APPENDIX 1: INHERITANCE OF ATTRIBUTES AND METHODS BY FRAME OBJECTS

TAnimal

TObject	TAnimal
Attributes	*Attributes* Name Description - sex - species - class - working - liveweight Meal WeightChange
Methods ~~Init~~ Done	*Methods* Init Load Store Weight FaecalN UrineN Excrete PopUpView PrintData

Notes

1. Each objects type is a descendant of the object or objects to its left in the table.
2. Objects can inherit (i.e. use) all attributes and methods defined in their ancestor objects.
3. Objects may overwrite methods to modify their function. Where this occurs the overwritten method is scored out in the ancestor descriptions.

25-Jul-94	Cow	Cow	Cow	Cow	Ox	Ox 2
Bean bark		1.6				
Mix 1	9.2	29.83		.565	8.	22.95
Patle						
Rice bran		.6				
Salt		.2				

TRuminant

TObject	TAnimal	TRuminant
Attributes	*Attributes* Name Description - sex - species - class - working liveweight Meal WeightChange	*Attributes* r Y_mcp_fme Kn
Methods ~~Init~~ Done	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ Weight ~~FaecalN~~ ~~UrineN~~ Excrete PopUpView PrintData	*Methods* Init Load Store FaecalN UrineN ME_maint ME_production MP_maint MP_gain Gain_Loss_ME Gain_Loss_MP MP_required ProdLevel ProteinStatus Perform UpDate AbortPreg DriedOff

TBovine

TObject	TAnimal	TRuminant	TBovine
Attributes	*Attributes* Name Description - sex - species - class - working - liveweight Meal WeightChange	*Attributes* r Y_mcp_fme Kn	*Attributes* WorkDetails
Methods ~~Init~~ Done	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ Weight ~~FaecalN~~ ~~UrineN~~ Excrete PopUpView PrintData	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ FaecalN UrineN ~~ME_Maint~~ ~~ME_Production~~ ~~MP_Maint~~ MP_Gain Gain_Loss_ME Gain_Loss_MP MP_required ProdLevel ProteinStatus Perform UpDate AbortPreg DriedOff	*Methods* Init Load Store ME_Maint ME_Production ME_Work MP_Maint

TGrowingBovine

TObject	TAnimal	TRuminant	TBovine	TGrowingBovine
Attributes	*Attributes* Name Description - sex - species - class - working - liveweight Meal WeightChange	*Attributes* r Y_mcp_fme Kn	*Attributes* WorkDetails	*Attributes*
Methods ~~Init~~ Done	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ Weight ~~FaecalN~~ ~~UrineN~~ Excrete PopUpView PrintData	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ FaecalN UrineN ~~ME_Maint~~ ~~ME_Production~~ ~~MP_Maint~~ ~~MP_Gain~~ ~~Gain_Loss_ME~~ ~~Gain_Loss_MP~~ MP_required ProdLevel ProteinStatus Perform ~~UpDate~~ AbortPreg DriedOff	*Methods* Init Load Store ME_Maint ME_Production ME_Work MP_Maint	*Methods* MP_Gain Gain_Loss_ME Gain_Loss_MP UpDate

TMatureBovine

TObject	TAnimal	TRuminant	TBovine	TMatureBovine
Attributes	*Attributes* Name Description - sex - species - class - working - liveweight Meal WeightChange	*Attributes* r Y_mcp_fme Kn	*Attributes* WorkDetails	*Attributes*
Methods ~~Init~~ Done	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ Weight ~~FaecalN~~ ~~UrineN~~ Excrete PopUpView PrintData	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ FaecalN UrineN ~~ME_Maint~~ ~~ME_Production~~ ~~MP_Maint~~ ~~MP_Gain~~ ~~Gain_Loss_ME~~ ~~Gain_Loss_MP~~ MP_required ProdLevel ProteinStatus Perform UpDate AbortPreg DriedOff	*Methods* Init Load Store ME_Maint ME_Production ME_Work MP_Maint	*Methods* MP_Gain Gain_Loss_ME Gain_Loss_MP

TMaleBovine

TObject	TAnimal	TRuminant	TBovine	TMatureBovine	TMaleBovine
Attributes	*Attributes* Name Description - sex - species - class - working - liveweight Meal WeightChange	*Attributes* r Y_mcp_fme Kn	*Attributes* WorkDetails	*Attributes*	
Methods ~~Init~~ Done	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ Weight ~~FaecalN~~ ~~UrineN~~ Excrete PopUpView PrintData	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ FaecalN UrineN ~~ME_Maint~~ ~~ME_Production~~ ~~MP_Maint~~ ~~MP_Gain~~ ~~Gain_Loss_ME~~ ~~Gain_Loss_MP~~ MP_required ProdLevel ProteinStatus Perform ~~UpDate~~ AbortPreg DriedOff	*Methods* Init Load Store ME_Maint ME_Production ME_Work MP_Maint	*Methods* MP_Gain Gain_Loss_ME Gain_Loss_MP	*Methods* UpDate

TFemaleBovine

TObject	TAnimal	TRuminant	TBovine	TMatureBovine	TFemaleBovine
Attributes	*Attributes* Name Description - sex - species - class - working - liveweight Meal WeightChange	*Attributes* r Y_mcp_fme Kn	*Attributes* WorkDetails	*Attributes*	*Attributes* ReproStat - DayOfPreg - DayOfLact MilkYield MilkFat MilkProt
Methods ~~Init~~ Done	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ Weight ~~FaecalN~~ ~~UrineN~~ Excrete PopUpView ~~PrintData~~	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ FaecalN UrineN ~~ME_Maint~~ ~~ME_Production~~ ~~MP_Maint~~ ~~MP_Gain~~ ~~Gain_Loss_ME~~ ~~Gain_Loss_MP~~ ~~MP_required~~ ~~ProdLevel~~ ~~ProteinStatus~~ Perform ~~UpDate~~ AbortPreg DriedOff	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ ME_Maint ~~ME_Production~~ ME_Work MP_Maint	*Methods* MP_Gain ~~Gain_Loss_ME~~ ~~Gain_Loss_MP~~	*Methods* Init Load Store ME_Preg ME_Lact Gain_Loss_ME MP_Preg MP_Lact ME_Production MP_Required ProdLevel ProteinStatus UpDate PrintData

TCaprine

TObject	TAnimal	TRuminant	TCaprine
Attributes	*Attributes* Name Description - sex - species - class - working liveweight Meal WeightChange	*Attributes* r Y_mcp_fme Kn	*Attributes*
Methods ~~Init~~ Done	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ Weight ~~FaecalN~~ ~~UrineN~~ Excrete PopUpView PrintData	*Methods* Init Load Store FaecalN UrineN ~~ME_Maint~~ ME_Production ~~MP_Maint~~ MP_Gain Gain_Loss_ME Gain_Loss_MP MP_required ProdLevel ProteinStatus Perform UpDate AbortPreg DriedOff	*Methods* ME_Maint MP_Maint

TGrowingCaprine

TObject	TAnimal	TRuminant	TCaprine	TGrowingCaprine
Attributes	*Attributes* Name Description - sex - species - class - working - liveweight Meal WeightChange	*Attributes* r Y_mcp_fme Kn	*Attributes*	*Attributes*
Methods ~~Init~~ Done	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ Weight ~~FaecalN~~ ~~UrineN~~ Excrete PopUpView PrintData	*Methods* Init Load Store FaecalN UrineN ~~ME_Maint~~ ME_Production ~~MP_Maint~~ ~~MP_Gain~~ ~~Gain_Loss_ME~~ ~~Gain_Loss_MP~~ MP_required ProdLevel ProteinStatus Perform ~~UpDate~~ AbortPreg DriedOff	*Methods* ME_Maint MP_Maint	*Methods* MP_Gain Gain_Loss_ME Gain_Loss_MP UpDate

TMatureCaprine

TObject	TAnimal	TRuminant	TCaprine	TMatureCaprine
Attributes	*Attributes* Name Description - sex - species - class - working - liveweight Meal WeightChange	*Attributes* r Y_mcp_fme Kn	*Attributes*	*Attributes*
Methods ~~Init~~ Done	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ Weight ~~FaecalN~~ ~~UrineN~~ Excrete PopUpView PrintData	*Methods* Init Load Store FaecalN UrineN ~~ME_Maint~~ ME_Production ~~MP_Maint~~ ~~MP_Gain~~ ~~Gain_Loss_ME~~ ~~Gain_Loss_MP~~ MP_required ProdLevel ProteinStatus Perform UpDate AbortPreg DriedOff	*Methods* ME_Maint MP_Maint	*Methods* MP_Gain Gain_Loss_ME Gain_Loss_MP

TMaleCaprine

TObject	TAnimal	TRuminant	TCaprine	TMatureCaprine	TMaleCaprine
Attributes	*Attributes* Name Description - sex - species - class - working - liveweight Meal WeightChange	*Attributes* r Y_mcp_fme Kn	*Attributes*	*Attributes*	*Attributes*
Methods ~~Init~~ Done	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ Weight ~~FaecalN~~ ~~UrineN~~ Excrete PopUpView PrintData	*Methods* Init Load Store FaecalN UrineN ~~ME_Maint~~ ME_Production ~~MP_Maint~~ ~~MP_Gain~~ ~~Gain_Loss_ME~~ ~~Gain_Loss_MP~~ MP_required ProdLevel ProteinStatus Perform ~~UpDate~~ AbortPreg DriedOff	*Methods* ME_Maint MP_Maint	*Methods* MP_Gain Gain_Loss_ME Gain_Loss_MP	*Methods* UpDate

TFemaleCaprine

TObject	TAnimal	TRuminant	TCaprine	TMatureCaprine	TFemaleCaprine
Attributes	*Attributes* Name Description - sex - species - class - working - liveweight Meal WeightChange	*Attributes* r Y_mcp_fme Kn	*Attributes*	*Attributes*	*Attributes* ReproStat - DayOfPreg - DayOfLact MilkYield MilkFat MilkProt
Methods ~~Init~~ Done	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ Weight ~~FaecalN~~ ~~UrineN~~ Excrete PopUpView ~~PrintData~~	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ FaecalN UrineN ~~ME_Maint~~ ME_Production ~~MP_Maint~~ ~~MP_Gain~~ ~~Gain_Loss_ME~~ ~~Gain_Loss_MP~~ ~~MP_required~~ ~~ProdLevel~~ ~~ProteinStatus~~ Perform UpDate AbortPreg DriedOff	*Methods* ME_Maint MP_Maint	*Methods* MP_Gain Gain_Loss_ME Gain_Loss_MP	*Methods* Init Load Store ME_Preg ME_Lact Gain_Loss_ME MP_Preg MP_Lact ME_Production MP_Required ProdLevel ProteinStatus UpDate PrintData

TOvine

TObject	TAnimal	TRuminant	TOvine
Attributes	*Attributes* Name Description - sex - species - class - working - liveweight Meal WeightChange	*Attributes* r Y_mcp_fme Kn	*Attributes*
Methods ~~Init~~ Done	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ Weight ~~FaecalN~~ ~~UrineN~~ Excrete PopUpView PrintData	*Methods* Init Load Store FaecalN UrineN ~~ME_Maint~~ ME_Production ~~MP_Maint~~ MP_Gain Gain_Loss_ME Gain_Loss_MP MP_required ProdLevel ProteinStatus Perform UpDate AbortPreg DriedOff	*Methods* ME_Maint MP_Maint

TGrowingOvine

TObject	TAnimal	TRuminant	TOvine	TGrowingOvine
Attributes	*Attributes* Name Description - sex - species - class - working liveweight Meal WeightChange	*Attributes* r Y_mcp_fme Kn	*Attributes*	*Attributes*
Methods ~~Init~~ Done	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ Weight ~~FaecalN~~ ~~UrineN~~ Excrete PopUpView PrintData	*Methods* Init Load Store FaecalN UrineN ~~ME_Maint~~ ME_Production ~~MP_Maint~~ ~~MP_Gain~~ ~~Gain_Loss_ME~~ ~~Gain_Loss_MP~~ MP_required ProdLevel ProteinStatus Perform ~~UpDate~~ AbortPreg DriedOff	*Methods* ME_Maint MP_Maint	*Methods* MP_Gain Gain_Loss_ME Gain_Loss_MP UpDate

TMatureOvine

TObject	TAnimal	TRuminant	TOvine	TMatureOvine
Attributes	*Attributes* Name Description - sex - species - class - working - liveweight Meal WeightChange	*Attributes* r Y_mcp_fme Kn	*Attributes*	*Attributes*
Methods ~~Init~~ Done	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ Weight ~~FaecalN~~ ~~UrineN~~ Excrete PopUpView PrintData	*Methods* Init Load Store FaecalN UrineN ~~ME_Maint~~ ME_Production ~~MP_Maint~~ ~~MP_Gain~~ ~~Gain_Loss_ME~~ ~~Gain_Loss_MP~~ MP_required ProdLevel ProteinStatus Perform UpDate AbortPreg DriedOff	*Methods* ME_Maint MP_Maint	*Methods* MP_Gain Gain_Loss_ME Gain_Loss_MP

TMaleOvine

TObject	TAnimal	TRuminant	TOvine	TMatureOvine	TMaleOvine
Attributes	*Attributes* Name Description - sex - species - class - working - liveweight Meal WeightChange	*Attributes* r Y_mcp_fme Kn	*Attributes*	*Attributes*	*Attributes*
Methods ~~Init~~ Done	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ Weight ~~FaecalN~~ ~~UrineN~~ Excrete PopUpView PrintData	*Methods* Init Load Store FaecalN UrineN ~~ME_Maint~~ ME_Production ~~MP_Maint~~ ~~MP_Gain~~ ~~Gain_Loss_ME~~ ~~Gain_Loss_MP~~ MP_required ProdLevel ProteinStatus Perform ~~UpDate~~ AbortPreg DriedOff	*Methods* ME_Maint MP_Maint	*Methods* MP_Gain Gain_Loss_ME Gain_Loss_MP	*Methods* UpDate

TFemaleOvine

TObject	TAnimal	TRuminant	TOvine	TMatureOvine	TFemaleOvine
Attributes	*Attributes* Name Description - sex - species - class - working - liveweight Meal WeightChange	*Attributes* r Y_mcp_fme Kn	*Attributes*	*Attributes*	*Attributes* ReproStat - DayOfPreg - DayOfLact MilkYield MilkFat MilkProt
Methods ~~Init~~ Done	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ Weight ~~FaecalN~~ ~~UrineN~~ Excrete PopUpView ~~PrintData~~	*Methods* ~~Init~~ ~~Load~~ ~~Store~~ FaecalN UrineN ~~ME_Maint~~ ME_Production ~~MP_Maint~~ ~~MP_Gain~~ ~~Gain_Loss_ME~~ ~~Gain_Loss_MP~~ ~~MP_required~~ ~~ProdLevel~~ ~~ProteinStatus~~ Perform UpDate AbortPreg DriedOff	*Methods* ME_Maint MP_Maint	*Methods* MP_Gain Gain_Loss_ME Gain_Loss_MP	*Methods* Init Load Store ME_Preg ME_Lact Gain_Loss_ME MP_Preg MP_Lact ME_Production MP_Required ProdLevel ProteinStatus UpDate PrintData

APPENDIX 2: PROGRAMME LISTING

The complete Borland Pascal with Objects 7.0™ programme listing follows at this appendix. As an aid to its logical development and for clarity of understanding, the complete programme is divided into a number of units containing code for a number of discreet functions or groups of functions. These are, in the order in which they are listed in this appendix:

File name	Contents
ANIMALS.PAS	Simulations of FRAME animal types
DATABASE.PAS	Feeds database
DATES.PAS	Structure for feed calendar
FEEDS.PAS	Description of feeds
FRAMERES.PAS	User interface
FRAME.PAS	Main programme file
GLOBALS.PAS	Global variables used by other FRAME units
HELPCONT.PAS	Constant used to define context sensitive help
MUCK.PAS	Manure-compost component
SIMULAT.PAS	Basic simulation structure
TOOLS.PAS	Interface components

```
unit ANIMALS;

interface

uses Objects, MsgBox,
    Dos,
    Globals,
    Feeds,
    Muck;

type
 PAnimal = ^TAnimal;
 TAnimal = object(TObject)
  Name : String;
  Description : TANRec;
  Meal : PFeeds;
  WeightChange : Integer;
  constructor Init(De : PANRec);
  constructor Load(var S : TStream);
  procedure Store(var S : TStream);
  function Weight : Real;
  function FaecalN : Real; virtual;
  function UrineN : Real; virtual;
  procedure Excrete(CH: PCompostHeap);
  procedure PopUpView; virtual;
  procedure PrintData(var F : Text); virtual;
 end;

 PRuminant = ^TRuminant;
 TRuminant = object(TAnimal)
  r : Real;
  Ymcpfme : Real;
  Kn : Real;
  constructor Init(De : PANRec);
  constructor Load(var S : TStream);
  procedure Store(var S : TStream);
  function ME_Maint : Real; virtual;
  function ME_Production : Real; virtual;
  function MP_Maint : Real; virtual;
  function MP_Gain : Real; virtual;
  function Gain_Loss_ME(ME : Real): Real; virtual;
  function Gain_Loss_MP(MPG : Real): Real; virtual;
  function MPRequired : Real; virtual;
  procedure ProductionLevel; virtual;
  procedure ProteinStatus; virtual;
  function FaecalN : Real; virtual;
  function UrineN : Real; virtual;
  function Perform(NumDays : Word) : PANRec; virtual;
  function UpDate(NumDays : Word) : PAnimal; virtual;
  function AbortPregnancy(EnergyProt : String): Word;
  function DriedOff(EnergyProt : String): Word;
 end;

 PBovine = ^TBovine;
 TBovine = object(TRuminant)
  WorkDetails : TWKRec;
```

```
  constructor Init(De : PANRec);
  constructor Load(var S : TStream);
  procedure Store(var S : TStream);
  function ME_Maint : Real; virtual;
  function ME_Production : Real; virtual;
  function MP_Maint : Real; virtual;
  function ME_Work : Real; virtual;
end;

PGrowingBovine = ^TGrowingBovine;
TGrowingBovine = object(TBovine)
  function MP_Gain: Real; virtual;
  function Gain_Loss_ME(ME : Real) : Real; virtual;
  function Gain_Loss_MP(MPG : Real) : Real; virtual;
  function UpDate(NumDays : Word) : PAnimal; virtual;
end;

PMatureBovine = ^TMatureBovine;
TMatureBovine = object(TBovine)
  function Gain_Loss_ME(ME : Real) : Real; virtual;
  function Gain_Loss_MP(MPG : Real) : Real; virtual;
  function MP_Gain : Real; virtual;
end;

PMaleBovine = ^TMaleBovine;
TMaleBovine = Object(TMatureBovine)
  function UpDate(NumDays : Word) : PAnimal; virtual;
end;

PFemaleBovine = ^TFemaleBovine;
TFemaleBovine = Object(TMatureBovine)
  ReproStat : TMFRec;
  MilkYield : Real;
  MilkFat : Real;
  MilkProt : Real;
  constructor Init(De : PANRec; Mf : PMFRec);
  constructor Load(var S : TStream);
  procedure Store(var S : TStream);
  function ME_Preg : Real;
  function ME_Lact : Real;
  function Gain_Loss_ME(ME : Real) : Real; virtual;
  function MP_Preg : Real;
  function MP_Lact : Real;
  function ME_Production : Real; virtual;
  function MPRequired : Real; virtual;
  procedure ProductionLevel; virtual;
  procedure ProteinStatus; virtual;
  function UpDate(NumDays : Word) : PAnimal; virtual;
  procedure PrintData(var F : Text);  virtual;
end;

PCaprine = ^TCaprine;
TCaprine = object(TRuminant)
  function ME_Maint : Real; virtual;
  function MP_Maint : Real; virtual;
end;
```

```
 PGrowingCaprine = ^TGrowingCaprine;
 TGrowingCaprine = object(TCaprine)
  function MP_Gain : Real; virtual;
  function Gain_Loss_ME(ME : Real) : Real; virtual;
  function Gain_Loss_MP(MPG : Real) : Real; virtual;
  function UpDate(NumDays : Word) : PAnimal; virtual;
 end;

 PMatureCaprine = ^TMatureCaprine;
 TMatureCaprine = object(TCaprine)
  function Gain_Loss_ME(ME : Real) : Real; virtual;
  function MP_Gain : Real; virtual;
  function Gain_Loss_MP(MPG : Real) : Real; virtual;
end;

 PMaleCaprine = ^TMaleCaprine;
 TMaleCaprine = Object(TMatureCaprine)
  function UpDate(NumDays : Word) : PAnimal; virtual;
 end;

 PFemaleCaprine = ^TFemaleCaprine;
 TFemaleCaprine = object(TMatureCaprine)
  ReproStat : TMFRec;
  MilkYield : Real;
  MilkFat : Real;
  MilkProt : Real;
  constructor Init(De : PANRec; Mf : PMFRec);
  constructor Load(var S : TStream);
  procedure Store(var S : TStream);
  function ME_Preg : Real;
  function ME_Lact : Real;
  function MP_Preg : Real;
  function MP_Lact : Real;
  function ME_Production : Real; virtual;
  function MPRequired : Real; virtual;
  procedure ProductionLevel; virtual;
  procedure ProteinStatus; virtual;
  function UpDate(NumDays : Word) : PAnimal; virtual;
  procedure PrintData(var F : Text);  virtual;
end;

 POvine = ^TOvine;
 TOvine = object(TRuminant)
  function ME_Maint : Real; virtual;
 end;

 PGrowingOvine = ^TGrowingOvine;
 TGrowingOvine = object(TOvine)
  function MP_Gain : Real; virtual;
  function Gain_Loss_ME(ME : Real) : Real; virtual;
  function Gain_Loss_MP(MPG : Real) : Real; virtual;
  function UpDate(NumDays : Word) : PAnimal; virtual;
 end;

 PMatureOvine = ^TMatureOvine;
```

```
 TMatureOvine = object(TOvine)
  function Gain_Loss_ME(ME: Real) : Real; virtual;
  function MP_Maint : Real; virtual;
  function MP_Gain : Real; virtual;
  function Gain_Loss_MP(MPG : Real): Real; virtual;
 end;

 PMaleOvine = ^TMaleOvine;
 TMaleOvine = Object(TMatureOvine)
  function UpDate(NumDays : Word) : PAnimal; virtual;
 end;

 PFemaleOvine = ^TFemaleOvine;
 TFemaleOvine = object(TMatureOvine)
  ReproStat : TMFRec;
  MilkYield: Real;
  MilkFat: Real;
  MilkProt: Real;
  constructor Init(De : PANRec; Mf : PMFRec);
  constructor Load(var S: TStream);
  procedure Store(var S: TStream);
  function ME_Preg : Real;
  function ME_Lact : Real;
  function MP_Preg : Real;
  function MP_Lact : Real;
  function ME_Production : Real; virtual;
  function MPRequired : Real; virtual;
  procedure ProductionLevel; virtual;
  procedure ProteinStatus; virtual;
  function UpDate(NumDays : Word) : PAnimal; virtual;
  procedure PrintData(var F : Text);  virtual;
 end;

 PHerd = ^THerd;
 THerd = object(TSortedCollection)
  function KeyOf(Item: Pointer): Pointer; virtual;
  function Compare(Key1, Key2: Pointer): Integer; virtual;
  procedure MakeViewerList(var L: PStringCollection);
  procedure ViewData(Var F : Text);
 end;


const

 hdNew = 0;
 hdAdd = 1;
 hdEdit = 2;

 RAnimal: TStreamRec = (
  ObjType: 700;
  VmtLink: Ofs(TypeOf(TAnimal)^);
  Load: @TAnimal.Load;
  Store: @TAnimal.Store
 );

 RRuminant: TStreamRec = (
```

```
    ObjType: 710;
    VmtLink: Ofs(TypeOf(TRuminant)^);
    Load: @TRuminant.Load;
    Store: @TRuminant.Store
  );

  RBovine: TStreamRec = (
    ObjType: 720;
    VmtLink: Ofs(TypeOf(TBovine)^);
    Load: @TBovine.Load;
    Store: @TBovine.Store
  );

  RGrowingBovine: TStreamRec = (
    ObjType: 730;
    VmtLink: Ofs(TypeOf(TGrowingBovine)^);
    Load: @TGrowingBovine.Load;
    Store: @TGrowingBovine.Store
  );

  RMatureBovine: TStreamRec = (
    ObjType: 731;
    VmtLink: Ofs(TypeOf(TMatureBovine)^);
    Load: @TMatureBovine.Load;
    Store: @TMatureBovine.Store
  );

  RMaleBovine: TStreamRec = (
    ObjType: 740;
    VmtLink: Ofs(TypeOf(TMaleBovine)^);
    Load: @TMaleBovine.Load;
    Store: @TMaleBovine.Store
  );

  RFemaleBovine: TStreamRec = (
    ObjType: 741;
    VmtLink: Ofs(TypeOf(TFemaleBovine)^);
    Load: @TFemaleBovine.Load;
    Store: @TFemaleBovine.Store
  );

  RCaprine: TStreamRec = (
    ObjType: 721;
    VmtLink: Ofs(TypeOf(TCaprine)^);
    Load: @TCaprine.Load;
    Store: @TCaprine.Store
  );

  RGrowingCaprine: TStreamRec = (
    ObjType: 732;
    VmtLink: Ofs(TypeOf(TGrowingCaprine)^);
    Load: @TGrowingCaprine.Load;
    Store: @TGrowingCaprine.Store
  );

  RMatureCaprine: TStreamRec =
```

```
    ObjType: 733;
    VmtLink: Ofs(TypeOf(TMatureBovine)^);
    Load: @TMatureCaprine.Load;
    Store: @TMatureCaprine.Store
  );

  RMaleCaprine: TStreamRec = (
    ObjType: 742;
    VmtLink: Ofs(TypeOf(TMaleCaprine)^);
    Load: @TMaleCaprine.Load;
    Store: @TMaleCaprine.Store
  );

  RFemaleCaprine: TStreamRec = (
    ObjType: 743;
    VmtLink: Ofs(TypeOf(TFemaleCaprine)^);
    Load: @TFemaleCaprine.Load;
    Store: @TFemaleCaprine.Store
  );

  ROvine: TStreamRec = (
    ObjType: 722;
    VmtLink: Ofs(TypeOf(TOvine)^);
    Load: @TOvine.Load;
    Store: @TOvine.Store
  );

  RGrowingOvine: TStreamRec = (
    ObjType: 734;
    VmtLink: Ofs(TypeOf(TGrowingOvine)^);
    Load: @TGrowingOvine.Load;
    Store: @TGrowingOvine.Store
  );

  RMatureOvine: TStreamRec = (
    ObjType: 735;
    VmtLink: Ofs(TypeOf(TMatureOvine)^);
    Load: @TMatureOvine.Load;
    Store: @TMatureOvine.Store
  );

  RMaleOvine: TStreamRec = (
    ObjType: 744;
    VmtLink: Ofs(TypeOf(TMaleOvine)^);
    Load: @TMaleOvine.Load;
    Store: @TMaleOvine.Store
  );

  RFemaleOvine: TStreamRec = (
    ObjType: 745;
    VmtLink: Ofs(TypeOf(TFemaleOvine)^);
    Load: @TFemaleOvine.Load;
    Store: @TFemaleOvine.Store
  );

  RHerd: TStreamRec = (
```

```
    ObjType: 780;
    VmtLink: Ofs(TypeOf(THerd)^);
    Load: @THerd.Load;
    Store: @THerd.Store
  );

  procedure RegisterAnimals;


implementation

{*** TAnimal ***}

constructor TAnimal.Init(De: PANRec);
begin
  inherited Init;
  Description := De^;
  with Description do
    Name := ClassNames[Class] + SexNames[Sex] + SpeciesNames[Species];
  Meal := New(PFeeds, Init(10, 2));
  WeightChange := 0
end;

constructor TAnimal.Load(var S: TStream);
begin
  S.Read(Name, SizeOf(String));
  S.Read(Description, SizeOf(TANRec));
  S.Read(WeightChange, SizeOf(Integer));
  Meal := New(PFeeds, Init(10, 2));
  Meal^.Load(S)
end;

procedure TAnimal.Store(var S: TStream);
begin
  S.Write(Name, SizeOf(String));
  S.Write(Description, SizeOf(TANRec));
  S.Write(WeightChange, SizeOf(Integer));
  Meal^.Store(S)
end;

function TAnimal.Weight: Real;
Var
  Error : Integer;
  Wt : Real;
begin
  val(Description.LiveWeight, Wt, Error);
  if Error = 0 then Weight := Wt
end;

function TAnimal.FaecalN: Real;
begin
  {*** Dummy for descendant object types ***}
end;

function TAnimal.UrineN : Real;
begin
```

```
    {*** Dummy for descendant object types ***}
  end;

  procedure TAnimal.Excrete(CH: PCompostHeap);
  begin
   CH^.Insert(new(PManure, init(Name, FaecalN)));
   CH^.Insert(new(PUrine, init(Name, UrineN)))
  end;

  procedure TAnimal.PopUpView;
  Var
   Bounds : TRect;
   T : String;

   procedure MealComposition(Item : PAvailableFeed); far;
   begin
    T := T + Item^.Name + ' '
   end;

 begin
  T := Name + #13'Liveweight: ' + Description.Liveweight + 'kg';
  if Meal^.Count = 0 then T := T + #13#13'No Feeds Offered'
  else
  begin
    T := T + #13#13'Feeds Offered: ';
    Meal^.ForEach(@MealComposition)
  end;
  Bounds.Assign(14, 5, 65, 16);
  MessageBoxRect(Bounds, T, nil, mfInformation + mfOKButton);
 end;

 procedure TAnimal.PrintData(var F : Text);
 begin
  write(F, Name);
  FillString(40 - Length(Name), F);
  write(F, Description.LiveWeight)
 end;

 {*** TRuminant ***}

 constructor TRuminant.Init(De: PANRec);
 begin
  inherited Init(De);
  r:=0;
  Ymcpfme:=0;
  Kn:=0
 end;

 constructor TRuminant.Load(var S: TStream);
 begin
  inherited Load(S);
  S.Read(r, SizeOf(Real));
  S.Read(Ymcpfme, SizeOf(Real));
  S.Read(Kn, SizeOf(Real))
 end;
```

```
procedure TRuminant.Store(var S: TStream);
begin
  inherited Store(S);
  S.Write(r, SizeOf(Real));
  S.Write(Ymcpfme, SizeOf(Real));
  S.Write(Kn, SizeOf(Real))
end;

function TRuminant.ME_Maint: Real;
begin
  {*** Dummy for descendant object types ***}
end;

function TRuminant.ME_Production: Real;
begin
  ME_Production := Meal^.MetEnergy - ME_Maint
end;

function TRuminant.MP_Maint: Real;
begin
  MP_Maint := 2.1875 * exp(0.75 * ln(Weight))
end;

function TRuminant.MP_Gain: Real;
begin
  {*** Dummy for descendant object types ***}
end;

function TRuminant.MPRequired: Real;
begin
  {*** This procedure is used by all growing and mature male animals ***}
  MPRequired := MP_Maint + MP_Gain
end;

function TRuminant.Gain_Loss_ME(ME: Real): Real;
begin
  {*** Dummy for descendant object types ***}
end;

function TRuminant.Gain_Loss_MP(MPG: Real): Real;
begin
  {*** Dummy for descendant object types ***}
end;

Procedure TRuminant.ProductionLevel;
begin
  WeightChange:=round(Gain_Loss_ME(ME_Production))
end;


Procedure TRuminant.ProteinStatus;
var TWC: Real;
begin

  {*** This procedure is used by all growing and mature male animals ***}
```

```
  if Meal^.MP < MPRequired then
  begin
    TWC := round(Gain_Loss_MP(Meal^.MP - MP_Maint));
    if TWC < WeightChange then WeightChange := round(TWC)
  end;

  {*** Need to calculate a value for KN so that urine N can be estimated ***}

  if MP_Maint<Meal^.MP then
  begin
    KN := (MP_Maint/Meal^.MP) * 0.85;
    if WeightChange > 0 then KN := KN + (MP_Gain / Meal^.MP) * 0.59
  end
  else
  KN := 0.85
end;

function TRuminant.FaecalN: Real;
var BEN: Real;
begin
{  MFN:=0.13*exp(0.75*ln(Weight));          Dewhurst and Thomas, 1992 }
   BEN := 0.35 * exp(0.75 * ln(Weight));    { AFRC_TCORN, 1992 }
   FaecalN := BEN + 0.15 * Meal^.MTP + Meal^.ADIN
end;

function TRuminant.UrineN : Real;
var EUN: Real;
begin

  {*** Look at estimates of endogenous urinary nitrogen in more detail ***}

  EUN := 0.13 * exp(0.75 * ln(Weight));      Dewhurst and Thomas, 1992 }

{ EUN:=0.02348*Weight+0.54;           for sheep   ARC, 1980 }

{ EUN:=5.9206* log10(Weight)-6.76;     for cattle   ARC, 1980 }

  UrineN := EUN + (Meal^.MP / 6.25) * (1 - KN)
end;

function TRuminant.Perform(NumDays : Word) : PANRec;
var L : Real;
    MEForProd : Real;
    OldWeight, NewWeight : Word;
    Error : Integer;
    NewDescription : PANRec;
begin
  L := Meal^.MetEnergy / ME_Maint;

  r := -0.024 + 0.179 * (1-exp(-0.278 * L));
  if r > 0.08 then r := 0.08;
  if r < 0.02 then r := 0.02;

  Ymcpfme := 7 + 6 * (1-exp(-0.35 * L));
  if Ymcpfme > 11 then Ymcpfme := 11;
  if Ymcpfme < 9 then Ymcpfme := 9;
```

```
  {*** need to look at these for animals on sub-maintenance diets ***}
  ProductionLevel;
  Meal^.CalculateMP(Ymcpfme);
  ProteinStatus;

  { Common to all? }
  val(Description.LiveWeight, OldWeight, Error);
  NewWeight := round(OldWeight + WeightChange * NumDays / 1000);
  New(NewDescription);
  NewDescription^ := Description;
  str(NewWeight, NewDescription^.LiveWeight);
  Perform := NewDescription;

  {implemented for different animal types only ?

  Perform := New(PRuminant, Init(NewDescription));}
end;

function TRuminant.UpDate(NumDays : Word) : PAnimal;
begin
  {*** Dummy for descendant object types ***}
end;


function TRuminant.AbortPregnancy(EnergyProt: String): Word;
Var
  MBStr : String;
begin
  MBStr := Name + ': ' + EnergyProt +
       ' supply inadequate. Pregnancy terminated';;
  MessageBox(MBStr, nil, mfInformation + mfOKButton);
  AbortPregnancy:=0
end;

function TRuminant.DriedOff(EnergyProt: String): Word;
Var
  MBStr : String;
begin
  MBStr:=Name + ': ' + EnergyProt +
       ' supply inadequate. Lactation terminated';
  MessageBox(MBStr, nil, mfInformation + mfOKButton);
  DriedOff:=0
end;


{*** PBovine ***}

constructor TBovine.Init(De : PANRec);
begin
  inherited Init(De);
  with WorkDetails do
  begin
    DaysWorked := '0';
    TimeWorked := '0';
    Implement := 0
```

```
  end
 end;

 constructor TBovine.Load(var S: TStream);
 begin
  inherited Load(S);
  S.Read(WorkDetails, SizeOf(TWKRec))
 end;

 procedure TBovine.Store(var S: TStream);
 begin
  inherited Store(S);
  S.Write(WorkDetails, SizeOf(TWKRec))
 end;

 function TBovine.ME_Maint: Real;

 begin
  ME_Maint := (0.42 * exp(0.73 * ln(Weight))) / (0.35 * Meal^.Q + 0.503)
 end;

 function TBovine.ME_Production: Real;
 var
  MEP, DW : Real;
  Error : Integer;
 begin
  val(WorkDetails.DaysWorked, DW, Error);
  if DW <> 0 then ME_Production := TRuminant.ME_Production - ME_Work
  else ME_Production := TRuminant.ME_Production
 end;

 function TBovine.MP_Maint: Real;
 var
  MPd: Real;
 begin
  MPd := 0.1125 * exp(0.75 * ln(Weight));
  MP_Maint := TRuminant.MP_Maint + MPd
 end;

 function TBovine.ME_Work: Real;
 var DraughtForce: Word;
    DistanceCropped: Word;
    DW, TW : Real;
    Error : Integer;
 begin
  {*** 0 = traditional plough; 1 = mouldboard plough; 2 =  spike toothed
    harrow ***}
  with WorkDetails do
  begin
   val(TimeWorked, TW, Error);
   val(DaysWorked, DW, Error);
   case Implement of
    0: DraughtForce := 65;
    1: DraughtForce := 90;
    2: DraughtForce := 50
   end
```

```
  end;
  case Description.Species of
    0: DistanceCropped := round(1.2 * TW * DW);
    1: DistanceCropped := round(1.2 * TW * DW);
    2: DistanceCropped := round(0.8 * TW * DW)
  end;
  ME_Work:=((2 * DistanceCropped * Weight)+
        (9.81 * DistanceCropped * DraughtForce / 0.298)){ /
        (1000 * ThisMonth.NoOfDays)}
end;


{*** TGrowingBovine ***}

function TGrowingBovine.MP_Gain: Real;
var CF, TWC: Real;
begin
 if WeightChange <> 0 then
 begin
   TWC := WeightChange;
   if WeightChange > 0 then
   begin
     CF := 1;
     Case Description.Species of
       0 : CF := CF + 0.10;          {*** buffalo ***}
       1, 2 : CF := CF - 0.10        {*** Bos taurus, Bos indicus ***}
     end;
     Case Description.Sex of
       0 : CF := CF + 0.10;          {*** bulls ***}
       1 : CF := CF - 0.10           {*** females ***}
     end;
     MP_Gain := CF * (1.695 *(TWC / 1000) *
           (168.07 - 0.16869 * Weight + 0.0001633 * Sqr(Weight)) *
           (1.12 - 0.1223 * TWC / 1000))
   end
   else MP_Gain := TWC * 138 / 1000
 end
 else MP_Gain := 0
end;

function TGrowingBovine.Gain_Loss_ME(ME: Real): Real;
var
 KFat: Real;
 CF: Real;    {*** Correction factor ***}
begin
 if ME <> 0 then
 begin
   if ME > 0 then
   begin
     CF := 1;
     Case Description.Species of
       0 : CF := CF + 0.15;          {*** buffalo ***}
       1, 2 : CF := CF - 0.15        {*** Bos taurus, Bos indicus ***}
     end;
     Case Description.Sex of
       0 : CF := CF + 0.15;          {*** bulls ***}
```

```
      1 : CF := CF - 0.15          {*** females ***}
     end;
     KFat := 0.78 * Meal^.Q + 0.0006;
     Gain_Loss_ME := (ME * CF) * KFat /
                (4.1 + 0.0332 * Weight - 0.000009 * Sqr(Weight) +
                (ME * CF) * Kfat * 0.1475) * 1000
    end
    else Gain_Loss_ME:=ME*1000/35.4    {*** as per dry suckler cow ***}
  end
  else Gain_Loss_ME:=0
end;

function TGrowingBovine.Gain_Loss_MP(MPG: Real): Real;
var
  CF: Real;
begin
  if MPG <> 0 then
  begin
    if MPG > 0 then
    begin
      CF := 1;
      Case Description.Species of
        0 : CF := CF - 0.10;          {*** buffalo ***}
        1, 2 : CF := CF + 0.10        {*** Bos taurus, Bos indicus ***}
      end;
      Case Description.Sex of
        0 : CF := CF - 0.10;          {*** bulls ***}
        1 : CF := CF + 0.10           {*** females ***}
      end;
      Gain_Loss_MP := 1000 * ((-1.8984 + Sqrt(3.6039 - 0.8292 * (MPG*CF) /
                (168.07 - 0.16869 * Weight + 0.0001633 *
                Sqr(Weight)))) / -0.4146)
    end
    else Gain_Loss_MP := round(1000 * MPG / 138)
  end
  else Gain_Loss_MP := 0
end;

function TGrowingBovine.UpDate(NumDays : Word) : PAnimal;
begin
  UpDate := New(PGrowingBovine, Init(Perform(NumDays)))
end;


{*** TMatureBovine ***}

function TMatureBovine.Gain_Loss_ME(ME : Real) : Real;
begin
  if ME > 0 then Gain_Loss_ME := 1000 * ME * (0.78 * Meal^.Q + 0.0006) / 27
  else
  begin
    if ME < 0 then Gain_Loss_ME := ME * 1000 / 35.4
    else Gain_Loss_ME := 0
  end
end;
```

```
function TMatureBovine.MP_Gain : Real;
begin
  If WeightChange <> 0 then
  begin
    if WeightChange > 0 then MP_Gain := 233 * WeightChange / 1000
    else MP_Gain := 138 * WeightChange / 1000
  end
  else MP_Gain := 0
end;

function TMatureBovine.Gain_Loss_MP(MPG : Real): Real;
begin
  if MPG <> 0 then
  begin
    if MPG > 0 then
    begin
      Gain_Loss_MP := 1000 * MPG / 233
    end
    else Gain_Loss_MP := round(1000 * MPG / 138)
  end
  else Gain_Loss_MP := 0
end;


{*** TMaleBovine ***}

function TMaleBovine.UpDate(NumDays : Word) : PAnimal;
begin
  UpDate := New(PMaleBovine, Init(Perform(NumDays)))
end;


{*** TFemaleBovine ***}

constructor TFemaleBovine.Init(De : PANRec; Mf : PMFRec);
begin
  inherited Init(De);
  ReproStat := Mf^;
  MilkYield:=0;
  case Description.Species of
    0: begin
         MilkFat:=80;
         MilkProt:=40
       end;
    1: begin
         MilkFat:=30;
         MilkProt:=35
       end;
    2: begin
         MilkFat:=50;        {*** Zebu; Preston and Leng, 1987 ***}
         MilkProt:=30
       end
  end
end;

constructor TFemaleBovine.Load(var S: TStream);
```

```
  begin
    inherited Load(S);
    S.Read(ReproStat, SizeOf(TMFRec));
    S.Read(MilkYield, SizeOf(Real));
