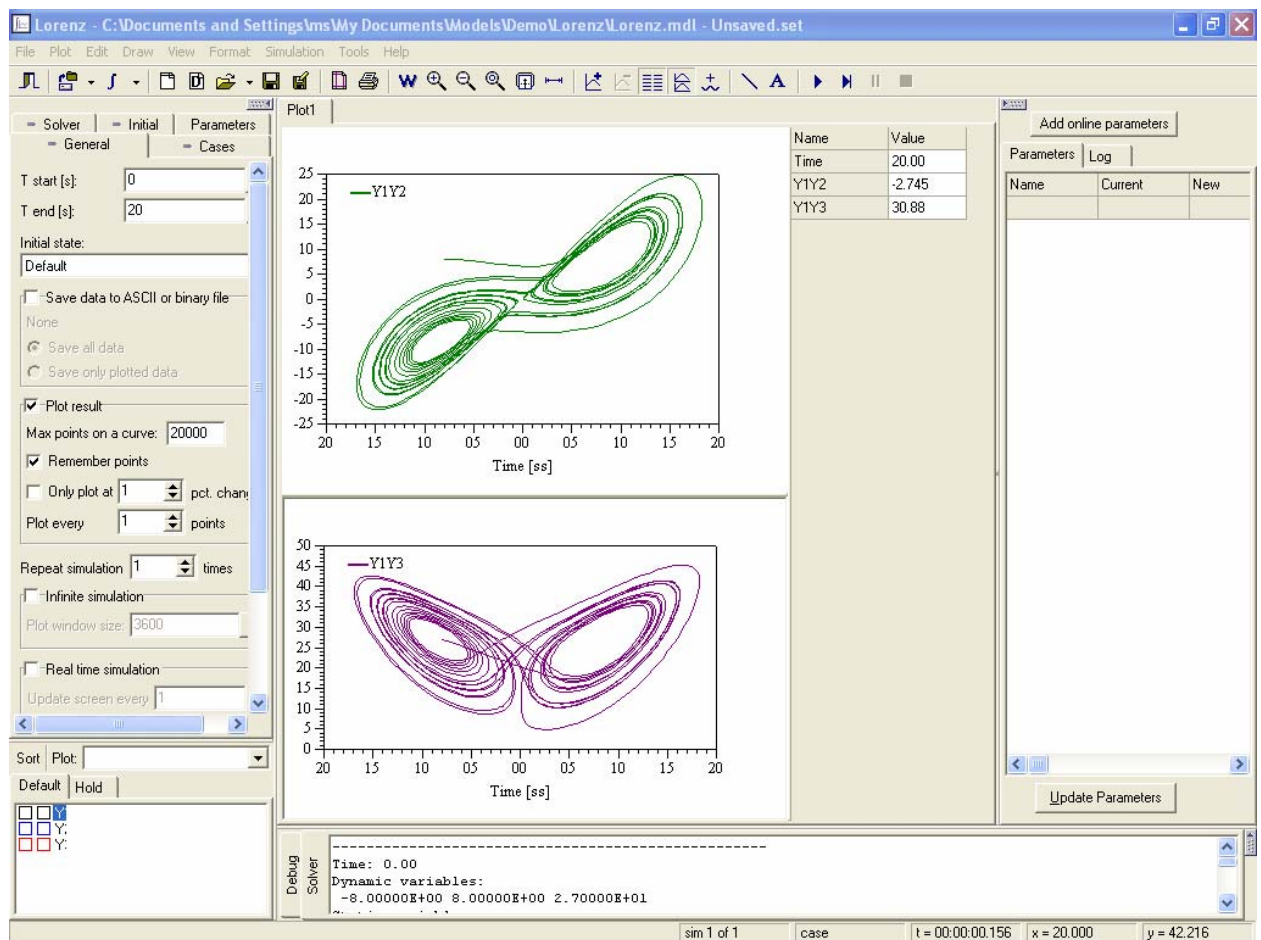




WinDali

A Modeling and Simulation System for
Microsoft Windows
Version 3.00

MORTEN JUEL SKOVRUP



Contents

1 Disclaimer	1
1.1 Contact information	1
2 Version information	3
2.1 Changes in version 3.00	3
2.2 Changes in version 2.20	3
2.3 Changes in version 2.10	4
3 Introduction	5
3.1 Typing convention	6
3.2 Terms used in this document	7
4 System structure	9
5 Creating a simple model	11
5.1 SetupProblem.....	15
5.2 ModelEquations.....	16
5.3 EndCalc.....	16
5.4 Compiling	18
5.5 Simulation.....	18
6 Model file format.....	21
6.1 Common parameters and datatypes.....	23
6.2 SetupProblem.....	23
6.2.1 SetupModel	24
6.2.2 SetupState	24
6.2.3 SetTimeFactor.....	24
6.2.4 SolverSettings	24
6.2.5 Dynamic variables	25
6.2.6 States	26
6.2.7 Static variables	26
6.2.8 Parameter pages.....	28
6.2.9 Initial Parameters	28
6.2.10 Floating point parameters	28
6.2.11 Integer parameters.....	30
6.2.12 Boolean parameters.....	31
6.2.13 List parameters	32
6.2.14 Enumerated parameters	33
6.2.15 Enumerated choice parameters	34
6.2.16 Explicit variables	36
6.2.17 Action buttons	36
6.2.18 Info Labels.....	37
6.2.19 HideSampleTime.....	37
6.2.20 Model help file	38
6.3 PreCalc.....	38
6.3.1 SetStartState	38
6.3.2 AddExplicitVar	39
6.3.3 SetSampleTime.....	39
6.4 ModelEquations.....	39
6.5 StateShift.....	41

6.6 OnStateChange	42
6.7 OnSolution	42
6.8 OnSample	42
6.9 EndCalc	44
6.10 OnQuit	44
6.11 OnUIValueChange.....	45
6.11.1 Running simulations from the model	47
6.12 OnSaveSettings.....	48
6.13 OnLoadSettings	48
6.14 Using Initial parameters.....	49
6.14.1 SetInitial.....	51
6.14.2 SetGuess.....	52
6.14.3 AddDynVar	52
6.14.4 AddStatVar	53
6.15 Mathematical text.....	54
6.16 Debugging	55
7 Common problems	56
8 Using refrigerant equations	57
9 WinDali Model Editor	59
9.1 Compiler Options.....	61
9.2 Environment Options	64
10 WinDali Simulation Interface.....	67
10.1 Menu commands	70
10.2 Online parameters	72
10.3 Varying parameters	72
10.4 Dali solver	76
11 Using Profiles in models	77
