
    DBGrid1.DataSource := SDM.RawDataSource;
    ProcessBtn.Enabled := True;
    InputLines.Free;
    ProgressDlg.Free;
  end
  else begin
    // MessageDlg( 'leere Datei ?!', mtError, [mbCancel], 0 );
  end;
end;

//---------------------------------------------------------------------
procedure TImpMainForm.ProcessBtnClick(Sender: TObject);
begin
  DBGrid1.DataSource := nil;
  ProgressDlg := TProgressDlg.Create( self );
  ProgressDlg.StartProcessing;
  ProgressDlg.Free;
  DBGrid1.DataSource := SDM.StudiDataSource;
  ProcessBtn.Enabled := false;
end;

//---------------------------------------------------------------------
procedure TImpMainForm.LeistImpBtnClick(Sender: TObject);
var
  InputLines: TStringList;
begin
  InputLines := OpenFile( '[Leistungen]' );
  DBGrid1.DataSource := nil;
  if InputLines.Count > 0 then begin
    ProgressDlg := TProgressDlg.Create( self );
    ProgressDlg.StartLeistImport( InputLines );
    InputLines.Free;
    ProgressDlg.Free;
    DBGrid1.DataSource := SDM.PersLeistDataSource;
  end
  else begin
    // MessageDlg( 'leere Datei ?!', mtError, [mbCancel], 0 );
  end;
end;

//---------------------------------------------------------------------
procedure TImpMainForm.QuitBtnClick(Sender: TObject);
begin
  Close;
end;

//---------------------------------------------------------------------
procedure TImpMainForm.OpenRawTableBtnClick(Sender: TObject);
begin
  if SDM.TryOpenTable( SDM.RawTable ) then begin
    DBGrid1.DataSource := SDM.RawDataSource;
    ProcessBtn.Enabled := True;
  end;
end;

//---------------------------------------------------------------------
procedure TImpMainForm.OpenStudiTableBtnClick(Sender: TObject);
begin
  if SDM.TryOpenTable( SDM.StudiTable ) then
    DBGrid1.DataSource := SDM.StudiDataSource;
end;

//---------------------------------------------------------------------
procedure TImpMainForm.OpenPersLeistTableBtnClick(Sender: TObject);
begin
  if SDM.TryOpenTable( SDM.PersLeistTable ) then
    DBGrid1.DataSource := SDM.PersLeistDataSource;
end;

//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : Leistung.pas
// Vorlesungen, etc., Abschlüsse
//  alle Knoten im StrukturBaum eines Abschlusses
//  Blätter sind Vorlesungen, etc.
//  restl. Knoten: Teilabschlüsse
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit Leistung;

```
interface

uses  StudiDataMod;
type
  //---------------------------------------------------------------------
  TLeistung = class(TObject)
  public
    LeistId: Integer;
    LeistText,
    LeistAbk,
    LeistZusatz: String;
    constructor Create;
    constructor Make( Id: Integer; Txt, Abk, Zusatz: String );
    procedure Save;
  end;

implementation

//---------------------------------------------------------------------
constructor TLeistung.Create;
begin
  inherited Create;
  LeistId := -1;
  LeistText := '';
  LeistAbk := '';
  LeistZusatz := '';
end;

//---------------------------------------------------------------------
constructor TLeistung.Make( Id: Integer; Txt, Abk, Zusatz: String );
begin
  Create;
  LeistId := Id;
  LeistText := Txt;
  LeistAbk := Abk;
  LeistZusatz := Zusatz;
end;

//---------------------------------------------------------------------
procedure TLeistung.Save;
begin
  with SDM do begin
    if TryOpenTable( LeistungTable ) then begin
      LeistungTable.Append;
      LeistungTable['LeistId']   := LeistId;
      LeistungTable['LeistText'] := LeistText;
      LeistungTable['LeistAbk'] := LeistAbk;
      LeistungTable['LeistZusatz'] := LeistZusatz;
      LeistungTable.Post;
    end
    else begin // open LeistT ?
    end;
  end;
end;

//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : Main.pas
// MainForm mit Auswahl: Personendaten, Reglemente, Import,
//                       Migration, LeistungsAusweise
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit Main;

```
interface

uses
  StudiDataMod, Personen, ImpMainForm, Reglement, Auswertung,
  Migration, MigrationDataMod,
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, StdCtrls, IniFiles;

type
  //---------------------------------------------------------------------
  TMainForm = class(TForm)
    PersDataBtn: TButton;
    ReglementeBtn: TButton;
    ImportBtn: TButton;
    LABtn: TButton;
    AnmeldenBtn: TButton;
    QuitBtn: TButton;
    Label1: TLabel;
    Label2: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    HeaderLogo: TImage;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Bevel1: TBevel;
    Bevel2: TBevel;
    MigrationBtn: TButton;
    procedure QuitBtnClick(Sender: TObject);
    procedure ImportBtnClick(Sender: TObject);
    procedure PersDataBtnClick(Sender: TObject);
    procedure ReglementeBtnClick(Sender: TObject);
    procedure LABtnClick(Sender: TObject);
    procedure MigrationBtnClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  end;

var
  MainForm: TMainForm;


implementation

{$R *.DFM}

//---------------------------------------------------------------------
procedure TMainForm.QuitBtnClick(Sender: TObject);
begin
  Application.Terminate;
end;

//---------------------------------------------------------------------
procedure TMainForm.ImportBtnClick(Sender: TObject);
var
  ImportForm: TImpMainForm;
begin
  ImportForm := TImpMainForm.Create( self );
  ImportForm.ShowModal;
  ImportForm.Free;
end;

//---------------------------------------------------------------------
```

```
procedure TMainForm.PersDataBtnClick(Sender: TObject);
begin
  PersDataForm := TPersDataForm.Create( self );

  if SDM.TryOpenTable( SDM.StudiTable ) then
    PersDataForm.ShowModal
  else
    ShowMessage('kann StudiTable nicht öffnen');

  PersDataForm.Free;
end;

//---------------------------------------------------------------------
procedure TMainForm.ReglementeBtnClick(Sender: TObject);
begin
  RegleForm := TRegleForm.Create( self );
  RegleForm.ShowModal;
  RegleForm.Free;
end;

//---------------------------------------------------------------------
procedure TMainForm.LABtnClick(Sender: TObject);
begin
  AuswertungsForm.ShowModal;
end;

//---------------------------------------------------------------------
procedure TMainForm.MigrationBtnClick(Sender: TObject);
begin
  MigrationForm := TMigrationForm.Create( self );
  MigrationForm.ShowModal;
  MigrationForm.Free;
end;

//---------------------------------------------------------------------
procedure TMainForm.FormCreate(Sender: TObject);
var
  IniFile: TIniFile;
begin
  // 'Studis3.ini' mus im Win-Dir sein !
  IniFile       := TIniFile.Create('Studis3.ini');
  ROOT_PATH     := IniFile.ReadString( 'Paths', 'ROOT', '---' );
  OLD_ROOT_PATH := IniFile.ReadString( 'Paths', 'OLD_ROOT', '---' );
  IniFile.Free;
//  ShowMessage( 'Paths:'#10#13'root: ' + ROOT_PATH
//                      + #10#13'old_root: ' + OLD_ROOT_PATH  );
end;

//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : Migration.pas
// Migration der Tables von Studis 2
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit Migration;

```
interface

uses
  StudiDataMod, MigrationDataMod, Reglement, Abschluss, Node,
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls, Grids, DBGrids, DBTables, DB;

type
  //---------------------------------------------------------------------
  TMigrationForm = class( TForm )
    CreateTablesBtn: TButton;
    OKBtn: TButton;
    SideTablesBtn: TButton;
    PSDBtn: TButton;
    ReglementeBtn: TButton;
    LeistungenBtn: TButton;
    ProgressBar1: TProgressBar;
    Label1: TLabel;
    Memo1: TMemo;
    Label2: TLabel;
    AllButton: TButton;
    SetReglementBtn: TButton;
    CancelBtn: TButton;
    Button1: TButton;
    procedure CreateTablesBtnClick(Sender: TObject);
    procedure SideTablesBtnClick(Sender: TObject);
    procedure PSDBtnClick(Sender: TObject);
    procedure LeistungenBtnClick(Sender: TObject);
    procedure ReglementeBtnClick(Sender: TObject);
    procedure OKBtnClick(Sender: TObject);
    procedure AllButtonClick(Sender: TObject);
    procedure SetReglementBtnClick(Sender: TObject);
    procedure CancelBtnClick(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    procedure CopySideTable( FromTable, ToTable: TTable;
                             FromId, ToId, FromName, ToName: String );
    procedure CopyFaTable( FromTable, ToTable: TTable;
                         FromA, ToA, FromB, ToB, FromC, ToC: String );
    procedure SideTables;
    procedure CopyFachTable;
    procedure CopyLeistungTable;
    procedure MigPSD;
    procedure MigReglemente;
    procedure MigTPN;
    procedure ToLog( Msg: String );
    procedure InitProgrBar(Maximum: Integer);
  end;

var
  MigrationForm: TMigrationForm;

implementation
{$R *.DFM}

//---------------------------------------------------------------------
procedure TMigrationForm.CreateTablesBtnClick(Sender: TObject);
begin
  OKBtn.Enabled := false;
  InitProgrBar( 11 );
  with SDM do begin
    MakeStudiTable;
    ProgressBar1.Position := 1;
```

```
    MakeImmatStatTable;
    ProgressBar1.Position := 2;
    MakeKategorieTable;
    ProgressBar1.Position := 3;
    MakeStudZielTable;
    ProgressBar1.Position := 4;
    MakeFachTable;
    ProgressBar1.Position := 5;
    MakeSpracheTable;
    ProgressBar1.Position := 6;
    MakePersLeistTable;
    ProgressBar1.Position := 7;
    MakeRegleTable;
    ProgressBar1.Position := 8;
    MakeAbschTable;
    ProgressBar1.Position := 9;
    MakeStruktTable;
    ProgressBar1.Position := 10;
    MakeLeistTable;
    ProgressBar1.Position := 11;
  end;
  ToLog( '* Tabellen neu initialisiert' );
  OKBtn.Enabled := true;
end;

//---------------------------------------------------------------------
procedure TMigrationForm.MigReglemente;
var
  Regle: TReglement;
  Absch: TAbschluss;
  S: String;
begin

// RegleTable !!!!!!!!!!!!!!!!!!!!

  Sessions.CurrentSession.AddPassWord( OLD_PW );
  with SDM do with MigDM do begin

    if TryOpentable( CalcTable ) then begin         // from
      if TryOpentable( LeistungTable ) then begin   // to

 // ------ loop (je Reg.) ------------
        // vorl. nur 1 Reglement
        Regle := TReglement.Create( 'RegName', DEFAULT_REGLE );

        with Query1 do begin             // zu ladende Abschlüsse
          DataSource := DataSource5;
          Close;
          SQL.Clear;
          SQL.Add( 'select distinct AbschlussNr from ''' + CALC_DB
             + ''' where ReglementsNr =' + IntToStr( Regle.RegId ) );
          Open;
          First;
        end;

        InitProgrBar( Query1.RecordCount );
        while not Query1.Eof do begin    // Abschlüsse laden
          if LeistungTable.Locate( 'LeistId',
                                   Query1['AbschlussNr'],
                                   [loPartialKey] )
            then S := LeistungTable['LeistText']
            else S := 'Leistung nicht gefunden in <Leistung.db>';

          ToLog( 'Reglement ' + IntToStr( DEFAULT_REGLE )
            + ' : Abschluss "' + S
            + '" übertragen' );
          Absch := TAbschluss.Make( Regle.RegId,
                                    Query1['AbschlussNr'] );
          Absch.MigrateFromCALC_DB;      //***

          Regle.AbschList.Add( Absch );
          ProgressBar1.Position :=  Query1.RecNo;
          Query1.Next;
        end;
```

```
        ToLog('* Reglement ' + IntToStr( DEFAULT_REGLE )
                             + ' : alle Abschlüsse geladen');
        Regle.SaveAbschluesse;

 // ------ loop (je Reg.) ------------

      end
      else ToLog('kann Leistung.db nicht öffnen')
    end
    else ToLog('kann Calc.db nicht öffnen')
  end;
end;

//---------------------------------------------------------------------
procedure TMigrationForm.CopySideTable( FromTable, ToTable: TTable;
                            FromId, ToId, FromName, ToName: String );
begin
  if MigDM.TryOpenTable( FromTable ) then begin
    if SDM.TryOpenTable( ToTable ) then begin
      while not FromTable.Eof do begin
        ToTable.Append;
        ToTable[ToId]   := FromTable[FromId];
        ToTable[ToName] := FromTable[FromName];
        ToTable.Post;
        FromTable.Next;
      end;
      ToLog( ToTable.Name + ' : '
          + IntToStr( ToTable.RecordCount ) + ' Recs copied' );
    end
    else     // ... ToTable
  end
  else begin // ... FromTable
  end
end;

//---------------------------------------------------------------------
procedure TMigrationForm.CopyFaTable( FromTable, ToTable: TTable;
                        FromA, ToA, FromB, ToB, FromC, ToC: String );
begin
  if MigDM.TryOpenTable( FromTable ) then begin
    if SDM.TryOpenTable( ToTable ) then begin
      while not FromTable.Eof do begin
        ToTable.Append;
        ToTable[ToA]   := FromTable[FromA];
        ToTable[ToB]   := FromTable[FromB];
        ToTable[ToC]   := FromTable[FromC];
        ToTable.Post;
        FromTable.Next;
      end;
      ToLog( ToTable.Name + ' : '
          + IntToStr( ToTable.RecordCount ) + ' Recs copied' );
    end
    else     // ... ToTable
  end
  else begin // ... FromTable
  end
end;

//---------------------------------------------------------------------
procedure TMigrationForm.CopyFachTable;
begin
  if MigDM.TryOpenTable( MigDM.FachTab ) then begin
    if SDM.TryOpenTable( SDM.FachTable ) then begin
      while not MigDM.FachTab.Eof do begin
        SDM.FachTable.Append;
        SDM.FachTable['FachId']      := MigDM.FachTab['FachNr'];
        SDM.FachTable['Fach']        := MigDM.FachTab['Bezeichnung'];
        SDM.FachTable['FakultaetsId']:= MigDM.FachTab['FakultätsNr'];
        SDM.FachTable.Post;
        MigDM.FachTab.Next;
      end;
      ToLog( SDM.FachTable.Name + ' : '
          + IntToStr( SDM.FachTable.RecordCount ) + ' Recs copied' );
    end
    else     // ... ToTable
```

```
  end
  else begin // ... FromTable
  end
end;

//---------------------------------------------------------------------
procedure TMigrationForm.CopyLeistungTable;
begin
  if MigDM.TryOpenTable( MigDM.LeistungTab ) then begin
    if SDM.TryOpenTable( SDM.LeistungTable ) then begin
      while not MigDM.LeistungTab.Eof do begin
        SDM.LeistungTable.Append;
        SDM.LeistungTable['LeistId']
                        := MigDM.LeistungTab['LeistungsNr'];
        SDM.LeistungTable['LeistText']
                        := MigDM.LeistungTab['Bezeichnung'];
        SDM.LeistungTable['LeistAbkürzung']
                        := MigDM.LeistungTab['Abkürzung'];
        SDM.LeistungTable['LeistZusatz']
                        := MigDM.LeistungTab['Zusatz'];
        SDM.LeistungTable.Post;
        MigDM.LeistungTab.Next;
      end;

      // Alternativ-Nodes für AbschlussStruktur
      SDM.LeistungTable.Append;
      SDM.LeistungTable['LeistId'] := ALTERNATIVE;
      SDM.LeistungTable['LeistText']   := 'Alternative';
      SDM.LeistungTable['LeistAbkürzung'] := '';
      SDM.LeistungTable['LeistZusatz']    := '';
      SDM.LeistungTable.Post;

      ToLog( SDM.LeistungTable.Name + ' : '
          + IntToStr( SDM.LeistungTable.RecordCount )
          + ' Recs copied' );
    end
    else     // ... ToTable
  end
  else begin // ... FromTable
  end
end;

//---------------------------------------------------------------------
procedure TMigrationForm.SideTables;
// Vor.: ZielTables(SDM) leer (Index...)
var S: String;
begin
  InitProgrBar( 6 );
  Sessions.CurrentSession.AddPassWord( OLD_PW );

  CopySideTable( MigDM.KategorieTab, SDM.KategorieTable,
                 'KategorieNr', 'KategorieId',
                 'Bezeichnung', 'Kategorie' );
  ToLog( 'KategorieTable ok' );
  ProgressBar1.Position := 1;

  CopySideTable( MigDM.StatusTab, SDM.ImmatStatTable,
                 'StatusNr',    'ImmatStatusId',
                 'Bezeichnung', 'ImmatStatus' );
  ToLog( 'ImmatStatTable ok' );
  ProgressBar1.Position := 2;

  CopySideTable( MigDM.ZielTab, SDM.StudZielTable,
                 'ZielNr',      'StudienZielId',
                 'Bezeichnung', 'StudienZiel' );
  ToLog( 'StudZielTable ok' );
  ProgressBar1.Position := 3;

  CopyFaTable( MigDM.FachTab, SDM.FachTable,
                 'FachNr',      'FachId',
                 'Bezeichnung', 'Fach',
                 'FakultätsNr', 'FakultaetsId' );
  ToLog( 'FachTable ok' );
  ProgressBar1.Position := 4;
```

```
  CopyLeistungTable;
  ToLog( 'LeistungTable ok' );
  ProgressBar1.Position := 5;

  CopySideTable( MigDM.RglmtTab, SDM.RegleTable,
                 'ReglementsNr', 'RegleId',
                 'Bezeichnung',  'RegleName' );
  ToLog( 'RegleTable ok' );
  ProgressBar1.Position := 6;

  ToLog( '* SideTables ok' );
end;

//---------------------------------------------------------------------
procedure TMigrationForm.MigPSD;
var S: String;
begin
  S := '';
  with MigDM do with SDM do begin
    if MigDM.TryOpenTable( PSDTab ) then begin
      if SDM.TryOpenTable( StudiTable ) then begin
        PSDTab.First;

        InitProgrBar( PSDTab.RecordCount );
        while not PSDTab.Eof do begin
// ??!!
if PSDTab['ImmatStatusNr'] > 0 then begin
         try

          // if HF oder eins der NF = 710  oder HF Chemie ... !!

          StudiTable.Append;                    // ist leer!
          StudiTable['MatrikelNr']  := PSDTab['MatrikelNr'];
          if PSDTab['Name_2'] <> NULL then
            StudiTable['Name']      := PSDTab['Name_1']
                                       + '-' + PSDTab['Name_2']
          else StudiTable['Name']   := PSDTab['Name_1'];
          StudiTable['Vorname']     := PSDTab['Vorname'];
          if PSDTab['Adresse_1'] <> NULL then
            StudiTable['Adresse_1']   := PSDTab['Adresse_1'];
          if PSDTab['Adresse_2'] <> NULL then
            StudiTable['Adresse_2']   := PSDTab['Adresse_2'];
          if PSDTab['Telefon'] <> NULL then
            StudiTable['Tel']         := PSDTab['Telefon'];
          // Land
          if PSDTab['PLZ'] <> NULL then
            StudiTable['PLZ']         := StrToInt(PSDTab['PLZ']); // **
          if PSDTab['Wohnort'] <> NULL then
            StudiTable['Ort']         := PSDTab['Wohnort'];
          // GeburtsDatum
          // Heimatort
          // Elternwohnort
          if PSDTab['AnredeNr'] = 1
            then StudiTable['AnredeId'] := 2     // male
            else StudiTable['AnredeId'] := 1;    // female
          // SprachId
          if PSDTab['StudienZielNr'] <> NULL then
            StudiTable['StudienZielId'] := PSDTab['StudienZielNr'];
          if PSDTab['ImmatStatusNr'] <> NULL then
            StudiTable['ImmatStatusId'] := PSDTab['ImmatStatusNr'];
          if PSDTab['ImmatKategorieNr'] <> NULL then
            StudiTable['KategorieId']  := PSDTab['ImmatKategorieNr'];
          if PSDTab['ReglementsNr'] <> NULL then
            StudiTable['RegId']         := PSDTab['ReglementsNr'];
          if PSDTab['AbschlussNr'] <> NULL then
            StudiTable['ZielAbschId']   := PSDTab['AbschlussNr'];

          if PSDTab['HFNr'] <> NULL then begin
            StudiTable['HFId'] := PSDTab['HFNr'];
            StudiTable['HFSemAnzahl'] := PSDTab['SemesterHF'];
          end;
          if PSDTab['NF1Nr'] <> NULL then begin
            StudiTable['NF1Id'] := PSDTab['NF1Nr'];
            StudiTable['NF1SemAnzahl'] := PSDTab['SemesterNF1'];
          end;
```

```
          if PSDTab['NF2Nr'] <> NULL then begin
            StudiTable['NF2Id'] := PSDTab['NF2Nr'];
            StudiTable['NF2SemAnzahl'] := PSDTab['SemesterNF2'];
          end;

          StudiTable['InsertDate']      := PSDTab['TStampSDXInsert'];
          StudiTable['UpdateDate']      := PSDTab['TStampSDXUpdate'];
          // ??
          StudiTable['AbschChangeDate']
                              := PSDTab['TStampPSDAbschlussChange'];
          StudiTable['LA1Date']         := PSDTab['TStampLA1'];
          StudiTable['LA2Date']         := PSDTab['TStampLA2'];

          // *****  mit Bemerkung ex. genau ein rec. !
           StudiTable.Post;
         except
          on E : Exception do
            S := S + 'Personen-Migration: Fehler bei MatrNr '
                   + IntToStr(PSDTab['MatrikelNr'])
                   + ' : ' + E.Message + #10#13;
         end;
end;
          ProgressBar1.Position := PSDTab.RecNo;
          PSDTab.Next;
        end;
        if S > '' then
          ShowMessage( S )
        else
          ToLog('* PersonenStammDaten-Migration beendet');
      end;   // else 'cannot open' StudiTable
    end;   // else 'cannot open' PSDTable
  end;
end;

//---------------------------------------------------------------------
procedure TMigrationForm.MigTPN;
var S: String;
begin
  S := '';
  with MigDM do with SDM do begin
    if MigDM.TryOpenTable( TPNTab ) then begin
      if SDM.TryOpenTable( PersLeistTable ) then begin

        InitProgrBar( TPNTab.RecordCount );
        while not TPNTab.Eof do begin
         try
          PersLeistTable.Append;
          PersLeistTable['MatrikelNr']   := TPNTab['MatrikelNr'];
          PersLeistTable['LeistId']      := TPNTab['LeistungsNr'];
          if TPNTab['Art'] = 'T'
            then PersLeistTable['Note']  := 0
            else PersLeistTable['Note']  := TPNTab['Note'];
          PersLeistTable['Jahr']         := TPNTab['Jahr'];
          PersLeistTable['Termin']       := TPNTab['Termin'];

          // gueltig, first..., ...
         except
          on E : Exception do
            S := S + 'Leistungs-Migration: Fehler bei '
                   + IntToStr(TPNTab['MatrikelNr']) + ' / '
                   + IntToStr(TPNTab['LeistungsNr']) + ' / '
                   + IntToStr(TPNTab['Note']) + ' / '
                   + IntToStr(TPNTab['Jahr']) + ' / '
                   + IntToStr(TPNTab['Termin']) + ' / '
                   + TPNTab['Art'] + #10#13
                   + ' : ' + E.Message + #10#13;
         end;
          ProgressBar1.Position := TPNTab.RecNo;
          TPNTab.Next;
        end;
        if S > '' then
          ShowMessage( S )
        else
          ToLog('* Leistungsnachweis-Migration beendet');
      end;   // else 'cannot open' PersLeistTable
```

```
    end;   // else 'cannot open' TPNTab
  end;
end;

//---------------------------------------------------------------------
procedure TMigrationForm.SideTablesBtnClick(Sender: TObject);
begin
  OKBtn.Enabled := false;
  SideTables;
  OKBtn.Enabled := true;
end;

//---------------------------------------------------------------------
procedure TMigrationForm.PSDBtnClick(Sender: TObject);
begin
  OKBtn.Enabled := false;
  MigPSD;
  OKBtn.Enabled := true;
end;

//---------------------------------------------------------------------
procedure TMigrationForm.ReglementeBtnClick(Sender: TObject);
begin
  OKBtn.Enabled := false;
  MigReglemente;
  OKBtn.Enabled := true;
end;
//---------------------------------------------------------------------
procedure TMigrationForm.LeistungenBtnClick(Sender: TObject);
begin
  OKBtn.Enabled := false;
  MigTPN;
  OKBtn.Enabled := true;
end;

//---------------------------------------------------------------------
procedure TMigrationForm.OKBtnClick(Sender: TObject);
begin
  Close;    // Free?
end;
//---------------------------------------------------------------------
procedure TMigrationForm.InitProgrBar(Maximum: Integer);
begin
    ProgressBar1.Max := Maximum;
    ProgressBar1.Position := 0;
    ProgressBar1.Step := 1;
end;

//---------------------------------------------------------------------
procedure TMigrationForm.ToLog( Msg: String );
begin
  Memo1.Lines.Add( Msg );     // & to log-file
end;

//---------------------------------------------------------------------
procedure TMigrationForm.AllButtonClick(Sender: TObject);
begin
  CreateTablesBtnClick(Sender);
  SideTablesBtnClick(Sender);
  ReglementeBtnClick(Sender);
  PSDBtnClick(Sender);
  LeistungenBtnClick(Sender);
  ToLog( '  >>   alle Migrationsschritte beendet   <<' );
end;

//---------------------------------------------------------------------
procedure TMigrationForm.SetReglementBtnClick(Sender: TObject);
var Str: String;
begin
  Str := InputBox('Reglement',
                  'Bitte Reglements-Nummer eingeben',
                  IntToStr( DEFAULT_REGLE ) );

  DEFAULT_REGLE := StrToInt( Str );
end;
```

```
//---------------------------------------------------------------------
procedure TMigrationForm.CancelBtnClick(Sender: TObject);
begin
  // thread unterbrechen ?! ...
  OKBtnClick( Sender );
end;

//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : MigrationDataMod.pas
// DatenModul für die Migration
// alle Tables von Studis 2 öffnen (ReadOnly) & Daten übertragen
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit MigrationDataMod;