    S.Read(MilkFat, SizeOf(Real));
    S.Read(MilkProt, SizeOf(Real))
  end;

  procedure TFemaleBovine.Store(var S: TStream);
  begin
    inherited Store(S);
    S.Write(ReproStat, SizeOf(TMFRec));
    S.Write(MilkYield, SizeOf(Real));
    S.Write(MilkFat, SizeOf(Real));
    S.Write(MilkProt, SizeOf(Real))
  end;

  function TFemaleBovine.ME_Preg: Real;
  Var
    DP : Real;
    Error : Integer;
  {*** Needs more work ***}
  begin
  {***  ME_Preg:=(1.13*exp(DayOfPreg)*0.0106)/1000  Check this out ***}

    with ReproStat do val(DayOfPregnancy, DP, Error);

      if DP > 0 then       {*** from table in ARC, 1980 *** }
      begin
        if DP <= 140 then ME_Preg  := 6.5
          else if DP < 170 then ME_Preg := 11
            else if DP < 200 then ME_Preg := 19
              else if DP < 240 then ME_Preg := 34
      end
      else ME_Preg:=0
    {*** correction for birthweight is * birthweight / 40 ***}
  end;

  function TFemaleBovine.ME_Lact: Real;
  Var
    DL : Real;
    Error : Integer;
  begin
    with ReproStat do val(DayOfLactation, DL, Error);
    if DL <> 0 then
    ME_Lact := ((0.0405 * MilkFat + 0.0215 * (MilkProt + 46.4 {Lactose} ) -
            0.247) / (0.35 * Meal^.Q + 0.42)) * MilkYield
    else ME_Lact := 0
  end;

  function TFemaleBovine.MP_Preg: Real;

  var
    TPt: Real;
    DP : Real;
    Error : Integer;
```

```
 begin
   with ReproStat do val(DayOfPregnancy, DP, Error);
   if DP > 0 then
   begin
    TPt := exp((3.707 - 5.698 * exp(-0.00262 * DP)) * ln(10));
    MP_Preg := 40.4 * TPt * exp(-0.00262 * DP);
   end
   else MP_Preg := 0
   {*** correction for birthweight is * birthweight / 40 ***}
 end;

 function TFemaleBovine.MP_Lact: Real;
 Var
  DL : Real;
  Error : Integer;
 begin
  with ReproStat do val(DayOfLactation, DL, Error);
  if DL > 0 then MP_Lact := MilkYield * MilkProt * 0.95 * 1.471
  else MP_Lact := 0
 end;


 function TFemaleBovine.Gain_Loss_ME(ME: Real): Real;
 Var
  DL : Real;
  Error : Integer;
 begin
  with ReproStat do val(DayOfLactation, DL, Error);
  if DL > 0 then
  begin
    if ME > 0 then Gain_Loss_ME := 1000 * ME * (0.35 * Meal^.Q + 0.42) / 27.4
    else
    begin
     if ME < 0 then Gain_Loss_ME := 1000 * ME * 0.82 / 27
     else Gain_Loss_ME := 0
    end
  end
  else Gain_Loss_ME := TMatureBovine.Gain_Loss_ME(ME)
 end;

 function TFemaleBovine.ME_Production;
 begin
  ME_Production := TBovine.ME_Production - ME_Preg - ME_Lact
 end;

 function TFemaleBovine.MPRequired: Real;
 begin
  MPRequired := TMatureBovine.MPRequired + MP_Preg + MP_Lact
 end;

 Procedure TFemaleBovine.ProductionLevel;
 const
  MaxGain=500;
  MaxLoss=-500;    {*** should vary with bodyweight ***}
 var
```

```
  ME_Deficit, TWC, TMY: Real;
  DL, DP : Real;
  Error : Integer;
 begin
  with ReproStat do
  begin
   val(DayOfLactation, DL, Error);
   val(DayOfPregnancy, DP, Error)
  end;
  if DL > 0 then
   case Description.Species of
    0 : MilkYield := 4;
    1 : MilkYield := 2;
    2 : MilkYield := 2
   end;
  if ME_Production < 0 then
  begin
   ME_Deficit := ME_Production;
   TWC := (ME_Deficit * 1000 * 0.82) / 27;
   if TWC < MaxLoss then
   begin
    ME_Deficit := ME_Deficit - ((MaxLoss) * 27 / (1000 * 0.82));
    WeightChange := MaxLoss;
    if DL > 0 then
    begin
     TMY := ME_Deficit / ((0.405 * MilkFat + 0.215 * (MilkProt + 4.64) -
          0.247) / (0.35*Meal^.Q+0.42));
     if abs(TMY) < MilkYield then MilkYield := MilkYield - abs(TMY)
     else
     begin
      ME_Deficit := ME_Deficit - ME_Lact;
      MilkYield := 0;
      DL := DriedOff('Energy');
      Str(DL, ReproStat.DayOfLactation)
     end
    end;
    if DP > 0 then
     if ME_Deficit < 0 then
     begin
      DP := AbortPregnancy ('Energy');
      Str(DP, ReproStat.DayOfPregnancy)
     end
  end
  else WeightChange := round(TWC)
 end
 else WeightChange := round(Gain_Loss_ME(ME_Production))


 { Increment dop and dol}


end;
```

```
procedure TFemaleBovine.ProteinStatus;
const
  MaxGain=500;
  MaxLoss=-500;    {*** should vary with bodyweight ***}
var
  MP_Deficit, TWC, TMY: Real;
  DL, DP : Real;
  Error : Integer;
begin
  with ReproStat do
  begin
    val(DayOfLactation, DL, Error);
    val(DayOfPregnancy, DP, Error)
  end;
  if Meal^.MP < MPRequired then
  begin
    MP_Deficit := Meal^.MP - MPRequired;
    TWC := Gain_Loss_MP(MP_Deficit);

    {*** Priority 1 - maintain pregnancy ***}
    {*** Priority 2 - maintain milk yield ***}

    if TWC < MaxLoss then
    begin
      MP_Deficit := MP_Deficit - ((MaxLoss) * 138 / 1000);
      TWC := MaxLoss;

      {*** start by reducing milk yield ***}

      if DL > 0 then
      begin
        TMY := MilkYield + ((1.471 * 0.95 * MilkProt) / MP_Deficit);
        if TMY > 0 then MilkYield := TMY
        else
        begin
          MP_Deficit := MP_Deficit - MP_Lact;

          {***      (1.471*0.95*MilkProt)/MilkYield;    ***}

          MilkYield := 0;
          DL := DriedOff('Protein');
          Str(DL, ReproStat.DayOfLactation)
        end
      end;
      if DP > 0 then
        if MP_Deficit < 0 then
        begin
          DP:=AbortPregnancy('Protein');
          Str(DP, ReproStat.DayOfPregnancy)
        end
    end
  end
  else
  begin
    TWC := Gain_Loss_MP(Meal^.MP - MPRequired);
  end;
```

```
  if TWC < WeightChange then WeightChange := round(TWC);

  {*** need to calculate a value for KN ***}

  if MP_Maint < Meal^.MP then
  begin
    KN := (MP_Maint/Meal^.MP) * 0.85;
    if WeightChange > 0 then KN := KN + (MP_Gain / Meal^.MP) * 0.59;
    if MilkYield > 0 then KN := KN + (MP_Lact / Meal^.MP) * 0.68;
    if DP > 0 then KN := KN + (MP_Preg / Meal^.MP) * 0.85
  end
  else
  KN := 0.85
end;

function TFemaleBovine.UpDate(NumDays : Word) : PAnimal;
Var
  DL, DP, W : Real;
  Error : Integer;
  NewAnimal : PFemaleBovine;
begin
  NewAnimal := New(PFemaleBovine, Init(Perform(NumDays), nil));
  with ReproStat do
  begin
    val(DayOfLactation, DL, Error);
    val(DayOfPregnancy, DP, Error);
    if DL <> 0 then DL := DL + NumDays;
    if DP <> 0 then
    begin
      DP := DP + NumDays;
      {acount for growth of foetus to 40kg birthweight}
      if DP <= 160 then W := NewAnimal^.Weight;
      if (DP > 160) and (DP <= 240) then
        W := NewAnimal^.Weight + NumDays * 0.25;
      if (DP > 240) and (DP <= 285) then
        W := NewAnimal^.Weight + NumDays * 0.5;
      if DP > 285 then W := NewAnimal^.Weight + (NumDays - (DP - 285)) * 0.5;
      Str(W:3:0, NewAnimal^.Description.LiveWeight)
    end;
    if DP > 285 then
    begin
      {parturition}
      {create new offspring and insert into herd}
      DP := 0;
      DL := DP - 285;
      W := NewAnimal^.Weight - 40;
      Str(W:3:0, NewAnimal^.Description.LiveWeight)
    end;
    Str(DL:3:0, NewAnimal^.ReproStat.DayOfLactation);
    Str(DP:3:0, NewAnimal^.ReproStat.DayOfPregnancy)
  end;
  UpDate := NewAnimal
end;

procedure TFemaleBovine.PrintData(var F : Text);
begin
```

```
  inherited PrintData(F);
  FillString(20 - Length(Description.LiveWeight), F);
  write(F, MilkYield:4:1)
 end;

 {*** TCaprine ***}

 function TCaprine.ME_Maint: Real;
 {*** Currently same as sheep ***}
 begin
  ME_Maint := (0.293 * exp(0.73 * ln(Weight))) / (0.35 * Meal^.Q + 0.503)
 end;

 function TCaprine.MP_Maint: Real;
 var MPd : Real;
 begin
  MPd := 0.1125 * exp(0.75 * ln(Weight));
  MP_Maint := TRuminant.MP_Maint + MPd
 end;


{*** TGrowingCaprine ***}

function TGrowingCaprine.MP_Gain : Real;
begin
 if WeightChange <> 0 then
 begin
  if WeightChange > 0 then MP_Gain := 1.695 * (WeightChange / 1000) *
      (160.4 - 1.22 * Weight + 0.0105 * Sqr(Weight))
  else MP_Gain := WeightChange * 119 / 1000
 end
 else MP_Gain := 0
end;

function TGrowingCaprine.Gain_Loss_ME(ME : Real) : Real;
var KFat : Real;
begin
 if ME <> 0 then
 begin
  if ME > 0 then
  begin
   KFat := 0.78 * Meal^.Q + 0.0006;
   Case Description.Sex of
    0 : Gain_Loss_ME := (ME * KFat / (2.5 + 0.35 * Weight)) * 1000;  {*** Buck ***}
    1 : Gain_Loss_ME := (ME * KFat / (2.1 + 0.45 * Weight)) * 1000;  {*** doe ***}
    2 : Gain_Loss_ME := (ME * KFat / (4.5 + 0.32 * Weight)) * 1000   {*** castrate ***}
   end
  end
  else Gain_Loss_ME := ME * 1000 / 28
 end
 else Gain_Loss_ME := 0
end;

function TGrowingCaprine.Gain_Loss_MP(MPG: Real): Real;
begin
 if MPG <> 0 then
```

```
   begin
    if MPG > 0 then

    {*** fleece free lwg for sheep ***}

     case Description.Sex of
      0, 2 : Gain_Loss_MP := (1000 * MPG) / (1.695 * (160.4 -
                    1.22 * Weight + 0.0105 * Sqr(Weight)));
      1 : Gain_Loss_MP := (1000 * MPG) / (1.695 * (156.1 - 1.94 *
                  Weight + 0.0173 * Sqr(Weight)))
     end
    else Gain_Loss_MP := round(1000 * MPG / 119{???})
   end
   else Gain_Loss_MP := 0
 end;

function TGrowingCaprine.UpDate(NumDays : Word) : PAnimal;
begin
  UpDate := New(PGrowingCaprine, Init(Perform(NumDays)))
end;


{*** TMatureCaprine ***}

function TMatureCaprine.Gain_Loss_ME(ME : Real) : Real;
begin
  if ME > 0 then Gain_Loss_ME := 1000 * ME / 34
  else
  begin
    if ME < 0.then Gain_Loss_ME := 1000 * ME / 28
    else Gain_Loss_ME := 0
  end
end;

function TMatureCaprine.MP_Gain: Real;
begin
  If WeightChange <> 0 then
  begin
    if WeightChange > 0 then MP_Gain :=WeightChange * 140 / 1000
    else MP_Gain := WeightChange * 119 / 1000
  end
  else MP_Gain := 0
end;

function TMatureCaprine.Gain_Loss_MP(MPG : Real) : Real;
begin
  if MPG <> 0 then
  begin
    if MPG > 0 then Gain_Loss_MP := 1000 * MPG / 140
    else Gain_Loss_MP := round(1000 * MPG / 119)
  end
  else Gain_Loss_MP := 0
end;


{*** TMaleCaprine ***}
```

```
function TMaleCaprine.UpDate(NumDays : Word) : PAnimal;
begin
  UpDate := New(PMaleCaprine, Init(Perform(NumDays)))
end;


{*** TFemaleCaprine ***}

constructor TFemaleCaprine.Init(De : PANRec; Mf : PMFRec);
begin
  inherited Init(De);
  ReproStat := Mf^;
  MilkYield:=0;
  MilkFat:=70;      {*** Check appropriate value for does ***}
  MilkProt:=45
end;

constructor TFemaleCaprine.Load(var S: TStream);
begin
  inherited Load(S);
  S.Read(ReproStat, SizeOf(TMFRec));
  S.Read(MilkYield, SizeOf(Real));
  S.Read(MilkFat, SizeOf(Real));
  S.Read(MilkProt, SizeOf(Real))
end;

procedure TFemaleCaprine.Store(var S: TStream);
begin
  inherited Store(S);
  S.Write(ReproStat, SizeOf(TMFRec));
  S.Write(MilkYield, SizeOf(Real));
  S.Write(MilkFat, SizeOf(Real));
  S.Write(MilkProt, SizeOf(Real))
end;

function TFemaleCaprine.ME_Preg : Real;
Var
  DP : Real;
  Error : Integer;
begin
  with ReproStat do val(DayOfPregnancy, DP, Error);
  if DP > 0 then
    ME_Preg := ((exp((3.322 - 4.979 * (exp(-0.00643 * DP))) * ln(10))) *
          0.07372 * (exp(-0.00643 * DP))) / 0.133
  else ME_Preg := 0
end;

function TFemaleCaprine.ME_Lact : Real;
Var
  DL : Real;
  Error : Integer;
begin
  with ReproStat do val(DayOfLactation, DL, Error);
  if DL > 0 then
    ME_Lact := (((0.0328 * MilkFat) + 0.0025 * DL + 2.203) /
```

```
            ((0.35 * Meal^.Q) + 0.420)) * MilkYield
  else ME_Lact := 0
end;

function TFemaleCaprine.MP_Preg: Real;
var
  TPt: Real;
  DP : Real;
  Error : Integer;
begin

  {*** single foetus only ***}

  with ReproStat do val(DayOfPregnancy, DP, Error);
  if DP > 0 then
  begin
    TPt := exp((4.928 - 4.873 * exp(-0.00601 * DP)) * ln(10));
    MP_Preg := 0.079 * TPt * exp(-0.00601 * DP)
  end
  else MP_Preg := 0

  {*** correction for birthweight is * birthweight / 4 ***}

end;

function TFemaleCaprine.MP_Lact: Real;
Var
  DL : Real;
  Error : Integer;
begin
  with ReproStat do val(DayOfLactation, DL, Error);
  if DL > 0 then MP_Lact := MilkYield * MilkProt * 1.471
  else MP_Lact := 0
end;

function TFemaleCaprine.ME_Production : Real;
begin
  ME_Production := TRuminant.ME_Production - ME_Preg - ME_Lact
end;

function TFemaleCaprine.MPRequired : Real;
begin
  MPRequired := TMatureCaprine.MPRequired + MP_Preg + MP_Lact
end;


Procedure TFemaleCaprine.ProductionLevel
const
  MaxGain=50;    {*** ??? ***}
  MaxLoss=-50;   {*** should vary with bodyweight ***}
Var
  ME_Deficit, TWC, TMY : Real;
  DL, DP : Real;
  Error : Integer;
begin
  with ReproStat do
```

```
    begin
      val(DayOfLactation, DL, Error);
      val(DayOfPregnancy, DP, Error)
    end;
    if DL > 0 then MilkYield := 0.3;
    if ME_Production < 0 then
    begin
      ME_Deficit := ME_Production;
      TWC := Gain_Loss_ME(ME_Deficit);
      if TWC < MaxLoss then
      begin
        ME_Deficit := ME_Deficit - ((MaxLoss) * 28 / 1000);
        WeightChange := MaxLoss;
        if DL > 0 then
        begin
          TMY := ME_Deficit / (((0.0328 * 73) + 2.203) / ((0.35 * Meal^.Q) +
                0.420));
          if abs(TMY) < MilkYield then MilkYield := MilkYield - abs(TMY)
          else
          begin
            ME_Deficit := ME_Deficit - ME_Lact;
            MilkYield := 0;
            DL := DriedOff('Energy');
            str(DL, ReproStat.DayOfLactation)
          end
        end;
        if DP > 0 then
        begin
          if ME_Deficit < 0 then DP := AbortPregnancy('Energy');
          str(DP, ReproStat.DayOfPregnancy)
        end
      end
      else WeightChange := round(TWC)
    end
    else WeightChange := round(Gain_Loss_ME(ME_Production))
  end;


  procedure TFemaleCaprine.ProteinStatus;
  const
    MaxGain=50;    {*** ??? ***}
    MaxLoss=-50;   {*** should vary with bodyweight ***}
  Var
    MP_Deficit, TWC, TMY: Real;
    DL, DP : Real;
    Error : Integer;
  begin
    with ReproStat do
    begin
      val(DayOfLactation, DL, Error);
      val(DayOfPregnancy, DP, Error)
    end;
    if Meal^.MP < MPRequired then
    begin
      MP_Deficit := Meal^.MP - MPRequired;
      TWC := Gain_Loss_MP(MP_Deficit);
```

```
      {*** Priority 1 - maintain pregnancy ***}
      {*** Priority 2 - maintain milk yield ***}

      if TWC < MaxLoss then
      begin
       MP_Deficit := MP_Deficit - ((MaxLoss) * 119 / 1000);
       TWC := MaxLoss;

       {*** start by reducing milk yield ***}

       if DL > 0 then
       begin
        TMY := MilkYield + ((1.471 * MilkProt) / MP_Deficit);
        if TMY > 0 then MilkYield := TMY
        else
        begin
         MP_Deficit := MP_Deficit - MP_Lact;

         {***     (1.471*0.95*MilkProt)/MilkYield;   ***}

         MilkYield := 0;
         DL := DriedOff('Protein');
         str(DL, ReproStat.DayOfLactation)
        end
       end;
       if DP > 0 then
        if MP_Deficit < 0 then
        begin
         DP:=AbortPregnancy('Protein');
         str(DP, ReproStat.DayOfPregnancy)
        end
     end
    end
    else
    begin
     TWC := Gain_Loss_MP(Meal^.MP - MPRequired);
    end;
    if TWC < WeightChange then WeightChange := round(TWC);
    if MP_Maint < Meal^.MP then
    begin
     KN := (MP_Maint / Meal^.MP) * 0.85;
     if WeightChange > 0 then KN := KN + (MP_Gain / Meal^.MP) * 0.59;
     if MilkYield > 0 then KN := KN + (MP_Lact / Meal^.MP) * 0.68;
     if DP > 0 then KN := KN + (MP_Preg / Meal^.MP) * 0.85
    end
    else
    KN := 0.85
  end;

  function TFemaleCaprine.UpDate(NumDays : Word) : PAnimal;
  Var
    DL, DP : Real;
    Error : Integer;
    NewAnimal : PFemaleCaprine;
  begin
```

```
      NewAnimal := New(PFemaleCaprine, Init(Perform(NumDays), nil));
      with ReproStat do
      begin
       val(DayOfLactation, DL, Error);
       val(DayOfPregnancy, DP, Error);
       if DL <> 0 then DL := DL + NumDays;
       if DP <> 0 then DP := DP + NumDays;
       if DP > 285 then
       begin
        {parturition}
        {create new offspring and insert into herd}
        DP := 0;
        DL := DP - 285
       end;
      Str(DL, NewAnimal^.ReproStat.DayOfLactation);
      Str(DP, NewAnimal^.ReproStat.DayOfPregnancy)
    end;
    UpDate := NewAnimal
  end;

  procedure TFemaleCaprine.PrintData(var F : Text);
  begin
   inherited PrintData(F);
   FillString(20 - Length(Description.LiveWeight), F);
   write(F, MilkYield:4:1)
  end;


  {*** TOvine ***}

  function TOvine.ME_Maint: Real;
  begin
   ME_Maint := (0.293 * exp(0.73 * ln(Weight))) / (0.35 * Meal^.Q + 0.503)
  end;


  {*** TGrowingOvine ***}

  function TGrowingOvine.MP_Gain : Real;
  begin
   if WeightChange <> 0 then
   begin
    if WeightChange > 0 then
     case Description.Sex of
      0, 2 : MP_Gain :=(WeightChange / 1000) * ( 334 - 2.54 * Weight +
               0.022 * Sqr(Weight)) + 11.5;
      1 : MP_Gain :=(WeightChange / 1000) * (325 - 4.03 * Weight +
             0.036 * Sqr(Weight)) + 11.5
     end
    else MP_Gain := WeightChange * 119 / 1000
   end
   else MP_Gain := 0
  end;

  function TGrowingOvine.Gain_Loss_ME(ME : Real) : Real;
  Var
```

```
  KFat: Real;
begin
  if ME <> 0 then
  begin
    if ME > 0 then
    begin
      KFat := 0.78 * Meal^.Q + 0.0006;
      Case Description.Sex of
        0: Gain_Loss_ME := (ME * KFat / (2.5 + 0.35 * Weight)) * 1000;   {*** ram ***}
        1: Gain_Loss_ME := (ME * KFat / (2.1 + 0.45 * Weight)) * 1000;   {*** ewe ***}
        2: Gain_Loss_ME := (ME * KFat / (4.5 + 0.32 * Weight)) * 1000    {*** hoggett ***}
      end;
    end
    else Gain_Loss_ME := ME * 1000 / 28     {*** as per dry ewe ***}
  end
  else Gain_Loss_ME := 0
end;

function TGrowingOvine.Gain_Loss_MP(MPG : Real) : Real;
begin
  if MPG <> 0 then
  begin
    if MPG > 0 then
    case Description.Sex of
      0, 2 : Gain_Loss_MP := 1000 * ((MPG - 11.5) / (334 - 2.54 * Weight +
                    0.022 * Sqr(Weight)));
      1 : Gain_Loss_MP := 1000 * ((MPG - 11.5) / (325 - 4.03 * Weight +
                  0.036 * Sqr(Weight)))
    end
    else Gain_Loss_MP := round(1000 * MPG / 119)
  end
  else Gain_Loss_MP := 0
end;

function TGrowingOvine.UpDate(NumDays : Word) : PAnimal;
begin
  UpDate := New(PGrowingOvine, Init(Perform(NumDays)))
end;


{*** TMatureOvine ***}

function TMatureOvine.Gain_Loss_ME(ME : Real) : Real;
begin
  if ME > 0 then Gain_Loss_ME := 1000 * ME / 34
  else
  begin
    if ME < 0 then Gain_Loss_ME := 1000 * ME / 28
    else Gain_Loss_ME := 0
  end
end;

function TMatureOvine.MP_Maint : Real;
begin
  MP_Maint := TOvine.MP_Maint + 20.4
end;
```

```
function TMatureOvine.MP_Gain: Real;
begin
 If WeightChange <> 0 then
 begin
  if WeightChange > 0 then MP_Gain := WeightChange * 140 / 1000
  else MP_Gain := WeightChange * 119 / 1000
 end
 else MP_Gain := 0
end;

function TMatureOvine.Gain_Loss_MP(MPG : Real) : Real;
begin
 if MPG <> 0 then
 begin
  if MPG > 0 then
  begin
   Gain_Loss_MP := 1000 * MPG / 140
  end
  else Gain_Loss_MP := round(1000 * MPG / 119)
 end
 else Gain_Loss_MP := 0
end;


{*** TMaleOvine ***}

function TMaleOvine.UpDate(NumDays : Word) : PAnimal;
begin
 UpDate := New(PMaleOvine, Init(Perform(NumDays)))
end;


{*** TFemaleOvine ***}

constructor TFemaleOvine.Init(De : PANRec; Mf : PMFRec);
begin
 inherited Init(De);
 ReproStat := Mf^;
 MilkYield:=0;
 MilkFat:=73;     {*** Check appropriate value for ewes ***}
 MilkProt:=48.9
end;

constructor TFemaleOvine.Load(var S: TStream);
begin
 inherited Load(S);
 S.Read(ReproStat, SizeOf(TMFRec));
 S.Read(MilkYield, SizeOf(Real));
 S.Read(MilkFat, SizeOf(Real));
 S.Read(MilkProt, SizeOf(Real))
end;

procedure TFemaleOvine.Store(var S: TStream);
begin
 inherited Store(S);
```

```
  S.Write(ReproStat, SizeOf(TMFRec));
  S.Write(MilkYield, SizeOf(Real));
  S.Write(MilkFat, SizeOf(Real));
  S.Write(MilkProt, SizeOf(Real))
 end;

 function TFemaleOvine.ME_Preg: Real;
 Var
  DP : Real;
  Error : Integer;
 begin
  with ReproStat do val(DayOfPregnancy, DP, Error);
  if DP > 0 then
   ME_Preg := ((exp((3.322 - 4.979 * (exp(-0.00643 * DP))) * ln(10))) *
         0.07372 * (exp(-0.00643 * DP))) / 0.133
  else ME_Preg := 0
 end;

function TFemaleOvine.ME_Lact: Real;
Var
 DL : Real;
 Error : Integer;
begin
 with ReproStat do val(DayOfLactation, DL, Error);
 if DL <> 0 then
  ME_Lact := (((0.0328 * MilkFat) + 0.0025 * DL + 2.203) /
        ((0.35 * Meal^.Q) + 0.420)) * MilkYield
 else ME_Lact := 0
end;

function TFemaleOvine.MP_Preg: Real;
Var
 TPt: Real;
 DP : Real;
 Error : Integer;
begin
 with ReproStat do val(DayOfPregnancy, DP, Error);

  {*** single foetus only ***}

 if DP > 0 then
 begin
  TPt := exp((4.928 - 4.873 * exp(-0.00601 * DP)) * ln(10));
  MP_Preg := 0.079 * TPt * exp(-0.00601 * DP)
 end
 else MP_Preg := 0

  {*** correction for birthweight is *birthweight/4 ***}

end;

function TFemaleOvine.MP_Lact: Real;
Var
 DL : Real;
 Error : Integer;
begin
```

```
  with ReproStat do val(DayOfLactation, DL, Error);
  if DL > 0 then MP_Lact := MilkYield * MilkProt * 1.471
  else MP_Lact := 0
end;

function TFemaleOvine.ME_Production;
begin
  ME_Production := TRuminant.ME_Production - ME_Preg - ME_Lact
end;

function TFemaleOvine.MPRequired: Real;
begin
  MPRequired := TMatureOvine.MPRequired + MP_Preg + MP_Lact
end;


Procedure TFemaleOvine.ProductionLevel;
const
  MaxGain=50;    {*** ??? ***}
  MaxLoss=-50;   {*** should vary with bodyweight ***}
var
  ME_Deficit, TWC, TMY: Real;
  DL, DP : Real;
  Error : Integer;
begin
  with ReproStat do
  begin
    val(DayOfLactation, DL, Error);
