kleiner kommunikation
max@kleiner.com

1 Daten hinter Gittern

Wie man mit einfachen Mitteln ein persistentes StringGrid (Gitter) aufbaut, will ich im folgenden Bericht modellgetrieben aufzeigen. Für Echtzeit- oder Eingebettete Systeme sind Datenbanken, z.B. für die Darstellung von Meßwerten oder Statusinformationen, zu schwerfällig, so daß ein Gitter mit direktem Dateizugriff ein übliches Muster ist.

1.1 Direkter Dateizugriff

Der Begriff Direkter Dateizugriff bezieht sich auf ein programmiertes Lesen und Schreiben eines darunterliegenden Dateisystems. Nicht die neueste Technologie, aber ein solider Ansatz ;).Eine Datei besteht ja aus einer Aufeinanderfolge von Elementen desselben Typs. Ich wähle als erstes eine typisierte Datei, welche die Struktur des Application-Loaders unter DelphiWebStart (DWS) repräsentiert. Dieser Typ ist ein Typ fester Länge, so daß sich hier ein Record anbietet.:

```
type
 TAppData = record
   Name: string[50];
   Size: longint;
   Release: string[30];
   Descript: string[80];
 end;
```

Wenn ein Record-Typ mit dem voreingestellten Status {$A+} deklariert wird und die Deklaration nicht den Modifizierer `packed` enthält, handelt es sich um einen ungepackten Record-Typ. Die Felder des Records werden so ausgerichtet, daß die CPU möglichst effizient darauf zugreifen kann. Auf der anderen Seite erhöhen Sie mit `packed` die Kapazität, da der Compiler die Daten in komprimierter Form speichert. Ein typisches Optimierungsproblem zwischen „Speed und Space“, welches der Entwickler eines mobilen Systems mit berücksichtigen sollte.
Die Konstruktion `file of` ... läßt sich dann direkt in einer Variablendeklaration oder in einer Klasse als Member verwenden. Die folgende Klasse ist nach der PAC-Architektur aufgebaut. Mehr über den Aufbau und Einsatz von Patterns in Design und Architektur unter [i]. In der PAC Architektur ist jeder sogenannte Agent auf seiner Stufe eine ziemlich autonome informationsverarbeitende Komponente, die auf eine bestimmte Aufgabe spezialisiert ist. Sie verfügt über die Möglichkeit zum Empfangen und Versenden von Ereignissen. Jeder Agent besteht (wenn nötig) aus den drei Teilen:

- Die Präsentation stellt das sichtbare Verhalten dar.
- Der Controller als Vermittlerschicht, zudem kommuniziert er mit andern Agenten, die sich auf höherem oder tieferem Level befinden
- Die Abstraction hat Zugriff auf das Datenmodell oder eine Struktur

Die Präsentation ist verantwortlich für die Darstellung des Gitter inklusive seiner Initialisierung, der Controller vermittelt zwischen dem Gitter und der Datei innerhalb des Konstruktor und der Abstraktionsteil speichert die Daten des Record.
Die Klasse `TBuildAppGrid` ist nun in der Lage, die Struktur des Records zu lesen und zu speichern, zudem im Gitter darzustellen und sich auch Datenänderungen zu merken:

```
  TBuildAppGrid = class
  private
    aGrid: TStringGrid;
```

```
    app: TAppData;
    f: file of TAppData;
    FaDatfile: ShortString;
    Fmodified: Boolean;
  protected
    function GetaDatfile: ShortString;
    procedure SetaDatfile(const Value: ShortString);
  public
    constructor initGrid(vGrid: TStringGrid; vFile: shortString);
    procedure fillGrid;
    procedure storeGrid;
    property aDatfile: ShortString read GetaDatfile write SetaDatfile;
    property modified: Boolean read Fmodified write Fmodified;
  end;
```

Die Komponente `TStringGrid` ist ein Abkömmling von `TDrawGrid`, die über spezialisierte Funktionalität verfügt, um die Anzeige von Strings zu vereinfachen. Die Eigenschaft `Cells` listet die Strings für jede Zelle im Gitter auf. Die Eigenschaft `Objects` listet die Objekte auf, die jedem String zugeordnet sind. Auf alle Strings und zugeordneten Objekte einer bestimmten Spalte oder Zeile kann mit den Eigenschaften `Cols` oder `Rows` zugegriffen werden.