```
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, DB, DBTables;

type
  //---------------------------------------------------------------------
  TMigDM = class(TDataModule)
    PSDTab:         TTable;
    TPNTab:         TTable;
    KategorieTab:   TTable;
    StatusTab:      TTable;
    ZielTab:        TTable;
    FachTab:        TTable;
    FakultaetTab:   TTable;
    LeistungTab:    TTable;
    RglmtTab: TTable;

    CalcTable:      TTable;
    AequivalTable:  TTable;
    AequiListTable: TTable;

    Query1: TQuery;
    Query2: TQuery;
    DataSource1: TDataSource;
    DataSource5: TDataSource;
    DataSource6: TDataSource;
    procedure MigDMCreate(Sender: TObject);
  public
    function TryOpenTable( ATable: TTable ) : Boolean;
  private
    procedure SetTableName( ATable: TTable );
  end;

var
  MigDM: TMigDM;
  OLD_ROOT_PATH, OLD_PW,
  PSD_TAB, TPN_TAB, STATUS_TAB, KATEGORIE_TAB, ZIEL_TAB,
  FACH_TAB, FAKULTAET_TAB, LEISTUNG_TAB, RGLMT_TAB, CALC_DB,
  AEQUIV_DB, AEQUILIST_DB : String;
implementation
{$R *.DFM}

//---------------------------------------------------------------------
function TMigDM.TryOpenTable( ATable: TTable ): Boolean;
begin
  SetTableName( ATable );
  Result := true;      // default
  if ATable.Active = true then begin    // schon geöffnet
    ATable.First;
  end
  else begin                            // öffnen
    try
      ATable.Active := true;
      ATable.First;
    except
      on EDBEngineError do begin
        ShowMessage('ERROR: kann ' + ATable.Name + ' nicht öffnen!');
        ATable.Active := false;
        Result := false;
      end;
```

```
    end;
  end;
end;

//---------------------------------------------------------------------
procedure TMigDM.SetTableName( ATable: TTable );
begin
  with ATable do begin
    if not Active then begin
           if Name = 'PSDTab'         then TableName := PSD_TAB
      else if Name = 'TPNTab'         then TableName := TPN_TAB
      else if Name = 'KategorieTab'   then TableName := KATEGORIE_TAB
      else if Name = 'StatusTab'      then TableName := STATUS_TAB
      else if Name = 'ZielTab'        then TableName := ZIEL_TAB
      else if Name = 'FachTab'        then TableName := FACH_TAB
      else if Name = 'FakultaetTab'   then TableName := FAKULTAET_TAB
      else if Name = 'LeistungTab'    then TableName := LEISTUNG_TAB
      else if Name = 'RglmtTab'       then TableName := RGLMT_TAB

      else if Name = 'CalcTable'      then TableName := CALC_DB
      else if Name = 'AequivalTable'  then TableName := AEQUIV_DB
      else if Name = 'AequiListTable' then TableName := AEQUILIST_DB
    end;
  end;
end;

//---------------------------------------------------------------------
procedure TMigDM.MigDMCreate(Sender: TObject);
begin
  OLD_PW        := 'Kristall';
  PSD_TAB       := OLD_ROOT_PATH + 'PSD.db';
  TPN_TAB       := OLD_ROOT_PATH + 'TPN.db';
  STATUS_TAB    := OLD_ROOT_PATH + 'Status.db';
  KATEGORIE_TAB := OLD_ROOT_PATH + 'Kategori.db';
  ZIEL_TAB      := OLD_ROOT_PATH + 'Ziel.db';
  FACH_TAB      := OLD_ROOT_PATH + 'Fach.db';
  FAKULTAET_TAB := OLD_ROOT_PATH + 'Faculty.db';
  LEISTUNG_TAB  := OLD_ROOT_PATH + 'Leistung.db';
  RGLMT_TAB     := OLD_ROOT_PATH + 'RGLMT.db';
  CALC_DB       := OLD_ROOT_PATH + 'cc.db';
  AEQUIV_DB     := OLD_ROOT_PATH + 'Aequival.db';
  AEQUILIST_DB  := OLD_ROOT_PATH + 'AequiLst.db';
end;

//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : NeuDialog.pas
// universeller NeuDlg  mit 1 Integer- & 3 String-Eingabefeldern
//  -> für Abschluss, Leistung
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit NeuDialog;

```
interface

uses
  AuswahlDialog, StudiDataMod,
  Windows, SysUtils, Classes, Graphics, Forms, Controls, StdCtrls,
  Dialogs, Buttons, ExtCtrls;

type
  //---------------------------------------------------------------------
  TNeuDlg = class(TForm)
    OKBtn: TButton;
    CancelBtn: TButton;
    IdEdit: TEdit;
    TxEdit: TEdit;
    AbkEdit: TEdit;
    ZusatzEdit: TEdit;
    IdLabel: TLabel;
    TxLabel: TLabel;
    TitleLabel: TLabel;
    Bevel1: TBevel;
    CommentLabel: TLabel;
    SelectBtn: TButton;
    procedure OKBtnClick(Sender: TObject);
    procedure SelectBtnClick(Sender: TObject);
  public
    Id: Integer;
    Tx: String;
  end;

var
  NeuDlg: TNeuDlg;

implementation
{$R *.DFM}

//---------------------------------------------------------------------
procedure TNeuDlg.OKBtnClick(Sender: TObject);
begin
  Id := -1;
  if ( IdEdit.Text > '' ) and ( TxEdit.Text > '' ) then begin
    try
      Id := StrToInt( IdEdit.Text );
      Tx := TxEdit.Text;
      ModalResult := mrOK;
    except
      on EConvertError do begin
        MessageDlg( IdLabel.Caption + ' muss ganzzahlig sein !',
                    mtError, [mbOk], 0 );
      end;
    end;
  end;
end;

//---------------------------------------------------------------------
procedure TNeuDlg.SelectBtnClick(Sender: TObject);
begin
  with AuswahlDlg do begin
    LoadList( SDM.LeistungTable, 'LeistText', 'Abschluss' );
    SetLabel( 'Abschluss auswählen' );
    ShowModal;
    if ModalResult = mrOk then begin
      IdEdit.Text     := IntToStr( Id );
```

```
      TxEdit.Text     := Txt;
      SDM.LeistungTable.Locate( 'LeistId', Id, [loPartialKey] );
      if SDM.LeistungTable['LeistAbkürzung'] > NULL then
        AbkEdit.Text    := SDM.LeistungTable['LeistAbkürzung'];
      if SDM.LeistungTable['LeistZusatz'] > NULL then
        ZusatzEdit.Text := SDM.LeistungTable['LeistZusatz'];
    end;
  end;
end;

//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : Node.pas
// Nodes für Abschluss-Struktur-Baum
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit Node;

```
interface

uses
  StudiDataMod,
  SysUtils, Classes;

const
  LEER         =  -1;       // default ref's
  ALTERNATIVE  = 111;       // "Aequivalenz" LeistId in Leistung.db
  ALT_TXT      = 'Alternative';

type
//---------------------------------------------------------------------
  TNode = class(TObject)

    // Ref's auf umgebende Nodes im (linkslinearen) Baum
    NodeId, Parent, Sibling, Child:       Integer;

    // Node-Inhalt
    LeistId:                              Integer;
    NodeText:                             String;
    AnzNoten,
    AnzTestate:                           Integer;
    WahlNote, WahlTestat:                 Boolean;

    LA1, LA2, LA1Fertig, LA2Fertig:       Boolean;
    ZeitlimiteTestate, ZeitlimiteNoten:   Integer;
    ErfordNote, RundenAuf:                Real;
    Gewichtung:                           Integer;

    Error:                                Boolean;

    constructor CreateDefault;
    constructor Create( Id, Par, Sib, Chi: Integer;
                        LId, AnzN, AnzT : Integer;
                        WN, WT: Boolean;
                        Txt: String;
                        LAusw1, LAusw2,
                        LA1F, LA2F: Boolean;
                        LimiteT, LimiteN: Integer;
                        Erford, Rund: Real;
                        Gewicht: Integer    ); virtual;

    procedure Load( Id: Integer );
    procedure Save;
    function GetString : String;
    function GetLongString : String;
    function IsLeaf : Boolean;
    function GetStruktPos : String;   // 4 debugging
  end;

implementation

//---------------------------------------------------------------------
// Vor.: NodeId noch nicht vorhanden!
// vgl Abschl.SaveList
procedure TNode.Save;
begin
  try
    with SDM do begin
      if TryOpentable( StruktTable ) then begin
        StruktTable.Append;          // = goto last rec, insertmode
        StruktTable['NodeId']      := NodeId;
```

```
        StruktTable['Parent']      := Parent;
        StruktTable['Sibling']     := Sibling;
        StruktTable['Child']       := Child;

        StruktTable['LeistId']     := LeistId;
        StruktTable['AnzNoten']    := AnzNoten;
        StruktTable['AnzTestate']  := AnzTestate;
        StruktTable['WahlNote']    := WahlNote;
        StruktTable['WahlTestat']  := WahlTestat;

        StruktTable['LA1']                 := LA1;
        StruktTable['LA2']                 := LA2;
        StruktTable['LA1Fertig']           := LA1Fertig;
        StruktTable['LA2Fertig']           := LA2Fertig;
        StruktTable['ZeitlimiteTestate']   := ZeitlimiteTestate;
        StruktTable['ZeitlimiteNoten']     := ZeitlimiteNoten;
        StruktTable['ErfordNote']          := ErfordNote;
        StruktTable['RundenAuf']           := RundenAuf;
        StruktTable['Gewichtung']          := Gewichtung;
        // NodeText = LeistText der entspr. Leistung

        StruktTable.Post;
      end
      else begin // Log : StruktTable ...
      end;
    end;
  except
    on EDatabaseError do
      Error := true;
  end;
end;