    val(DayOfPregnancy, DP, Error)
  end;
  if DL > 0 then MilkYield := 0.3;
  if ME_Production < 0 then
  begin
    ME_Deficit := ME_Production;
    TWC := Gain_Loss_ME(ME_Deficit);
    if TWC < MaxLoss then
    begin
      ME_Deficit := ME_Deficit - ((MaxLoss) * 28 / 1000);
      WeightChange := MaxLoss;
      if DL > 0 then
      begin
        TMY := ME_Deficit / (((0.0328 * 73) + 2.203) /
             ((0.35 * Meal^.Q) + 0.420));
        if abs(TMY) < MilkYield then MilkYield := MilkYield - abs(TMY)
        else
        begin
          ME_Deficit := ME_Deficit - ME_Lact;
          MilkYield := 0;
          DL := DriedOff('Energy');
          str(DL, ReproStat.DayOfLactation)
        end
      end;
      if DP > 0 then
        if ME_Deficit < 0 then
        begin
          DP := AbortPregnancy('Energy');
```

```
        str(DP, ReproStat.DayOfPregnancy)
       end;
    end
    else WeightChange := round(TWC)
  end
  else WeightChange := round(Gain_Loss_ME(ME_Production))
end;

procedure TFemaleOvine.ProteinStatus;
const
  MaxGain=50;    {*** ??? ***}
  MaxLoss=-50;   {*** should vary with bodyweight ***}
var
  MP_Deficit, TWC, TMY: Real;
  DL, DP : Real;
  Error : Integer;
begin
  with ReproStat do
  begin
    val(DayOfLactation, DL, Error);
    val(DayOfPregnancy, DP, Error)
  end;
  if Meal^.MP < MPRequired then
  begin
    MP_Deficit := Meal^.MP - MPRequired;
    TWC := Gain_Loss_MP(MP_Deficit);

    {*** Priority 1 - maintain pregnancy ***}
    {*** Priority 2 - maintain milk yield ***}

    if TWC < MaxLoss then
    begin
      MP_Deficit := MP_Deficit - ((MaxLoss) * 119 / 1000);
      TWC := MaxLoss;

      {*** start by reducing milk yield ***}

      if DL > 0 then
      begin
        TMY := MilkYield + ((1.471 * MilkProt) / MP_Deficit);
        if TMY > 0 then MilkYield := TMY
        else
        begin
          MP_Deficit := MP_Deficit - MP_Lact;

        {***     (1.471*0.95*MilkProt)/MilkYield;    ***}

          MilkYield := 0;
          DL := DriedOff('Protein');
          str(DL, ReproStat.DayOfLactation)
        end
      end;
      if DP > 0 then
        if MP_Deficit < 0 then
        begin
          DP := AbortPregnancy('Protein');
```

```
          str(DP, ReproStat.DayOfPregnancy)
        end
      end
    end
    else
    begin
      TWC := Gain_Loss_MP(Meal^.MP - MPRequired);
    end;
    if TWC < WeightChange then WeightChange := round(TWC);

    if MP_Maint < Meal^.MP then
    begin
      KN := (MP_Maint / Meal^.MP) * 0.85;
      if WeightChange > 0 then KN := KN + (MP_Gain / Meal^.MP) * 0.59;
      if MilkYield > 0 then KN := KN + (MP_Lact / Meal^.MP) * 0.68;
      if DP > 0 then KN := KN + (MP_Preg / Meal^.MP) * 0.85
    end
    else
    KN := 0.85
  end;

  function TFemaleOvine.UpDate(NumDays : Word) : PAnimal;
  Var
    DL, DP : Real;
    Error : Integer;
    NewAnimal : PFemaleOvine;
  begin
    NewAnimal := New(PFemaleOvine, Init(Perform(NumDays), nil));
    with ReproStat do
    begin
      val(DayOfLactation, DL, Error);
      val(DayOfPregnancy, DP, Error);
      if DL <> 0 then DL := DL + NumDays;
      if DP <> 0 then DP := DP + NumDays;
      if DP > 285 then
      begin
        {parturition}
        {create new offspring and insert into herd}
        DP := 0;
        DL := DP - 285
      end;
      Str(DL, NewAnimal^.ReproStat.DayOfLactation);
      Str(DP, NewAnimal^.ReproStat.DayOfPregnancy)
  end;
  UpDate := NewAnimal
end;

procedure TFemaleOvine.PrintData(var F : Text);
begin
  inherited PrintData(F);
  FillString(20 - Length(Description.LiveWeight), F);
  write(F, MilkYield:4:1)
end;


{*** THerd ***}
```

```
function THerd.KeyOf(Item : Pointer) : Pointer;
begin
  KeyOf := NewStr(PAnimal(Item)^.Name);
end;

function THerd.Compare(Key1, Key2 : Pointer) : Integer;
begin
  if PString(Key1)^ = PString(Key2)^ then Compare := 0
  else if PString(Key1)^ < PString(Key2)^ then Compare := -1
  else Compare := 1;
end;

procedure THerd.MakeViewerList(var L: PStringCollection);

  procedure OneAnimal(Item : PAnimal); far;
  Var
    S : String;
  begin
    with Item^ do
    begin
      if Meal^.Count <> 0 then S := '*' else S := '';
      L^.Insert(NewStr(Name + S))
    end
  end;

begin
  L^.FreeAll;
  ForEach(@OneAnimal)
end;

procedure THerd.ViewData(Var F : Text);
var I : Integer;

  procedure OneAnimal(Item : PAnimal); far;
  begin
    Item^.PrintData(F);
    writeln(F)
  end;

begin
  writeln(F);
  write(F, 'Animal');
  FillString(34, F);
  write(F, 'Weight');
  FillString(14, F);
  writeln(F, 'Milk Yield');
  for I := 1 to 80 do write(F, '_');
  writeln(F);
  writeln(F);
  ForEach(@OneAnimal);
  for I := 1 to 80 do write(F, '_');
  writeln(F);
end;

{*** Stream Registration ***}
```

```
procedure RegisterAnimals;
begin
  RegisterType(RAnimal);
  RegisterType(RRuminant);
  RegisterType(RBovine);
  RegisterType(RGrowingBovine);
  RegisterType(RMatureBovine);
  RegisterType(RMaleBovine);
  RegisterType(RFemaleBovine);
  RegisterType(RCaprine);
  RegisterType(RGrowingCaprine);
  RegisterType(RMatureCaprine);
  RegisterType(RMaleCaprine);
  RegisterType(RFemaleCaprine);
  RegisterType(ROvine);
  RegisterType(RGrowingOvine);
  RegisterType(RMatureOvine);
  RegisterType(RMaleOvine);
  RegisterType(RFemaleOvine);
  RegisterType(RHerd)
end;

END.
```

```
unit DATABASE;

interface

uses Objects,
    Dos,
    Globals;

type
 PFeedData = ^TFeedData;
 TFeedData = object(TObject)

   Composition: TFCRec;

   constructor Init(C: PFCRec);
   constructor Load(var S: TStream);
   procedure Store(var S: TStream);
 end;

 PDataBase = ^TDataBase;
 TDataBase = object(TSortedCollection)

   FileName: PathStr;

   constructor Init(ALimit, ADelta: Integer);
   destructor Done; virtual;
   procedure Reset;
   constructor Load(var S: TStream);
   procedure Store(var S: TStream);
   function KeyOf(Item: Pointer): Pointer; virtual;
   function Compare(Key1, Key2: Pointer): Integer; virtual;
   procedure MakeViewerList(var L: PStringCollection);
 end;

const
 RFeedData: TStreamRec = (
   ObjType: 300;
   VmtLink: Ofs(TypeOf(TFeedData)^);
   Load: @TFeedData.Load;
   Store: @TFeedData.Store
 );

 RDataBase: TStreamRec = (
   ObjType: 310;
   VmtLink: Ofs(TypeOf(TDataBase)^);
   Load: @TDataBase.Load;
   Store: @TDataBase.Store
 );

 procedure RegisterDataBase;


implementation

{*** TFeedData ***}
```

```
constructor TFeedData.Init(C: PFCRec);
begin
  inherited Init;
  Composition:=C^
end;

constructor TFeedData.Load(var S: TStream);
begin
  S.Read(Composition, SizeOf(TFCRec))
end;

procedure TFeedData.Store(var S: TStream);
begin
  S.Write(Composition, SizeOf(TFCRec))
end;


{*** TDataBase ***}

constructor TDataBase.Init(ALimit, ADelta: Integer);
begin
  inherited Init(ALimit, ADelta);
  FileName:='*.FDB'
end;

destructor TDataBase.Done;
begin
  inherited Done
end;

procedure TDataBase.Reset;
begin
  FileName:='*.FDB';
  FreeAll
end;

constructor TDataBase.Load(var S: TStream);
begin
  inherited Load(S);
  S.Read(FileName, SizeOf(PathStr))
end;

procedure TDataBase.Store(var S: TStream);
begin
  inherited Store(S);
  S.Write(FileName, SizeOf(PathStr))
end;

function TDataBase.KeyOf(Item: Pointer): Pointer;
begin
  KeyOf:= NewStr(PFeedData(Item)^.Composition.Name);
end;

function TDataBase.Compare(Key1, Key2: Pointer): Integer;
begin
  if PString(Key1)^ = PString(Key2)^ then Compare:=0
```

```
  else if PString(Key1)^ < PString(Key2)^ then Compare:=-1
  else Compare:=1;
end;

procedure TDataBase.MakeViewerList(var L: PStringCollection);

  procedure OneFeed(Item: PFeedData); far;
  begin
    L^.Insert(NewStr(Item^.Composition.Name));
  end;

begin
  L^.FreeAll;
  ForEach(@OneFeed);
end;


{*** Stream Registration ***}

procedure RegisterDataBase;
begin
  RegisterType(RFeedData);
  RegisterType(RDataBase)
end;

END.
```

```
unit DATES;

interface

uses Objects, Feeds, Animals, Muck, Globals;

type

 DateStr = String[8];
 PartStr = String[2];

 PDate = ^TDate;
 TDate = object(TObject)

  Date : DateStr;
  DD : Word;
  MM : Word;
  YY : Word;
  Previous : DateStr;

  Feeds : PFeeds;
  Herd : PHerd;
  Compost : PCompostHeap;

  constructor Init(D, P: DateStr);
  constructor Load(var S: TStream);
  procedure Store(var S: TStream);
  procedure SplitDate;
  function Valid: Boolean;
  function DaysAfter: Word;

 end;

PCalendar = ^TCalendar;
TCalendar = object(TSortedCollection)
 function KeyOf(Item: Pointer): Pointer; virtual;
 function Compare(Key1, Key2: Pointer): Integer; virtual;
 procedure ViewProductionData(Var F : Text);
 procedure ViewCompostData(Var F : Text);
end;

PCBRec = ^TCBRec;          {Calendar list box}
TCBRec = record
 Items : PCalendar;
 Selected : Integer
end;

PBIRec = ^TBIRec;          {Basic information for new simulation}
TBIRec = record
 StartDate: DateStr;
 Notes: array[0..255] of Char;
 Title: String[20];
end;

PODRec = ^TODRec;          {Observation date}
TODRec = DateStr;
```

```
    PRSRec = ^TRSRec;
    TRSRec = record
     Date : DateStr;
     Observation : String[8];
     FeedList : PStringCollection;
     SelectedFeed : Integer;
     AnimalList : PStringCollection;
     SelectedAnimal : Integer;
    end;


  const
   BIInfo: TBIRec = (StartDate: 'DD/MM/YY';
             Notes: '';
             Title: NoName);

   ODInfo: TODRec = 'DD/MM/YY';

   DaysInMonths: Array[1..12] of Word = (31, 28, 31, 30,
                31, 30, 31, 31,
                30, 31, 30, 31);

  Leaps: set of 0 .. 100 = ([80, 84, 88, 92, 96, 00, 04]);

  RDate: TStreamRec = (
   ObjType: 600;
   VmtLink: Ofs(TypeOf(TDate)^);
   Load: @TDate.Load;
   Store: @TDate.Store
  );

  RCalendar: TStreamRec = (
   ObjType: 601;
   VmtLink: Ofs(TypeOf(TCalendar)^);
   Load: @TCalendar.Load;
   Store: @TCalendar.Store
  );

 procedure RegisterDates;


implementation

constructor TDate.Init(D, P: DateStr);
begin
 inherited Init;
 Date := D;
 Previous := P;
 Feeds := New(PFeeds, Init(5, 1));
 Herd := New(PHerd, Init(5, 1));
 Compost := New(PCompostHeap, Init(5, 1));
 SplitDate;
end;
```

```
constructor TDate.Load(var S: TStream);
begin
  S.Read(Date, SizeOf(DateStr));
  S.Read(DD, SizeOf(Word));
  S.Read(MM, SizeOf(Word));
  S.Read(YY, SizeOf(Word));
  S.Read(Previous, SizeOf(DateStr));
  Feeds := New(PFeeds, Init(5, 1));
  Feeds^.Load(S);
  Herd := New(PHerd, Init(5, 1));
  Herd^.Load(S);
  Compost := New(PCompostHeap, Init(5, 1));
  Compost^.Load(S)
end;

procedure TDate.Store(var S: TStream);
begin
  S.Write(Date, SizeOf(DateStr));
  S.Write(DD, SizeOf(Word));
  S.Write(MM, SizeOf(Word));
  S.Write(YY, SizeOf(Word));
  S.Write(Previous, SizeOf(DateStr));
  Feeds^.Store(S);
  Herd^.Store(S);
  Compost^.Store(S)
end;

procedure TDate.SplitDate;
Var
  TDD : PartStr;
  TMM : PartStr;
  TYY : PartStr;
  Error : Integer;
begin
  if Date[2] <> '/' then
  begin
    TDD := Date[1] + Date[2];
    if Date[5] <> '/' then
    begin
      TMM := Date[4] + Date[5];
      TYY := Date[7] + Date[8]
    end
    else
    begin
      TMM := Date[4];
      TYY := Date[6] + Date[7]
    end
  end
  else
  begin
    TDD := Date[1];
    if Date[4] <> '/' then
    begin
      TMM := Date[3] + Date[4];
      TYY := Date[6] + Date[7]
    end
```

```
      else
      begin
       TMM := Date[3];
       TYY := Date[5] + Date[6]
      end
    end;
    Val(TDD, DD, Error);
    Val(TMM, MM, Error);
    Val(TYY, YY, Error)
  end;

  function TDate.Valid: Boolean;
  Var
   P : PDate;
  begin
   if Previous = '' then Valid := True else
   begin
    P := New(PDate, Init(Previous, ''));
    P^.SplitDate;
    Valid := (YY > P^.YY)
          or ((YY = P^.YY) and (MM > P^.MM))
          or ((YY = P^.YY) and (MM = P^.MM) and (DD > P^.DD));
    P^.Done
   end
  end;


 function TDate.DaysAfter: Word;
 var
  NumYears: Word;
  NumMonths: Word;
  DaysThisYear: Word;
  NumDays: Word;
  M: Word;
  P : PDate;
 begin
  if Previous <> '' then
  begin
   P := New(PDate, Init(Previous, ''));
   P^.SplitDate;
   NumDays := 0;
   if YY <> P^.YY then
   begin
    for M := P^.MM to 12 do
    begin
     NumDays := NumDays + DaysInMonths[M];
     if M = P^.MM then NumDays := NumDays - P^.DD;
     if (M = 2) and (P^.YY in Leaps)
      then NumDays := NumDays + 1
    end;
    for M := 1 to MM do
    begin
     if M = MM then NumDays := NumDays + DD
     else
     begin
      NumDays := NumDays + DaysInMonths[M];
```

```
        if (M = 2) and (YY in Leaps) then NumDays := NumDays + 1;
       end;
      end
     end
     else
     begin
      if MM = P^.MM then NumDays := DD - P^.DD
      else
      begin
       for M := P^.MM to MM do
       begin
        if M = MM
         then NumDays := NumDays + DD
        else
        begin
         NumDays := NumDays + DaysInMonths[M];
         if M = P^.MM then NumDays := NumDays - P^.DD;
         if (M = 2) and (YY in Leaps) then NumDays := NumDays + 1;
        end
       end
      end
     end;
     DaysAfter:=NumDays;
   end
   else DaysAfter := 0;
end;


{*** TCalendar ***}

function TCalendar.KeyOf(Item: Pointer): Pointer;
begin
  KeyOf:= NewStr(PDate(Item)^.Date);
end;

function TCalendar.Compare(Key1, Key2: Pointer): Integer;
Var
  Date1, Date2: PDate;
begin
  Date1 := New(PDate, Init(PString(Key1)^, ''));
  Date2 := New(PDate, Init(PString(Key2)^, ''));
  Date1^.SplitDate;
  Date1^.SplitDate;
  if Date1^.YY <> Date2^.YY then
  begin
    if Date1^.YY < Date2^.YY then Compare := -1 else Compare := 1
  end
  else
  begin
    if Date1^.MM <> Date2^.MM then
    begin
      if Date1^.MM < Date2^.MM then Compare := -1 else Compare := 1;
    end
    else
    begin
```

```
  );

  RFeeds: TStreamRec = (
    ObjType: 501;
    VmtLink: Ofs(TypeOf(TFeeds)^);
    Load: @TFeeds.Load;
    Store: @TFeeds.Store
  );

  DegParmInfo : array[0..8] of DegParmRec =
                ((a: 0.178; b: 0.746; c: 0.033; ADINCoeff: 90),
                 (a: 0.190; b: 0.583; c: 0.042; ADINCoeff: 70),
                 (a: 0.413; b: 0.381; c: 0.091; ADINCoeff: 60),
                 (a: 0.035; b: 0.542; c: 0.034; ADINCoeff: 232),
                 (a: 0.100; b: 0.720; c: 0.080; ADINCoeff: 120),
                 (a: 0.170; b: 0.580; c: 0.022; ADINCoeff: 85),
                 (a: 0.200; b: 0.740; c: 0.040; ADINCoeff: 95),
                 (a: 0.600; b: 0.050; c: 0.015; ADINCoeff: 30),
                 (a: 0.220; b: 0.500; c: 0.025; ADINCoeff: 50));

  procedure RegisterFeeds;


implementation

{*** TAvailableFeed ***}

constructor TAvailableFeed.Init(Na : String; Qu : String);
begin
  inherited Init;
  Name := Na;
  Quantity := Qu
end;

constructor TAvailableFeed.Load(var S: TStream);
begin
  S.Read(Name, SizeOf(String));
  S.Read(Quantity, SizeOf(String))
end;

procedure TAvailableFeed.Store(var S: TStream);
begin
  S.Write(Name, SizeOf(String));
  S.Write(Quantity, SizeOf(String))
end;

{*** TFeeds ***}
constructor TFeeds.Init(Alimit, ADelta : Integer);
begin
  inherited Init(ALimit, ADelta);
  FreshWt := 0;
  DryMatter := 0;
  Nitrogen := 0;
  CrudeFibre := 0;
  EtherExtract := 0;
  MetEnergy := 0;
```

```
    Q := 0;
    FME := 0;
    ADIN := 0;
    UDP := 0;
    ERDP := 0;
    MTP := 0;
    MP := 0;
  end;

  function TFeeds.KeyOf(Item: Pointer): Pointer;
  begin
    KeyOf:= NewStr(PAvailableFeed(Item)^.Name);
  end;

  function TFeeds.Compare(Key1, Key2: Pointer): Integer;
  begin
    if PString(Key1)^ = PString(Key2)^ then Compare:=0
    else if PString(Key1)^ < PString(Key2)^ then Compare:=-1
    else Compare:=1;
  end;

  constructor TFeeds.Load(var S: TStream);
  begin
    inherited Load(S);
    S.Read(FreshWt, SizeOf(Real));
    S.Read(DryMatter, SizeOf(Real));
    S.Read(Nitrogen, SizeOf(Real));
    S.Read(CrudeFibre, SizeOf(Real));
    S.Read(EtherExtract, SizeOf(Real));
    S.Read(MetEnergy, SizeOf(Real));
    S.Read(Q, SizeOf(Real));
    S.Read(FME, SizeOf(Real));
    S.Read(ADIN, SizeOf(Real));
    S.Read(UDP, SizeOf(Real));
    S.Read(ERDP, SizeOf(Real));
    S.Read(MTP, SizeOf(Real));
    S.Read(MP, SizeOf(Real))
  end;

  procedure TFeeds.Store(var S: TStream);
  begin
    inherited Store(S);
    S.Write(FreshWt, SizeOf(Real));
    S.Write(DryMatter, SizeOf(Real));
    S.Write(Nitrogen, SizeOf(Real));
    S.Write(CrudeFibre, SizeOf(Real));
    S.Write(EtherExtract, SizeOf(Real));
    S.Write(MetEnergy, SizeOf(Real));
    S.Write(Q, SizeOf(Real));
    S.Write(FME, SizeOf(Real));
    S.Write(ADIN, SizeOf(Real));
    S.Write(UDP, SizeOf(Real));
    S.Write(ERDP, SizeOf(Real));
    S.Write(MTP, SizeOf(Real));
    S.Write(MP, SizeOf(Real))
  end;
```

```
procedure TFeeds.MakeViewerList(var L: PStringCollection);

  procedure OneFeed(Item: PAvailableFeed); far;
  begin
    with Item^ do
      L^.Insert(NewStr(Name + ' [' + Quantity + 'kg]'));
  end;

begin
  L^.FreeAll;
  ForEach(@OneFeed);
end;

procedure TFeeds.CalcNutrients(r : Real; var DB: PDatabase);
Var
  F                    : TBufStream;
  GE                   : Real;

  procedure OneFeed(Item: PAvailableFeed); far;
  Var
    Ind, Error         : Integer;
    Amount             : Real;
    N                  : Real;
    CurrentFeed        : PFeedData;
    QDP, SDP           : Real;

begin
  with Item^ do
  begin
    Val(Quantity, Amount, Error);
    FreshWt := FreshWt + Amount;
   if DB^.Search(NewStr(Name), Ind)
   then
   begin
     CurrentFeed := PFeedData(DB^.At(Ind));
     with CurrentFeed^.Composition do
     begin
       Val(DM, N, Error);
       Amount := N * Amount / 1000;
       DryMatter := DryMatter + Amount;
       Val(CP, N, Error);
       N := N / 6.25;
       Nitrogen := Nitrogen + N * Amount;
       ADIN := ADIN + DegParmInfo[FeedType].ADINCoeff* N * Amount / 1000;
       QDP := N * 6.25 * DegParmInfo[FeedType].a * Amount;
       SDP := N * 6.25 * DegParmInfo[FeedType].b *
              DegParmInfo[FeedType].c * Amount /
              (DegParmInfo[FeedType].c + r);
       UDP := UDP + N * 6.25 * Amount - (QDP + SDP);
       ERDP := ERDP + 0.8 * QDP + SDP;
       Val(CF, N, Error);
       CrudeFibre := CrudeFibre + N * Amount;
       Val(EE, N, Error);
       EtherExtract := EtherExtract + N * Amount;
       Val(ME, N, Error);
```

```
        MetEnergy := MetEnergy + N * Amount
       end
      end
     end
    end;

  begin
   FreshWt := 0;
   DryMatter := 0;
   Nitrogen := 0;
   CrudeFibre := 0;
   EtherExtract := 0;
   MetEnergy := 0;
   ADIN := 0;
   UDP := 0;
   ERDP := 0;
   ForEach(@OneFeed);
   if EtherExtract > 50 then GE := DryMatter * (EtherExtract * 36 /1000 +
                              (1000 - EtherExtract) * 18 / 1000)
   else GE := DryMatter * 18;
   Q := MetEnergy / GE;
   FME := MetEnergy - 0.035 * EtherExtract / 1000;
  end;

  procedure TFeeds.CalculateMP(Ymcpfme: Real);
  var MCP, DMTP : Real;
  begin
   if ERDP < Ymcpfme * FME then MCP := ERDP
   else MCP := Ymcpfme * FME;

    {Ymcpfme represents efficiency of FME utilisation
     for microbial protein sysnthesis; depends on animal type}

   MTP := MCP * 0.75;   { AFRC-TCORN, 1992 }
   DMTP := MTP * 0.85;
   MP := DMTP + 0.9 * (UDP - 6.25 * ADIN);
  end;



  {*** Stream Registration ***}

  procedure RegisterFeeds;
  begin
   RegisterType(RAvailableFeed);
   RegisterType(RFeeds);
  end;

  END.
```

```
Program FRAME;

uses Dos,
    Objects, App, Views, Drivers, Menus, Dialogs, Editors, Validate,
    StdDlg, MsgBox,
    Globals, Dates,
    Tools,
    DataBase, Simulat, Feeds, Animals,
    HelpFile, HelpCont;

{$I commands.inc}
{$I strings.inc}

type

 FM = object(TApplication)

   FeedBase                  : PDataBase;
   Sim                       : PSimulation;
   DBChanged                 : Boolean;
   SimChanged                : Boolean;

   constructor Init;
   destructor Done; virtual;
   procedure InitScreen; virtual;
   function GetPalette: PPalette; virtual;
   procedure GetEvent(var Event: TEvent); virtual;
   procedure InitDeskTop; virtual;
   procedure WriteShellMsg; virtual;
   procedure HandleEvent(var Event: TEvent); virtual;
   procedure Idle; virtual;
   procedure InitMenuBar; virtual;
   procedure InitStatusLine; virtual;
   procedure ExitFM;

   {--- Database ---}
   procedure AddFeed;
   procedure ViewEditFeed;
   procedure DeleteFeed;
   procedure LoadDatabase(WildCard: PathStr);
   procedure StoreDatabase(WildCard: PathStr);
   procedure ClearDatabase;

   {--- File ---}
   procedure OpenFile(WildCard: PathStr);
   procedure SaveFile(WildCard: PathStr);
   function OverWriteOK(Name: PathStr): Boolean;

   {--- Simulation ---}
   procedure BasicInformation(Edit: Boolean);
   procedure RunSimulation;
   procedure ViewProduction;
   procedure ViewCompost;
   function ClearSimulation: Boolean;
```

```
    {--- Feeds ---}
    procedure Availability(Edit: Boolean);


    {--- Herd ---}
    function CreateAnimal : PAnimal;
    procedure Herd(Operation : Word);
    procedure DeleteAnimal;

    {--- Setup ---}
    procedure ChangeScrColour;

    {--- Help ---}
    procedure DoAboutBox;

    procedure OpenWindow(WTitle: string; Bounds: TRect);
  end;

var ResFile                    : TResourceFile;