11.1 Generating profiles	77
11.2 Using profiles in a model	79
12 Using Post Process	81
13 Distributing models	83
14 References.....	85
Appendix A Used file types	87
Appendix B Files and directories created during installation.....	89

1 Disclaimer

This software is provided “as is” and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall Institute for Product Development or any person mentioned in this document be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

1.1 Contact information

Morten Juel Skovrup
Senior Engineer, Ph.D.

Institute for Product Development
Refrigeration and Energy Engineering
The Technical University of Denmark (DTU)
Nils Koppels Allé, Building 402
DK-2800 Kgs. Lyngby
Denmark

Phone: +45 45 25 41 20
Fax: +45 45 93 52 15
e-mail: mjs@ipu.dk
Web: <http://www.ipu.dk>

2 Version information

2.1 Changes in version 3.00

The following changes and some bug-fixes have been made:

- The software is no longer freeware
- FreePascal editor has changed name to WinDali Model Editor, and a lot of changes have been made so that it is more a model editor than a general programming editor
- The Simulation program has changed name to WinDali Simulation Interface
- Features regarding time and plotting in the WinDali Simulation Interface have been changed/added – see the section for the WinDali Simulation Interface
- Added a lot of (simplifying) changes to the model structure.
 - All extended parameter calls (for example `AddFloatParamExt`) does not take `IndexOnPage` as parameter any more.
 - Extended calls to `AddDynamic`, `AddStatic` etc. does not take `format`, `precision` and `digit` parameters any more.
- The component modeling interface is no longer supported.
- All datatypes have been changed to allow for change in floating point precision in the model (reserved for future versions). See chapter 6.1.
- Explicit variables have been renamed to Implicit variables
- Post Process and Profile Editor has been added.
- All programs have been prepared for installation on a multiuser-system.