//---------------------------------------------------------------------
procedure TNode.Load( Id: Integer );
begin
  with SDM do begin
    if TryOpentable( StruktTable ) then begin
      if TryOpentable( LeistungTable ) then begin
        if StruktTable.Locate( 'NodeId', Id, [loPartialKey] )
        then begin
          NodeId        := Id;
          Parent        := StruktTable['Parent'];
          Sibling       := StruktTable['Sibling'];
          Child         := StruktTable['Child'];

          LeistId       := StruktTable['LeistId'];
          AnzNoten      := StruktTable['AnzNoten'];
          AnzTestate    := StruktTable['AnzTestate'];
          WahlNote      := StruktTable['WahlNote'];
          WahlTestat    := StruktTable['WahlTestat'];

          LA1                 := StruktTable['LA1'];
          LA2                 := StruktTable['LA2'];
          LA1Fertig           := StruktTable['LA1Fertig'];
          LA2Fertig           := StruktTable['LA2Fertig'];
          ZeitlimiteTestate   := StruktTable['ZeitlimiteTestate'];
          ZeitlimiteNoten     := StruktTable['ZeitlimiteNoten'];
          ErfordNote          := StruktTable['ErfordNote'];
          RundenAuf           := StruktTable['RundenAuf'];
          Gewichtung          := StruktTable['Gewichtung'];

          if LeistungTable.Locate( 'LeistId',
                                   LeistId, [loPartialKey] )
          then NodeText := LeistungTable['LeistText']
          else Error := true;  // Leist
        end
        else Error := true;  // Strukt
      end
      else Error := true;  //  open LeistT
    end
    else Error := true;  //  open StruktT
  end; // with
  // tables Active lassen ...
end;
```

```
//---------------------------------------------------------------------
constructor TNode.Create( Id, Par, Sib, Chi: Integer;
                          LId, AnzN, AnzT : Integer;
                          WN, WT: Boolean;
                          Txt: String;
                          LAusw1, LAusw2,
                          LA1F, LA2F: Boolean;
                          LimiteT, LimiteN: Integer;
                          Erford, Rund: Real;
                          Gewicht: Integer    );
begin
  inherited Create;
  NodeId      := Id;
  Parent      := Par;
  Sibling     := Sib;
  Child       := Chi;

  LeistId     := LId;
  AnzNoten    := AnzN;
  AnzTestate  := AnzT;
  WahlNote    := WN;
  WahlTestat  := WT;
  NodeText    := Txt;

  LA1                 := LAusw1;
  LA2                 := LAusw2;
  LA1Fertig           := LA1F;
  LA2Fertig           := LA2F;
  ZeitlimiteTestate   := LimiteT;
  ZeitlimiteNoten     := LimiteN;
  ErfordNote          := Erford;
  RundenAuf           := Rund;
  Gewichtung          := Gewicht;

  Error       := false;
end;

//---------------------------------------------------------------------
constructor TNode.CreateDefault;
begin
  inherited Create;
  NodeId      := LEER;
  Parent      := LEER;
  Sibling     := LEER;
  Child       := LEER;

  LeistId     := 10001;
  AnzNoten    := LEER;
  AnzTestate  := LEER;
  WahlNote    := False;
  WahlTestat  := False;
  NodeText    := '';

  LA1                 := false;
  LA2                 := false;
  LA1Fertig           := false;
  LA2Fertig           := false;
  ZeitlimiteTestate   := 0;
  ZeitlimiteNoten     := 0;
  ErfordNote          := 0.0;
  RundenAuf           := 0.0;
  Gewichtung          := 0;

  Error       := false;
end;

//---------------------------------------------------------------------
function TNode.GetString : String;
var S: String;
begin
  if LeistId = ALTERNATIVE then begin
    S := NodeText;
  end
  else begin
```

```
    S := IntToStr( LeistId ) + ' : ' + NodeText;
  end;
  if Error then S := S + ' Error';
  Result := S;
end;

//---------------------------------------------------------------------
function TNode.GetLongString : String;
var S: String;
begin
  if LeistId = ALTERNATIVE then begin
    S := NodeText;
  end
  else begin
    S := IntToStr( LeistId ) + ' : ' + NodeText;
  end;
  S := S + '     >>    ';
  // if AnzNoten <> -1 then begin
    S := S + ' N: ';
    S := S + IntToStr( AnzNoten );
  // end;
  // if AnzTestate <> -1 then
    S := S + ', T: ' + IntToStr( AnzTestate );

  if WahlNote
    then S := S + ',  wN'
    else S := S + ',  oN';
  if WahlTestat
    then S := S + ',  wT'
    else S := S + ',  oT';

  if LA1
    then S := S + ',  auf LA1'
    else S := S + ',  not LA1';
  if LA2
    then S := S + ',  auf LA2'
    else S := S + ',  not LA2';

  if LA1Fertig
    then S := S + ',    LA1 fertig  '
    else S := S + ',  not LA1 fertig';
  if LA1Fertig
    then S := S + ',    LA2 fertig  '
    else S := S + ',  not LA2 fertig';

  S := S + '  ZeitlimiteTestate: ' + IntToStr( ZeitlimiteTestate );
  S := S + '  ZeitlimiteNoten: ' + IntToStr( ZeitlimiteNoten );
  S := S + '  Erford Note: ' + FloatToStr( ErfordNote );
  S := S + '  RundenAuf: ' + FloatToStr( RundenAuf );
  S := S + '  Gewichtung: ' + IntToStr( Gewichtung );

  // S := S + '  (' + IntToStr(NodeId) + ')';  ...
  if Error then S := S + ' * Error *';
  Result := S;
end;

//---------------------------------------------------------------------
function TNode.IsLeaf : Boolean;
begin
  Result := Child = LEER;
end;

//---------------------------------------------------------------------
function TNode.GetStruktPos : String;
var S: String;
begin
   S := '       ('   + IntToStr( NodeId ) + ') '
      + '    >     Par: ' + IntToStr( Parent )
      + ' ,  Sib: ' + IntToStr( Sibling )
      + ' ,  Chi: ' + IntToStr( Child );
   Result := S;
end;

//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : Personen.pas
// Uebersichts-Liste aller eingetragenen Studis
// -> aufrufen des Detailansicht ( StudiDlg )
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit Personen;

```
interface

uses
  Studi, StudiDialog, StudiDataMod, SearchDlg,
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Grids, DBGrids;

//---------------------------------------------------------------------
type
  TPersDataForm = class(TForm)
    DBGrid1: TDBGrid;
    Label1: TLabel;
    QuitBtn: TButton;
    DetailsBtn: TButton;
    SearchBtn: TButton;
    procedure FormCreate(Sender: TObject);
    procedure QuitBtnClick(Sender: TObject);
    procedure DetailsBtnClick(Sender: TObject);
    procedure SearchBtnClick(Sender: TObject);
  private
  public
  end;

var
  PersDataForm: TPersDataForm;

implementation
{$R *.DFM}

//---------------------------------------------------------------------
procedure TPersDataForm.FormCreate(Sender: TObject);
begin
  DBGrid1.DataSource := SDM.StudiDataSource;

{ // nur paar Angaben, Rest bei 'Details'
  // (suchen müsste angepasst werden)
  with SDM.StudiQuery do begin
    Close;
    SQL.Clear;
    SQL.Add( 'select '
          + 'MatrikelNr, Name, Vorname, Tel, '
          + 'Adresse_1, Adresse_2, HFId, NF1Id, NF2Id, '
          +'ImmatStatusId, KategorieId '
          + ' from ''' + STUDI_DB + ''''
          + ' order by MatrikelNr '
    );
    Open;
  end;
  DBGrid1.DataSource := SDM.StudiQueryDS;
}
end;

//---------------------------------------------------------------------
procedure TPersDataForm.QuitBtnClick(Sender: TObject);
begin
  Close;
end;

//---------------------------------------------------------------------
procedure TPersDataForm.DetailsBtnClick(Sender: TObject);
var
  m: Integer;
```

```
  Studi: TStudi;
begin
//  m := SDM.StudiQuery['MatrikelNr'];
  m := SDM.StudiTable['MatrikelNr'];
  Studi := TStudi.Load( m );
  StudiDlg := TStudiDlg.Make( self, Studi );
  StudiDlg.ShowModal;
  StudiDlg.Free;
end;

//---------------------------------------------------------------------
procedure TPersDataForm.SearchBtnClick(Sender: TObject);
var
  Matr: Integer;
begin
  SuchDlg.ShowModal;
  with SDM.StudiTable do begin
    if SuchDlg.ModRes = 'MatrikelNr' then begin
      try
        if IndexDefs.Count > 0 then
          IndexName := IndexDefs.Items[0].Name;
        Matr := StrToInt( SuchDlg.Value );
        while Matr < 10000000 do Matr := Matr * 10;
        FindNearest([Matr]);
      except
        on EConvertError do ShowMessage('falsche Eingabe: '
                                        + ' bitte nur Ziffern ');
      end;
    end
    else if SuchDlg.ModRes = 'Name' then begin
      IndexName := 'NameIndex';
      FindNearest([SuchDlg.Value]);
    end
    else if SuchDlg.ModRes = 'Vorname' then begin
      IndexName := 'VornameIndex';
      FindNearest([SuchDlg.Value]);
    end;
  end;
  // ShowMessage('#'+IntToStr(SDM.StudiTable.indexdefs.count));
end;

//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : ProcessThr.pas
// RawTable-Processing
// überprüfen und korrigieren des importierten RawTables
//   jeden Fremdschlüssel mit NebenTabelle checken & diese ergänzen
//     -> nach Value (Name, ...) und nicht nach Id !
// wenn alles ok -> je Record Studi generieren und speichern
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit ProcessThr;

```
interface

uses
  Studi, StudiDataMod,
  Dialogs, Classes, SysUtils, DBTables;

type
  ByteSet= Set of 0..255;
  TProcessThr = class(TThread)
  public
    constructor Make;
  private
    StudiList: TList;
    procedure Execute; override;

    procedure ProcessAnrede;
    procedure ProcessKategorie;
    procedure ProcessFach;
    // für Sprache-, Status- & StudZiel-Table
    procedure ProcessTable( ATable: TTable;
                            NameField, IdField: String );
    procedure MakeStudis;
    procedure SaveStudis;

    procedure SetLabel( S: String );
    function MakeNewKey( var IdSet: ByteSet ): Integer;
    procedure ShowProgress(i: Integer);
  end;

implementation

uses ImpMainForm, ProgrDlg;

//---------------------------------------------------------------------
constructor TProcessThr.Make;
begin
  Create(False);
  StudiList := TList.Create;
  FreeOnTerminate := True;
end;

//---------------------------------------------------------------------
procedure TProcessThr.Execute;
begin
  ProgressDlg.Label1.Caption := 'processing the new data ...';
  ProcessAnrede;
  ProcessKategorie;
  ProcessTable( SDM.SpracheTable, 'Sprache', 'SprachId' );
  ProcessTable( SDM.ImmatStatTable, 'ImmatStatus', 'ImmatStatusId' );
  ProcessTable( SDM.StudZielTable, 'StudienZiel', 'StudienZielId' );
  ProcessFach;
  ProgressDlg.Label1.Caption := 'generating Studis ...';
  MakeStudis;
  ProgressDlg.Label1.Caption := 'saving Studis ...';
  SaveStudis;
  StudiList.Free;
  ProgressDlg.OKBtn.Enabled := True;

end;
```

```
//---------------------------------------------------------------------
procedure TProcessThr.ProcessAnrede;
begin
// in Philnat.txt: nur 2: Frau & 3: Herrn
// in PSD.DB: auch, zusatzlich ein paar 0 (beiderlei Geschlechts)?!
// neu: 1 für 'Frau', 2 für 'Herrn' ohne Referenztabelle...
  with SDM do begin
    if TryOpenTable( RawTable ) then begin
      SetLabel( 'Anrede überprüfen ...' );
      with RawQuery do begin
        Close;
        SQL.Clear;
        SQL.Add( 'update ' + '''' + RAW_DB + '''' );
        SQL.Add( 'set AnredeId = 1 where Anrede = ''Frau''' );
        ExecSQL;
        SQL.Clear;    // nur 1 update aufs Mal ...
        SQL.Add( 'update ' + '''' + RAW_DB + '''' );
        SQL.Add( 'set AnredeId = 2 where Anrede = ''Herrn''' );
        ExecSQL;
        SQL.Clear;
        SQL.Add( 'update ' + '''' + RAW_DB + '''' );
        SQL.Add( 'set AnredeId = 0');
        SQL.Add( 'where not(Anrede = ''Herrn'' or '
               + 'Anrede = ''Frau'')');
        ExecSQL;
      end;
    end
    else begin
      // Msg: kann RawTable nicht öffnen
    end;
  end;
end;

//---------------------------------------------------------------------
procedure TProcessThr.ProcessKategorie;
// in RawTable KategoieId's setzen & neue in KategorieTable schreiben
//
// RawQuery: alle in RawTable vorh. Kat.
// Kat.Query: alle in Kat.Table vorh. Kat & K'Id -> IdSet
// Kat.Table & IdSet ergänzen
// RawTable ergänzen
var
  i: Integer;
  IdSet: ByteSet;
  S, T: String;
begin
  with SDM do begin
    if TryOpenTable( KategorieTable ) then begin
      if TryOpenTable( RawTable ) then begin
        // in RawTable vorhandene Kategorien laden
        SetLabel( 'vorhandene Kategorien laden ...' );
        with RawQuery do begin
          Close;
          SQL.Clear;
          SQL.Add( 'select distinct Kategorie from ' );
          SQL.Add( '''' + RAW_DB + '''');
          Open;
        end;
        // in KategorieTable vorhandene Kategorien laden
        with KategQuery do begin
          Close;
          SQL.Clear;
          SQL.Add( 'select distinct KategorieId, Kategorie from ' );
          SQL.Add( '''' + KATEGORIE_DB + '''' );
          Open;
          // alte Id's -> IdSet
          IdSet := [];
          First;
          while not Eof do begin
            i := Integer( FieldValues['KategorieId'] );
            IdSet := IdSet + [ i ];
            Next;
          end;
        end;
```

```
        // alle Kateg. im RawTable, die noch nicht im Kateg.Table
        // sind, dort mit neuer Id einfügen
        with RawQuery do begin
          First;
          while not Eof do begin
            if not KategQuery.Locate('Kategorie',
                        RawQuery['Kategorie'], [] ) then begin
              KategorieTable.Append;
              i := MakeNewKey( IdSet );
              KategorieTable['KategorieId'] := i;
              KategorieTable['Kategorie']:=FieldValues['Kategorie'];
              KategorieTable.Post;
            end;
            Next;
          end;
        end;
        // KategorieId's in RawTable eintragen
        SetLabel( 'Kategorien eintragen ...' );
        KategQuery.Close;
        KategQuery.Open;                // update
        KategQuery.First;
        while not KategQuery.EOF do begin
          S := IntToStr( KategQuery['KategorieId'] );
          T := KategQuery['Kategorie'];
          with RawQuery do begin
            Close;
            SQL.Clear;
            SQL.Add( 'update ' + '''' + RAW_DB + '''' );
            SQL.Add( ' set KategorieId = ' + S );
            SQL.Add( ' where Kategorie = ''' + T + '''');
            ExecSQL;
          end;
          KategQuery.Next;
        end;
        RawTable.Refresh;  // neu lesen -> Kat.Nr weden angezeigt
        KategQuery.Close;
        RawQuery.Close;
      end
      else begin
        // Msg: kann RawTable nicht öffnen
      end;
      KategorieTable.Active := false;    // Kateg.Table schliessen
    end
    else begin
      // Msg: kann KategorieTable nicht öffnen
    end;
    // ( RawTable bleibt Active )
  end;
end;

//---------------------------------------------------------------------
procedure TProcessThr.ProcessTable( ATable: TTable;
                                    NameField, IdField: String );
// sucht in RawTable die vorhanden (NameField), (IdField) Einträge
// & hängt gegebenenfalls noch nicht vorhanden in ATable an
begin
  SetLabel( 'prüfe '+ ATable.Name + ' ...' );
  with SDM do begin
    if TryOpenTable( RawTable ) then begin
      if TryOpenTable( ATable ) then begin
        // in ATable vorhandene (NameField), (IdField) laden
        with RawQuery do begin
          Close;
          SQL.Clear;
          SQL.Add( 'select distinct ' + IdField + ', ' + NameField );
          SQL.Add( ' from ''' + RAW_DB + '''');
          Open;
          First;
          while not Eof do begin
            // mit ATable vergl. & ev. anhängen
            if not ATable.Locate( IdField,
                 RawQuery[IdField], [loPartialKey] ) then begin
              ATable.Append;
              ATable[IdField] := RawQuery[IdField];
              ATable[NameField]  := RawQuery[NameField];
```

```
            ATable.Post;
          end
          else begin
            // ev. noch prüfen, ob 'NameField' identisch?!
          end;
          Next;
        end;
      end;
    end
    else begin    // Msg: kann ATable.Name nicht öffnen
    end;
    ATable.Active := false;
  end;     // RawTable bleibt Active
 end;
end;

//---------------------------------------------------------------------
procedure TProcessThr.ProcessFach;
var i: Integer; S: String;
begin
  SetLabel( 'prüfe Fachtabelle ...' );
  with SDM do begin
    if TryOpenTable( RawTable ) then begin
      if TryOpenTable( FachTable ) then begin
        with SDM.RawQuery do begin
          Close;
          SQL.Clear;
          SQL.Add( 'select distinct HFId, HF' );
          SQL.Add( 'from ''' + RAW_DB + '''');
          SQL.Add( 'where HFId > 0 ');
          SQL.Add( '  union all ');
          SQL.Add( 'select distinct NF1Id, NF1' );
          SQL.Add( 'from ''' + RAW_DB + '''');
          SQL.Add( 'where NF1Id > 0 ');
          SQL.Add( '  union all ');
          SQL.Add( 'select distinct NF2Id, NF2' );
          SQL.Add( 'from ''' + RAW_DB + '''');
          SQL.Add( 'where NF2Id > 0 ');
          Open;
          First;
          while not Eof do begin
            if not FachTable.Locate('FachId', RawQuery['HFId'],
                                    [loPartialKey] ) then begin
              FachTable.Append;
              FachTable['FachId']       := RawQuery['HFId'];
              FachTable['Fach']         := RawQuery['HF'];
              FachTable['FakultaetsId'] := 999; // keineFak. ...
              FachTable.Post;
            end
            else begin
              // ev Fach, Fakultaet prüfen ?!!!!!
            end;
            Next;
          end;
          Active := False; // FachTable schliessen
        end; // ... with SDM.RawQuery do
      end
      else begin
        // Msg: FachTable ...
      end
    end
    else begin
      // Msg: RawTable ...
    end;
  end;
end;