    {*** Dialog Box Data Globals ***}
    SCData                     : PSCRec;
    FCData                     : PFCRec;
    SBData                     : PSBRec;
    SWData                     : PSWRec;
    BIData                     : PBIRec;
    ODData                     : PODRec;
    CBData                     : PCBRec;
    AVData                     : PAVRec;
    ANData                     : PANRec;
    MFData                     : PMFRec;
    RSData                     : PRSRec;

    {*** Other Globals ***}
    Result                     : Integer;
    Strings                    : PStringList;

    CurrentDate                : PDate;
    NextDate                   : PDate;


function CalcHelpName: PathStr;
var
  EXEName                      : PathStr;
  Dir                          : DirStr;
  Name                         : NameStr;
  Ext                          : ExtStr;
begin
  if Lo(DosVersion) >= 3 then EXEName := ParamStr(0)
  else EXEName := FSearch('FRAME.EXE', GetEnv('PATH'));
  FSplit(EXEName, Dir, Name, Ext);
  if Dir[Length(Dir)] = '\' then Dec(Dir[0]);
  CalcHelpName := FSearch('FRAME.HLP', Dir);
end;
```

```
{*** Resource MessageBox Wrappers ***}

function RMsgBox(StrNum: Word; Param: Pointer; Flags: Word): Word;
begin
  RMsgBox := MessageBox(Strings^.Get(StrNum), Param, Flags);
end;

function RMsgBoxRect(var Rect: TRect; StrNum: Word; Param: Pointer;
  Flags: Word): Word;
begin
  RMsgBoxRect := MessageBoxRect(Rect, Strings^.Get(StrNum), Param,
    Flags);
end;

{*** FM ***}

constructor FM.Init;
begin

  {Stream registrations}
  RegisterMenus;
  RegisterObjects;
  RegisterDialogs;
  RegisterViews;
  RegisterApp;
  RegisterValidate;
  RegisterEditors;
  RegisterStdDlg;

  RegisterType(RStringList);

  RegisterTools;
  RegisterDates;
  RegisterDataBase;
  RegisterSimulation;
  RegisterFeeds;
  RegisterAnimals;

  {Open main resource file}
  ResFile.Init(New(PBufStream, Init('FRAME.TVR', stOpenRead, 3074)));
  Strings := PStringList(ResFile.Get('Strings'));



  {Default screen colour scheme}
  New(SCData);
  SCData^:=0;

  Inherited Init;

  RegisterHelpfile;

  {Create pointer variables for dialog box data}

  New(FCData);
  New(SBData);
```

```
  New(SWData);
  New(BIData);
  New(ODData);
  New(CBData);
  New(AVData);
  New(ANData);
  New(MFData);
  New(RSData);

  {Create operating variables}

  FeedBase := New(PDataBase, init(50,10));
  DBChanged := False;

  Sim := New(PSimulation, Init);
  SimChanged := False;

end;

destructor FM.Done;
begin

  {Close resource file}
  ResFile.Done;

  {Dispose of pointer variables for dialog box data}
  Dispose(SCData);
  Dispose(FCData);
  Dispose(SBData);
  Dispose(SWData);
  Dispose(BIData);
  Dispose(ODData);
  Dispose(CBData);
  Dispose(AVData);
  Dispose(ANData);
  Dispose(MFData);
  Dispose(RSData);

  FeedBase^.Done;
  Sim^.Done;

  Inherited Done
end;

procedure FM.InitScreen;
begin
  Inherited InitScreen;
  Case SCData^ of
    0: AppPalette:=apColor;
    1: AppPalette:=apBlackWhite;
    2: AppPalette:=apMonoChrome
  end;
end;

function FM.GetPalette: PPalette;
const
```

```
  CNewColor = CColor + CHelpColor;
  CNewBlackWhite = CBlackWhite + CHelpBlackWhite;
  CNewMonochrome = CMonochrome + CHelpMonochrome;
  P: array[apColor..apMonochrome] of string[Length(CNewColor)] =
    (CNewColor, CNewBlackWhite, CNewMonochrome);
begin
  GetPalette := @P[AppPalette];
end;

procedure FM.GetEvent(var Event: TEvent);
var
  W: PWindow;
  HFile: PHelpFile;
  HelpStrm: PDosStream;
const
  HelpInUse: Boolean = False;
begin
  inherited GetEvent(Event);
  case Event.What of
    evCommand:
      case Event.Command of
        cmHelp: if not HelpInUse then
        begin
          HelpInUse := True;
          HelpStrm := New(PDosStream, Init(CalcHelpName, stOpenRead));
          HFile := New(PHelpFile, Init(HelpStrm));
          if HelpStrm^.Status <> stOk then
          begin
            RMsgBox(smbHelpOpenFailed, nil, mfError+mfOKButton);
            Dispose(HFile, Done);
          end
          else
          begin
            W := New(PHelpWindow,Init(HFile, GetHelpCtx));
            if ValidView(W) <> nil then
            begin
              ExecView(W);
              Dispose(W, Done);
            end;
            ClearEvent(Event);
          end;
          HelpInUse := False;
        end;


      end;
    evMouseDown:
      if Event.Buttons <> 1 then Event.What := evNothing;
  end;
end;

procedure FM.InitDeskTop;
begin
  inherited InitDeskTop;
  DeskTop^.BackGround^.Pattern:= ':';
end;
```

```
procedure FM.WriteShellMsg;
begin
  PrintStr('Leaving FRAME for DOS. Type EXIT to return.');
end;

procedure FM.HandleEvent(var Event: TEvent);

  procedure ChangeDir;
  var
    D: PChDirDialog;
  begin
    ExecuteDialog(PDialog(ResFile.Get('DIALOG_ChangeDir')), nil);
  end;

  procedure TileWindows;
  var R: TRect;
  begin
    GetExtent(R);
    R.Grow(0, -1);
    R.Move(0, -1);
    DeskTop^.Tile(R)
  end;

  procedure CascadeWindows;
  var R: TRect;
  begin
    GetExtent(R);
    R.Grow(0, -1);
    R.Move(0, -1);
    DeskTop^.Cascade(R)
  end;

begin
  Inherited HandleEvent(Event);
  if Event.What = evCommand then
  begin
    case Event.Command of

      {--- Debug ---}
      cmTestNew : RMsgBox(smbNoDebugTest, nil, mfError+mfOKButton);

      {--- Database ---}
      cmAddFeed : AddFeed;
      cmViewEditFeed : ViewEditFeed;
      cmDeleteFeed : DeleteFeed;
      cmLoadDatabase : LoadDataBase('*.FDB');
      cmStoreDatabase : StoreDataBase('*.FDB');
      cmStoreAs : StoreDataBase(FeedBase^.FileName);
      cmClearDatabase : ClearDataBase;

      {--- File ---}
      cmOpen : OpenFile('*.SIM');
      cmSave : SaveFile('*.SIM');
      cmSaveAs : SaveFile(Sim^.FileName);
      cmChangeDir : ChangeDir;
```

```
    cmDosShell : DosShell;
    cmExitFM : ExitFM;

    {--- Simulation ---}
    cmNewBasicInfo : BasicInformation(False);
    cmEditBasicInfo : BasicInformation(True);
    cmRunSimulation : RunSimulation;
    cmViewProduction : ViewProduction;
    cmViewCompost : ViewCompost;
    cmClearSimulation : ClearSimulation;


    {--- Feeds ---}
    cmNewAvailability : Availability(False);
    cmEditAvailability : Availability(True);

    {--- Herd ---}
    cmNewHerd : Herd(hdNew);
    cmAddToHerd : Herd(hdAdd);
    cmEditAnimal : Herd(hdEdit);
    cmDeleteAnimal : DeleteAnimal;

    {--- Setup ---}
    cmChangeScrColour : ChangeScrColour;

    {--- Help ---}
    cmAbout : DoAboutBox;

   else
    begin
     MessageBox('Command with no procedure attached',
           nil, mfWarning + mfOkButton);
     Exit;
    end;
   end;
   ClearEvent(Event)
  end
end;


procedure FM.Idle;
const
  DBInMem: TCommandSet = [cmViewEditFeed, cmClearDatabase,
                  cmStoreDatabase, cmStoreAs,
                  cmDeleteFeed];
  BISet: TCommandSet = [cmSave, cmSaveAs, cmClearSimulation,
                cmNewAvailability, cmNewHerd,
                cmEditBasicInfo, cmEditAvailability,
                cmAddToHerd, cmEditAnimal];

  function IsTileable(P: PView): Boolean; far;
  begin
    IsTileable:= P^.Options and ofTileable<>0
  end;

begin
```

```
  Inherited Idle;

  if FeedBase^.Count=0 then DisableCommands(DBInMem)
  else EnableCommands(DBInMem);

  if Sim^.BasicInfoSet then EnableCommands(BISet)
  else DisableCommands(BISet);

  if Sim^.FeedsSet then EnableCommands([cmEditAvailability])
  else DisableCommands([cmEditAvailability]);

  if Sim^.HerdSet then EnableCommands([cmAddToHerd, cmEditAnimal, cmDeleteAnimal])
  else DisableCommands([cmAddToHerd, cmEditAnimal, cmDeleteAnimal]);

  if Sim^.AllSet then EnableCommands([cmRunSimulation])
  else DisableCommands([cmRunSimulation]);

  if DeskTop^.FirstThat(@IsTileable) <> nil then
    EnableCommands([cmTile, cmCascade])
  else
    DisableCommands([cmTile, cmCascade])
end;


procedure FM.InitMenuBar;
var R: TRect;
   M: TRect;
begin
  MenuBar := PMenuBar(ResFile.Get('MENU_Main'));
end;

procedure FM.InitStatusLine;
var
  R: TRect;
begin
  StatusLine := PStatusLine(ResFile.Get('STATUS_Line'));
  GetExtent(R);
  StatusLine^.MoveTo(0, R.B.Y - 1);
end;

procedure FM.ExitFM;
begin
  if DBChanged or SimChanged then
  begin
    Result:=RMsgBox(smbNotSaved, nil, mfWarning + mfYesNoCancel);
    if Result = 12 then
    begin
      if FeedBase^.Count <> 0 then StoreDatabase('*.FDB');
      if SimChanged then SaveFile('*.SIM');
    end
  end;
  if Result <> 11 then EndModal(cmQuit)
end;

procedure FM.AddFeed;
var
```

```
  D : PDialog;
  C : Word;
  I : Integer;
label 1;
begin
  FCData^ := FCInfo;
1:D := PDialog(ResFile.Get('DIALOG_FeedComp'));
  D^.Title := NewStr('Add Feed');
  C := ExecuteDialog(D, FCData);
  if C<>cmCancel then with FeedBase^ do
  begin
    if not Search(NewStr(FCData^.Name), I) then
      Insert(New(PFeedData, Init(FCData)))
    else
    begin
      Result := RMsgBox(smbFeedNameUsed, nil, mfWarning + mfYesNoCancel);
      Case Result of
        11 : Exit;
        12 : AtPut(I, New(PFeedData, Init(FCData)));
        13 : Goto 1
      end
    end;
    DBChanged := true
  end
end;

procedure FM.ViewEditFeed;
var
  D : PDialog;
  C : Word;
begin
  with SBData^ do
  begin
    Items := New(PStringCollection, Init(50,10));
    FeedBase^.MakeViewerList(Items);
    Selected := 0
  end;
  D := PDialog(ResFile.Get('DIALOG_StringBox'));
  D^.Title := NewStr('View / Edit Feed');
  C := ExecuteDialog(D, SBData);
  if C <> cmCancel then
  begin
    FCData^ := PFeedData(FeedBase^.At(SBData^.Selected))^.Composition;
    D := PDialog(ResFile.Get('DIALOG_FeedComp'));
    D^.Title := NewStr('View / Edit Feed');
    C := ExecuteDialog(D, FCData);
    if C <> cmCancel then
    begin
      FeedBase^.AtPut(SBData^.Selected, New(PFeedData, Init(FCData)));
      DBChanged := true
    end
  end;
end;

procedure FM.DeleteFeed;
var
```

```
  D : PDialog;
  C : Word;
begin
  with SBData^ do
  begin
    Items := New(PStringCollection, Init(50,10));
    FeedBase^.MakeViewerList(Items);
    Selected := 0
  end;
  D := PDialog(ResFile.Get('DIALOG_StringBox'));
  D^.Title := NewStr('Delete Feed');
  C := ExecuteDialog(D, SBData);
  if C <> cmCancel then
  begin
    Result := RMsgBox(smbRemoveFeed, nil, mfConfirmation+mfYesButton+mfNoButton);
    if Result = 12 then
    begin
      FeedBase^.AtDelete(SBData^.Selected);
      DBChanged := true
    end
  end
end;

procedure FM.LoadDatabase(WildCard: PathStr);
var
  D : PDialog;
  FN : FNameStr;
  F : TBufStream;
begin
  with FeedBase^ do
    if Count > 0 then
    begin
      Result := RMsgBox(smbDBInMem, nil, mfWarning + mfOkCancel);
      if Result = 10 then Reset else Exit
    end;
  if WildCard <> '*.FDB' then FN := WildCard
  else
  begin
    FN := '*.FDB';
    D := PDialog(ResFile.Get('DIALOG_Open'));
    D^.Title := NewStr('Load Database');
    D^.HelpCtx := hcLoadDatabaseDlg;
    if ExecuteDialog(D, @FN) = cmCancel then
    begin
      RMsgBox(smbOpenDBError, nil, mfError + mfOkButton);
      Exit
    end
  end;
  begin
    F.Init(FN, stOpen, 1024);
    if F.Status = StOK then
    begin
      FeedBase := PDataBase(F.Get);
      F.Done
    end
    else
```

```
  end
end;

procedure FM.StoreDatabase(WildCard: PathStr);
var
  D : PFileDialog;
  F : TBufStream;
  FN : FNameStr;
  Dn : DirStr;
  Nn : NameStr;
  En : ExtStr;
begin
  if (WildCard <> '*.FDB') or (FeedBase^.FileName = '*.FDB') then
  begin
    FSplit(WildCard, Dn, Nn, En);
    FN := Nn + En;
    D := PFileDialog(ResFile.Get('DIALOG_Save'));
    D^.WildCard := '*.FDB';
    D^.Title := NewStr('Store Database');
    D^.HelpCtx := hcStoreDatabaseDlg;
    if ExecuteDialog(D, @FN) = cmCancel then Exit
    else if OverWriteOK(FN) then FeedBase^.FileName := FN else Exit
  end;
  F.Init(FeedBase^.FileName, stCreate, 1024);
  F.Put(FeedBase);
  F.Done;
  DBChanged := False
end;

procedure FM.ClearDatabase;
begin
  Result := RMsgBox(smbClearDB, nil, mfConfirmation+mfYesButton+mfNoButton);
  if Result = 12 then
  begin
    if DBChanged then
    begin
      Result := RMsgBox(smbNotSaved, nil, mfWarning + mfYesNoCancel);
      Case Result of
        11 : Exit;
        12 : StoreDatabase(FeedBase^.FileName)
      end
    end;
    FeedBase^.Reset;
    DBChanged := False
  end
end;

procedure FM.OpenFile(WildCard: PathStr);
var
  D : PDialog;
  FN : FNameStr;
  F : TBufStream;
begin
  if Sim^.BasicInfoSet then
  begin
    Result := RMsgBox(smbSimInMem, nil, mfWarning + mfOkCancel);
```

```
    if Result <> 10 then exit
  end;
  FN := '*.SIM';
  D := PDialog(ResFile.Get('DIALOG_Open'));
  D^.Title := NewStr('Open File');
  D^.HelpCtx := hcOpenDlg;
  if ExecuteDialog(D, @FN) <> cmCancel then
  begin
    F.Init(FN, stOpen, 1024);
    if F.Status = StOK then
    begin
      Sim := PSimulation(F.Get);
      F.Done;
    end
    else RMsgBox(smbOpenFileError, nil, mfError + mfOkButton)
  end
end;

procedure FM.SaveFile(WildCard: PathStr);
var
  D : PFileDialog;
  F : TBufStream;
  FN : FNameStr;
  Dn : DirStr;
  Nn : NameStr;
  En : ExtStr;
begin
  if (WildCard <> '*.SIM') or (Sim^.FileName = '*.SIM') then
  begin
    FSplit(WildCard, Dn, Nn, En);
    FN := Nn + En;
    D := PFileDialog(ResFile.Get('DIALOG_Save'));
    D^.WildCard := '*.SIM';
    D^.Title := NewStr('Save File');
    D^.HelpCtx := hcSaveDlg;
    if ExecuteDialog(D, @FN) = cmCancel then Exit
    else if OverWriteOK(FN) then Sim^.FileName := FN else Exit;
  end;
  F.Init(Sim^.FileName, stCreate, 1024);
  F.Put(Sim);
  F.Done;
  SimChanged := False
end;

function FM.OverWriteOK(Name: PathStr): Boolean;
var
  SR : SearchRec;
begin
  FindFirst(Name, 0, SR);
  if DosError = 0 then
  begin
    Result := RMsgBox(smbFileOverWrite, nil, mfWarning+mfYesButton+mfNoButton);
    OverWriteOK := (Result<>13)
  end
  else OverWriteOK := true
end;
```

```
procedure FM.BasicInformation(Edit: Boolean);
Var
 R                    : TRect;
 C                    : Word;
 D                    : PDialog;
 NewDate, OldDate          : PDate;
 ODSet, DBSet              : Boolean;
 FN                   : FNameStr;

 procedure NewCalendar;
 begin
 if not ODSet then
  begin
   with BIData^ do
   begin
    ODData^ := StartDate;
    NewDate := New(PDate, Init(StartDate, ''))
   end;
   with CBData^ do
   begin
    Items := New(PCalendar, Init(10, 2));
    Items^.Insert(NewDate)
   end;
   Repeat
    OldDate := NewDate;
    D := PDialog(ResFile.Get('DIALOG_Calendar'));
    D^.Title := NewStr('New Calendar');
    D^.HelpCtx := hcNewCalendarDlg;
    C := ExecuteDialog(D, ODData);
    NewDate := New(PDate, Init(ODData^, OldDate^.Date));
    if (C = cmOK) or (C = cmCalendarComplete) then
    begin
     if NewDate^.Valid then
     begin
      CBData^.Items^.Insert(NewDate);
      ODSet := True
     end
     else
     if C <> cmCalendarComplete then
     begin
      RMsgBox(smbInvalidDate, nil, mfError+mfOKButton);
      ODData^ := OldDate^.Date
     end
    end
    else C := cmCalendarComplete;
   Until C = cmCalendarComplete
  end
  else RMsgBox(smbDatesAlreadySet, nil, mfError+mfOKButton);
 end;

 procedure EditCalendar;
 begin
  Repeat
   with CBData^ do
   begin
```

```
      Items := Sim^.Calendar;
      Selected := 0
    end;
    D := PDialog(ResFile.Get('DIALOG_CalendarBox'));
    D^.Title := NewStr('Edit Calendar');
    C := ExecuteDialog(D, CBData);
    if C <> cmCancel then
    begin
      OldDate := PDate(CBData^.Items^.At(CBData^.Selected));
      ODData^ := OldDate^.Date;
      D := PDialog(ResFile.Get('DIALOG_Calendar'));
      D^.Title := NewStr('Edit Calendar');
      D^.HelpCtx := hcEditCalendarDlg;
      C := ExecuteDialog(D, ODData);
      if C <> cmCancel then
      begin
        NewDate := New(PDate, Init(ODData^, OldDate^.Previous));
        if Newdate^.Valid then
        begin
          with CBData^ do Items^.AtPut(Selected, NewDate);
          with BIData^ do
            if NewDate^.Previous = '' then StartDate := NewDate^.Date;
          SimChanged := true
        end
        else
        begin
          RMsgBox(smbInvalidDate, nil, mfError+mfOKButton);
          ODData^ := OldDate^.Date
        end
      end
    end;
  Until (C = cmCalendarComplete) or (C = cmCancel)
end;

begin
  if Edit then
  begin
    ODSet := True;
    DBSet := True
  end
  else
  begin
    if Sim^.BasicInfoSet then if not ClearSimulation then Exit;
    ODSet := False;
    DBSet := False
  end;
  BIData^ := Sim^.BasicInfo;
  Repeat
    D := PDialog(ResFile.Get('DIALOG_BasicInformation'));
    if Edit then D^.Title := NewStr('Edit Simulation')
    else D^.Title := NewStr('New Simulation');
    C := ExecuteDialog(D, BIData);
    Case C of
      cmMakeCalendar : if Edit then EditCalendar else NewCalendar;
      cmSetDatabase :
      begin
```

```
        D := PDialog(ResFile.Get('DIALOG_Open'));
        D^.Title := NewStr('Set Database');
        D^.HelpCtx := hcSetDatabaseDlg;
        DBSet := ExecuteDialog(D, @Sim^.DBFilename) <> cmCancel
      end;
      cmCancel : Exit;
      cmOK:
      begin
       if not Edit then
        if (ODSet = False) or (DBSet = False) then
        begin
         R.Assign(13, 6, 68, 16);
         RMsgBoxRect(R, smbBasicDataNotSet, nil, mfError+mfOKButton);
         C := cmCancel
        end
     end
    end;
  Until C = cmOK;
  Sim^.SetBasicInfo(BIData, CBData^.Items);
  SimChanged := True;
end;

procedure FM.Availability(Edit: Boolean);
var
  D                          : PDialog;
  C, AC                      : Word;
  F                          : TBufStream;
  DB                         : PDataBase;
  DBList                     : PStringCollection;
  DateList                   : PStringCollection;

 procedure OneDate(Item: PDate); far;
 Var
   Ind                       : Integer;

   procedure FlushDB(Item: PString); far;
   begin
    if DBList^.Search(Item, Ind) then DBList^.AtDelete(Ind);
   end;

begin
  if C <> cmCancel then
  begin
    DB^.MakeViewerList(DBList);
    if Edit then
    begin
      Item^.Feeds^.MakeViewerList(DateList);
      DateList^.ForEach(@FlushDB)
    end;
    with SWData^ do
    begin
      List1:=DateList;
      Selected1 := 0;
      List2 := DBList;
      Selected2 :=0
    end;
```

```
    Repeat
      D := PDialog(ResFile.Get('DIALOG_SwapBox'));
      D^.Title := NewStr('Feed Availability (' + Item^.Date + ')');
      D^.HelpCtx := hcAvailabilityDlg;
      C := ExecuteDialog(D, SWData);
      Case C of
       cmRight : if DBList^.Count <> 0 then
       begin
         AVData^ := '';
         D := PDialog(ResFile.Get('DIALOG_Availability'));
         D^.Title := NewStr(PString(DBList^.At(SWData^.Selected2))^ + ' (' + Item^.Date + ')');
         D^.HelpCtx := hcQuantityDlg;
         AC:= ExecuteDialog(D , AVData);
         if AC <> cmCancel then
         begin
          DateList^.Insert(NewStr(
           PString(DBList^.At(SWData^.Selected2))^ +
           ' [' + AVData^ + 'kg]'));
          Item^.Feeds^.Insert(New(PAvailableFeed,
Init(PString(DBList^.At(SWData^.Selected2))^ ,AVData^)));
          DBList^.AtDelete(SWData^.Selected2)
         end
       end;
       cmLeft : if DateList^.Count <> 0 then
       begin
         DBList^.Insert(PString(DateList^.At(SWData^.Selected1)));
         if Item^.Feeds^.Search(PString(DateList^.At(SWData^.Selected1)), Ind)
         then Item^.Feeds^.AtDelete(Ind);
         DateList^.AtDelete(SwData^.Selected1)
        end;
       cmCancel : Exit;
      end
    Until C = cmOK;
    DBList^.FreeAll;
    DateList^.FreeAll
  end
 end;

begin
 if (Sim^.FeedsSet and not Edit) then
 begin
  Result:=RMsgBox(smbFeedsInMem, nil, mfWarning+mfYesButton+mfNoButton);
  if Result = 13 then Exit
 end;
 F.Init(Sim^.DBFileName, stOpen, 1024);
 if F.Status=StOK then
 begin
  DB:=PDataBase(F.Get);
  F.Done;
 end
 else RMsgBox(smbOpenDBError, nil, mfWarning + mfOkButton);
 DBList := New(PStringCollection, Init(50,10));
 DateList := New(PStringCollection, Init(50,10));
 C := cmOK;
 Sim^.Calendar^.ForEach(@OneDate);
 Sim^.FeedsSet := True;
```

```
  DBList^.Done;
  DateList^.Done
end;

function FM.CreateAnimal : PAnimal;
var
  D: PDialog;
  C : Word;
  Title : PString;
begin
  with ANData^ do
    Title := NewStr(ClassNames[Class] + SexNames[Sex] + SpeciesNames[Species]);
  with ANData^ do case Species of
    0,1,2 : case Class of
            0 : CreateAnimal := (new(PGrowingBovine, init(ANData)));
            1 : begin
                  if Sex = 1 then
                  begin
                    D:=PDialog(ResFile.Get('DIALOG_MatureFemale'));
                    D^.Title := Title;
                    C:=ExecuteDialog(D, MFData);
                    if C<>cmCancel then
                      CreateAnimal := (new(PFemaleBovine, init(ANData, MFData)))
                    else CreateAnimal := nil
                  end
                  else CreateAnimal := (new(PMaleBovine, init(ANData)));
                end
          end;
    3 : case Class of
          0 : CreateAnimal := (new(PGrowingCaprine, init(ANData)));
          1 : if Sex = 1 then
              begin