2.2 Changes in version 2.20

The following changes and some bug-fixes have been made:

- The meaning of max-min values for static variables has been changed. They do not any longer affect only the user-interface – they are also used in the static-equation solver to limit the values of the static variables within the Newton-Raphson iterations.
- The function `AddStaticVarExt` has been added.
- Most `Add...` procedures have been changed to functions. They now return the number assigned to the variable/parameter added.
- Try to right-click on the curve window in the simulation program. You will see a menu, which enables you to easier select the variables you want to plot.
- In the simulation program on the General page a new option called “Remember points” has been added. If checked then all values of all variables in the curve window will be remembered – even if the variable is not selected (plotted). If “Remember points” is unchecked then only values of selected (plotted) variables will be stored.

2.3 Changes in version 2.10

The model file format has changed. This means that models build with previous versions have to be changed to make them work in version 2.10. Note that the changes do not affect the component file format, i.e. models created using components do not have to be changed.

The changes all have to do with specifying the number of variables and parameters. It is no longer required (in fact it is not possible) to specify the number of for example dynamic variables by calling `SetNumDynamic`. The number of dynamic variables are automatically counted when you call `AddDynamic`. The same apply for parameters.

More precisely the following functions are obsolete:

```
SetNumActionBtns(Num : TInteger);
SetNumBoolParams(Num : TInteger);
SetNumChoice(EnumParam, ItemIndex, Num : TInteger);
SetNumDynamic(Num : TInteger);
SetNumEnumChoiceParams(Num : TInteger);
SetNumEnumParams(Num : TInteger);
SetNumExtra(Num : TInteger);
SetNumFloatParams(Num : TInteger);
SetNumInfoLabels(Num : TInteger);
SetNumInitialParams(Num : TInteger);
SetNumIntParams(Num : TInteger);
SetNumListParams(Num : TInteger);
SetNumStatic(StateNum, NumStatic : TInteger);
```

And all the `Add...` functions have been changed, so that you no longer have to pass the variable number in the function call. For example:

```
AddFloatParam(Num : TInteger; var Parameter : TFloat; DefaultValue : TFloat;
               Name : PChar; ParamPage : TInteger);
```

Has been changed to:

```
AddFloatParam(var Parameter : TFloat; DefaultValue : TFloat;
               Name : PChar; ParamPage : TInteger);
```

See the details under the different functions in chapter 6.

3 Introduction

WinDali is a modeling and simulation system for Microsoft Windows™ 95, 98, ME, NT 4.0, XP or later. The basic features of the system are:

- Solves a system of semi-explicit differential algebraic equations (DAE's).
- Solves initial value problems.
- Handles discontinuities.
- Equation based modeling.
- Graphical simulation program.
- Creates distributable files (exe-files).
- Models can be created in practical any programming language (Pascal, C++, Fortran...).
- A user-supplied solver may replace the accompanying equation solver.

These issues will be covered in depth in the following chapters.

Examples in this document are included in the `\All users\Documents\Models\Demo` directory.

WinDali consists of four programs:

1. **WinDali Model Editor**, in which the models are formulated – note that you can use Borland Delphi™, Microsoft Visual C++™ or another programming environment instead.
2. **WinDali Simulation Interface**, in which the models are loaded, and the simulation performed.
3. **Post Process**, for displaying saved binary data.
4. **Profile Editor**, for creating profiles to be used in models.

An important limitation in this version of WinDali is that it only handles semi-explicit DAE's. This means that WinDali only can solve problems on the form:

$$\begin{aligned}\frac{d y}{d t} &= f(t, x, y, p) \\ 0 &= g(t, x, y, p, s)\end{aligned}\tag{2.1}$$

The main reason for this limitation is that the solver that comes with this version has this limitation. A future release will include possibility to formulate implicit problems:

$$\begin{aligned}0 &= f\left(t, x, y, \frac{d y}{d t}, p\right) \\ 0 &= g(t, x, y, p, s)\end{aligned}\tag{2.2}$$

The notation used in equation (2.1) and (2.2), and the terms used in the rest of this document will be explained in the following.