//---------------------------------------------------------------------
procedure TProcessThr.MakeStudis;
// je Rec. in RawTable einen Studi def. -> wenn Info : to StudiList
// -> Validierung der Keys muss vorher gemacht werden!

var
  i: Integer;
  TmpStudi: TStudi;
```

```
    //-------------------------------------------------
    function GetIntVal( FieldName: String ) : Integer;
    begin
      if SDM.RawTable[Fieldname] <> NULL    // *
        then Result := SDM.RawTable[Fieldname]
        else Result := -1;
    end;
    //-------------------------------------------------
    function GetStrVal( FieldName: String ) : String;
    begin
      if SDM.RawTable[Fieldname] > ''
        then Result := SDM.RawTable[Fieldname]
        else Result := '';
    end;
    //-------------------------------------------------
begin
  with SDM do begin
    if TryOpenTable( RawTable ) then begin
      i := 0;
      ProgressDlg.InitProgBar( RawTable.RecordCount );
      while not RawTable.Eof do begin
        inc( i );
        ShowProgress( i );
        if ( (RawTable['HFId']  = 710) or
             (RawTable['NF1Id'] = 710) or
             (RawTable['NF2Id'] = 710) ) then begin
          TmpStudi := TStudi.Create;
          with TmpStudi do begin
            MatrikelNr    := GetIntVal( 'MatrikelNr'     );
            AnredeId      := GetIntVal( 'AnredeId'       );
            Anrede        := GetStrVal( 'Anrede'         );
            TmpStudi.Name := GetStrVal( 'Name'      ); // with SDM ..
            Vorname       := GetStrVal( 'Vorname'        );
            Adresse_1     := GetStrVal( 'Adresse_1'      );
            Adresse_2     := GetStrVal( 'Adresse_2'      );
            if SDM.RawTable['Land'] > ''
              then Land := RawTable['Land']
              else Land := 'CH';
            PLZ           := GetIntVal( 'PLZ'            );
            Ort           := GetStrVal( 'Ort'            );
            HeimatOrt     := GetStrVal( 'HeimatOrt'      );
            ElternWohnort := GetStrVal( 'ElternWohnort'  ); // ...
            GeburtsDatum  := GetStrVal( 'GeburtsDatum'   );
            Tel           := GetStrVal( 'Tel'            );
            SprachId      := GetIntVal( 'SprachId'       );
            Sprache       := GetStrVal( 'Sprache'        );
            StudienZielId := GetIntVal( 'StudienZielId'  );
            StudienZiel   := GetStrVal( 'StudienZiel'    );
            ImmatStatusId := GetIntVal( 'ImmatStatusId'  );
            ImmatStatus   := GetStrVal( 'ImmatStatus'    );
            KategorieId   := GetIntVal( 'KategorieId'    );
            Kategorie     := GetStrVal( 'Kategorie'      );
            HFId          := GetIntVal( 'HFId'           );
            HF            := GetStrVal( 'HF'             );
            HFSemAnzahl   := GetIntVal( 'HFSemAnzahl'    );
            NF1Id         := GetIntVal( 'NF1Id'          );
            NF1           := GetStrVal( 'NF1'            );
            NF1SemAnzahl  := GetIntVal( 'NF1SemAnzahl'   );
            NF2Id         := GetIntVal( 'NF2Id'          );
            NF2           := GetStrVal( 'NF2'            );
            NF2SemAnzahl  := GetIntVal( 'NF2SemAnzahl'   );
          end;
          StudiList.Add( TmpStudi );
        end;  // else: kein Info-Studi
        RawTable.Next;
      end;
    end
    else begin
      // err log?
    end;
  end;
  // ShowMessage( 'StudiList.Count: ' + IntToStr( StudiList.Count ) );
end;

//-----------------------------------------------------------------
```

```
procedure TProcessThr.SaveStudis;
// StudiListe durchgehen & je Studi.Save aufrufen
var
  i: Integer; S: String;
  Studi: TStudi;
begin
  ProgressDlg.InitProgBar( StudiList.Count );
  with SDM do begin
    if SDM.TryOpenTable( SDM.StudiTable ) then begin
      for i := 0 to StudiList.Count-1 do begin
        ShowProgress( i + 1 );
        Studi := TStudi( StudiList.Items[i] );
        Studi.Save;
      end;
    end;
    SDM.StudiTable.Active := false;
  end;
end;

//---------------------------------------------------------------------
function TProcessThr.MakeNewKey( var IdSet: ByteSet ): Integer;
// return an unused Key & add it to IdSet
var i: Integer;
begin
  i := 1;
  while i in IdSet do inc(i);
  IdSet := IdSet + [i];
  Result := i;
end;

//---------------------------------------------------------------------
procedure TProcessThr.SetLabel( S: String );
begin
  ProgressDlg.Label2.Caption := S;
end;

//---------------------------------------------------------------------
procedure TProcessThr.ShowProgress(i: Integer);
begin
  ProgressDlg.ProgBar.Position := i;
  ProgressDlg.Label2.Caption := '... Studi ' + IntToStr(i)
               + ' of total ' + IntToStr( ProgressDlg.ProgBar.Max );
end;

//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : ProgrDlg.pas
// modaler Dlg mit Fortschrittsanzeige
// ruft die Import-Threads auf
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit ProgrDlg;

```
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls, Grids, DBGrids;

type
  //---------------------------------------------------------------------
  TProgressDlg = class(TForm)
    ProgBar: TProgressBar;
    Label1: TLabel;
    Label2: TLabel;
    OKBtn: TButton;
    procedure OKBtnClick(Sender: TObject);
  public
    procedure StartPersImport( Lines: TStringList );
    procedure StartProcessing;
    procedure StartLeistImport( Lines: TStringList );
    procedure InitProgBar(Maximum: Integer);
  end;

var
  ProgressDlg: TProgressDlg;

implementation
{$R *.DFM}
uses
  ImpMainForm, RawImpThr, ProcessThr, TatenImpThr, StudiDataMod;

//---------------------------------------------------------------------
procedure TProgressDlg.StartPersImport( Lines: TStringList );
var
  RawImportThread: TRawImportThr;
begin
  if not Lines.Count > 0 then
    // err.
  else
  begin
    Caption := 'Personendaten importieren';
    InitProgBar( Lines.Count - 1 );
    OKBtn.Enabled := False;
    RawImportThread := TRawImportThr.Make( Lines );
    ShowModal;  // Dlg. anzeigen & Rest blockieren
  end;
end;

//---------------------------------------------------------------------
procedure TProgressDlg.StartLeistImport( Lines: TStringList );
var
  TatenImpThr: TTatenImportThr;
begin
  if not Lines.Count > 0 then
    // err.
  else
  if SDM.TryOpenTable( SDM.PersLeistTable ) then begin
    Label1.Caption := 'Leistungsdaten importieren';
    InitProgBar( Lines.Count - 1 );
    OKBtn.Enabled := False;
    TatenImpThr := TTatenImportThr.Make( Lines );
    ShowModal;
  end;
end;
```

```
//---------------------------------------------------------------------
procedure TProgressDlg.StartProcessing;
var
  ProcessThread: TProcessThr;
begin
  Caption := Caption + ' processing data ';
  OKBtn.Enabled := False;
  ProcessThread := TProcessThr.Make;
  ShowModal;
end;

//---------------------------------------------------------------------
procedure TProgressDlg.InitProgBar(Maximum: Integer);
begin
    ProgBar.Max := Maximum;
    ProgBar.Position := 0;
    ProgBar.Step := 1;
end;

//---------------------------------------------------------------------
procedure TProgressDlg.OKBtnClick(Sender: TObject);
begin
  Close;
end;

//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : PropertiesDlg.pas
// Dlg zum Anzeigen einer einz. Node des Strukturbaums
// -> Dlg lädt nur 1x die Leistg.Liste & bleibt bestehen
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit PropertiesDlg;

```
interface

uses
  Node, Leistung, StudiDataMod,
  Windows, SysUtils, Classes, Graphics, Forms, Controls, StdCtrls,
  Dialogs, Buttons, ExtCtrls, Mask, Spin;

type
//---------------------------------------------------------------------
  TPropDlg = class(TForm)
    Bevel1: TBevel;
    Label2: TLabel;
    Label1: TLabel;
    OKBtn: TBitBtn;
    CancelBtn: TBitBtn;
    Edit1: TEdit;
    LeistungCombo: TComboBox;
    NeueLeistBtn: TButton;
    Label5: TLabel;
    GroupBox1: TGroupBox;
    GroupBox2: TGroupBox;
    Label4: TLabel;
    AnzNotenSpin: TSpinEdit;
    Label7: TLabel;
    LimitTestateSpin: TSpinEdit;
    Label12: TLabel;
    GewichtungSpin: TSpinEdit;
    RundenAuf: TEdit;
    Label11: TLabel;
    LimitNotenSpin: TSpinEdit;
    Label8: TLabel;
    AnzTestateSpin: TSpinEdit;
    Label3: TLabel;
    WahlNoteCheck: TCheckBox;
    WahlTestatCheck: TCheckBox;
    Label9: TLabel;
    LA1Check: TCheckBox;
    LA2Check: TCheckBox;
    Label6: TLabel;
    LA1FertigCheck: TCheckBox;
    LA2FertigCheck: TCheckBox;
    Label10: TLabel;
    ErfordNoteEdit: TEdit;
    procedure CancelBtnClick(Sender: TObject);
    procedure OKBtnClick(Sender: TObject);
    procedure NeueLeistBtnClick(Sender: TObject);
    procedure SetModifiedFlag(Sender: TObject);
    procedure ErfordNoteEditChange(Sender: TObject);
    procedure RundenAufChange(Sender: TObject);
    procedure ErfordNoteEditExit(Sender: TObject);
    procedure FloatEditExit( Edit: TEdit; Min, Max: Real );
    procedure RundenAufExit(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormShow(Sender: TObject);
  public
    TheNode: TNode;
    NodeModified: Boolean;
  private
    LeistList: TList;
    procedure LoadLeistList;
    procedure FillLeistungCombo;
    procedure SetToDlgValues;
```

```
    procedure SetToNodeValues;
    procedure SetToLeistungCombo;
    procedure SetEditToFloat( Edit: TEdit; R: Real );
    function GetItemIndex( LId: Integer ) : Integer;
  end;
//---------------------------------------------------------------------

var
  PropDlg: TPropDlg;

implementation
{$R *.DFM}

//---------------------------------------------------------------------
procedure TPropDlg.FormCreate(Sender: TObject);
begin
  LeistList := TList.Create;
  LoadLeistList;
  FillLeistungCombo;
end;

//---------------------------------------------------------------------
procedure TPropDlg.FormShow(Sender: TObject);
begin
  SetToNodeValues;
  NodeModified := false;
end;

//---------------------------------------------------------------------
// kein TLeistung.Load, da nie einz. Leistungen geladen werden
procedure TPropDlg.LoadLeistList;
var Leist: TLeistung;
    Abk, Zusatz: String;
begin
  with SDM do begin
    if TryOpenTable( LeistungTable ) then begin
      //
      while not LeistungTable.Eof do begin
        //
        if not ( LeistungTable['LeistAbkürzung'] = NULL ) then
          Abk := LeistungTable['LeistAbkürzung'];
        if not ( LeistungTable['LeistZusatz'] = NULL ) then
          Zusatz := LeistungTable['LeistZusatz'];
        Leist := TLeistung.Make( LeistungTable['LeistId'],
                                 LeistungTable['LeistText'],
                                 Abk, Zusatz );
        LeistList.Add( Leist );
        LeistungTable.Next;
      end; // while
    end
    else begin // open LeistungTable ...
    end;
  end;  // with
end;

//---------------------------------------------------------------------
procedure TPropDlg.FillLeistungCombo;
var i: Integer;
begin
 for i := 0 to LeistList.Count-1 do
  LeistungCombo.Items.Add( TLeistung(LeistList.Items[i]).LeistText );
end;

//---------------------------------------------------------------------
procedure TPropDlg.SetToNodeValues;
var i: Integer;
begin
  LeistungCombo.ItemIndex := GetItemIndex( TheNode.LeistId );

  AnzNotenSpin.Value   := TheNode.AnzNoten;
  AnzTestateSpin.Value := TheNode.AnzTestate;

  if TheNode.WahlNote
    then WahlNoteCheck.State := cbChecked
    else WahlNoteCheck.State := cbUnchecked;
```

```
  if TheNode.WahlTestat
    then WahlTestatCheck.State := cbChecked
    else WahlTestatCheck.State := cbUnchecked;

  if TheNode.LA1
    then LA1Check.State := cbChecked
    else LA1Check.State := cbUnchecked;
  if TheNode.LA2
    then LA2Check.State := cbChecked
    else LA2Check.State := cbUnchecked;
  if TheNode.LA1Fertig
    then LA1FertigCheck.State := cbChecked
    else LA1FertigCheck.State := cbUnchecked;
  if TheNode.LA2Fertig
    then LA2FertigCheck.State := cbChecked
    else LA2FertigCheck.State := cbUnchecked;

  LimitTestateSpin.Value := TheNode.ZeitlimiteTestate;
  LimitNotenSpin.Value := TheNode.ZeitlimiteNoten;
  SetEditToFloat( ErfordNoteEdit, TheNode.ErfordNote );
  SetEditToFloat( RundenAuf, TheNode.RundenAuf );
  GewichtungSpin.Value := TheNode.Gewichtung;

  Edit1.Text := TheNode.GetLongString;    // 4 debugging
end;

//---------------------------------------------------------------------
function TPropDlg.GetItemIndex( LId: Integer ) : Integer;
var i: Integer;
begin
  Result := -1;
  for i := 0 to LeistList.Count-1 do
    if ( TLeistung( LeistList.Items[i] ) ).LeistId = LId then
      Result := i;
end;

//---------------------------------------------------------------------
procedure TPropDlg.SetToLeistungCombo;
var L: TLeistung;
begin
  L := TLeistung( LeistList.Items[ LeistungCombo.ItemIndex ] );
  TheNode.NodeText := L.LeistText;
  TheNode.LeistId := L.LeistId;
end;

//---------------------------------------------------------------------
procedure TPropDlg.SetToDlgValues;
var i: Integer;
begin
  SetToLeistungCombo;
  TheNode.AnzTestate := AnzTestateSpin.Value;
  TheNode.AnzNoten   := AnzNotenSpin.Value;
  TheNode.WahlNote   := WahlNoteCheck.State = cbChecked;
  TheNode.WahlTestat := WahlTestatCheck.State = cbChecked;
end;

//---------------------------------------------------------------------
procedure TPropDlg.OKBtnClick(Sender: TObject);
begin
  if NodeModified then begin
    SetToDlgValues;
    ModalResult := mrOK;
  end
  else ModalResult := mrCancel;
end;

//---------------------------------------------------------------------
procedure TPropDlg.CancelBtnClick(Sender: TObject);
begin
  ModalResult := mrCancel;
end;

//---------------------------------------------------------------------
procedure TPropDlg.NeueLeistBtnClick(Sender: TObject);
begin
```

```
// NeuDialog
end;

//---------------------------------------------------------------------
procedure TPropDlg.SetModifiedFlag(Sender: TObject);
begin
  NodeModified := true;
end;

//---------------------------------------------------------------------
procedure TPropDlg.ErfordNoteEditChange(Sender: TObject);
begin
  // check if val_ok  -> zw 1 & 6, 0.1 incr
  SetModifiedFlag( Sender );
end;

//---------------------------------------------------------------------
procedure TPropDlg.RundenAufChange(Sender: TObject);
begin
  // check if val_ok
  SetModifiedFlag( Sender );
end;

//---------------------------------------------------------------------
procedure TPropDlg.FloatEditExit( Edit: TEdit; Min, Max: Real );
var
  R: Real;
  Code: Integer;
begin
  if Length( Edit.Text ) > 0 then begin
    // SysUtils ...
    Val( Edit.Text, R, Code );
    if Code <> 0 then begin
      ShowMessage( 'Fehler an Position: ' + IntToStr(Code) );
    Edit.SetFocus;
    end
    else begin
      if R > Max then R := Max;
      if R < Min then R := Min;
      SetEditToFloat( Edit, R );
    end;
  end;
end;

//---------------------------------------------------------------------
procedure TPropDlg.SetEditToFloat( Edit: TEdit; R: Real );
begin
  Edit.Text := FloatToStrF( Abs(R), ffFixed, 7, 2 );
end;

//---------------------------------------------------------------------
procedure TPropDlg.ErfordNoteEditExit(Sender: TObject);
begin
  FloatEditExit( ErfordNoteEdit, 0.0, 6.0 );
end;

//---------------------------------------------------------------------
procedure TPropDlg.RundenAufExit(Sender: TObject);
begin
  FloatEditExit( RundenAuf, 0.1, 1.0 );
end;