                D:=PDialog(ResFile.Get('DIALOG_MatureFemale'));
                D^.Title := Title;
                C:=ExecuteDialog(D, MFData);
                if C<>cmCancel then
                  CreateAnimal := (new(PFemaleCaprine, init(ANData, MFData)))
                else CreateAnimal := nil
              end
              else CreateAnimal := (new(PMaleCaprine, init(ANData)))
    end;
    4 : case Class of
          0 : CreateAnimal := (new(PGrowingOvine, init(ANData)));
          1 : if Sex = 1 then
              begin
                D:=PDialog(ResFile.Get('DIALOG_MatureFemale'));
                D^.Title := Title;
                C:=ExecuteDialog(D, MFData);
                if C<>cmCancel then
                  CreateAnimal := (new(PFemaleOvine, init(ANData, MFData)))
                else CreateAnimal := nil
              end
              else CreateAnimal := (new(PMaleOvine, init(ANData)))
        end
    end
end;
```

```
procedure FM.Herd(Operation : Word);
var
  D: PDialog;
  C : Word;
  ThisDate : PDate;
  Hd : PHerd;
  An : PAnimal;
  NewAnimal : PAnimal;
  HerdList : PStringCollection;
  Ind : Integer;
Label 1;
begin
  with Sim^ do
  begin
    Hd := PDate(Calendar^.At(0))^.Herd;
    if (HerdSet and not ((Operation = hdAdd) or (Operation = hdEdit))) then
    begin
      Result := RMsgBox(smbHerdInMem, nil, mfWarning + mfYesButton + mfNoButton);
      if Result = 12 then Hd^.FreeAll
      else Exit
    end;
  end;
  1: if (Operation = hdEdit) then
  begin
    with SBData^ do
    begin
      Items := New(PStringCollection, Init(5, 1));
      Hd^.MakeViewerList(Items);
      Selected := 0
    end;
    D:=PDialog(ResFile.Get('DIALOG_StringBox'));
    D^.Title := NewStr('Edit Herd');
    C:=ExecuteDialog(D, SBData);
    if C<>cmCancel then
    begin
      if Hd^.Search(PString(SBData^.Items^.At(SBData^.Selected)), Ind)
      then
      begin
        ANData^ := PAnimal(Hd^.At(Ind))^.Description;
        with ANData^ do
        begin
          if (Class = 1) and (Sex = 1) then
          Case Species of
            0, 1, 2 : MFData^ := PFemaleBovine(Hd^.At(Ind))^.ReproStat;
            3 : MFData^ := PFemaleCaprine(Hd^.At(Ind))^.ReproStat;
            4 : MFData^ := PFemaleOvine(Hd^.At(Ind))^.ReproStat
          end
        end
      end
    end
    else Exit
  end
  else
  begin
    ANData^ := ANInfo;
```

```
    MFData^ := MFInfo
  end;
  D := PDialog(ResFile.Get('DIALOG_Animal'));
  D^.Title := NewStr('Animal Description');
  D^.HelpCtx := hcAnimalDlg;
  C := ExecuteDialog(D, ANData);
  if C <> cmCancel then
  begin
    NewAnimal := CreateAnimal;
    if NewAnimal <> nil then
    begin
      if Operation = hdEdit then Hd^.AtPut(Ind, NewAnimal)
      else Hd^.Insert(NewAnimal);
      Sim^.HerdSet := True;
      SimChanged := True
    end
  end;
  if Operation = hdNew then
    if RMsgBox(smbNextAnimal, nil, mfConfirmation + mfYesButton + mfNoButton) = 12
    then Goto 1;
end;

procedure FM.DeleteAnimal;
var
  D : PDialog;
  C : Word;
  Hd : PHerd;
begin
  Hd := PDate(Sim^.Calendar^.At(0))^.Herd;
  with SBData^ do
  begin
    Items := New(PStringCollection, Init(50,10));
    Hd^.MakeViewerList(Items);
    Selected := 0
  end;
  D := PDialog(ResFile.Get('DIALOG_StringBox'));
  D^.Title := NewStr('Delete Animal');
  C := ExecuteDialog(D, SBData);
  if C <> cmCancel then
  begin
    Result := RMsgBox(smbRemoveAnimal, nil, mfConfirmation+mfYesButton+mfNoButton);
    if Result = 12 then
    begin
      Hd^.AtDelete(SBData^.Selected);
      SimChanged := true
    end
  end
end;


procedure FM.RunSimulation;
Var
  D                          : PDialog;
  C, C2                      : Word;
  CD                         : Integer;
  HerdBkp, FeedsBkp                : TEmsStream;
```

```
An                          : PAnimal;
Fd                          : PAvailableFeed;
FdName                        : String[20];
Ind, Error                   : Integer;
Allocation, Total              : Real;
R                           : TRect;

procedure OneAnimal(CurrentAnimal: PRuminant); far;
begin
 with CurrentAnimal^ do
 begin
  Meal^.CalcNutrients(r, FeedBase);
  with NextDate^ do
   if NextDate <> nil then
   begin
    Herd^.Insert(UpDate(DaysAfter));
   end;
  Excrete(CurrentDate^.Compost)
 end
end;

procedure LeftOvers(CurrentFeed: PAvailableFeed); far;
begin
  CurrentDate^.Compost^.Insert(CurrentFeed)
end;

procedure AddAnimal;
Var
 D                          : PDialog;
 C                          : Word;
begin
 ANData^ := ANInfo;
 D := PDialog(ResFile.Get('DIALOG_Animal'));
 D^.Title := NewStr('Animal Description');
 D^.HelpCtx := hcAnimalDlg;
 C := ExecuteDialog(D, ANData);
 if C <> cmCancel then CurrentDate^.Herd^.Insert(CreateAnimal)
end;

procedure CullAnimal;
begin
 Result := RMsgBox(smbCullAnimal, nil, mfConfirmation+mfYesButton+mfNoButton);
 if Result = 12 then
 with RSData^ do
 begin
  CurrentDate^.Herd^.AtDelete(SelectedAnimal);
  if SelectedAnimal <> 0 then SelectedAnimal := SelectedAnimal - 1
 end
end;

procedure ViewAnimals;
begin
 with RSData^ do
  PAnimal(CurrentDate^.Herd^.At(SelectedAnimal))^.PopUpView;
end;
```

```
procedure AllocateFeeds;
Var
  D                          : PDialog;
  C                          : Word;
begin
  if CurrentDate^.Feeds^.Count <> 0 then
  begin
    Repeat
      with CurrentDate^.Feeds^ do
        AVData^ := PAvailableFeed(At(RSData^.SelectedFeed))^.Quantity;
      D := PDialog(ResFile.Get('DIALOG_Availability'));
      with RSData^ do
      begin
        An := CurrentDate^.Herd^.At(SelectedAnimal);
        with CurrentDate^.Feeds^ do
          FdName := PAvailableFeed(At(RSData^.SelectedFeed))^.Name;
        D^.Title := NewStr(Date + ' - ' + FdName + ' - ' + An^.Name)
      end;
      D^.HelpCtx := hcQuantityDlg;
      C:= ExecuteDialog(D , AVData);
      if C <> cmCancel then
      begin
        val(AVData^, Allocation, Error);
        val(PAvailableFeed(CurrentDate^.Feeds^.At(RSDAta^.SelectedFeed))^.Quantity, Total,
Error);
        if Allocation > Total then
          RMsgBox(smbTooMuchOffered, nil, mfError+mfOKButton);
      end;
    Until Allocation <= Total;
    with RSData^ do
    begin
      Fd := New(PAvailableFeed, Init(FdName, AVData^));
      An^.Meal^.Insert(Fd);
      if Allocation = Total then
      begin
        CurrentDate^.Feeds^.AtDelete(SelectedFeed);
        if SelectedFeed <>0 then SelectedFeed := SelectedFeed - 1
      end
      else
      begin
        Str(Total - Allocation:8:3, AVData^);
        AVData^ := TidyNumStr(AVData^);
        Fd := New(PAvailableFeed, Init(FdName, AVData^));
        CurrentDate^.Feeds^.AtPut(SelectedFeed, Fd)
      end
    end
  end
  else RMsgBox(smbAllAllocated, nil, mfError+mfOKButton);
end;

procedure ResetSim;
begin
  HerdBkp.Seek(0);
  FeedsBkp.Seek(0);
  with CurrentDate^ do
  begin
```

```
    Feeds^.FreeAll;
    Feeds := PFeeds(FeedsBkp.Get);
    Herd^.FreeAll;
    Herd := PHerd(HerdBkp.Get);
   end
  end;

begin
  CD := 0;
  CurrentDate := Sim^.Calendar^.At(0);
  NextDate := Sim^.Calendar^.At(1);
  while (CD <> Sim^.Calendar^.Count) and (C <> cmCancel) do
  begin

   HerdBkp.Init(1, 1024);
   FeedsBkp.Init(1, 1024);

   with CurrentDate^ do
   begin
    HerdBkp.Put(Herd);
    FeedsBkp.Put(Feeds)
   end;


   with RSData^ do
   begin
    SelectedFeed := 0;
    SelectedAnimal := 0;
    Date := CurrentDate^.Date;
    str(Sim^.Calendar^.IndexOf(CurrentDate) + 1, Observation);
    AnimalList := New(PStringCollection, Init(10, 2));
    FeedList := New(PStringCollection, Init(10, 2))
   end;

   Repeat
    with RSData^ do
    begin
     with CurrentDate^ do
     begin
      Feeds^.MakeViewerList(FeedList);
      Herd^.MakeViewerList(AnimalList)
     end
    end;
    D := PFRAMEDlg(ResFile.Get('DIALOG_RunSimulation'));
    D^.HelpCtx := hcRunSimulationDlg;
    C := ExecuteDialog(D, RSData);
    case C of
     cmCullAnimal : CullAnimal;
     cmAddAnimal : AddAnimal;
     cmViewAnimals :  ViewAnimals;
     cmAllocateFeeds : AllocateFeeds;
     cmResetSim : ResetSim;
     cmCancel : begin
                FeedsBkp.Done;
                HerdBkp.Done;
                Exit
```

```
            end;
        cmOK : begin
            if FeedBase^.Count <> 0 then ClearDatabase;
            LoadDatabase(Sim^.DBFileName);

{crashes here with loaded simulation}

            CurrentDate^.Herd^.ForEach(@OneAnimal);
          end
      end;
      Until C = cmNext;

      if CurrentDate^.Feeds^.Count <> 0 then
      begin
        {Put leftovers in compost heap}
        CurrentDate^.Feeds^.ForEach(@Leftovers);
      end;

      CD := CD + 1;
      CurrentDate := NextDate;
      if CD + 1 < Sim^.Calendar^.Count then NextDate := Sim^.Calendar^.At(CD + 1)
      else NextDate := nil;

      FeedBase^.Reset;
      RSData^.AnimalList^.Done;
      FeedsBkp.Done;
      HerdBkp.Done;

  end;
  RMsgBox(smbSimComplete, nil, mfInformation + mfOKButton)
end;

procedure FM.ViewProduction;
var F : Text;
    I : Integer;
    R : TRect;
    Title : String;
begin
  Assign(F, WindowFN);
  ReWrite(F);
  Sim^.Calendar^.ViewProductionData(F);
  Close(F);
  DeskTop^.GetExtent(R);
  R.Assign(numWindows, numWindows, R.B.X, R.B.Y);
  Title := 'HerdPerformance - ' + Sim^.BasicInfo.Title;
  OpenWindow(Title, R)
end;

procedure FM.ViewCompost;
var F : Text;
    I : Integer;
    R : TRect;
    Title : String;
begin
  Assign(F, WindowFN);
  ReWrite(F);
```

```
  Sim^.Calendar^.ViewCompostData(F);
  Close(F);
  DeskTop^.GetExtent(R);
  R.Assign(numWindows, numWindows, R.B.X, R.B.Y);
  Title := 'Compost Production - ' + Sim^.BasicInfo.Title;
  OpenWindow(Title, R)
 end;

function FM.ClearSimulation: Boolean;
begin
  Result := RMsgBox(smbClearSim, nil, mfConfirmation+mfYesButton+mfNoButton);
  if Result = 12 then
  begin
    if SimChanged then
    begin
      Result := RMsgBox(smbNotSaved, nil, mfWarning + mfYesNoCancel);
      Case Result of
        11 : Exit;
        12 : SaveFile('*.SIM')
      end
    end;
    Sim^.Reset;
    SimChanged := False;
    ClearSimulation := True
  end
  else ClearSimulation := False
end;

procedure FM.ChangeScrColour;
begin
  ExecuteDialog(PDialog(ResFile.Get('DIALOG_ChangeScrColour')), SCData);
  Case SCData^ of
    0: AppPalette:=apColor;
    1: AppPalette:=apBlackWhite;
    2: AppPalette:=apMonoChrome
  end;
  DeskTop^.Redraw;
  ReDraw
end;

procedure FM.DoAboutBox;
var AboutBox: PDialog;
begin
  ExecuteDialog(PDialog(ResFile.Get('DIALOG_AboutBox')), nil);
end;

procedure FM.OpenWindow(WTitle: string; Bounds: TRect);
var W : PFRAMEWindow;
    R : TRect;
begin
  if numWindows < maxWindows then
  begin
    R:=Bounds;
    New(W, Init(WTitle, R));
    DeskTop^.Insert(W)
  end
```

```
  else MessageBox('Only nine windows may be open concurrently. ' +
                  'Close unwanted windows and try again.',
       nil, mfError + mfOkButton);

end;


var
  FeedModel: FM;

BEGIN
  FeedModel.Init;
  FeedModel.Run;
  FeedModel.Done
END.
```

```
program FRAMERES;

uses Drivers, Objects, Views, Dialogs, Editors, Validate, Menus, App,
   StdDlg,
   HelpCont,
   Tools, Globals;

{$I commands.inc}
{$I strings.inc}

var
 ResFile : TResourceFile;

{*** Resource procedures ***}
procedure CreateStrings;
var
 S: PStrListMaker;
begin
 S := New(PStrListMaker, Init(10000, 100));

 S^.Put(smbNotSaved, 'Changes not saved.' + #13#13'Save now?');
 S^.Put(smbFeedNameUsed, 'Feed name already used.' + #13#13'Replace existing feed of this
name?');
 S^.Put(smbRemoveFeed, 'Remove feed from database');
 S^.Put(smbClearDB, 'Clear current database from memory?');
 S^.Put(smbDBInMem, 'Database currently in memory' + #13#13'Select OK to overwrite');
 S^.Put(smbOpenDBError, 'Unable to open database');
 S^.Put(smbFileOverWrite, 'This file already exists.' + #13#13'Overwrite?');
 S^.Put(smbInvalidDate, 'Date not acceptable. Must be later' + #13'previous date');
 S^.Put(smbBasicDataNotSet, 'Observation dates or associated database not set.' +
#13#13'Complete or select cancel.');
 S^.Put(smbClearSim, 'Clear current simulation from memory?');
 S^.Put(smbDatesAlreadySet, 'Dates already set.' + #13#13'Use Edit | Basic data to change.');
 S^.Put(smbFeedsInMem, 'Feed availability already entered' + #13#13'Erase and reset?');
 S^.Put(smbNextAnimal, 'Enter data for another animal');
 S^.Put(smbHerdInMem, 'Initial herd structure in memory' + #13#13'Erase and reset?');
 S^.Put(smbHelpOpenFailed, 'Could not open help file.');
 S^.Put(smbNoDebugTest, 'There is no procedure attached to' + #13#13'Debug|Test');
 S^.Put(smbOpenFileError, 'Unable to open file');
 S^.Put(smbSimInMem, 'Simulation currently in memory' + #13#13'Select OK to overwrite');
 S^.Put(smbTooMuchOffered, 'Amount offered must not exceed amount available');
 S^.Put(smbCullAnimal, 'Cull animal from herd?');
 S^.Put(smbRemoveAnimal, 'Remove animal from database');
 S^.Put(smbSimComplete, 'Simulation complete');
 S^.Put(smbAllAllocated, 'All feeds allocated for this month');

 ResFile.Put(S, 'Strings');
 Dispose(S, Done);
end;

procedure CreateMenu;
var
 R: TRect;
 P: PView;
begin
 R.Assign(0, 0, 80, 1);
```

```
  P := New(PMenuBar, Init(R,
   NewMenu(
    NewSubMenu('~D~ata '#179, hcDataBase, NewMenu(
     NewItem('~A~dd...','',kbNoKey, cmAddFeed, hcAddFeed,
     NewItem('View / ~E~dit...','',kbNoKey,cmViewEditFeed, hcViewEditFeed,
     NewItem('~D~elete...','',kbNoKey,cmDeleteFeed, hcDeleteFeed,
     NewLine(
     NewItem('~L~oad...','',kbNoKey, cmLoadDatabase, hcLoadDatabase,
     NewItem('~S~tore','',kbNoKey, cmStoreDatabase, hcStoreDatabase,
     NewItem('Store ~a~s...','',kbNoKey, cmStoreAs, hcStoreAs,
     NewItem('~C~lear','',kbNoKey, cmClearDatabase, hcClearDatabase,
    nil))))))))),
    NewSubMenu('~F~ile', hcFile, NewMenu(
     NewItem('~O~pen...', '', kbNoKey, cmOpen, hcOpen,
     NewItem('~S~ave', '', kbNoKey, cmSave, hcSave,
     NewItem('Save ~a~s...', '', kbNoKey, cmSaveAs, hcSaveAs,
     NewLine(
     NewItem('~C~hange dir...', '', kbNoKey, cmChangeDir, hcChangeDir,
     NewItem('~D~os shell', '', kbNoKey, cmDosShell, hcDosShell,
     NewLine(
     NewItem('E~x~it', 'Alt-X', kbAltX, cmExitFM, hcExit,
    nil))))))))),
    NewSubMenu('~S~imulate ', hcSimulation, NewMenu(
     NewItem('~N~ew...', '', kbNoKey, cmNewBasicInfo, hcNewBasicInfo,
     NewItem('~E~dit...', '', kbNoKey, cmEditBasicInfo, hcEditBasicInfo,
     NewItem('~C~lear','',kbNoKey, cmClearSimulation, hcClearSimulation,
     NewLine(
     NewItem('~R~un...', '', kbNoKey, cmRunSimulation, hcRunSimulation,
     NewSubMenu('~S~ummary', hcSummary, NewMenu(
      NewItem('~P~roduction', '', kbNoKey, cmViewProduction, hcViewProduction,
      NewItem('~C~ompost', '', kbNoKey, cmViewCompost, hcViewCompost,
      nil))),
     nil)))))))),
    NewSubMenu('F~e~eds', hcFeeds, NewMenu(
     NewItem('~A~vailability...', '', kbNoKey, cmNewAvailability, hcNewAvailability,
     NewItem('~E~dit...', '', kbNoKey, cmEditAvailability, hcEditAvailability,
    nil))),
    NewSubMenu('He~r~d', hcHerd, NewMenu(
     NewItem('~N~ew...', '', kbNoKey, cmNewHerd, hcNewHerd,
     NewItem('~A~dd animal...', '', kbNoKey, cmAddToHerd, hcAddToHerd,
     NewItem('~E~dit animal...', '', kbNoKey,  cmEditAnimal, hcEditAnimal,
     NewItem('~D~elete animal...', '', kbNoKey,  cmDeleteAnimal, hcDeleteAnimal,
    nil))))),
    NewSubMenu('~C~onfigure', hcConfigure, NewMenu(
     NewItem('Change c~o~lours...', '', kbNoKey, cmChangeScrColour, hcChangeScrColour,
    nil)),
    NewSubMenu('~H~elp', hcHelp, NewMenu(
     NewItem('~A~bout FRAME', '', kbNoKey, cmAbout, hcAbout,
    nil)),
   nil))))
  ))))));
  ResFile.Put(P, 'MENU_Main');
  Dispose(P, Done)
end;

procedure CreateStatusLine;
```

```
var
  R: TRect;
  P: PView;
begin
  R.Assign(0, 24, 80, 25);
   P:=New(PHintLine, Init(R,
    NewStatusDef(0, 0,
      NewStatusKey('~F1~ Help', kbF1, cmHelp,
      NewStatusKey('~F3~ Test', kbF3, cmTestNew,
      NewStatusKey('~Alt-X~ Exit', kbAltX, cmExitFM,
    nil))),
    NewStatusDef(1, $FFFF,
      NewStatusKey('~F1~ Help', kbF1, cmHelp,
    nil),
  nil))));
  ResFile.Put(P, 'STATUS_Line');
  Dispose(P, Done);
end;

procedure CreateOpenDlg;
var
  D: PDialog;
begin
  D := New(PFileDialog, Init('*.FDB', '',
        '~N~ame', fdOKButton + fdHelpButton, 100));
  ResFile.Put(D, 'DIALOG_Open');
  Dispose(D, Done);
end;

procedure CreateSaveDlg;
var
  D: PFileDialog;
begin
  D := New(PFileDialog, Init('*.*', '', '~N~ame',
    fdOkButton + fdHelpButton, 101));
  ResFile.Put(D, 'DIALOG_Save');
  Dispose(D, Done);
end;

procedure CreateChangeDirDlg;
var
  R: TRect;
  D: PDialog;
begin
  D := New(PChDirDialog, Init(cdNormal + cdHelpButton + cdNoLoadDir, 101));
  D^.HelpCtx := hcChangeDirDlg;
  ResFile.Put(D, 'DIALOG_ChangeDir');
  Dispose(D, Done);
end;

procedure CreateFeedCompDlg;
var
  R: TRect;
  D: PDialog;
  WHAM : PView;
  I: PInputLine;
```

```
begin
 R.Assign(0, 0, 62, 19);
 D:=New(PDialog, Init(R, ''));
 with D^ do
 begin
   Options := Options or ofCentered;
   R.Assign(2, 5, 25, 14);

   { Problem with help contexts }

   WHAM:= New(PRadioButtons, Init(R,
     NewSItem('~G~rass',
     NewSItem('Grass ~h~ay',
     NewSItem('Legume ha~y~',
     NewSItem('~C~ereal residue',
     NewSItem('~L~egume residue',
     NewSItem('~T~ree fodder',
     NewSItem('G~r~ain',
     NewSItem('~A~nimal protein',
     NewSItem('~V~egetable protein',
     nil)))))))))
   ));
   WHAM^.HelpCtx := hcFeedTypeRb;
   Insert(WHAM);
   R.Assign(2, 4, 36, 5);
   Insert(New(Plabel, Init(r, 'Type of Feed', WHAM)));

   R.Assign(53, 5, 60, 6);
   I:=New(PForceInputLine, Init(R, 5));
   I^.SetValidator(New(PFilterValidator, Init(NumSet)));
   I^.HelpCtx := hcDryMatterII;
   Insert(I);
   R.Assign(26, 5, 52, 6);
   Insert(New(PLabel, Init(R, 'Dry matter (g/kg)', I)));

   R.Assign(53, 7, 60, 8);
   I:=New(PForceInputLine, Init(R, 5));
   I^.SetValidator(New(PFilterValidator, Init(NumSet)));
   I^.HelpCtx := hcCrudeProteinII;
   Insert(I);
   R.Assign(26, 7, 52, 8);
   Insert(New(PLabel, Init(R, 'Crude protein (g/kg DM)', I)));

   R.Assign(53, 9, 60, 10);
   I:=New(PForceInputLine, Init(R, 5));
   I^.SetValidator(New(PFilterValidator, Init(NumSet)));
   I^.HelpCtx := hcCrudeFibreII;
   Insert(I);
   R.Assign(26, 9, 52, 10);
   Insert(New(PLabel, Init(R, 'Crude fibre (g/kg DM)', I)));

   R.Assign(53, 11, 60, 12);
   I:=New(PForceInputLine, Init(R, 5));
   I^.SetValidator(New(PFilterValidator, Init(NumSet)));
   I^.HelpCtx := hcEtherExtractII;
   Insert(I);
```

```
    R.Assign(26, 11, 52, 12);
    Insert(New(PLabel, Init(R, 'Ether Extract (g/kg DM)', I)));

    R.Assign(53, 13, 60, 14);
    I:=New(PForceInputLine, Init(R, 5));
    I^.SetValidator(New(PFilterValidator, Init(NumSet)));
    I^.HelpCtx := hcMetEnergyII;
    Insert(I);
    R.Assign(26, 13, 52, 14);
    Insert(New(PLabel, Init(R, 'ME - Ruminant (MJ/kg DM)', I)));

    R.Assign(10, 16, 22, 18);
    Insert(New(POKButton, Init(R)));
    R.Assign(25, 16, 37, 18);
    Insert(New(PCancelButton, Init(R)));
    R.Assign(40, 16, 52, 18);
    Insert(New(PHelpButton, Init(R)));

    R.Assign(17, 2, 39, 3);
    WHAM:=New(PForceInputLine, Init(R, 18));
    WHAM^.HelpCtx := hcFeedNameII;
    Insert(WHAM);
    R.Assign(2, 2, 16, 3);
    Insert(New(PLabel, Init(R, 'Name of Feed', WHAM)));
  end;
  ResFile.Put(D, 'DIALOG_FeedComp');
  Dispose(D, Done)
end;

procedure CreateStringBoxDlg;
var
  R: TRect;
  D: PDialog;
  WHAM : PView;
  ScrollBar: PScrollBar;
begin
  R.Assign(0, 0, 48, 11);
  D:=New(PDialog, Init(R, ''));
  with D^ do
  begin
    Options := Options or ofCentered;

    R.Assign(3, 8, 15, 10);
    Insert(New(POKButton, Init(R)));
    R.Assign(18, 8, 30, 10);
    Insert(New(PCancelButton, Init(R)));
    R.Assign(33, 8, 45, 10);
    Insert(New(PHelpButton, Init(R)));

    R.Assign(40, 2, 41, 6);
    New(ScrollBar, Init(R));
    Insert(ScrollBar);

    R.Assign(8, 2, 40, 6);
    WHAM:=New(PStringBox, Init(R, 1, ScrollBar, bxSingle));
    WHAM^.HelpCtx := hcStringBox;
```

```
    Insert(WHAM);
  end;
  ResFile.Put(D, 'DIALOG_StringBox');
  Dispose(D, Done)
end;

procedure CreateSwapBoxDlg;
var
  R: TRect;
  D: PSwapDlg;
  WHAM : PView;
  ScrollBar: PScrollBar;
begin
  R.Assign(0, 0, 70, 14);
  D:=New(PSwapDlg, Init(R, ''));
  with D^ do
  begin
    Options := Options or ofCentered;

    R.Assign(67, 3, 68, 9);
    New(ScrollBar, Init(R));
    Insert(ScrollBar);

    R.Assign(31, 3, 40, 5);
    Insert(New(PButton, Init(R, #196#196#196#196#16, cmRight, bfNormal)));

    R.Assign(42, 3, 67, 9);
    WHAM:=New(PStringBox, Init(R, 1, ScrollBar, bxLeft));
    WHAM^.HelpCtx := hcStringBox;
    Insert(WHAM);
    R.Assign(42, 2, 67, 3);
    Insert(New(PLabel, Init(R, 'Feeds to use', WHAM)));

    R.Assign(31, 5, 40, 7);
    Insert(New(PButton, Init(R, #17#196#196#196#196, cmLeft, bfNormal)));

    R.Assign(13, 11, 25, 13);
    Insert(New(POKButton, Init(R)));
    R.Assign(28, 11, 40, 13);
    Insert(New(PCancelButton, Init(R)));
    R.Assign(43, 11, 55, 13);
    Insert(New(PHelpButton, Init(R)));