3.1 Typing convention

This report contains some examples written in source-code. The examples are all written in the programming language Object Pascal [3]. When this is the case the following typeface is used:

Item	Example of typeface:
Source code	constructor TOneObject.Create; var AStr : string ; ANum : TInteger; begin AStr := 'This is a string - next is a number'; ANum := 10; end ;
Source code in text	This is an example of source code in normal text

3.2 Terms used in this document

Term	Explanation
Dynamic variables	Variables that appears differentiated (with respect to time) in the equations. The symbol y will be used for dynamic variables.
Static variables	Variables that do not appear differentiated in the equations. The symbol x will be used for static variables.
Independent variable	The variable that y is differentiated with respect to. Normally this equals time, and the symbol t is used for the independent variable.
Parameters	Quantities that are set to a constant before simulation. The symbol p will be used for parameters.
States	For example a valve may be in on of two states: Open or Closed. These logical states will normally change the equations describing the physical system. The number of logical states for a model is the number of different sets of equations used to define the model. The symbol s will be used for states.
Discontinuities	Discontinuities are a way to describe abrupt changes in the physical system that is modeled. For example, by describing the process of a valve suddenly closing as a discontinuity, one avoids describing in detail the valve position while it closes. A discontinuity indicates that the physical system shifts to another logic state. So describing abrupt changes as discontinuities involves a description of the possible states of the physical system. The conditions causing the change of logical state must also be formulated.
Initial value problems	A problem where the present state of the system is known, and the future state is to be determined. Problems that can be formulated as initial value problems can be solved by WinDali (note that initial value problems does not in general require that the independent variable is time).
DAE	Differential Algebraic Equation. An equation system that consists of both an ordinary differential equation and an algebraic equation, for example: $\frac{d y}{d t} = x$ $x = \sqrt{x + y}$
Solver	The numeric code that solves the system of DAE's. The solver integrates the differential equations, solves the algebraic equations and is handling discontinuities.