//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : RawImpThr.pas
// Raw-Import-Thread:
// Import raw pers.data from PhilNat.txt
//  & put it into RawTable
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit RawImpThr;

```
interface

uses
  StudiDataMod, Tokenizers,
  Classes, SysUtils, Dialogs;

type
  //-------------------------------------------------------------------
  TRawImportThr = class( TThread )
    constructor Make( Lines: TStringList );
    procedure ImportRawData;
  private
    InputLines: TStringList;
    procedure Execute; override;
    procedure ShowProgress(i: Integer);
  end;

implementation

uses
  ImpMainForm, ProgrDlg;

//---------------------------------------------------------------------
constructor TRawImportThr.Make( Lines: TStringList );
begin
  Create(False);
  FreeOnTerminate := True;
  InputLines := Lines;
end;

//---------------------------------------------------------------------
procedure TRawImportThr.Execute;
begin
  ProgressDlg.Label1.Caption := ' creating RawImport table ... ';
  SDM.MakeRawTable;
  ProgressDlg.Label1.Caption := ' importing data ... ';
  ImportRawData;
  ProgressDlg.OKBtn.Enabled := True;
end;

//---------------------------------------------------------------------
procedure TRawImportThr.ImportRawData;   // from 'InputLines'
var
  PTok:          TPersTokenizer;
  i:             Integer;
  NofErr:        Integer;
  Msg:           String;
begin
  NofErr := 0;
  with SDM do begin
    TryOpenTable ( RawTable );
    // 1. Zeile ('0.') : Headers ... ev Kontrolle
    for i := 1 to InputLines.Count-1 do begin
      ShowProgress(i);
      PTok := TPersTokenizer.Create( Inputlines[i] );
      if PTok.Error then begin
        inc( NofErr );          // -> ErrLog mit LineNr
      end
      else begin
        with RawTable do begin
          // prüf, ob gl. MatrNr. schon vorhanden & pos. cursor
```

```
          if Locate( 'MatrikelNr', PTok.MatrikelNr, [loPartialKey] )
          then begin
            Edit;
            if PTok.FachStatus = 'Hauptfach' then  begin
              FieldValues['HFId']         := PTok.FachId;
              FieldValues['HF']           := PTok.Fach;
              FieldValues['HFSemAnzahl']  := PTok.SemAnzahl;
            end
            // wenn noch kein NF eingetragen
            // 1. angetroffenes NF wird '1. NF'; keine andere Angabe
            else if not FieldValues['NF1Id'] = NULL then begin
              FieldValues['NF1Id']        := PTok.FachId;
              FieldValues['NF1']          := PTok.Fach;
              FieldValues['NF1SemAnzahl'] := PTok.SemAnzahl;
            end
            else begin
              FieldValues['NF2Id']        := PTok.FachId;
              FieldValues['NF2']          := PTok.Fach;
              FieldValues['NF2SemAnzahl'] := PTok.SemAnzahl;
            end;
            Post;
          end
          else begin       // noch kein Rec. mit gl. Matr.Nr
            Append;
            FieldValues['MatrikelNr']     := PTok.MatrikelNr;
            FieldValues['AnredeId']       := PTok.AnredeId;
            FieldValues['Anrede']         := PTok.Anrede;
            FieldValues['BriefAnrede']    := PTok.BriefAnrede;
            FieldValues['Name']           := PTok.Name;
            FieldValues['Vorname']        := PTok.Vorname;
            FieldValues['Adresse_1']      := PTok.Adresse_1;
            FieldValues['Adresse_2']      := PTok.Adresse_2;
            FieldValues['PLZ']            := PTok.PLZ;
            FieldValues['Land']           := PTok.Land;
            FieldValues['Ort']            := PTok.Ort;
            FieldValues['GeburtsDatum']   := PTok.GeburtsDatum;
            FieldValues['HeimatOrt']      := PTok.HeimatOrt;
            FieldValues['Tel']            := PTok.Tel;
            FieldValues['SprachId']       := PTok.SprachId;
            FieldValues['Sprache']        := PTok.Sprache;
            FieldValues['StudienZielId']  := PTok.StudienZielId;
            FieldValues['StudienZiel']    := PTok.StudienZiel;
            FieldValues['SemesterId']     := PTok.SemesterId;
            FieldValues['SemesterBez']    := PTok.SemesterBez;
            FieldValues['ImmatStatusId']  := PTok.ImmatStatusId;
            FieldValues['ImmatStatus']    := PTok.ImmatStatus;
            //  KategorieId bleibt offen
            FieldValues['Kategorie']      := PTok.Kategorie;
            FieldValues['ElternWohnort']  := PTok.ElternWohnort;
            FieldValues['StudienJahr']    := PTok.StudienJahr;
            if PTok.FachStatus = 'Hauptfach' then begin
              FieldValues['HFId']         := PTok.FachId;
              FieldValues['HF']           := PTok.Fach;
              FieldValues['HFSemAnzahl']  := PTok.SemAnzahl;
            end
            else begin // FCFS ...
              FieldValues['NF1Id']        := PTok.FachId;
              FieldValues['NF1']          := PTok.Fach;
              FieldValues['NF1SemAnzahl'] := PTok.SemAnzahl;
            end;
            Post;
          end;
        end;
      end;
    end;
  end;
  Msg:= '-->  Imported : '
      + IntToStr( InputLines.Count - 1 ) + ' Lines'#13#10
      + '     -->  Errors  : ' + IntToStr( NofErr );
  ProgressDlg.Label2.Caption := Msg;
end;

//---------------------------------------------------------------------
procedure TRawImportThr.ShowProgress(i: Integer);
begin
```

```
  ProgressDlg.ProgBar.Position := i;
  ProgressDlg.Label2.Caption := ' importing line ' + IntToStr(i)
                  + ' of total ' + IntToStr(InputLines.Count - 1);
end;

//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : Reglement.pas
// einz. Reglement: mehrere Abschluesse in Liste
// vom RegleForm aus: Verwaltung, Modifikation der zugehoerigen Absch
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit Reglement;

```
interface

uses
  Abschluss, AbschlussForm, StudiDataMod, NeuDialog,
  MigrationDataMod,
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, ComCtrls, Spin;

type
  //---------------------------------------------------------------------
  TReglement = class( TObject )
    RegName: String;
    RegId: Integer;
    // Datum, ...
    AbschList: TList;
  public
    constructor Create( RName: String; RId: Integer );
    constructor MakeNew( RName: String; RId: Integer );
    procedure LoadAbschluesse;
    procedure SaveAbschluesse;  //-

  end;

//---------------------------------------------------------------------
  TRegleForm = class(TForm)
    AbschListBox: TListBox;
    Quit: TButton;
    ViewAbschBtn: TBitBtn;
    NeuerAbschBtn: TBitBtn;
    RegleCombo: TComboBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    NeuesReglementBtn: TButton;
    Label4: TLabel;
    DelAbschBtn: TButton;

    procedure SetFormToReglement( ReglementsId: Integer );
    procedure ViewAbschBtnClick(Sender: TObject);
    procedure NeuerAbschBtnClick(Sender: TObject);
    procedure QuitClick(Sender: TObject);

    procedure AbschListBoxClick(Sender: TObject);
    procedure AbschListBoxDblClick(Sender: TObject);
    procedure RegleComboChange(Sender: TObject);
    procedure NotYetImplementedMsg(Sender: TObject);
  public
    Selection: TAbschluss;
    constructor Create( AOwner: TComponent ); override;
  private
    TheRegle: TReglement;
    SomethingSelected: Boolean;
    procedure FillListBox;
    procedure FillRegleCombo;
    procedure SetComboTo( ReglementsId: Integer );
    function AbschIdNotUsed( Id: Integer ): Boolean;
  end;

//---------------------------------------------------------------------
var
  RegleForm: TRegleForm;
  DEFAULT_REGLE: Integer;         // vorl. nur 1 Reglement gl.zeitig
```

```
implementation
{$R *.DFM}

//---------------------------------------------------------------------
constructor TReglement.Create( RName: String; RId: Integer );
begin
  inherited Create;
  RegName := RName;
  RegId := RId;
  AbschList := TList.Create;
//  LoadAbschluesse;
end;


//---------------------------------------------------------------------
constructor TReglement.MakeNew( RName: String; RId: Integer );
// ev. mehr Param.:  Date, ... ?
begin
  Create( RName, RId );
  //  MakeDBEntrys in RegleTable
end;

//---------------------------------------------------------------------
procedure TReglement.LoadAbschluesse;
// Abschl. des akt. Regl. laden & in AbschList fuellen
var
  TmpAbsch: TAbschluss;
begin
  with SDM do begin
    if TryOpenTable( AbschTable ) then begin
      // --> besser mit Query !!!!

      AbschList.Free;
      AbschList := TList.Create;
      AbschTable.First;                  // was, wenn leer?
      while not AbschTable.Eof do begin
        if AbschTable['RegId'] = RegId then
        begin
          TmpAbsch := TAbschluss.Make(RegId, AbschTable['AbschId']);
          //TmpAbsch := TAbschluss.Make( FieldValues['AbschName'],
          // RegId, FieldValues['AbschId'],FieldValues['StruktId'] );
          AbschList.Add( TmpAbsch );
        end;
        AbschTable.Next;
      end;
      AbschTable.Active := false;
    end;
  end;
end;

//---------------------------------------------------------------------
procedure TReglement.SaveAbschluesse;
var
  i: Integer;
  Absch: TAbschluss;
begin
  // AbschList durchgehen
  //    if Absch.Id = -1 -> neuer -> append
  //    else RootId korrigieren

  with SDM do begin
    if TryOpenTable( AbschTable ) then begin

      for i := 0 to AbschList.Count-1 do begin

        Absch := TAbschluss( AbschList.Items[i] );

        if AbschTable.Locate( 'AbschId', Absch.AbschId,
                              [loPartialKey] ) then
        begin
          AbschTable.Edit;
          AbschTable['StruktId'] := Absch.RootId;
          AbschTable.Post;
        end
```

```
        else begin
          AbschTable.Append;
          AbschTable['RegId']     := RegId;
          AbschTable['AbschId']   := Absch.AbschId;
          AbschTable['StruktId']  := Absch.RootId;
          AbschTable.Post;
        end;
      end;
      AbschTable.Active := false;
    end;
  end;
end;

////////////////////////////////////////////////////////////////////

constructor TRegleForm.Create( AOwner: TComponent );
begin
  inherited Create( Owner );
  // Default-Reglement öffnen
  SetFormToReglement( DEFAULT_REGLE );
  FillRegleCombo;
  SetComboTo( TheRegle.RegId );  // set RegleCombo to current Reglem.
  SomethingSelected := false;
end;

//---------------------------------------------------------------------
procedure TRegleForm.SetFormToReglement( ReglementsId: Integer );
begin
  TheRegle.Free;
  TheRegle := TReglement.Create( '...', ReglementsId);
  //                              ^ aus DB holen!
  TheRegle.LoadAbschluesse;   // alle Abschl. -> AbschList
  FillListBox;                // AbschList -> AbschListBox
  Label1.Caption := 'Reglement: ' + IntToStr( ReglementsId );
end;

//---------------------------------------------------------------------
procedure TRegleForm.FillRegleCombo;
begin
  with SDM do begin
    if TryOpenTable( RegleTable ) then begin
      while not RegleTable.Eof do begin
        RegleCombo.Items.Add( IntToStr( RegleTable['RegleId'] ) );
        RegleTable.Next;
      end;
    end
    else begin  // report error
    end;
  end;
end;

//---------------------------------------------------------------------
procedure TRegleForm.SetComboTo( ReglementsId: Integer );
begin
  RegleCombo.ItemIndex :=
                RegleCombo.Items.IndexOf( IntToStr( ReglementsId ) );
end;

//---------------------------------------------------------------------
procedure TRegleForm.RegleComboChange(Sender: TObject);
var Reglement: Integer;
begin
  ShowMessage( 'zZ nur ein Reglement im System'#10#13#10#13
             + '  ->  Migration: SetReglementsNr' );
  SetComboTo( TheRegle.RegId );
{ // mit allen Reglementen:
  Reglement := StrToInt( RegleCombo.Items[RegleCombo.ItemIndex] );
  SetFormToReglement( Reglement );
}
end;

//---------------------------------------------------------------------
procedure TRegleForm.FillListBox;
var
  i: Integer;
```

```
  Absch: TAbschluss;
begin
  AbschListBox.Items.Clear;
  for i := 0 to TheRegle.AbschList.Count-1 do begin
    Absch := TAbschluss( TheRegle.AbschList.Items[i] );
    AbschListBox.Items.Add( IntToStr( Absch.AbschId )
                          + ' :  ' + Absch.AbschName );
  end;
  SomethingSelected := false;
end;

//---------------------------------------------------------------------
procedure TRegleForm.ViewAbschBtnClick(Sender: TObject);
begin
  if SomethingSelected then begin
    AbschForm := TAbschForm.Make( self, Selection );
    AbschForm.ShowModal;
    AbschForm.Free;
  end
  else
    ShowMessage( 'bitte einen Abschluss auswählen!' );
end;

//---------------------------------------------------------------------
procedure TRegleForm.NeuerAbschBtnClick(Sender: TObject);
var
  Absch: TAbschluss;
  i: Integer;
  OK: Boolean;
  ModRes: Integer;
begin
  NeuDlg := TNeuDlg.Create( self );
  NeuDlg.TitleLabel.Caption := 'Neuer Abschluss';
  NeuDlg.CommentLabel.Caption := 'mind. Id und Text eingeben';
  OK := false;
  while not OK do begin
    ModRes := NeuDlg.ShowModal;
    if ModRes = mrCancel then OK := true;
    if ModRes = mrOK then begin
      if ( NeuDlg.Id > -1 ) then begin
        if ( AbschIdNotUsed( NeuDlg.Id ) ) then begin
          Absch := TAbschluss.Make( TheRegle.RegId, NeuDlg.Id );
          TheRegle.AbschList.Add( Absch );
          AbschForm := TAbschForm.Make( self, Absch );   //*
          AbschForm.ShowModal;
          OK := true;
        end
        else begin
          ShowMessage( 'Abschluss schon definiert in '
                     + 'diesem Reglement' );
        end;
      end
      else ShowMessage( 'ungültige Id' );
    end;
  end;
  NeuDlg.Free;
  FillListBox;
  // Save Absch autom ...
end;

//---------------------------------------------------------------------
function TRegleForm.AbschIdNotUsed( Id: Integer ): Boolean;
var i: Integer;
begin
  Result := true;
  for i := 0 to TheRegle.AbschList.Count-1 do
    if TAbschluss( TheRegle.AbschList.Items[i] ).AbschId = Id then
      Result := false;
end;

//---------------------------------------------------------------------
procedure TRegleForm.AbschListBoxClick(Sender: TObject);
var i: Integer;
begin
  SomethingSelected := true;
```

```
  for i := 0 to AbschListBox.Items.Count-1 do
    if AbschListBox.Selected[i] then
      Selection := TAbschluss( TheRegle.AbschList.Items[i] );
end;

//---------------------------------------------------------------------
procedure TRegleForm.AbschListBoxDblClick(Sender: TObject);
begin
  ViewAbschBtnClick( Sender );
end;

//---------------------------------------------------------------------
procedure TRegleForm.QuitClick(Sender: TObject);
begin
  TheRegle.SaveAbschluesse;
  Close;
end;

//---------------------------------------------------------------------
procedure TRegleForm.NotYetImplementedMsg(Sender: TObject);
begin
  ShowMessage( 'noch nicht implementiert' );
end;

//---------------------------------------------------------------------
initialization
  DEFAULT_REGLE := 1995;   // default single Reglem.

//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : SearchDlg.pas
// best. Studi suchen von Uebersichtsdarstellung aus
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit SearchDlg;

```
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls;

type
  //---------------------------------------------------------------------
  TSuchDlg = class(TForm)
    RadioGroup1: TRadioGroup;
    GroupBox1: TGroupBox;
    Edit1: TEdit;
    GroupBox2: TGroupBox;
    CancelBtn: TButton;
    OKBtn: TButton;
    procedure CancelBtnClick(Sender: TObject);
    procedure OKBtnClick(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure RadioGroup1Click(Sender: TObject);
  public
    Value,
    ModRes: String;
  end;

var
  SuchDlg: TSuchDlg;

implementation
{$R *.DFM}

//---------------------------------------------------------------------
procedure TSuchDlg.CancelBtnClick(Sender: TObject);
begin
  ModRes := 'cancel';
  Close;
end;

//---------------------------------------------------------------------
procedure TSuchDlg.OKBtnClick(Sender: TObject);
begin
  Value := Edit1.Text;
  Close;
end;

//---------------------------------------------------------------------
procedure TSuchDlg.FormActivate(Sender: TObject);
begin
  Value := '';
  Edit1.Text := '';
  ModRes := RadioGroup1.Items[0];
end;

//---------------------------------------------------------------------
procedure TSuchDlg.RadioGroup1Click(Sender: TObject);
begin
  ModRes := RadioGroup1.Items[RadioGroup1.ItemIndex];
end;

//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : Studi.pas
// TStudi: Kapselung einer einzelnen Studi-Entität
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit Studi;

```
interface

uses
  StudiDataMod, Taten, SearchDlg, AuswahlDialog,
  DBTables,
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Grids, DBGrids, ExtCtrls, ComCtrls;

type
  TStudi = class( TObject )
  public
    AnredeId:       Integer;
    Anrede:         String;
    Name:           String;
    Vorname:        String;
    Adresse_1:      String;
    Adresse_2:      String;
    PLZ:            Integer;
    Land:           String;
    Ort:            String;
    HeimatOrt:      String;
    ElternWohnort:  String;
    GeburtsDatum:   String;
    Tel:            String;
    SprachId:       Integer;
    Sprache:        String;
    StudienZielId:  Integer;
    StudienZiel:    String;
    ImmatStatusId:  Integer;
    ImmatStatus:    String;
    KategorieId:    Integer;
    Kategorie:      String;
    HFId:           Integer;
    HF:             String;
    HFSemAnzahl:    Integer;
    NF1Id:          Integer;
    NF1:            String;
    NF1SemAnzahl:   Integer;
    NF2Id:          Integer;
    NF2:            String;
    NF2SemAnzahl:   Integer;
    RegId:          Integer;
    RegName:        String;
    ZielAbschId:    Integer;
    ZielAbschluss:  String;
    InsertDate,
    UpdateDate,
    AbschChangeDate,
    LA1Date,
    LA2Date:        TDateTime;
  private
    MatrNr: Integer;
    function MatrNrOk( Nr: Integer ): Boolean;
    procedure SetMatrNr( Value: Integer );
  public
    TatenListe: TList;
    Error: Boolean;
    constructor Create;
    constructor Load( MatrNr: Integer );
    procedure LoadTatenListe;             // nicht automatisch
    procedure Save;
    function ToString : String;
    property MatrikelNr: Integer read MatrNr write SetMatrNr;
  end;
```

```
//---------------------------------------------------------------------

implementation

constructor TStudi.Create;
begin
  inherited Create;
  Error := false;
end;