    R.Assign(28, 3, 29, 9);
    New(ScrollBar, Init(R));
    Insert(ScrollBar);

    R.Assign(2, 3, 28, 9);
    WHAM:=New(PStringBox, Init(R, 1, ScrollBar, bxRight));
    WHAM^.HelpCtx := hcStringBox;
    Insert(WHAM);
    R.Assign(2, 2, 28, 3);
    Insert(New(PLabel, Init(R, 'Feeds available', WHAM)));

  end;
  ResFile.Put(D, 'DIALOG_SwapBox');
```

```
  Dispose(D, Done)
end;

procedure CreateBasicInformationDlg;
var
  R: TRect;
  D: PDialog;
  WHAM : PView;
  I: PInputLine;
  Memo: PMemo;
  ScrollBar: PScrollBar;
begin
  R.Assign(0, 0, 50, 19);
  D:=New(PFRAMEDlg, Init(R, ''));
  with D^ do
  begin
    Options := Options or ofCentered;

    R.Assign(37, 4, 47, 5);
    I:=New(PForceInputLine, Init(R, 8));
    I^.HelpCtx := hcStartDateII;
    Insert(I);
    I^.SetValidator(New(PPXPictureValidator,
      Init('{#[#]}/{#[#]}/{##[##]}', True)));
    R.Assign(2, 4, 24, 5);
    Insert(New(PLabel, Init(R, 'Start date', I)));

    R.Assign(3, 6, 23, 7);
    Insert(New(PStaticText, Init(R, 'Observation calendar')));
    R.Assign(36, 6, 48, 8);
    WHAM := New(PButton, Init(R, '~D~ates', cmMakeCalendar, bfNormal));
    WHAM^.HelpCtx := hcMakeCalendarButton;
    Insert(WHAM);

    R.Assign(3, 8, 23, 9);
    Insert(New(PStaticText, Init(R, 'Associated database')));
    R.Assign(36, 8, 48, 10);
    WHAM := New(PButton, Init(R, '~S~et', cmSetDatabase, bfNormal));
    WHAM^.HelpCtx := hcSetDatabaseButton;
    Insert(WHAM);

    R.Assign(46, 11, 47, 15);
    New(ScrollBar, Init(R));
    Insert(ScrollBar);


    R.Assign(3, 11, 46, 15);
    Memo := New(PMemo, Init(R, nil, ScrollBar, nil, 255));
    Memo^.HelpCtx := hcSimNotes;
    Insert(Memo);
    R.Assign(2, 10, 30, 11);
    Insert(New(PLabel, Init(R, 'Notes', Memo)));

    R.Assign(4, 16, 16, 18);
    Insert(New(POKButton, Init(R)));
    R.Assign(19, 16, 31, 18);
```

```
    Insert(New(PCancelButton, Init(R)));
    R.Assign(34, 16, 46, 18);
    Insert(New(PHelpButton, Init(R)));

    R.Assign(25, 2, 47, 3);
    WHAM:=New(PForceInputLine, Init(R, 18));
    WHAM^.HelpCtx := hcSimTitleII;
    Insert(WHAM);
    R.Assign(2, 2, 24, 3);
    Insert(New(PLabel, Init(R, 'Title', WHAM)));
  end;
  ResFile.Put(D, 'DIALOG_BasicInformation');
  Dispose(D, Done)
end;

procedure CreateCalendarDlg;
var
  R : TRect;
  D : PDialog;
  Button : PButton;
  I : PInputLine;
begin
  R.Assign(0, 0, 47, 11);
  D:=New(PFRAMEDlg, Init(R, ''));
  with D^ do
  begin
    Options := Options or ofCentered;

    R.Assign(2, 5, 14, 7);
    Button := New(PButton, Init(R, '~N~ext', cmOK, bfDefault));
    Button^.HelpCtx := hcNextDate;
    Insert(Button);
    R.Assign(32, 5, 44, 7);
    Button := New(PButton, Init(R, 'C~o~mplete', cmCalendarComplete, bfNormal));
    Button^.HelpCtx := hcDatesComplete;
    Insert(Button);

    R.Assign(10, 8, 22, 10);
    Insert(New(PCancelButton, Init(R)));
    R.Assign(24, 8, 36, 10);
    Insert(New(PHelpButton, Init(R)));

    R.Assign(34, 2, 44, 3);
    I:=New(PForceInputLine, Init(R, 8));
    Insert(I);
    I^.SetValidator(New(PPXPictureValidator,
      Init('{#[#]}/{#[#]}/{##[##]}', True)));
    R.Assign(2, 2, 33, 3);
    Insert(New(PLabel, Init(R, 'Date .....', I)));
  end;
  ResFile.Put(D, 'DIALOG_Calendar');
  Dispose(D, Done)
end;

procedure CreateCalendarBoxDlg;
var
```

```
  R: TRect;
  D: PDialog;
  WHAM : PView;
  ScrollBar: PScrollBar;
begin
  R.Assign(0, 0, 48, 11);
  D:=New(PDialog, Init(R, ''));
  with D^ do
  begin
    Options := Options or ofCentered;

    R.Assign(3, 8, 15, 10);
    Insert(New(POKButton, Init(R)));
    R.Assign(18, 8, 30, 10);
    Insert(New(PCancelButton, Init(R)));
    R.Assign(33, 8, 45, 10);
    Insert(New(PHelpButton, Init(R)));

    R.Assign(34, 2, 35, 6);
    New(ScrollBar, Init(R));
    Insert(ScrollBar);

    R.Assign(14, 2, 34, 6);
    WHAM:=New(PCalendarBox, Init(R, 1, ScrollBar, 2));
    WHAM^.HelpCtx := hcStringBox;
    Insert(WHAM);
  end;
  ResFile.Put(D, 'DIALOG_CalendarBox');
  Dispose(D, Done)
end;

procedure CreateAvailabilityDlg;
var
  R: TRect;
  D: PDialog;
  WHAM : PView;
  I : PInputLine;
begin
  R.Assign(0, 0, 65, 8);
  D:=New(PDialog, Init(R, ''));
  with D^ do
  begin
    Options := Options or ofCentered;
    {HelpCtx := hcViewEditDlg;}

    R.Assign(12, 5, 24, 7);
    Insert(New(POKButton, Init(R)));
    R.Assign(27, 5, 39, 7);
    Insert(New(PCancelButton, Init(R)));
    R.Assign(42, 5, 54, 7);
    Insert(New(PHelpButton, Init(R)));

    R.Assign(48, 2, 58, 3);
    I:=New(PForceInputLine, Init(R, 8));
    I^.SetValidator(New(PFilterValidator, Init(NumSet)));
    Insert(I);
```

```
    R.Assign(8, 2, 43, 3);
    Insert(New(PLabel, Init(R, 'Quantity (kg)', I)));

  end;
  ResFile.Put(D, 'DIALOG_Availability');
  Dispose(D, Done)
end;


procedure CreateAnimalDlg;
var
  WHAM: PView;
  D: PDialog;
  R: TRect;
  I: PInputLine;
begin
  R.Assign(9, 2, 70, 17);
  D := New(PDialog, Init(R, ''));
  with D^ do
  begin
    Options := Options or ofCentered;
    {HelpCtx := hcViewEditDlg;}

    R.Assign(22, 3, 39, 6);
    WHAM := New(PRadioButtons, Init(R,
      NewSItem('~M~ale',
      NewSItem('~F~emale',
      NewSItem('~C~astrate',
      nil)))
    ));
    Insert(WHAM);
    R.Assign(21, 2, 38, 3);
    Insert(New(PLabel, Init(R, 'Sex', WHAM)));

    R.Assign(41, 3, 58, 5);
    WHAM := New(PRadioButtons, Init(R,
      NewSItem('G~r~owing',
      NewSItem('M~a~ture',
      nil))
    ));
    Insert(WHAM);
    R.Assign(40, 2, 57, 3);
    Insert(New(PLabel, Init(R, 'Class', WHAM)));


    R.Assign(41, 7, 58, 8);
    WHAM:=new(PCheckBoxes, init(R,
      NewSItem('Working',
      nil)
    ));
    Insert(WHAM);

    R.Assign(36, 9, 44, 10);
    I:=New(PForceInputLine, Init(R, 6));
    I^.SetValidator(New(PFilterValidator, Init(NumSet)));
    Insert(I);
```

```
    R.Assign(16, 9, 35, 10);
    Insert(New(PLabel, Init(R, 'Liveweight (kg)', I)));

    R.Assign(9, 12, 21, 14);
    Insert(New(POKButton, Init(R)));
    R.Assign(24, 12, 36, 14);
    Insert(New(PCancelButton, Init(R)));
    R.Assign(39, 12, 51, 14);
    Insert(New(PHelpButton, Init(R)));

    R.Assign(3, 3, 20, 8);
    WHAM := New(PRadioButtons, Init(R,
     NewSItem('~B~uffalo',
     NewSItem('Bos ~t~aurus',
     NewSItem('Bos ~i~ndicus',
     NewSItem('~G~oat',
     NewSItem('~S~heep',
     nil)))
    ))));
    Insert(WHAM);
    R.Assign(2, 2, 10, 3);
    Insert(New(PLabel, Init(R, 'Species', WHAM)));
  end;
  ResFile.Put(D, 'DIALOG_Animal');
  Dispose(D, Done)
end;

procedure CreateMatureFemaleDlg;
var
  WHAM: PView;
  D: PDialog;
  R: TRect;
  I: PInputLine;
begin
  R.Assign(0, 0, 47, 10);
  D := New(PDialog, Init(R, ''));
  with D^ do
  begin
    Options := Options or ofCentered;
    {HelpCtx := hcViewEditDlg;}

    R.Assign(37, 4, 43, 5);
    I:=New(PForceInputLine, Init(R, 6));
    I^.SetValidator(New(PRangeValidator, Init(0, 300)));
    Insert(I);
    R.Assign(2, 4, 36, 5);
    Insert(new(PLabel, Init(R, 'Day of pregnancy', I)));

    R.Assign(2, 7, 14, 9);
    Insert(New(POKButton, Init(R)));
    R.Assign(17, 7, 29, 9);
    Insert(New(PCancelButton, Init(R)));
    R.Assign(32, 7, 44, 9);
    Insert(New(PHelpButton, Init(R)));

    R.Assign(37, 2, 43, 3);
```

```
    I:=New(PForceInputLine, Init(R, 6));
    I^.SetValidator(New(PRangeValidator, Init(0, 600)));
    Insert(I);
    R.Assign(2, 2, 36, 3);
    Insert(New(PLabel, Init(R, 'Day of lactation', I)));

  end;
  ResFile.Put(D, 'DIALOG_MatureFemale');
  Dispose(D, Done)
end;

procedure CreateRunSimDlg;
var
  R: TRect;
  D: PDialog;
  WHAM : PView;
  ScrollBar: PScrollBar;
  C : Integer;
begin
  R.Assign(0, 0, 77, 18);
  D:=new(PFRAMEDlg, Init(R, 'Run Simulation'));
  with D^ do
  begin
    Options := Options or ofCentered;

    for C := 3 to 73 do
    begin
      R.Assign(C, 3, C+1, 4);
      if C <> 39 then Insert(New(PStaticText, Init(R, #196)))
      else Insert(New(PStaticText, Init(R, #194)));
    end;
    for C := 3 to 73 do
    begin
      R.Assign(C, 13, C+1, 14);
      if C <> 39 then Insert(New(PStaticText, Init(R, #196)))
      else Insert(New(PStaticText, Init(R, #193)));
    end;
    for C := 4 to 12 do
    begin
      R.Assign(39, C, 40, C + 1);
      Insert(New(PStaticText, Init(R, #179)));
    end;

    R.Assign(18, 2, 28, 3);
    WHAM:=New(PInputLine, Init(R, 8));
    WHAM^.Options := WHAM^.Options-OfSelectable;
    Insert(WHAM);
    R.Assign(2, 2, 16, 3);
    Insert(New(PLabel, Init(R, 'Current date', WHAM)));

    R.Assign(64, 2, 74, 3);
    WHAM:=New(PInputLine, Init(R, 8));
    WHAM^.Options := WHAM^.Options-OfSelectable;
    Insert(WHAM);
    R.Assign(49, 2, 63, 3);
    Insert(New(PLabel, Init(R, 'Observation', WHAM)));
```

```
R.Assign(5, 11, 13, 13);
WHAM := New(PButton, Init(R, 'C~u~ll', cmCullAnimal, bfNormal));
WHAM^.HelpCtx := hcCullAnimal;
Insert(WHAM);
R.Assign(16, 11, 24, 13);
WHAM := New(PButton, Init(R, '~A~dd', cmAddAnimal, bfNormal));
WHAM^.HelpCtx := hcAddAnimal;
Insert(WHAM);
R.Assign(27, 11, 35, 13);
WHAM := New(PButton, Init(R, '~V~iew', cmViewAnimals, bfNormal));
WHAM^.HelpCtx := hcViewAnimals;
Insert(WHAM);

R.Assign(73, 5, 74, 10);
New(ScrollBar, Init(R));
Insert(ScrollBar);

R.Assign(42, 5, 73, 10);
WHAM:=New(PListBox, Init(R, 1, ScrollBar));
Insert(WHAM);
R.Assign(41, 4, 73, 5);
Insert(New(PLabel, Init(R, 'Feeds available', WHAM)));

R.Assign(44, 11, 56, 13);
WHAM := New(PButton, Init(R, '~A~llocate', cmAllocateFeeds, bfNormal));
WHAM^.HelpCtx := hcAllocateFeeds;
Insert(WHAM);
R.Assign(59, 11, 71, 13);
WHAM := New(PButton, Init(R, '~C~onserve', cmConserveFeeds, bfNormal));
WHAM^.HelpCtx := hcConserveFeeds;
Insert(WHAM);

R.Assign(3, 15, 15, 17);
WHAM := New(PButton, Init(R, '~R~un', cmOK, bfDefault));
WHAM^.HelpCtx := hcRun;
Insert(WHAM);
R.Assign(18, 15, 30, 17);
WHAM := New(PButton, Init(R, 'Rese~t~', cmResetSim, bfNormal));
WHAM^.HelpCtx := hcReset;
Insert(WHAM);
R.Assign(33, 15, 45, 17);
WHAM := New(PButton, Init(R, '~N~ext', cmNext, bfNormal));
WHAM^.HelpCtx := hcNext;
Insert(WHAM);



R.Assign(47, 15, 59, 17);
Insert(New(PCancelButton, Init(R)));
R.Assign(62, 15, 74, 17);
Insert(New(PHelpButton, Init(R)));

R.Assign(36, 5, 37, 10);
New(ScrollBar, Init(R));
```

```
    Insert(ScrollBar);

    R.Assign(3, 5, 36, 10);
    WHAM:=New(PListBox, Init(R, 1, ScrollBar));
    Insert(WHAM);
    R.Assign(2, 4, 36, 5);
    Insert(New(PLabel, Init(R, 'Herd (* = fed)', WHAM)));

  end;
  ResFile.Put(D, 'DIALOG_RunSimulation');
  Dispose(D, Done)
end;

procedure CreateChangeScreenDlg;
var
  R: TRect;
  D: PDialog;
  WHAM : PView;
begin
  R.Assign(0, 0, 38, 11);
  D:=new(PDialog, Init(R, 'Change Colours'));
  with D^ do
  begin
    Options := Options or ofCentered;
    HelpCtx := hcChangeScrColourDlg;
    R.Assign(24, 2, 34, 4);
    Insert(New(POKButton, Init(R)));
    R.Assign(24, 5, 34, 7);
    Insert(New(PCancelButton, Init(R)));
    R.Assign(24, 8, 34, 10);
    Insert(New(PHelpButton, Init(R)));
    R.Assign(3, 2, 22, 5);
    WHAM := New(PRadioButtons, Init(R,
          NewSItem('~C~olour',
          NewSItem('~B~lack / White',
          NewSItem('~M~onochrome',
          nil)))
        ));
    Insert(WHAM)
  end;
  ResFile.Put(D, 'DIALOG_ChangeScrColour');
  Dispose(D, Done)
end;

procedure CreateAboutBox;
var
  R: TRect;
  D: PDialog;
begin
  R.Assign(0, 0, 40, 11);
  D := New(PDialog, Init(R, 'About FRAME'));
  with D^ do
  begin
    Options := Options or ofCentered;
    HelpCtx := hcAboutDlg;
    R.Assign(2, 2, 38, 4);
```

```
    Insert(New(PStaticText, Init(R, #3'-* FRAME *-'#13#3'Simulation of Feed Allocation')));
    R.Assign(2, 5, 38, 7);
    Insert(New(PStaticText, Init(R, #3'Copyright 1995'#13#3'Natural Resources Institute')));
    R.Assign(15, 8, 25, 10);
    Insert(New(POKButton, Init(R)));
  end;
  ResFile.Put(D, 'DIALOG_AboutBox');
  Dispose(D, Done)
end;

BEGIN
  ResFile.Init(New(PBufStream, Init('FRAME.TVR', stCreate, 3074)));

  RegisterViews;
  RegisterMenus;
  RegisterDialogs;
  RegisterValidate;
  RegisterStdDlg;
  RegisterEditors;
  RegisterType(RStrListMaker);

  RegisterTools;

  CreateStrings;
  CreateMenu;
  CreateStatusLine;
  CreateOpenDlg;
  CreateSaveDlg;
  CreateChangeDirDlg;
  CreateFeedCompDlg;
  CreateStringBoxDlg;
  CreateSwapBoxDlg;
  CreateBasicInformationDlg;
  CreateCalendarDlg;
  CreateCalendarBoxDlg;
  CreateAvailabilityDlg;
  CreateAnimalDlg;
  CreateMatureFemaleDlg;
  CreateRunSimDlg;
  CreateChangeScreenDlg;
  CreateAboutBox;

  ResFile.Done
END.
```

```
unit Globals;

interface

uses objects;

type

{*** Dialog Box Data Storage ***}

 {--- Setup ---}

 PSCRec = ^TSCRec;        {Screen colour scheme}
 TSCRec = Word;

 {--- Database ---}

 PFCRec = ^TFCRec;        {Feed composition}
 TFCRec = record
  FeedType: Word;
  DM : string[5];
  CP : string[5];
  CF : string[5];
  EE : string[5];
  ME : string[5];
  Name : string[20];
 end;

 PSBRec = ^TSBRec;          {String list box}
 TSBRec = record
  Items : PStringCollection;
  Selected : Integer
 end;

 PSWRec = ^TSWRec;          {Swap box}
 TSWRec = record
  List1 : PStringCollection;
  Selected1 : Integer;
  List2 : PStringCollection;
  Selected2 : Integer
 end;

 PAVRec = ^TAVRec;
 TAVRec = String[8];

 PANRec = ^TANRec;
 TANRec = record
  Sex : Word;
  Class : Word;
  Working : Word;
  LiveWeight : String[6];
  Species : Word
 end;

 PMFRec = ^TMFRec;
 TMFRec = record
```

```
    DayOfPregnancy : String[6];
    DayOfLactation : String[6]
  end;

  PWKRec = ^TWKRec;
  TWKRec = record
    DaysWorked : String[6];
    TimeWorked : String[6];
    Implement : Word
  end;

  DegParmRec = record
    a : Real;
    b : Real;
    c : Real;
    ADINCoeff : Real
  end;


const
  NoName = '';

  FCInfo: TFCRec = (FeedType : 0;
                    DM : NoName;
                    CP : NoName;
                    CF : NoName;
                    EE : NoName;
                    ME : NoName;
                    Name : NoName);

  ANInfo: TANRec = (Sex: 0;
                    Class: 0;
                    Working: 0;
                    Liveweight: NoName;
                    Species: 0);

  MFInfo: TMFRec = (DayOfPregnancy: '0';
                    DayOfLactation: '0');

  NumSet: TCharSet = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '.'];

  SpeciesNames: array[0..4] of String = ('Buffalo',
                                         'Bos taurus',
                                         'Bos indicus',
                                         'Goat',
                                         'Sheep');
  SexNames: array[0..2] of String = ('Male ',
                                     'Female ',
                                     'Castrated ');
  ClassNames: array[0..1] of String = ('Growing ',
                                       'Mature ');

procedure FillString(S: Word; var F: Text);

implementation
```

```
procedure FillString(S: Word; var F: Text);
Var
  Spaces : Word;
begin
  for Spaces:=1 to S do write(F, ' ')
end;

END.
```

```
unit HELPCONT;

interface

const


  {--- general ---}
  hcDebug                          = 1001;
    hcTestNew                      = 1002;
  hcStringBox                      = 1003;

  {--- Database ---}
  hcDatabase                       = 1101;
    hcAddFeed                      = 1102;
      hcFeedNameII                 = 1103;
      hcFeedTypeRb                 = 1104;
      hcDryMatterII                = 1105;
      hcCrudeProteinII             = 1106;
      hcCrudeFibreII               = 1107;
      hcEtherExtractII             = 1108;
      hcMetEnergyII                = 1109;
    hcViewEditFeed                 = 1110;
    hcDeleteFeed                   = 1111;
    hcLoadDatabase                 = 1112;
      hcLoadDatabaseDlg            = 1113;
    hcStoreDatabase                = 1114;
    hcStoreAs                      = 1115;
      hcStoreDatabaseDlg           = 1116;
    hcClearDatabase                = 1117;

  {--- File ---}
  hcFile                           = 1201;
     hcOpenDlg                     = 1202;
     hcSaveDlg                     = 1203;
     hcChangeDirDlg                = 1204;

  {--- Simulation ---}
  hcSimulation                     = 1301;
    hcNewBasicInfo                 = 1302;
      hcBasicInfoDlg               = 1303;
        hcSimTitleII               = 1304;
        hcStartDateII              = 1305;
        hcSimNotes                 = 1306;
        hcMakeCalendarButton       = 1307;
          hcNewCalendarDlg         = 1308;
          hcEditCalendarDlg        = 1309;
          hcNextDate               = 1310;
          hcDatesComplete          = 1311;
        hcSetDataBaseButton        = 1313;
          hcSetDatabaseDlg         = 1314;
    hcEditSim                      = 1315;
      hcEditBasicInfo              = 1316;
    hcRunSimulation                = 1317;
      hcRunSimulationDlg           = 1318;
        hcCullAnimal               = 1319;
```

```
        hcAddAnimal                 = 1320;
        hcViewAnimals               = 1321;
        hcAllocateFeeds             = 1322;
        hcConserveFeeds             = 1323;
        hcRun                       = 1324;
        hcReset                     = 1325;
        hcNext                      = 1326;
      hcSummary                     = 1327;
        hcViewProduction            = 1328;
        hcViewCompost               = 1329;
      hcClearSimulation             = 1330;

    {--- Feeds ---}
    hcFeeds                         = 1401;
      hcNewAvailability             = 1402;
        hcAvailabilityDlg           = 1403;
        hcQuantityDlg               = 1404;
      hcEditAvailability            = 1405;

    {--- Herd ---}
    hcHerd                          = 1501;
      hcNewHerd                     = 1502;
        hcAnimalDlg                 = 1503;
      hcAddToHerd                   = 1504;
      hcEditAnimal                  = 1505;
      hcDeleteAnimal                = 1506;

    {--- Configure ---}
    hcConfigure                     = 1801;
      hcChangeScrColour             = 1802;
        hcChangeScrColourDlg        = 1803;

    {--- Help ---}
    hcHelp                          = 1901;
      hcAbout                       = 1902;
        hcAboutDlg                  = 1903;

    {--- Buttons ---}
    hcOKButton                      = 1951;
    hcCancelButton                  = 1952;
    hcHelpButton                    = 1953;


implementation

END.
```

```
unit MUCK;

interface

uses Objects,
    Dos,
    Globals,
    Feeds;

type
 PExcreta = ^TExcreta;
 TExcreta = object(TObject)
  Name : String[40];
  Nitrogen : Real;
  constructor Init(Na : String; N : Real);
  constructor Load(var S : TStream);
  procedure Store(var S : TStream);
  procedure PrintData(var F : Text);
 end;

 PManure = ^TManure;
 TManure = object(TExcreta)
  constructor Init(Na : String; N : Real);
 end;

 PUrine = ^TUrine;
 TUrine = object(TExcreta)
  constructor Init(Na : String; N : Real);
 end;

 PCompostHeap = ^TCompostHeap;
 TCompostHeap = object(TSortedCollection)
  function KeyOf(Item: Pointer): Pointer; virtual;
  function Compare(Key1, Key2: Pointer): Integer; virtual;
  procedure MakeViewerList(var L: PStringCollection);
  procedure ViewData(Var F : Text);
 end;

const

 RExcreta: TStreamRec = (
  ObjType: 800;
  VmtLink: Ofs(TypeOf(TExcreta)^);
  Load: @TExcreta.Load;
  Store: @TExcreta.Store
 );

 RManure: TStreamRec = (
  ObjType: 810;
  VmtLink: Ofs(TypeOf(TManure)^);
  Load: @TManure.Load;
  Store: @TManure.Store
 );

 RUrine: TStreamRec = (
  ObjType: 820;
```

```
    VmtLink: Ofs(TypeOf(TUrine)^);
    Load: @TUrine.Load;
    Store: @TUrine.Store
  );

  RCompostHeap: TStreamRec = (
    ObjType: 830;
    VmtLink: Ofs(TypeOf(TCompostHeap)^);
    Load: @TCompostHeap.Load;
    Store: @TCompostHeap.Store
  );

  procedure RegisterMuck;


implementation

{*** TExcreta ***}

constructor TExcreta.Init(Na : String; N : Real);
begin
  Name := Na;
  Nitrogen := N
end;

constructor TExcreta.Load(var S: TStream);
begin
  S.Read(Name, SizeOf(String));
  S.Read(Nitrogen, SizeOf(Real))
end;

procedure TExcreta.Store(var S: TStream);
begin
  S.Write(Name, SizeOf(String));
  S.Write(Nitrogen, SizeOf(Real))
end;

procedure TExcreta.PrintData(var F : Text);
begin
  write(F, Name);
  FillString(44 - Length(Name), F);
  write(F, Nitrogen:6:1)
end;


{*** TManure ***}

constructor TManure.Init(Na : String; N : Real);
begin
  inherited Init(Na, N);
  Name := Name + ' (manure)'
end;


{*** TUrine ***}
```

```
constructor TUrine.Init(Na : String; N : Real);
begin
 inherited Init(Na, N);
 Name := Name + ' (urine)'
end;


{*** TCompostHeap ***}

function TCompostHeap.KeyOf(Item : Pointer) : Pointer;
begin
 KeyOf := NewStr(PExcreta(Item)^.Name);
end;

function TCompostHeap.Compare(Key1, Key2 : Pointer) : Integer;
begin
 if PString(Key1)^ = PString(Key2)^ then Compare := 0
 else if PString(Key1)^ < PString(Key2)^ then Compare := -1
 else Compare := 1;
end;

procedure TCompostHeap.MakeViewerList(var L: PStringCollection);

 procedure OneItem(Item : PExcreta); far;
 begin
  with Item^ do
  begin