Das Gitter dient nebst der Darstellung und dem Editieren der Daten auch als Datenspeicher, das die Routine `fillGrid` beim Auslesen ausnützen kann. Das Speichern aus dem Gitter in die Datei ist mit Hilfe des Properties `aDatfile`, das die Existenz der Datei überprüft, ziemlich direkt implementiert:

```
procedure TBuildAppGrid.storeGrid;
var
  crow: Integer;
begin
  if FModified then
  if MessageDlg('Save Changes in ' +
          aDatFile, mtConfirmation,mbOkCancel,0) = mrOK then begin
    AssignFile(f, aDatfile);
    Rewrite(f);
    try
      for crow:= 1 to Pred(aGrid.RowCount) do begin
        app.Name:= aGrid.Cells[0, crow];
        app.size:= strToInt(aGrid.Cells[1, crow]);
        app.Release:= aGrid.Cells[2, crow];
        app.descript:= aGrid.Cells[3, crow];
        Write (f, app);
      end;
    finally
      CloseFile(f);
    end;
  end; //if MessageDlg...
end;
```

1.2 Gitter mit Ereignis

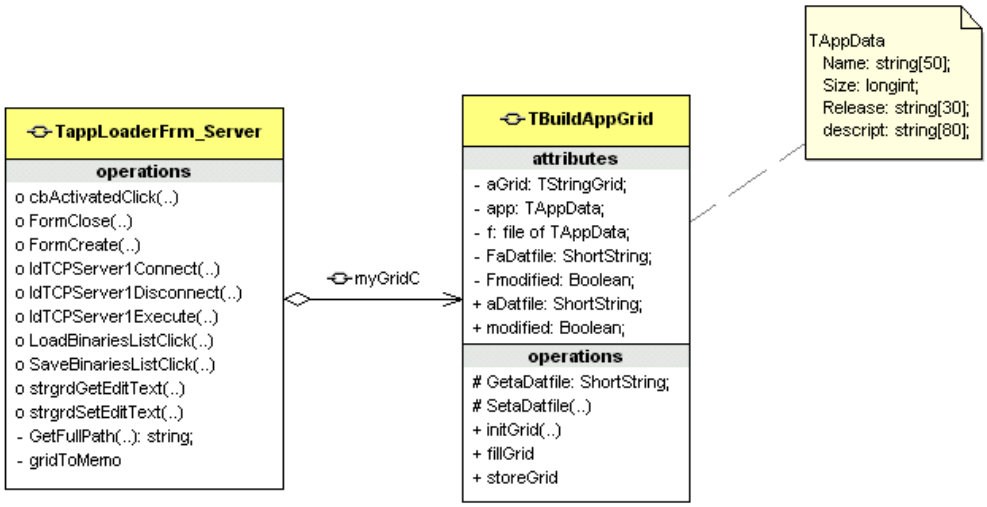
Als nächstes benötige ich das geeignete Ereignis, um das `modified` Flag zu setzen, aber auch um zusätzlich Validierungen, wie das Prüfen auf den richtigen Typ, durchzuführen. Mit der Ereignisbehandlung für `OnSetEditText` kann ich bestimmte Aktionen mit dem persistenten Text

durchführen, der vom Benutzer vorgängig bearbeitet wurde. Das Ereignis `OnSetEditText` wird also jedes Mal ausgelöst, wenn ein Anwender den Text im Gitter bearbeitet.
Dieses Ereignis tritt aber nur ein, wenn in der Eigenschaft `Options` das Flag `goEditing` gesetzt ist.

```
procedure TAppLoaderFrm_Server.strgrdSetEditText(Sender: TObject;
                        ACol, ARow: Integer; const Value: WideString);
begin
  if ACol = 1 then begin
    try
      strtoInt(Value);
    except
      raise Exception.Create('must be an integer');
    end;
  end;
 myGridC.modified:= True;
end;
```