//---------------------------------------------------------------------
constructor TStudi.Load( MatrNr: Integer );
    //-------------------------------------------------
    function GetIntVal( FieldName: String ) : Integer;
    begin
      if SDM.StudiTable[Fieldname] <> NULL    // *
        then Result := SDM.StudiTable[Fieldname]
        else Result := -1;
    end;
    //-------------------------------------------------
    function GetStrVal( FieldName: String ) : String;
    begin
      if SDM.StudiTable[Fieldname] <> NULL
        then Result := SDM.StudiTable[Fieldname]
        else Result := '';
    end;
    //-------------------------------------------------
    function GetDateTimeVal( FieldName: String ) : TDateTime;
    begin
      if SDM.StudiTable[Fieldname] <> NULL
        then Result := SDM.StudiTable[Fieldname]
        else Result := StrToDateTime( '01.01.01 0:0:0' );
    end;
    //-------------------------------------------------
begin
  Create;
  MatrikelNr := MatrNr;
  with SDM do begin
    if TryOpenTable( StudiTable ) then begin
      if StudiTable.Locate('MatrikelNr', MatrNr, [loPartialKey])
      then begin
        self.Name     := GetStrVal( 'Name' );    // with SDM do ...
        Vorname       := GetStrVal( 'Vorname'        );
        Adresse_1     := GetStrVal( 'Adresse_1'      );
        Adresse_2     := GetStrVal( 'Adresse_2'      );
        if StudiTable['Land'] > ''
          then Land   := StudiTable['Land']
          else Land   := 'CH';
        Tel           := GetStrVal( 'Tel'            );
        PLZ           := GetIntVal( 'PLZ'            );
        Ort           := GetStrVal( 'Ort'            );

        GeburtsDatum  := GetStrVal( 'GeburtsDatum'  );
        HeimatOrt     := GetStrVal( 'HeimatOrt'     );
        ElternWohnort := GetStrVal( 'ElternWohnort' );

        AnredeId      := GetIntVal( 'AnredeId'       );
        if AnredeId = 1
          then Anrede := 'Frau'
          else Anrede := 'Herr';
        SprachId      := GetIntVal( 'SprachId'       );
        StudienZielId := GetIntVal( 'StudienZielId'  );
        ImmatStatusId := GetIntVal( 'ImmatStatusId'  );
        KategorieId   := GetIntVal( 'KategorieId'    );
        RegId         := GetIntVal( 'RegId'    );
        ZielAbschId   := GetIntVal( 'ZielAbschId'    );

        HFId          := GetIntVal( 'HFId'           );
        HFSemAnzahl   := GetIntVal( 'HFSemAnzahl'    );
        NF1Id         := GetIntVal( 'NF1Id'          );
        NF1SemAnzahl  := GetIntVal( 'NF1SemAnzahl'   );
        NF2Id         := GetIntVal( 'NF2Id'          );
        NF2SemAnzahl  := GetIntVal( 'NF2SemAnzahl'   );

        InsertDate      := GetDateTimeVal( 'InsertDate' );
```

```
        UpdateDate      := GetDateTimeVal( 'UpdateDate' );
        AbschChangeDate := GetDateTimeVal( 'AbschChangeDate' );
        LA1Date         := GetDateTimeVal( 'LA1Date' );
        LA2Date         := GetDateTimeVal( 'LA2Date' );
      end
      else Error := true;
    end
    else Error := true;
    if not Error then begin
      if TryOpenTable( ImmatStatTable ) and
              TryOpenTable( LeistungTable ) and
              TryOpenTable( RegleTable ) and
              TryOpenTable( KategorieTable ) and
              TryOpenTable( StudZielTable )  and
              TryOpenTable( FachTable )      and
              TryOpenTable( SpracheTable ) then begin
        if ImmatStatTable.Locate('ImmatStatusId',
                  ImmatStatusId, [loPartialKey])
          then ImmatStatus := ImmatStatTable['ImmatStatus']
          else Error := true;
        if SpracheTable.Locate('SprachId',
                  SprachId, [loPartialKey])
          then Sprache := SpracheTable['Sprache']
          else Error := true;
        if StudZielTable.Locate('StudienZielId',
                  StudienZielId, [loPartialKey])
          then StudienZiel := StudZielTable['StudienZiel']
          else Error := true;
        if KategorieTable.Locate('KategorieId',
                  KategorieId, [loPartialKey])
          then Kategorie := KategorieTable['Kategorie']
          else Error := true;
        if FachTable.Locate('FachId', HFId, [loPartialKey]) then
          HF := FachTable['Fach'];
        if FachTable.Locate('FachId', NF1Id, [loPartialKey]) then
          NF1 := FachTable['Fach'];
        if FachTable.Locate('FachId', NF2Id, [loPartialKey]) then
          NF2 := FachTable['Fach'];
        if LeistungTable.Locate('LeistId', ZielAbschId,
                                [loPartialKey]) then
          ZielAbschluss := LeistungTable['LeistText']
        else Error := true;
        if RegleTable.Locate('RegleId', RegId, [loPartialKey]) then
          RegName := RegleTable['RegleName']
        else Error := true;
      end
      else Error := true;  // TryOpen..
    end;
  end; // with SDM
end;

//---------------------------------------------------------------------
procedure TStudi.LoadTatenListe;
var t: TTat;
begin
  TatenListe.Free;  // ?
  TatenListe := TList.Create;
  with SDM do begin
    if TryOpenTable( PersLeistTable ) then begin
      if PersLeistTable.Locate('MatrikelNr',
                               MatrikelNr, [loPartialKey]) then begin
        while ( not PersLeistTable.Eof ) and
              ( PersLeistTable['MatrikelNr'] = MatrikelNr ) do begin
          t := TTat.Create( MatrikelNr,
                            PersLeistTable['LeistId'],
                            PersLeistTable['Jahr'],
                            PersLeistTable['Termin'],
                            PersLeistTable['Note'] );
          TatenListe.Add( t );
          PersLeistTable.Next;
        end;
      end;
    end; // else 'cannot open ...'
  end;
end;
```

```
//---------------------------------------------------------------------
procedure TStudi.Save;
var NewRec: Boolean;
begin
  with SDM do begin
    if TryOpenTable( StudiTable ) then begin
      // prüfen, ob schon vorhanden & allenfalls positionieren
      if StudiTable.Locate('MatrikelNr', MatrikelNr, [loPartialKey])
      then begin
        NewRec := False;
        StudiTable.Edit;      // vorhandenen rec. updaten
      end
      else begin
        NewRec := True;
        StudiTable.Append;   // neuen anhaengen
      end;
      StudiTable['MatrikelNr']     :=  MatrikelNr;
      StudiTable['Name']           :=  self.Name;  // with SDM ...
      StudiTable['Vorname']        :=  Vorname;
      StudiTable['Adresse_1']      :=  Adresse_1;
      StudiTable['Adresse_2']      :=  Adresse_2;
      StudiTable['Tel']            :=  Tel;
      StudiTable['Land']           :=  Land;
      StudiTable['PLZ']            :=  PLZ;
      StudiTable['Ort']            :=  Ort;

      StudiTable['GeburtsDatum']   :=  GeburtsDatum;
      StudiTable['HeimatOrt']      :=  HeimatOrt;
      StudiTable['ElternWohnort']  :=  ElternWohnort;

      StudiTable['AnredeId']       :=  AnredeId;
      StudiTable['SprachId']       :=  SprachId;
      StudiTable['StudienZielId']  :=  StudienZielId;

      StudiTable['ImmatStatusId']  :=  ImmatStatusId;
      StudiTable['KategorieId']    :=  KategorieId;

      if HFId > 0 then begin
        StudiTable['HFId']           :=  HFId;
        StudiTable['HFSemAnzahl']    :=  HFSemAnzahl;
        StudiTable['ZielAbschId']    :=  80300;
        // ZielAbschId kann nur bei HF gesetzt werden;
        // sonst unbekannt -> von Hand ...
      end;
      if NF1Id > 0 then begin
        StudiTable['NF1Id']          :=  NF1Id;
        StudiTable['NF1SemAnzahl']   :=  NF1SemAnzahl;
      end;
      if NF2Id > 0 then begin
        StudiTable['NF2Id']          :=  NF2Id;
        StudiTable['NF2SemAnzahl']   :=  NF2SemAnzahl;
      end;
      // TimeStamps
      if NewRec then StudiTable['InsertDate'] := Now;
      StudiTable['UpdateDate']      := Now;
{
      StudiTable['AbschChangeDate'] := AbschChangeDate;
      StudiTable['LA1Date']         := LA1Date;
      StudiTable['LA2Date']         := LA2Date;
}
      StudiTable.Post;
    end
    else begin
      Error := true;
    end;       // Table offen lassen
  end;
end;

//---------------------------------------------------------------------
procedure TStudi.SetMatrNr( Value: Integer );
begin
  if MatrNrOk( Value ) then MatrNr := Value
  else MessageDlg( 'ungueltige MatrNr!', mtError, [mbOk], 0 );
end;
```

```
//---------------------------------------------------------------------
function TStudi.MatrNrOk( Nr: Integer ): Boolean;
var
  z1,z2,z3,z4,z5,z6,z7,z8: Integer;
  q1,q2,q3,q4,q5,q6,q7,p : Integer;
    //-----------------------------------
    function Quer(z: Integer): Integer;
    begin
      Result := -1;
      if z < 100 then  Result := (z mod 10) + (z div 10);
    end;
    //-----------------------------------
begin
  z1 := (Nr div 10000000) mod 10;
  z2 := (Nr div 1000000) mod 10;
  z3 := (Nr div 100000) mod 10;
  z4 := (Nr div 10000) mod 10;
  z5 := (Nr div 1000) mod 10;
  z6 := (Nr div 100) mod 10;
  z7 := (Nr div 10) mod 10;
  z8 := Nr mod 10;
  q1 := Quer( 2 * z1 );
  q2 := Quer( z2 );
  q3 := Quer( 2 * z3 );
  q4 := Quer( z4 );
  q5 := Quer( 2 * z5 );
  q6 := Quer( z6 );
  q7 := Quer( 2 * z7 );
  p := ( 100 - ( q1 +q2 +q3 +q4 +q5 +q6 +q7 ) ) mod 10;
  if p = z8
    then Result := true
    else Result := false;
end;

//---------------------------------------------------------------------
function TStudi.ToString : String;
var S : String;
begin
  S := IntToStr( MatrikelNr ) + ' - ';
  S := S + Name + ' - ';
  S := S + IntToStr( AnredeId ) + ' - ';
  S := S + Name  + ' - ';
  S := S + Vorname  + ' - ';
  S := S + Adresse_1  + ' - ';
  S := S + Adresse_2  + ' - ';
  S := S + IntToStr( PLZ ) + ' - ';
  S := S + Land  + ' - ';
  S := S + Ort  + ' - ';
  S := S + HeimatOrt  + ' - ';
  S := S + ElternWohnort  + ' - ';
  S := S + GeburtsDatum  + ' - ';
  S := S + Tel  + ' - ';
  S := S + IntToStr( SprachId ) + ' - ';
  S := S + IntToStr( StudienZielId ) + ' - ';
  S := S + IntToStr( ZielAbschId ) + ' - ';
  S := S + IntToStr( ImmatStatusId ) + ' - ';
  S := S + IntToStr( KategorieId ) + ' - ';
  S := S + IntToStr( HFId ) + ' - ';
  S := S + IntToStr( HFSemAnzahl ) + ' - ';
  S := S + IntToStr( NF1Id ) + ' - ';
  S := S + IntToStr( NF1SemAnzahl ) + ' - ';
  S := S + IntToStr( NF2Id ) + ' - ';
  S := S + IntToStr( NF2SemAnzahl );
  // & TimeStamps ...!
  Result := S;
end;