   L^.Insert(NewStr(Name))
  end
 end;

begin
 L^.FreeAll;
 ForEach(@OneItem)
end;

procedure TCompostHeap.ViewData(Var F : Text);
var I : Integer;

 procedure OneItem(Item : PExcreta); far;
 begin
  Item^.PrintData(F);
  writeln(F)
 end;

begin
 writeln(F);
 write(F, 'Material');
 FillString(44, F);
 writeln(F, 'Nitrogen');
 for I := 1 to 80 do write(F, '_');
 writeln(F);
 writeln(F);
 ForEach(@OneItem);
 for I := 1 to 80 do write(F, '_');
 writeln(F);
```

```
end;

{*** Stream Registration ***}

procedure RegisterMuck;
begin
  RegisterType(RExcreta);
  RegisterType(RManure);
  RegisterType(RUrine);
  RegisterType(RCompostHeap)
end;


END.
```

```
unit SIMULAT;

interface

uses Dos,
    Objects,
    Globals, Dates;

type
  PSimulation = ^TSimulation;
  TSimulation = object(TObject)

    FileName : FNameStr;
    DBFileName : FNameStr;

    BasicInfo : TBIRec;

    BasicInfoSet : Boolean;
    FeedsSet : Boolean;
    HerdSet : Boolean;

    Calendar : PCalendar;

    constructor Init;
    constructor Load(var S: TStream);
    procedure Store(var S: TStream);
    procedure SetBasicInfo(BI: PBIRec; CA: PCalendar);
    function AllSet: Boolean;
    procedure Reset;
  end;

const
  RSimulation: TStreamRec = (
    ObjType: 400;
    VmtLink: Ofs(TypeOf(TSimulation)^);
    Load: @TSimulation.Load;
    Store: @TSimulation.Store
  );

procedure RegisterSimulation;

implementation

{*** TSimulation ***}

constructor TSimulation.Init;
begin
  inherited Init;
  FileName := '*.SIM';
  DBFileName := '*.FDB';
  BasicInfo := BIInfo;
  BasicInfoSet := False;
  FeedsSet := False;
  HerdSet := False;
  Calendar := New(PCalendar, Init(10, 2));
end;
```

```
constructor TSimulation.Load(var S: TStream);
begin
  S.Read(FileName, SizeOf(FNameStr));
  S.Read(DBFileName, SizeOf(FNameStr));
  S.Read(BasicInfo, SizeOf(TBIRec));
  S.Read(BasicInfoSet, SizeOf(Boolean));
  S.Read(FeedsSet, SizeOf(Boolean));
  S.Read(HerdSet, SizeOf(Boolean));
  Calendar := New(PCalendar, Init(10, 2));
  Calendar^.Load(S)
end;

procedure TSimulation.Store(var S: TStream);
begin
  S.Write(FileName, SizeOf(FNameStr));
  S.Write(DBFileName, SizeOf(FNameStr));
  S.Write(BasicInfo, SizeOf(TBIRec));
  S.Write(BasicInfoSet, SizeOf(Boolean));
  S.Write(FeedsSet, SizeOf(Boolean));
  S.Write(HerdSet, SizeOf(Boolean));
  Calendar^.Store(S)
end;

procedure TSimulation.SetBasicInfo(BI: PBIRec; CA: PCalendar);
Var
  Dn: DirStr;
  Nn: NameStr;
  En: ExtStr;
begin
  if DBFileName <> '*.FDB' then
  begin
    FSplit(DBFileName, Dn, Nn, En);
    DBFileName:= Nn + En
  end;
  BasicInfo := BI^;
  BasicInfoSet := True;
  Calendar := CA;
end;

function TSimulation.AllSet: Boolean;
begin
  AllSet:=(BasicInfoSet and FeedsSet and HerdSet)
end;

procedure TSimulation.Reset;
begin
  FileName := '*.SIM';
  DBFileName := '*.FDB';
  BasicInfo := BIInfo;
  BasicInfoSet := False;
  FeedsSet := False;
  HerdSet := False;
  Calendar^.FreeAll          {Reset method needed?}
end;
```

```
{*** Stream Registration ***}

procedure RegisterSimulation;
begin
  RegisterType(RSimulation)
end;

END.
```

```
unit TOOLS;

interface

uses Objects, App, Views, Drivers, Menus, Dialogs, MsgBox,
    Dates,
    Globals,
    HelpCont;

{$I commands.inc}

Const
 MaxWindows = 9;
 MaxLines = 200;
 numWindows: Integer = 0;
 windowNums: set of 1 .. MaxWindows = [];
 WindowFN = 'WINDOW';

type

 PForceInputLine = ^TForceInputLine;
 TForceInputLine = object(TInputLine)
  function Valid(Command: Word): Boolean; virtual;
 end;

 POKButton = ^TOKButton;
 TOKButton = object(TButton)
  constructor Init(var Bounds: TRect);
 end;

 PCancelButton = ^TCancelButton;
 TCancelButton = object(TButton)
  constructor Init(var Bounds: TRect);
 end;

 PHelpButton = ^THelpButton;
 THelpButton = object(TButton)
  constructor Init(var Bounds: TRect);
 end;

 PStringBox = ^TStringBox;
 TStringBox = object(TListBox)
  Left, Right: Boolean;
  constructor Init(var Bounds: TRect; ANumCols: Word;
                   AScrollBar: PScrollBar; LeftOrRight: Word);
  constructor Load(var S: TStream);
  procedure Store(var S: TStream);
  procedure HandleEvent(var Event: TEvent); virtual;
 end;

 PCalendarBox = ^TCalendarBox;
 TCalendarBox = object(TStringBox)
  function GetText(Item: Integer; MaxLen: Integer): String; virtual;
 end;

 PFRAMEDlg = ^TFRAMEDlg;
```

```
  TFRAMEDlg = object(TDialog)
    procedure HandleEvent(var Event: TEvent); virtual;
  end;

  PSwapDlg = ^TSwapDlg;
  TSwapDlg = object(TDialog)
    procedure HandleEvent(var Event: TEvent); virtual;
  end;

  PFRAMEWindow = ^TFRAMEWindow;
  TFRAMEWindow = object(TWindow)
    constructor Init(WTitle: string;
                     Bounds: TRect);
    procedure MakeInterior(Bounds: TRect);
    procedure close; virtual;
    function GetWindowNumber: Integer;
    procedure PutWindowNumber;
  end;

  PFRAMEInterior = ^TFRAMEInterior;
  TFRAMEInterior = object(TScroller)
    LineCount : Integer;
    Lines : array[0..MaxLines - 1] of PString;
    constructor Init(var Bounds: TRect;
                     AHScrollBar, AVScrollBar: PScrollBar);
    destructor Done; Virtual;
    procedure Draw; virtual;
  end;

  PHintLine = ^THintLine;
  THintLine = object(TStatusLine)
    function Hint(AHelpCtx: Word): String; virtual;
  end;


const

  bxLeft = 0;
  bxRight = 1;
  bxSingle = 2;


  RForceInputLine: TStreamRec = (
    ObjType: 15000;
    VmtLink: Ofs(TypeOf(TForceInputLine)^);
    Load:   @TForceInputLine.Load;
    Store:  @TForceInputLine.Store
  );

  ROKButton: TStreamRec = (
    ObjType: 15002;
    VmtLink: Ofs(TypeOf(TOKButton)^);
    Load:   @TOKButton.Load;
    Store:  @TOKButton.Store
  );
```

```
  RCancelButton: TStreamRec = (
    ObjType: 15003;
    VmtLink: Ofs(TypeOf(TCancelButton)^);
    Load:    @TCancelButton.Load;
    Store:   @TCancelButton.Store
  );

  RHelpButton: TStreamRec = (
    ObjType: 15004;
    VmtLink: Ofs(TypeOf(THelpButton)^);
    Load:    @THelpButton.Load;
    Store:   @THelpButton.Store
  );

  RStringBox: TStreamRec = (
    ObjType: 15005;
    VmtLink: Ofs(TypeOf(TStringBox)^);
    Load:    @TStringBox.Load;
    Store:   @TStringBox.Store
  );

  RCalendarBox: TStreamRec = (
    ObjType: 15006;
    VmtLink: Ofs(TypeOf(TCalendarBox)^);
    Load:    @TCalendarBox.Load;
    Store:   @TCalendarBox.Store
  );

  RFRAMEDlg: TStreamRec = (
    ObjType: 15007;
    VmtLink: Ofs(TypeOf(TFRAMEDlg)^);
    Load:    @TFRAMEDlg.Load;
    Store:   @TFRAMEDlg.Store
  );

  RSwapDlg: TStreamRec = (
    ObjType: 15008;
    VmtLink: Ofs(TypeOf(TSwapDlg)^);
    Load:    @TSwapDlg.Load;
    Store:   @TSwapDlg.Store
  );

  RHintLine: TStreamRec = (
    ObjType: 993;
    VmtLink: Ofs(TypeOf(THintLine)^);
    Load: @THintLine.Load;
    Store: @THintLine.Store
  );

procedure RegisterTools;
function TidyNumStr(S: String): String;

implementation

{*** TForceInputLine ***}
```

```
function TForceInputLine.Valid(Command: Word): Boolean;
var
  Ok: Boolean;
begin
  Ok := True;
  if (Command <> cmCancel) and (Command <> cmValid) then
  begin
    if Data^ = '' then
    begin
      Select;
      MessageBox('Field or fields not completed.', nil, mfError + mfOkButton);
      Ok := False
    end
  end;
  if Ok then Valid := TInputLine.Valid(Command) else Valid := False
end;


{*** TOKButton ***}

constructor TOKButton.Init(var Bounds: TRect);
begin
  Inherited Init(Bounds, '~O~k', cmOK, bfDefault);
  HelpCtx := hcOKButton
end;


{*** TCancelButton ***}

constructor TCancelButton.Init(var Bounds: TRect);
begin
  Inherited Init(Bounds, '~C~ancel', cmCancel, bfNormal);
  HelpCtx := hcCancelButton
end;


{*** THelpButton ***}

constructor THelpButton.Init(var Bounds: TRect);
begin
  Inherited Init(Bounds, '~H~elp', cmHelp, bfNormal);
  HelpCtx := hcHelpButton
end;


{*** TStringBox ***}

constructor TStringBox.Init(var Bounds: TRect; ANumCols: Word;
                    AScrollBar: PScrollBar; LeftOrRight: Word);
begin
  inherited Init(Bounds, ANumCols, AScrollBar);
  Case LeftOrRight of
    bxLeft : begin
               Left := True;
               Right := False
             end;
```

```
   bxRight : begin
             Left := False;
             Right := True
           end;
   bxSingle : begin
              Left := False;
              Right := False
            end
 end
end;

constructor TStringBox.Load(var S: TStream);
begin
 inherited Load(S);
 S.Read(Left, SizeOf(Boolean));
 S.Read(Right, SizeOf(Boolean))
end;

procedure TStringBox.Store(var S: TStream);
begin
 inherited Store(S);
 S.Write(Left, SizeOf(Boolean));
 S.Write(Right, SizeOf(Boolean))
end;

procedure TStringBox.HandleEvent(var Event: TEvent);
begin
 if List^.Count<>0 then
   if ((Event.What = evMouseDown) and (Event.Double)) then
   begin
     SelectItem(Focused);
     Event.What := evCommand;
     if Left then Event.Command := cmLeft
     else
     begin
       if Right then Event.Command := cmRight
       else Event.Command := cmOK
     end;
     PutEvent(Event);
     ClearEvent(Event)
   end
  else TListBox.HandleEvent(Event)
end;


{*** TCalendarBox ***}

function TCalendarBox.GetText(Item: Integer; MaxLen: Integer): String;

var D: PDate;

begin
   D:=List^.At(Item);
   GetText:=NewStr(D^.Date)^;
   {D^.Done}
end;
```

```
{*** TFRAMEDlg ***}

procedure TFRAMEDlg.HandleEvent(var Event: TEvent);
begin
  if (Event.What = evCommand) then
    Case Event.Command of
      cmSetDatabase, cmMakeCalendar,
      cmCalendarComplete,
      cmAllocateFeeds,
      cmCullAnimal, cmAddAnimal, cmViewAnimals,
      cmResetSim, cmNext : EndModal(Event.Command)
    end;
  TDialog.HandleEvent(Event)
end;


{*** TSwapDlg ***}

procedure TSwapDlg.HandleEvent(var Event: TEvent);
begin
  if (Event.What = evCommand) then
    Case Event.Command of
      cmRight, cmLeft: EndModal(Event.Command)
    end;
  TDialog.HandleEvent(Event)
end;


{*** TFRAMEWindow ***}

constructor TFRAMEWindow.Init(WTitle: string; Bounds: TRect);
Var R: TRect;
begin
  R:=Bounds;
{  DeskTop^.GetExtent(R);
  R.Assign(numWindows, numWindows, R.B.X, R.B.Y);  }
  inc(numWindows);
  TWindow.Init(R, WTitle, GetWindowNumber);
  MakeInterior(R);
  Options:=Options or ofTileable;
  if numWindows >= maxWindows then
    DisableCommands([cmOpenWin])
end;

procedure TFRAMEWindow.MakeInterior(Bounds: TRect);
Var
  HScrollBar, VScrollBar: PScrollBar;
  Interior: PFRAMEInterior;
  R: TRect;
begin
  VscrollBar:= StandardScrollBar(sbVertical + sbHandleKeyBoard);
  HscrollBar:= StandardScrollBar(sbHorizontal + sbHandleKeyBoard);
  GetExtent(Bounds);
  Bounds.Grow(-1, -1);
```

```
  Interior:= new(PFRAMEInterior, Init(Bounds, HScrollBar, VScrollBar));
  Insert(Interior)
end;

procedure TFRAMEWindow.Close;
begin
  TWindow.Close;
  PutWindowNumber;
  dec(numWindows);
  EnableCommands([cmOpenWin])
end;

function TFRAMEWindow.GetWindowNumber: Integer;
Var I: Integer;
 begin
  for I := 1 to maxWindows do
    if not (I in windowNums) then
    begin
      GetWindowNumber := I;
      windowNums := windowNums + [I];
      exit
    end;
  GetWindowNumber := wnNoNumber
end;

procedure TFRAMEWindow.PutWindowNumber;
begin
  if (1 <= Number) and (Number <= maxWindows) then
    windowNums := windowNums - [Number]
end;


{*** TFRAMEInterior ***}

constructor TFRAMEInterior.Init(var Bounds: TRect;
                          AHScrollBar, AVScrollBar: PScrollBar);
 function ReadText : Integer;
 Var
   F : Text;
   S : String;
 begin
   LineCount := 0;
   Assign(F, WindowFN);
   Reset(F);
   while not Eof(F) and (LineCount < MaxLines) do
   begin
     ReadIn(F, S);
     Lines[LineCount] := NewStr(S);
     inc(LineCount)
   end;
   Close(F);
   Erase(F);
   Readtext := LineCount
 end;

begin
```

```
  TScroller.Init(Bounds, AHScrollBar, AVScrollBar);
  GrowMode:= gfGrowHiX + gfGrowHiY;
  SetLimit(128, ReadText);      {*** Limit should reflect ***}
                                {*** items in collection ***}
end;

destructor TFRAMEInterior.Done;

  procedure FlushLines;
  Var I : Integer;
  begin
    for I:=0 to LineCount-1 do
      if Lines[I]<> nil then DisposeStr(Lines[I])
  end;

begin
  TView.Done;
  FlushLines
end;


procedure TFRAMEInterior.Draw;
Var
  Color : Byte;
  Y, I : Integer;
  B : TDrawBuffer;
begin
  Color := GetColor(1);
  for Y := 0 to Size.Y - 1 do
  begin
    MoveChar(B, ' ', Color, Size.X);
    I := Delta.Y + Y;
    if (I < LineCount) and (Lines[I] <> nil) then
      MoveStr(B, Copy(Lines[I]^, Delta.X + 1, Size.X), Color);
      WriteLine(0, Y, Size.X, 1, B)
  end
end;


{*** THintLine ***}

function THintLine.Hint(AHelpCtx: Word): String;
begin
  case AHelpCtx of

    {--- General ---}
    hcDebug:              Hint := 'Debug commands (to test new features)';
      hcTestNew:          Hint := 'Test newly implemented feature';
    hcStringBox:          Hint := 'Double click mouse or press spacebar and OK button to make
selection';

    {--- Database ---}
    hcDatabase:           Hint := 'Describe the compositions of feeds that can be used in
simulations';
      hcAddFeed:          Hint := 'Describe the composition of a new feed and add it to the
database';
```

```
      hcFeedNameIl:       Hint := 'Enter the name of the feed (duplicates not allowed)';
      hcFeedTypeRb:       Hint := 'Select the type which describes the feed most closely';
      hcDryMatterIl:      Hint := 'Enter the dry matter content in g/kg';
      hcCrudeProteinIl:   Hint := 'Enter the crude protein content in g/kg of dry matter';
      hcCrudeFibreIl:     Hint := 'Enter the crude fibre content in g/kg of dry matter';
      hcEtherExtractIl:   Hint := 'Enter the ether extract content in g/kg of dry matter';
      hcMetEnergyIl:      Hint := 'Enter the metabolisable energy content in MJ/kg of dry matter';
    hcViewEditFeed:       Hint := 'View or edit the composition of a feed in the current database';
    hcDeleteFeed:         Hint := 'Delete a feed description from the current database';
    hcLoadDatabase:       Hint := 'Locate and load an existing feed composition database';
     hcLoadDatabaseDlg:   Hint := 'Select or enter name of feed composition database to load';
    hcStoreDatabase:      Hint := 'Store the current feed composition database';
    hcStoreAs:           Hint := 'Store the current feed composition database under a new name';
     hcStoreDatabaseDlg:  Hint := 'Enter name of feed compostion database to store';
    hcClearDatabase:      Hint := 'Clear the current feed composition database from memory';

  {--- File ---}
  hcFile:               Hint := 'File management commands (open and save simulation files etc.)';
    hcOpen:             Hint := 'Locate and open a file containing an existing simulation';
     hcOpenDlg:         Hint := 'Select or enter name of simulation file to open';
    hcSave:             Hint := 'Save the current simulation';
     hcSaveAs:          Hint := 'Save the current simulation under a new name';
     hcSaveDlg:         Hint := 'Enter name of simulation file to save';
    hcChangeDir:        Hint := 'Select a new default directory';
     hcChangeDirDlg:    Hint := 'Enter drive and/or directory path';
    hcDosShell:         Hint := 'Temporary exit to DOS command line';
    hcExit:             Hint := 'Exit FRAME';

  {--- Simulation ---}
  hcSimulation:         Hint := 'Setup, run and adjust FRAME simulations';
    hcNewBasicInfo:     Hint := 'Enter the basic data required to setup a new simulation';
     hcBasicInfoDlg:    Hint := 'Use tab key or mouse to complete the basic simulation
description';
      hcStartDateIl:    Hint := 'Enter the date of the first observation';
      hcSimTitleIl:     Hint := 'Enter a descriptive title for the simulation';
      hcMakeCalendarButton:Hint := 'Enter the dates on which observations were made';
      hcNewCalendarDlg:  Hint := 'Enter the date of the observation following that currently
displayed';
      hcEditCalendarDlg: Hint := 'Edit the date of the observation currently displayed';
      hcNextDate:       Hint := 'Enter the date of another observation';
      hcDatesComplete:   Hint := 'Finish entering observation dates';
      hcSetDatabaseButton: Hint := 'Select the feed composition database to use';
      hcSetDatabaseDlg:  Hint := 'Select or enter the name of the database to use for the
simulation';
      hcSimNotes:       Hint := 'Enter optional descriptive notes';
    hcEditBasicInfo:    Hint := 'Edit the basic, setup data for the current simulation';
    hcClearSimulation:   Hint := 'Clear the current simulation from memory';
    hcRunSimulation:    Hint := 'Run a FRAME simulation based on the current setup data (+
options)';
      hcRunSimulationDlg: Hint := 'Use controls to update herd structure and allocate feeds';
       hcCullAnimal:    Hint := 'Remove the currently selected animal from the herd';
       hcAddAnimal:     Hint := 'Describe a new animal and add it to the current herd';
       hcAllocateFeeds: Hint := 'Offer the selected feed to the selected animal';
       hcConserveFeeds: Hint := 'Conserve the selected feed for alloacation at a later
observation';
       hcRun:           Hint := 'Simulate performance based on current allocation';
```

```
        hcReset:          Hint := 'Abandon changes and retrieve the original data for this
observation';
        hcViewAnimals:    Hint := 'View the current state of the herd';
        hcNext:           Hint := 'Accept simulated results and move to next observation date';
      hcSummary:          Hint := '';
       hcViewProduction:  Hint := '';

    {--- Feeds ---}
    hcFeeds:              Hint := 'Describe the availability of feeds for the current simulation';
      hcNewAvailability:  Hint := 'Enter data describing the pattern of feed availability';
       hcAvailabilityDlg: Hint := 'Double click mouse or highlight and use switch buttons to select
feeds';
       hcQuantityDlg:     Hint := 'Enter quantity of feed available on current date';
      hcEditAvailability: Hint := 'Edit the data describing the pattern of feed availability';

    {--- Herd ---}
    hcHerd:               Hint := 'Describe the initial state of the herd for the current simulation';
      hcNewHerd:          Hint := 'Complete animal descriptions for a new, initial herd structure';
       hcAnimalDlg:       Hint := 'Use tab key or mouse to complete the animal description';
      hcAddToHerd:        Hint := 'Describe a new animal and add it to the initial herd';
      hcEditAnimal:       Hint := 'Edit the description of an animal in the initial herd';
      hcDeleteAnimal:     Hint := 'Remove an animal from the initial herd';

     {--- Configure ---}
    hcConfigure:          Hint := 'Configure the FRAME working environment';
      hcChangeScrColour:  Hint := 'Change screen colour scheme';
       hcChangeScrColourDlg: Hint := 'Use cursor keys or mouse to select ' +
                          'colour scheme';

    {--- Help ---}
    hcHelp:               Hint := 'Access FRAME''s online help system';
      hcAbout:            Hint := 'Show version and copyright information';
       hcAboutDlg:        Hint := 'Press ENTER to continue';

    {--- Buttons ---}
    hcOKButton:           Hint := 'Accept the settings in this dialog box';
    hcCancelButton:       Hint := 'Close the dialog box without making any changes';
    hcHelpButton:         Hint := 'View a help screen about this dialog box';

  else
    Hint := '';
  end
end;


procedure RegisterTools;
begin
  RegisterType(RForceInputLine);
  RegisterType(ROKButton);
  RegisterType(RCancelButton);
  RegisterType(RHelpButton);
  RegisterType(RStringBox);
  RegisterType(RCalendarBox);
  RegisterType(RFRAMEDlg);
  RegisterType(RSwapDlg);
  RegisterType(RHintLine);
```

```
end;

procedure FillString(S: Word; var F: Text);
Var
  Spaces : Word;
begin
  for Spaces:=1 to S do write(F, ' ')
end;

function TidyNumStr(S: String): String;
Var
  C, CTS: Word;
  DecSpotted, NumSpotted : Boolean;
  TempStr : String;
  Num : Real;
  NumDecs, NumChars : Integer;
  Error : Integer;
begin
  NumSpotted := false;
  DecSpotted := false;
  NumDecs := 0;
  NumChars := 0;
  CTS := 1;
  for C := Length(S) downto 1 do
  begin
    DecSpotted := S[C] = '.';
    if not NumSpotted then
      NumSpotted := (S[C] in NumSet) and (S[C] <> '0') and (S[C] <> '.');
    if DecSpotted then
    begin
      if NumSpotted then NumDecs := Length(S) - C;
      DecSpotted := False;
    end;
    if (NumSpotted) and (S[C] <> ' ') then NumChars := NumChars + 1;
  end;
  Val(S, Num, Error);
  Str(Num:CTS-1:NumDecs, TempStr);
  TidyNumStr := TempStr
end;

END.
```

APPENDIX 3: INPUT DATA FOR TEST SIMULATION

The complete input data for the sample FRAME simulation presented in the report are shown in the following three tables.

Table A1: Chemical compositions of available feeds (C.B Bista, 6 March - 25 July, 1994)

Table A2: Availability of feeds during the simulation period feeds (C.B. Bista, 6 March - 25 July, 1994)

Table A3: Feed allocations during the simulation period C.B. Bista, 6 March - 25 July, 1994)