Das folgende Klassendiagramm zeigt die Klasse `TBuildAppGrid` mit dem zugehörigen Record und dem eigentlichen Application-Loader, der die zu verwaltenden Applikationen mit ihren Namen somit aus dem Gitter holt und via einem TCP-Socket zum Client transportiert und startet. Mehr zu Sockets mit Delphi unter [[ii]]. Modellgetrieben ist der Ansatz deshalb, da eine Änderung im Record `TAppData` auch eine Änderung in der Datei sowie in der Präsentation nach sich zieht, die sich aus ModelMaker steuern läßt. Solch eine Änderung könnte das Hinzufügen eines Feldes „Pfadinformation der Applikation“ im Record sein. Das Modell aus der Serversicht besteht aus den zwei Modulen:

- Die Unit *udwsserver1.pas* beinhaltet das Gitter und die zugehörigen Ereignisse.
- Die Unit *udwsfiler.pas*, beinhaltet die Klasse `TBuildAppGrid` und den zugehörigen Record



// server_appload.tif

Das Klassendiagramm der Gitterverarbeitung

Als letztes Beispiel sei die Möglichkeit erwähnt, Daten auch untypisiert durch ein Betriebssystem-Datei-Handle in ein Gitter zu lesen. Beim Klicken auf die Methode fordert man den Benutzer zur Eingabe eines Dateinamens auf. Bestätigt er anschließend mit OK, öffnet die Anwendung die angegebene Datei, liest die Datei in einen Puffer und schließt sie wieder. Der Inhalt des Puffers läßt sich dann in zwei Spalten des Gitters anzeigen. Die erste Spalte enthält die Zeichenwerte im Puffer. Die zweite Spalte enthält die numerischen Werte der Zeichen im Puffer.

```
procedure TappLoaderFrm_Server.Button1Click(Sender: TObject);
var
  iFileHandle, iFileLength, iBytesRead, i: Integer;
  Buffer: PChar;
begin
  if OpenDialog1.Execute then begin
    try
      iFileHandle:= FileOpen(OpenDialog1.FileName, fmOpenRead);
      iFileLength:= FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      Buffer:= PChar(AllocMem(iFileLength + 1));
      iBytesRead:= FileRead(iFileHandle, Buffer^, iFileLength);
      FileClose(iFileHandle);
      for i:= 0 to iBytesRead-1 do begin
        StrGrid1.RowCount:= StrGrid1.RowCount + 1;
        StrGrid1.Cells[1,i+1]:= Buffer[i];
        StrGrid1.Cells[2,i+1]:= IntToStr(Integer(Buffer[i]));
      end;
    finally
      FreeMem(Buffer);
    end;
  end;
end;
```

Kombinieren Sie aber in Delphi auf keinen Fall Routinen, die mit Dateihandles (wie oben) arbeiten, mit Routinen, die Dateivariablen (normalerweise var F) verwenden. Um aus einer Datei zu lesen, die durch eine Dateivariable angegeben wird, verwenden Sie statt dessen die Prozedur `BlockRead` oder `Read`. Mit Ausnahme von `Read` und `Write` lassen sich alle Standardroutinen für typisierte Dateien auch für untypisierte Dateien verwenden. Anstelle von `Read` und `Write` stehen mit `BlockRead` und `BlockWrite` zwei Prozeduren für besonders schnelle Lese- und Schreiboperationen zur Verfügung.

Max Kleiner

Beispiel auf der CD-ROM

---

[i] Kleiner, M.: Patterns konkret, 2003, Software&Support Verlag

[ii] Gollub, L.: Messen, Steuern und Regeln mit TCP/IP – WinSock und Delphi, 2003, Franzis