//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : StudiDataMod.pas
// DatenModul für Studis-3
//   Tables etc., MakeTable proc's
//   DB-Konstanten
//   TableGenerierungsProzeduren, Oeffnen, ...
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit StudiDataMod;

```
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, DB, DBTables;

type
  //-------------------------------------------------------------------
  TSDM = class(TDataModule)
    RawTable: TTable;
    StudiTable: TTable;
    SpracheTable: TTable;
    ImmatStatTable: TTable;
    KategorieTable: TTable;
    StudZielTable: TTable;
    FachTable: TTable;
    PersLeistTable: TTable;
    AbschTable: TTable;
    LeistungTable: TTable;
    RegleTable: TTable;
    StruktTable: TTable;

    RawQuery: TQuery;
    KategQuery: TQuery;
    StudiQuery: TQuery;
    TatenQuery: TQuery;

    RawDataSource: TDataSource;
    KategDataSource: TDataSource;
    StudiDataSource: TDataSource;
    PersLeistDataSource: TDataSource;
    RawDS: TDataSource;
    StudiQueryDS: TDataSource;
    TatenQueryDS: TDataSource;
    procedure SDMCreate(Sender: TObject);

  public
    function TryOpenTable( ATable: TTable ): Boolean;
    procedure MakeRawTable;
    procedure MakeStudiTable;
    procedure MakeSpracheTable;
    procedure MakeImmatStatTable;
    procedure MakeKategorieTable;
    procedure MakeStudZielTable;
    procedure MakeFachTable;
    procedure MakePersLeistTable;
    procedure MakeAbschTable;
    procedure MakeLeistTable;

    procedure MakeRegleTable;
    procedure MakeStruktTable;

  private
    procedure SetTableName( ATable: TTable );
    procedure InitTable( ATable: TTable );
  end;

var
  SDM: TSDM;
  ROOT_PATH, RAW_DB, STUDI_DB, SPRACHE_DB, IMMSTAT_DB,
```

```
  KATEGORIE_DB, STUDZIEL_DB, FACH_DB, FAKULTAET_DB,
  PERSLEIST_DB, LEISTUNG_DB,  // neues Format!
  REGLE_DB, ABSCH_DB, STRUKT_DB  : String;


implementation
{$R *.DFM}

//---------------------------------------------------------------------
procedure TSDM.SetTableName( ATable: TTable );
// setzt TableName auf den effektiven File-Name
begin
  with ATable do begin
    if not Active then begin
      if Name = 'RawTable'            then TableName := RAW_DB
      else if Name = 'StudiTable'     then TableName := STUDI_DB
      else if Name = 'SpracheTable'   then TableName := SPRACHE_DB
      else if Name = 'ImmatStatTable' then TableName := IMMSTAT_DB
      else if Name = 'KategorieTable' then TableName := KATEGORIE_DB
      else if Name = 'StudZielTable'  then TableName := STUDZIEL_DB
      else if Name = 'FachTable'      then TableName := FACH_DB
      else if Name = 'PersLeistTable' then TableName := PERSLEIST_DB
      else if Name = 'AbschTable'     then TableName := ABSCH_DB
      else if Name = 'LeistungTable'  then TableName := LEISTUNG_DB

      else if Name = 'RegleTable'     then TableName := REGLE_DB
      else if Name = 'StruktTable'    then TableName := STRUKT_DB;

    end;
    // case-statement nur mit ordinal-typen :-(
  end;
end;

//---------------------------------------------------------------------
function TSDM.TryOpenTable( ATable: TTable ): Boolean;
// öffnet ATable (Active setzen)
begin
  if not ATable.Active = True then begin    // noch nicht aktiv
    SetTableName( ATable );                 // Name setzen
    // ATable.Exclusive := True;   // ****
    try
      ATable.Active := True;                // öffnen
    except
      on EDBEngineError do begin
        ShowMessage('ERROR: kann ' + ATable.Name + ' nicht öffnen!');
        ATable.Active := False;
      end;
    end;
  end;
  if ATable.Active = True then begin
    ATable.First;                           // auf 1. Rec. pos.
    Result := True;
  end
  else Result := False;
end;

//---------------------------------------------------------------------
procedure TSDM.InitTable( ATable: TTable );
// löscht Table (falls vorhanden), FieldDefs, IndexDefs
begin
  SetTableName( ATable );
  with ATable do begin
    Active := False;
    TableType := ttParadox;
    // (kompletter Pfad in TableName)
    if FileExists( ATable.TableName )
      then DeleteTable
      else CreateTable;
    FieldDefs.Clear;
    IndexDefs.Clear;
  end;
end;

//---------------------------------------------------------------------
// folgt Erstellung der neuen Tables
```

```
//---------------------------------------------------------------------
procedure TSDM.MakeRawTable;
begin
  InitTable( RawTable );
  with RawTable do begin
    with FieldDefs do begin
      // Add(Name:Str.; DataType: TFieldType; Size:Word; Req.: Bool.)
      // ->  Size nur bei ftString oder ftBlob; für numerische: 0
      Add( 'MatrikelNr',    ftInteger,     0,     True  );
      Add( 'Name',          ftString,      24,    True  );
      Add( 'Vorname',       ftString,      24,    True  );

      Add( 'AnredeId',      ftSmallint,    0,     False );
      Add( 'Anrede',        ftString,      10,    False );
      Add( 'BriefAnrede',   ftString,      24,    False );
      Add( 'Adresse_1',     ftString,      32,    False );
      Add( 'Adresse_2',     ftString,      32,    False );
      Add( 'Land',          ftString,      4,     False );
      Add( 'PLZ',           ftInteger,     0,     False );
      Add( 'Ort',           ftString,      32,    False );
      Add( 'GeburtsDatum',  ftString,      10,    False );
      Add( 'HeimatOrt',     ftString,      32,    False );
      Add( 'Tel',           ftString,      20,    False );
      Add( 'SprachId',      ftSmallint,    0,     False );
      Add( 'Sprache',       ftString,      24,    False );
      Add( 'StudienZielId', ftSmallint,    0,     False );
      Add( 'StudienZiel',   ftString,      32,    False );

      Add( 'HFId',          ftSmallint,    0,     False );
      Add( 'HF',            ftString,      30,    False );
      Add( 'HFSemAnzahl',   ftSmallint,    0,     False );

      Add( 'NF1Id',         ftSmallint,    0,     False );
      Add( 'NF1',           ftString,      30,    False );
      Add( 'NF1SemAnzahl',  ftSmallint,    0,     False );

      Add( 'NF2Id',         ftSmallint,    0,     False );
      Add( 'NF2',           ftString,      30,    False );
      Add( 'NF2SemAnzahl',  ftSmallint,    0,     False );

      Add( 'SemesterId',    ftSmallint,    0,     False );
      Add( 'SemesterBez',   ftString,      25,    False );
      Add( 'ImmatStatusId', ftSmallint,    0,     False );
      Add( 'ImmatStatus',   ftString,      25,    False );

      Add( 'KategorieId',   ftSmallint,    0,     False );
      Add( 'Kategorie',     ftString,      75,    False );

      Add( 'ElternWohnort', ftString,      32,    False );
      Add( 'StudienJahr',   ftSmallint,    0,     False );
    end;
    IndexDefs.Add('Key', 'MatrikelNr', [ixPrimary, ixUnique]);
    CreateTable;
  end;
end;

//---------------------------------------------------------------------
procedure TSDM.MakeStudiTable;
begin
  InitTable( StudiTable );
  with StudiTable do begin
    with FieldDefs do begin
      Add( 'MatrikelNr',    ftInteger,     0,     True  );
      Add( 'Name',          ftString,      24,    True  );
      Add( 'Vorname',       ftString,      24,    True  );
      Add( 'Adresse_1',     ftString,      32,    False );
      Add( 'Adresse_2',     ftString,      32,    False );
      Add( 'Tel',           ftString,      20,    False );
      Add( 'Land',          ftString,      4,     False );
      Add( 'PLZ',           ftInteger,     0,     False );
      Add( 'Ort',           ftString,      32,    False );
//
      Add( 'GeburtsDatum',  ftString,      10,    False );
      Add( 'HeimatOrt',     ftString,      32,    False );
      Add( 'ElternWohnort', ftString,      32,    False );
```

```
//
      Add( 'AnredeId',      ftSmallint,    0,     False );
      Add( 'SprachId',      ftSmallint,    0,     False );
      Add( 'StudienZielId', ftSmallint,    0,     False );
      Add( 'ImmatStatusId', ftSmallint,    0,     False );
      Add( 'KategorieId',   ftSmallint,    0,     False );

      Add( 'ZielAbschId',   ftInteger,     0,     False );
      Add( 'RegId',         ftInteger,     0,     False );

      Add( 'HFId',          ftSmallint,    0,     False );
      Add( 'HFSemAnzahl',   ftSmallint,    0,     False );
      Add( 'NF1Id',         ftSmallint,    0,     False );
      Add( 'NF1SemAnzahl',  ftSmallint,    0,     False );
      Add( 'NF2Id',         ftSmallint,    0,     False );
      Add( 'NF2SemAnzahl',  ftSmallint,    0,     False );

      Add( 'InsertDate',      ftDateTime,  0,     False );
      Add( 'UpdateDate',      ftDateTime,  0,     False );
      Add( 'AbschChangeDate', ftDateTime,  0,     False );
      Add( 'LA1Date',         ftDateTime,  0,     False );
      Add( 'LA2Date',         ftDateTime,  0,     False );
    end;

    with IndexDefs do begin
      Add('Key', 'MatrikelNr', [ixPrimary, ixUnique]);
      Add('NameIndex', 'Name', [ixCaseInsensitive]);
      Add('VornameIndex', 'Vorname', [ixCaseInsensitive]);
    end;
    CreateTable;
  end;
end;

//---------------------------------------------------------------------
procedure TSDM.MakeSpracheTable;
begin
  InitTable( SpracheTable );
  with SpracheTable do begin
    with FieldDefs do begin
      Add( 'SprachId',      ftSmallint,    0,     True  );
      Add( 'Sprache',       ftString,      24,    True  );
    end;
    IndexDefs.Add('Key', 'SprachId', [ixPrimary, ixUnique]);
    CreateTable;
  end;
end;

//---------------------------------------------------------------------
procedure TSDM.MakeImmatStatTable;
begin
  InitTable( ImmatStatTable );
  with ImmatStatTable do begin
    with FieldDefs do begin
      Add( 'ImmatStatusId', ftSmallint,    0,     True  );
      Add( 'ImmatStatus',   ftString,      30,    True  );
    end;
    IndexDefs.Add('Key', 'ImmatStatusId', [ixPrimary, ixUnique]);
    CreateTable;
  end;
end;

//---------------------------------------------------------------------
procedure TSDM.MakeKategorieTable;
begin
  InitTable( KategorieTable );
  with KategorieTable do begin
    with FieldDefs do begin
      Add( 'KategorieId',   ftSmallint,    0,     True  );
      Add( 'Kategorie',     ftString,      75,    True  );
    end;
    IndexDefs.Add('Key', 'KategorieId', [ixPrimary, ixUnique]);
    CreateTable;
  end;
end;
```

```
//---------------------------------------------------------------------
procedure TSDM.MakeStudZielTable;
begin
  InitTable( StudZielTable );
  with StudZielTable do begin
    with FieldDefs do begin
      Add( 'StudienZielId', ftSmallint,    0,     True  );
      Add( 'StudienZiel',   ftString,      48,    True  );
    end;
    IndexDefs.Add('Key', 'StudienZielId', [ixPrimary, ixUnique]);
    CreateTable;
  end;
end;

//---------------------------------------------------------------------
procedure TSDM.MakeFachTable;
begin
  InitTable( FachTable );
  with FachTable do begin
    with FieldDefs do begin
      Add( 'FachId',          ftSmallint,    0,   True  );
      Add( 'Fach',            ftString,     48,   True  );
      Add( 'FakultaetsId',    ftSmallint,    0,   True  );
    end;
    IndexDefs.Add('Key', 'FachId', [ixPrimary, ixUnique]);
    CreateTable;
  end;
end;

//---------------------------------------------------------------------
procedure TSDM.MakePersLeistTable;
begin
  InitTable( PersLeistTable );
  with PersLeistTable do begin
    with FieldDefs do begin
      Add( 'MatrikelNr',  ftInteger,     0,     True );
      Add( 'LeistId',     ftInteger,     0,     True );
      Add( 'Jahr',        ftSmallint,    0,     True );
      Add( 'Termin',      ftSmallint,    0,     True );
      Add( 'Note',        ftFloat,       0,     True );
    end;
    IndexDefs.Add('Key',
                  'MatrikelNr;LeistId;Jahr;Termin;Note',
                  [ixPrimary, ixUnique]);
    CreateTable;
  end;
end;

//---------------------------------------------------------------------
procedure TSDM.MakeAbschTable;
begin
  InitTable( AbschTable );
  with AbschTable do begin
    with FieldDefs do begin
      Add( 'AbschId',        ftInteger,     0,     True  );
      Add( 'StruktId',       ftInteger,     0,     True  );
      Add( 'RegId',          ftInteger,     0,     True  );
//      Add( 'AbschName',      ftString,     20,     True  );
    end;
    IndexDefs.Add( 'MNIndex', 'AbschId', [ixPrimary, ixUnique] );
    CreateTable;
  end;
end;

//---------------------------------------------------------------------
procedure TSDM.MakeLeistTable;
begin
  InitTable( LeistungTable );
  with LeistungTable do begin
    with FieldDefs do begin
      Add( 'LeistId',        ftInteger,     0,     True  );
      Add( 'LeistText',      ftString,     72,     True  );
      Add( 'LeistAbkürzung', ftString,      5,     False );
      Add( 'LeistZusatz',    ftString,     25,     False );
    end;
```

```
    IndexDefs.Add('MNIndex', 'LeistId', [ixPrimary, ixUnique]);
    CreateTable;
  end;
end;

//---------------------------------------------------------------------
procedure TSDM.MakeRegleTable;
begin
  InitTable( RegleTable );
  with RegleTable do begin
    with FieldDefs do begin
      Add( 'RegleId',        ftInteger,     0,     True  );
      Add( 'RegleName',      ftString,     96,     True  );
      // gültig von, bis ?
    end;
    IndexDefs.Add('MNIndex', 'RegleId', [ixPrimary, ixUnique]);
    CreateTable;
  end;
end;

//---------------------------------------------------------------------
procedure TSDM.MakeStruktTable;
begin
  InitTable( StruktTable );
  with StruktTable do begin
    with FieldDefs do begin
      Add( 'NodeId',        ftInteger,     0,     True  );
      Add( 'Parent',        ftInteger,     0,     True  );
      Add( 'Sibling',       ftInteger,     0,     True  );
      Add( 'Child',         ftInteger,     0,     True  );

      Add( 'LeistId',       ftInteger,     0,     True  );
      Add( 'AnzNoten',      ftInteger,     0,     False  );
      Add( 'AnzTestate',    ftInteger,     0,     False  );
      Add( 'WahlNote',      ftBoolean,     0,     False  );
      Add( 'WahlTestat',    ftBoolean,     0,     False  );

      Add( 'LA1',               ftBoolean,     0,     False  );
      Add( 'LA2',               ftBoolean,     0,     False  );
      Add( 'LA1Fertig',         ftBoolean,     0,     False  );
      Add( 'LA2Fertig',         ftBoolean,     0,     False  );

      Add( 'ZeitlimiteTestate', ftInteger,     0,     False  );
      Add( 'ZeitlimiteNoten',   ftInteger,     0,     False  );
      Add( 'ErfordNote',        ftFloat,       0,     False  );
      Add( 'RundenAuf',         ftFloat,       0,     False  );
      Add( 'Gewichtung',        ftInteger,     0,     False  );
    end;
    IndexDefs.Add('MNIndex', 'NodeId', [ixPrimary, ixUnique]);
    CreateTable;
  end;
end;

//---------------------------------------------------------------------
procedure TSDM.SDMCreate(Sender: TObject);
begin
  RAW_DB        := ROOT_PATH + 'RawImport.db';
  STUDI_DB      := ROOT_PATH + 'Studi.db';
  SPRACHE_DB    := ROOT_PATH + 'Sprach.db';
  IMMSTAT_DB    := ROOT_PATH + 'ImmatStat.db';
  KATEGORIE_DB  := ROOT_PATH + 'Kategorie.db';
  STUDZIEL_DB   := ROOT_PATH + 'StudZiel.db';
  FACH_DB       := ROOT_PATH + 'Fach.db';
  FAKULTAET_DB  := ROOT_PATH + 'Fakultaet.db';
  PERSLEIST_DB  := ROOT_PATH + 'PersLeist.db';

  // noch migrieren -> Attr-Namen
  LEISTUNG_DB   := ROOT_PATH + 'Leistung.db';    // neues Format!

  REGLE_DB      := ROOT_PATH + 'Reglement.db';
  ABSCH_DB      := ROOT_PATH + 'Absch.db';
  STRUKT_DB     := ROOT_PATH + 'AbschStrukt.db';
end;
//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : StudiDialog.pas
// Detail-Anzeige für TStudi
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit StudiDialog;

```
interface

uses
  Studi, StudiDataMod, Taten, SearchDlg, AuswahlDialog,
  DBTables,
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Grids, DBGrids, ExtCtrls, ComCtrls;

type
  TStudiDlg = class(TForm)
    CloseBtn: TButton;
    Label1: TLabel;
    Label2: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Label3: TLabel;
    Edit3: TEdit;
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    TabSheet2: TTabSheet;
    Label4: TLabel;
    Edit4: TEdit;
    DBGrid1: TDBGrid;
    Edit5: TEdit;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Edit6: TEdit;
    Edit7: TEdit;
    Label9: TLabel;
    Edit8: TEdit;
    Edit9: TEdit;
    Label10: TLabel;
    Edit10: TEdit;
    Bevel1: TBevel;
    Label11: TLabel;
    Label12: TLabel;
    Label13: TLabel;
    Edit11: TEdit;
    Edit12: TEdit;
    Edit13: TEdit;
    Edit14: TEdit;
    Edit15: TEdit;
    Edit16: TEdit;
    Label14: TLabel;
    Edit17: TEdit;
    Edit18: TEdit;
    Edit19: TEdit;
    Label15: TLabel;
    Label16: TLabel;
    Bevel2: TBevel;
    Label17: TLabel;
    Label18: TLabel;
    Edit20: TEdit;
    Label19: TLabel;
    Edit21: TEdit;
    Label20: TLabel;
    Edit22: TEdit;
    Edit23: TEdit;
    Label21: TLabel;
    Label22: TLabel;
    Edit24: TEdit;
    Label23: TLabel;
```

```
    Label24: TLabel;
    Label25: TLabel;
    Edit25: TEdit;
    Edit26: TEdit;
    TabSheet3: TTabSheet;
    Label26: TLabel;
    Label27: TLabel;
    Edit27: TEdit;
    Edit28: TEdit;
    Label28: TLabel;
    Bevel3: TBevel;
    Bevel4: TBevel;
    Label29: TLabel;
    Label30: TLabel;
    Edit29: TEdit;
    Edit30: TEdit;
    Label31: TLabel;
    Memo1: TMemo;
    Edit31: TEdit;
    Edit32: TEdit;
    ModLabel: TLabel;
    CancelBtn: TButton;
    Edit33: TEdit;
    Label32: TLabel;
    Label33: TLabel;
    Edit34: TEdit;
    procedure CloseBtnClick(Sender: TObject);
    procedure SetModified(Sender: TObject);
    procedure SetAbschModified(Sender: TObject);
    procedure Edit20DblClick(Sender: TObject);
    procedure Edit21DblClick(Sender: TObject);
    procedure Edit24DblClick(Sender: TObject);
    procedure Edit25DblClick(Sender: TObject);
    procedure Edit26DblClick(Sender: TObject);
    procedure CancelBtnClick(Sender: TObject);
    procedure Edit11DblClick(Sender: TObject);
    procedure Edit12DblClick(Sender: TObject);
    procedure Edit13DblClick(Sender: TObject);
    procedure Memo1DblClick(Sender: TObject);
  public
    constructor Make( Owner: TComponent; Studi: TStudi );
  private
    TheStudi: TStudi;
    Modified: Boolean;
    procedure OpenChoiceDlg( IdEdit, TxtEdit: TEdit;
                             RefTable: TTable;
                             IdField, TxtField, LabelTxt: String );
    procedure FachDialog( E1, E2, E3: TEdit; Tit: String );
    procedure SetToStudi;
    procedure SetToDlg;
  end;

var
  StudiDlg: TStudiDlg;

implementation

{$R *.DFM}

//---------------------------------------------------------------------
constructor TStudiDlg.Make( Owner: TComponent; Studi: TStudi );
begin
  inherited Create( Owner );
  TheStudi := Studi;
  Modified := True;
  SetToStudi;
  Modified := False;
end;

//---------------------------------------------------------------------
procedure TStudiDlg.SetToStudi;
begin
  // set Dlg-Fields to TheStudi-values
  with TheStudi do begin
    // Immatrikulationsdaten-Tab
```

```
    Edit1.Text := IntToStr( MatrikelNr );
    Edit2.Text := Name;
    Edit3.Text := Vorname;
    Edit6.Text := Adresse_1;
    Edit7.Text := Land;
    Edit8.Text := IntToStr( PLZ );
    Edit9.Text := Ort;
    Edit4.Text := Anrede;
    Edit5.Text := Sprache;
    Edit10.Text := Tel;
    if HFId  <> -1 then Edit11.Text := IntToStr( HFId );
    if NF1Id <> -1 then Edit12.Text := IntToStr( NF1Id );
    if NF1Id <> -1 then Edit13.Text := IntToStr( NF2Id );
    Edit14.Text := HF;
    Edit15.Text := NF1;
    Edit16.Text := NF2;
    if HFId  <> -1 then Edit17.Text := IntToStr( HFSemAnzahl );
    if NF1Id <> -1 then Edit18.Text := IntToStr( NF1SemAnzahl );
    if NF1Id <> -1 then Edit19.Text := IntToStr( NF2SemAnzahl );
    Edit20.Text := IntToStr( KategorieId );
    Edit22.Text := Kategorie;
    Edit21.Text := IntToStr( ImmatStatusId );
    Edit23.Text := ImmatStatus;
    Edit24.Text := IntToStr( StudienZielId );
    Edit31.Text := StudienZiel;
    Edit25.Text := IntToStr( ZielAbschId );
    Edit32.Text := ZielAbschluss;
    Edit26.Text := IntToStr( RegId );
    Edit33.Text := RegName;

    // Leistungen-Tab
    with SDM.TatenQuery do begin
      Close;
      SQL.Clear;
      SQL.Add( 'select P.LeistId, L.LeistText,'
            +        ' P.Jahr, P.Termin, P.Note'
            + ' from ''' + PERSLEIST_DB + ''' P, '''
                         + LEISTUNG_DB  + ''' L'
            + ' where P.MatrikelNr = ' + IntToStr( MatrikelNr )
            + '   and P.LeistId = L.LeistId'
            + ' order by  Jahr, Termin, P.LeistId'
      );
      Open;
    end;
    DBGrid1.DataSource := SDM.TatenQueryDS;

    // Änderungen-Tab
    Edit27.Text := DateTimeToStr( InsertDate );
    Edit28.Text := DateTimeToStr( UpdateDate );
    Edit34.Text := DateTimeToStr( AbschChangeDate );
    Edit29.Text := DateTimeToStr( LA1Date );
    Edit30.Text := DateTimeToStr( LA2Date );
    // Memo1.Lines ...            // Bemerkungen
  end;
end;

//---------------------------------------------------------------------
procedure TStudiDlg.SetToDlg;
begin
  // set TheStudi-values to Dlg-Fields
  with TheStudi do begin
    // Immatrikulationsdaten-Tab
    // MatrikelNr should not be changed
    Name      := Edit2.Text;
    Vorname   := Edit3.Text;
    Adresse_1 := Edit6.Text;
    Land      := Edit7.Text;
    PLZ       := StrToInt( Edit8.Text );  //*
    Ort       := Edit9.Text;
    Anrede    := Edit4.Text;
    Sprache   := Edit5.Text;
    Tel       := Edit10.Text;
    HF        := Edit14.Text;
    NF1       := Edit15.Text;
    NF2       := Edit16.Text;
```

```
    if Edit11.Text > '' then
      HFId      := StrToInt( Edit11.Text );
    if Edit12.Text > '' then
      NF1Id     := StrToInt( Edit12.Text );
    if Edit13.Text > '' then
      NF2Id     := StrToInt( Edit13.Text );
    if HFId  > -1 then HFSemAnzahl  := StrToInt( Edit17.Text );
    if NF1Id > -1 then NF1SemAnzahl := StrToInt( Edit18.Text );
    if NF1Id > -1 then NF2SemAnzahl := StrToInt( Edit19.Text );

    if Edit20.Text > '' then
      KategorieId := StrToInt( Edit20.Text );
    Kategorie    := Edit22.Text;
    if Edit21.Text > '' then
      ImmatStatusId  := StrToInt( Edit21.Text );
    ImmatStatus  := Edit23.Text;

    if Edit24.Text > '' then
      StudienZielId  := StrToInt( Edit24.Text );
    Edit31.Text  := StudienZiel;
    if Edit25.Text > '' then
      Edit25.Text  := IntToStr( ZielAbschId );
    Edit32.Text  := ZielAbschluss;
    if Edit26.Text > '' then
      Edit26.Text  := IntToStr( RegId );
    if Edit33.Text > '' then
      Edit33.Text  := RegName;

    // Leistungen-Tab entfällt
    // Änderungen-Tab

    InsertDate       := StrToDateTime( Edit27.Text );
    LA1Date          := StrToDateTime( Edit29.Text );
    LA2Date          := StrToDateTime( Edit30.Text );

    UpdateDate       := Now;
    AbschChangeDate  := StrToDateTime( Edit34.Text );

    Save;
  end;
end;

//---------------------------------------------------------------------
procedure TStudiDlg.CloseBtnClick(Sender: TObject);
begin
  if Modified then SetToDlg;
  Close;
end;

//---------------------------------------------------------------------
procedure TStudiDlg.CancelBtnClick(Sender: TObject);
begin
  Close;
end;

//---------------------------------------------------------------------
procedure TStudiDlg.SetModified(Sender: TObject);
begin
  if not Modified then begin
    ModLabel.Caption := 'Daten geändert';
    CloseBtn.Caption := 'speichern';
    CancelBtn.Visible := True;
    Modified := True;
  end;
end;

//---------------------------------------------------------------------
procedure TStudiDlg.SetAbschModified(Sender: TObject);
begin
  SetModified( Sender );
  Edit34.Text := DateTimeToStr( Now );
end;


//---------------------------------------------------------------------
```

```
procedure TStudiDlg.OpenChoiceDlg( IdEdit, TxtEdit: TEdit;
                                   RefTable: TTable;
                                   IdField, TxtField, LabelTxt: String );
begin
  with AuswahlDlg do begin
    LoadList( RefTable, TxtField, IdField );
    SetLabel( LabelTxt );
    ShowModal;
    if ModalResult = mrOk then begin
      IdEdit.Text := IntToStr( Id );
      TxtEdit.Text := Txt;
    end;
  end;
end;

//---------------------------------------------------------------------
procedure TStudiDlg.FachDialog( E1, E2, E3: TEdit; Tit: String );
begin
  OpenChoiceDlg( E1, E2, SDM.FachTable, 'FachId', 'Fach', Tit );
  E3.SetFocus;
end;

//---------------------------------------------------------------------
procedure TStudiDlg.Edit11DblClick(Sender: TObject);
begin
  FachDialog( Edit11, Edit14, Edit17, 'Hauptfach' );
end;

//---------------------------------------------------------------------
procedure TStudiDlg.Edit12DblClick(Sender: TObject);
begin
  FachDialog( Edit12, Edit15, Edit18, '1. Nebenfach' );
end;

//---------------------------------------------------------------------
procedure TStudiDlg.Edit13DblClick(Sender: TObject);
begin
  FachDialog( Edit13, Edit16, Edit19, '2. Nebenfach' );
end;

//---------------------------------------------------------------------
procedure TStudiDlg.Edit20DblClick(Sender: TObject);
begin
  OpenChoiceDlg( Edit20, Edit22, SDM.KategorieTable,
        'KategorieId', 'Kategorie', 'Kategorie' );
end;

//---------------------------------------------------------------------
procedure TStudiDlg.Edit21DblClick(Sender: TObject);
begin
  OpenChoiceDlg( Edit21, Edit23, SDM.ImmatStatTable,
        'ImmatStatusId', 'ImmatStatus', 'ImmatrikulationsStatus' );
end;

//---------------------------------------------------------------------
procedure TStudiDlg.Edit24DblClick(Sender: TObject);
begin
  OpenChoiceDlg( Edit24, Edit31, SDM.StudZielTable,
        'StudienZielId', 'StudienZiel', 'StudienZiel' );
end;

//---------------------------------------------------------------------
procedure TStudiDlg.Edit25DblClick(Sender: TObject);
begin
  OpenChoiceDlg( Edit25, Edit32, SDM.LeistungTable,
        'Abschluss', 'LeistText', 'Abschluss in Informatik' );
end;
//---------------------------------------------------------------------
procedure TStudiDlg.Edit26DblClick(Sender: TObject);
begin
  OpenChoiceDlg( Edit26, Edit33, SDM.RegleTable,
        'RegleId', 'RegleName', 'Reglement' );
end;
//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : Taten.pas
// vollbrachte Leistungen ;-)
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit Taten;

```
interface

uses
  SysUtils;

//---------------------------------------------------------------------
type
  TTat = class( TObject )
  public
    MatrNr,
    LeistId,
    Jahr,
    Termin       : Integer;
    Note         : Real;
    constructor Create( m,l,j,t: Integer; n: Real);
    function GetArt: char;
    function GetString: String;
    property Art: char read GetArt;
  end;

implementation

//---------------------------------------------------------------------
constructor TTat.Create( m,l,j,t: Integer; n: Real);
begin
  MatrNr    := m;
  LeistId   := l;
  Jahr      := j;
  Termin    := t;
  Note      := n;
end;

//---------------------------------------------------------------------
function TTat.GetArt : char;
begin
  if Note = 0.0 then Result := 'T' else Result := 'N'
end;

//---------------------------------------------------------------------
function TTat.GetString : String;
begin
  Result := IntToStr( Jahr ) + '.' +
            IntToStr( Termin ) + '  ';
  if GetArt = 'T' then
    Result := Result + 'Testat'
  else
    Result := Result + FloatToStrF( Note, ffFixed, 15, 1 );
end;

//---------------------------------------------------------------------
end.
```

```
//-----------------------------------------------------------------
// Studis 3 : TatenImpThr.pas
// Taten-Import: Noten / Testate   -> PersLeistTable
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//-----------------------------------------------------------------
```

unit TatenImpThr;

```
interface

uses
  Tokenizers, StudiDataMod,
  Classes, SysUtils, Dialogs;

type
  //-----------------------------------------------------------------
  TTatenImportThr = class( TThread )
  public
    constructor Make( Lines: TStringList );
    procedure ImportTaten;
  private
    InputLines: TStringList;
    procedure Execute; override;
    procedure ShowProgress( I: Integer );
  end;

implementation

uses
  ImpMainForm, ProgrDlg;

//-----------------------------------------------------------------
constructor TTatenImportThr.Make( Lines: TStringList );
begin
  Create(False);
  InputLines := Lines;
  FreeOnTerminate := True;
  // ...
end;

//-----------------------------------------------------------------
procedure TTatenImportThr.Execute;
begin
  ProgressDlg.Label2.Caption := ' Noten & Testate importieren ';
  ImportTaten;
  ProgressDlg.OKBtn.Enabled := True;
end;

//-----------------------------------------------------------------
procedure TTatenImportThr.ImportTaten;
var
  LTok:        TLeistTokenizer;
  i:           Integer;
  StudiExists: Boolean;
  NofErr:      Integer;
  Msg, S:      String;
  Arr:         Variant;
begin
  NofErr := 0;
  // nur Daten-Zeilen, keine Headers
  for i := 0 to InputLines.Count-1 do
  begin
    if Inputlines[i] <> '' then        // skip blank lines
    begin
      ShowProgress(i);
      LTok := TLeistTokenizer.Create( Inputlines[i] );
      if LTok.Error then begin
        Inc( NofErr );                 // Log ...
      end
      else begin
        // ev. MatrNr mit Name, Vorname überprüfen in StudiTable
```

```
        with SDM.PersLeistTable do
        begin
          if not Locate ( 'LeistId', LTok.LeistId, [loPartialKey] )
          then begin
            // ToLog( 'Leistung ex. nicht: ' + ...)
            ShowMessage('Leistung ' + IntToStr(LTok.LeistId)
                      + ' ex. nicht');
          end
          else begin
            Arr := VarArrayOf([LTok.MatrikelNr,
                               LTok.LeistId,
                               LTok.Jahr,
                               LTok.Termin,
                               LTok.Note]);
            if Locate( 'MatrikelNr;LeistId;Jahr;Termin;Note',
                          Arr, [] ) then begin
              S := 'Leistung ( '    + IntToStr( LTok.LeistId )
                 + '  von Studi '   + IntToStr( LTok.MatrikelNr )
                 + '  Termin '      + IntToStr( LTok.Jahr )
                 + '.'              + IntToStr( LTok.Termin )
                 + '  Note '        + FloatToStr( LTok.Note )
                 + ' )  schon im System vorhanden!';
              // ToLog( S );
              Inc( NofErr );
              ShowMessage( S );
            end
            else begin
              Append;
              FieldValues['MatrikelNr']   := LTok.MatrikelNr;
              FieldValues['Note']         := LTok.Note;
              FieldValues['LeistId']      := LTok.LeistId;
              FieldValues['Jahr']         := LTok.Jahr;
              FieldValues['Termin']       := LTok.Termin;
              Post;
            end;
          end;
        end;
      end;
    end;
  end;
  Msg:= '-->  Imported : ' +  IntToStr( InputLines.Count - NofErr -1)
      + ' Lines;     -->  Errors  : ' + IntToStr( NofErr );
  ProgressDlg.Label2.Caption := Msg;
end;

//---------------------------------------------------------------------
procedure TTatenImportThr.ShowProgress(i: Integer);
begin
  ProgressDlg.ProgBar.Position := i;
  ProgressDlg.Label2.Caption := ' importing line ' + IntToStr(i) +
                       ' of total ' + IntToStr(InputLines.Count - 1);
end;

//---------------------------------------------------------------------
end.
```

```
//---------------------------------------------------------------------
// Studis 3 : Tokenizers.pas
// Aufspaltung von Text-Zeilen der ImportFormate
// PersTokenizer für PersonenDatenImport
// LeistTokenizer für TatenImport (erbrachte Leistungen)
//
// Author: Thomas F. Hofmann
// last mod 28.3.99
//---------------------------------------------------------------------
```

unit Tokenizers;

```
interface

uses
  SysUtils, Dialogs;
//Forms, Windows;

type
  //---------------------------------------------------------------------
  TTokenizer = class( TObject )
  public
    Error: Boolean; // 'has error'
    ErrMsg: String;
    constructor Create( S: String ); virtual;
  private
    Delimiter: Char;
    function CutToken( var S: String ): String;
    function TryIntConversion( IntStr, Msg: String ): Integer;
    procedure Tokenize( var Line: String ); virtual; abstract;
    procedure SetDelim;virtual; abstract;
  end;

  //---------------------------------------------------------------------
  TPersTokenizer = class( TTokenizer )
  public
    MatrikelNr:     Integer;
    AnredeId:       Integer;
    Anrede:         String;
    BriefAnrede:    String;
    Name:           String;
    Vorname:        String;
    Adresse_1:      String;
    Adresse_2:      String;
    PLZ:            Integer;
    Land:           String;  // 'CH', 'D', ...
    Ort:            String;
    GeburtsDatum:   String;
    HeimatOrt:      String;
    Tel:            String;
    SprachId:       Integer;
    Sprache:        String;
    StudienZielId:  Integer;
    StudienZiel:    String;
    SemesterId:     Integer;
    SemesterBez:    String;

    FachStatus:     String;  //  je Zeile 1x HF oder NF ...
    FachId:         Integer;
    Fach:           String;
    SemAnzahl:      Integer;

    ImmatStatusId:  Integer;
    ImmatStatus:    String;
    Kategorie:      String;  //
    ElternWohnort:  String;  // Wohnort der Eltern ?!
    StudienJahr:    Integer; //

  private
    procedure SetDelim; override;
    procedure Tokenize( var Line: String ); override;
    function GetPLZ(S: String): Integer;
    function GetLand(S: String): String;
```

```
  end;

  //---------------------------------------------------------------
  TLeistTokenizer = class( TTokenizer )
  public
    MatrikelNr:  Integer;
    Name:        String;
    Vorname:     String;
    Testat:      Boolean;  // True -> Note = 0.0
    Note:        Real;     // Note
    LeistId:     Integer;
    Jahr:        Integer;
    Termin:      Integer;  // 1: WS, 2: SS, 3: Ende Sommerferien

  private
    procedure SetDelim; override;
    procedure Tokenize( var Line: String ); override;
    function GetNote(S: String): Real;
  end;


implementation

//---------------------------------------------------------------
constructor TTokenizer.Create( S: String );
begin
  inherited Create;
  Error := False;
  ErrMsg := '';
  SetDelim;
  Tokenize( S );
end;

//---------------------------------------------------------------
function TTokenizer.CutToken(var S: String): String;
// liefert Anfangs-Substr von S (bis 1. Tab) UND verkuerzt S entspr.
var
  Pos: Integer;
begin
  Pos := 1;
  while ( Length(S) > 0       ) and
        ( S[Pos] <> Delimiter ) and
        ( Pos <= Length(S)    ) do    // search next delim
  begin
    inc( Pos );
  end;
  Result := Copy( S, 1, Pos-1 );      // return first token
  S := Copy( S, Pos+1, Length(S) );   // cut it from inp-str
end;

//---------------------------------------------------------------
function TTokenizer.TryIntConversion( IntStr, Msg: String ): Integer;
begin
  Result := -1;     // default: error
  try
    Result := StrToInt( IntStr );
  except
    on EConvertError do
    begin
      MessageDlg( 'StrToInt-conversion '+ Msg,
                  mtError, [mbOk], 0 );
      Error := True;
      // log error w/ Msg
    end;
  end;
end;

//---------------------------------------------------------------
//---------------------------------------------------------------
procedure TPersTokenizer.SetDelim;
begin
  Delimiter := Chr(9);   // Tab
end;

//---------------------------------------------------------------
```

```
procedure TPersTokenizer.Tokenize( var Line: String );
var
  Tok, PLZStr: String;
begin
  Tok := CutToken(Line);
  MatrikelNr    := TryIntConversion( Tok, '@ MatrNr: '+Tok );
  AnredeId      := TryIntConversion( CutToken(Line), '@ AnredeId');
  Anrede        := CutToken(Line);
  BriefAnrede   := CutToken(Line);

  Name          := CutToken(Line);
  Vorname       := CutToken(Line);
  Adresse_1     := CutToken(Line);
  Adresse_2     := CutToken(Line);
  PLZStr        := CutToken(Line);
  Land          := GetLand(PLZStr);
  PLZ           := GetPLZ(PLZStr);
  Ort           := CutToken(Line);

  GeburtsDatum  := CutToken(Line);
  HeimatOrt     := CutToken(Line);

  Tel           := CutToken(Line);
  SprachId      := TryIntConversion( CutToken(Line), '@ SprachId');
  Sprache       := CutToken(Line);

  StudienZielId := TryIntConversion( CutToken(Line),
                                     '@ StudienZielId' );
  StudienZiel   := CutToken(Line);

  FachStatus    := CutToken(Line);       // HF / NF
  FachId        := TryIntConversion( CutToken(Line), '@ FachId' );
  Fach          := CutToken(Line);
  SemAnzahl     := TryIntConversion( CutToken(Line), '@ SemAnzahl' );

  SemesterId    := TryIntConversion( CutToken(Line), '@ SemesterId');
  SemesterBez   := CutToken(Line);

  ImmatStatusId := TryIntConversion( CutToken(Line),
                                     '@ ImmatStatusId' );
  ImmatStatus   := CutToken(Line);
  Kategorie     := CutToken(Line);
  ElternWohnort    := CutToken(Line);
  StudienJahr   := TryIntConversion(CutToken(Line), '@ StudienJahr');
end;


//---------------------------------------------------------------------
function TPersTokenizer.GetPLZ(S: String): Integer;
var
  i: Integer;
begin
  i := 1;
  while not ( S[i] in ['0'..'9'] ) do   // Land & '-' überspringen
    inc(i);
  Result := TryIntConversion( Copy(S, i, Length(S)-i+1), '@ GetPLZ');
end;

//---------------------------------------------------------------------
function TPersTokenizer.GetLand(S: String): String;
var
  i: Integer;
begin
  i := 1;
  while not ( S[i] in ['0'..'9'] ) do inc(i);
  Result := Copy(S, 1, i-2);
end;

//---------------------------------------------------------------------
//---------------------------------------------------------------------
procedure TLeistTokenizer.SetDelim;
begin
  Delimiter := Chr(9);   // Tab
end;
```

```
//---------------------------------------------------------------------
procedure TLeistTokenizer.Tokenize( var Line: String );
var
  Tok: String;
begin
  MatrikelNr    := TryIntConversion( CutToken(Line), '@ MatrikelNr');
  Name          := CutToken(Line);
  Vorname       := CutToken(Line);

  // in StudiTable checken, ob MatrikelNr zu Name, Vorname passt
  // und ob LeistId existiert, muss im TatenImportThread passieren

  Tok           := CutToken(Line);
  if Tok = 'T' then
  begin
    Testat := true;
    Note  := 0.0;
  end
  else begin
    Testat := false;
    Note   := GetNote( Tok );
  end;

  LeistId   := TryIntConversion( CutToken(Line), '@ LeistId');
  Jahr      := TryIntConversion( CutToken(Line), '@ Jahr' );
  // check?
  Termin    := TryIntConversion( CutToken(Line), '@ Termin' );
end;

//---------------------------------------------------------------------
function TLeistTokenizer.GetNote(S: String): Real;
var
  i: Integer;
begin
  i := 1;
  while not( S[i] in ['0'..'9'] )
        and( i <= Length( S ) ) do inc(i);
  // Int-Anteil
  Result := TryIntConversion( S[i], '@ Note: Int-Anteil' );
  // Zehntel
  Result := Result + TryIntConversion( S[i + 2],
                       '@ Note: Zehntel' ) / 10;
end;

//---------------------------------------------------------------------
end.
```