4 System structure

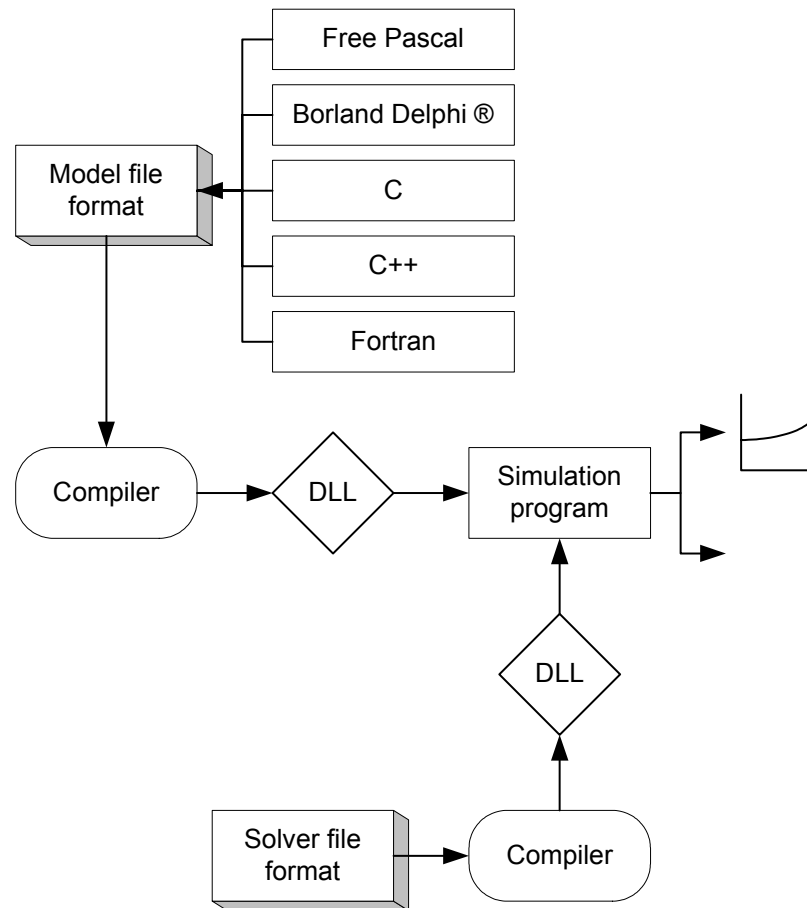


Figure 1. Structure of modeling system

Figure 1 shows the structure of the simulation system. The principle behind the structure of the simulation system is that all modules, or boxes in Figure 1, should be replaceable. The shadowed boxes in the structure are file formats or protocols that specify the interface between the other modules, i.e. when for example the structure of the Model file is fixed, third party programs could replace everything below and above the Model file box. Another important feature is that the solver is replaceable.

The boxes labeled Compiler are responsible for converting a file to some binary representation. In practice the boxes represents standard available compilers, capable of building Windows DLL's (Dynamic Link Libraries).

The different modules in Figure 1 have the following meaning:

Module	Explanation
Model file format	A file written in a standard programming language with a specified interface to the Simulation module.
Simulation program	A GUI application which is displaying the results of a simulation to the user as Graphs, animations, numbers, etc.
Solver	The numerical code which is solving the equations specified in the Model file.
Solver file format	A file format with a specified interface understood by the Simulation module.

In the current release of WinDali, the user starts at the model file level, specifying the model in a standard programming language. With the program comes a Pascal compiler, which is freeware, but other programming languages can be used.

5 Creating a simple model

This chapter describes the process of creating a simple model in the WinDali Model Editor. Only the basic features in the Model file format will be described, the details are left to chapter 6.

Suppose you want to cool a block of some material, as showed in Figure 2:

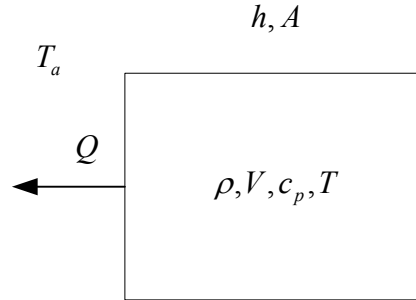


Figure 2. Cooling a block of some material

The block has density ρ , volume V , specific heat c_p and temperature T . The block is exposed to the ambient temperature T_a (constant) and has the surface area A . The heat transfer coefficient between the block and the surroundings is h (constant), and it is assumed that the block has a spatial uniform temperature at all times. This assumption is also known as the lumped capacitance method, and the error introduced by the assumption is small if the Biot number is less than 0.1.

The Biot number is defined as:

$$Bi = \frac{h L_c}{\lambda} \quad (4.1)$$

Where L_c is a characteristic length calculated as $L_c = V/A$ and λ is the conductivity of the material.

The energy balance for the problem is:

$$\rho V c_p \frac{dT}{dt} = h A (T_a - T) \quad (4.2)$$

For this problem there is one dynamic variable T and 7 parameters ρ, V, c_p, h, A, T_a and λ (we will also calculate the Biot number to evaluate the uniform temperature assumption)

To create the model file you have to go through the following steps:

1. Open WinDali Model Editor
2. Select File|New|New Model

This will bring up the following dialog:

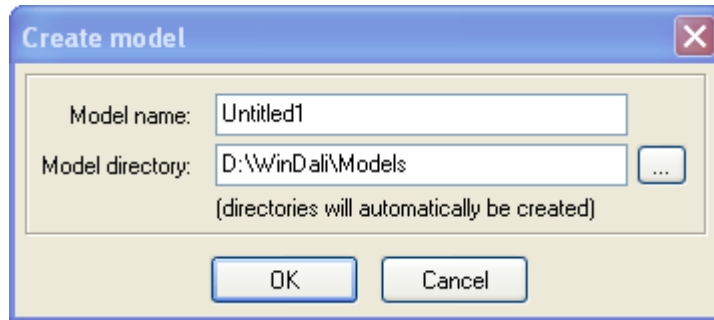


Figure 3. Create model dialog

Here you specify the name of your model and the directory where your files should be located. As default the program selects the Models directory, which is located in the directory where you installed WinDali. For now input “Test” as the model name, and add “\TestDir” to the default model directory:

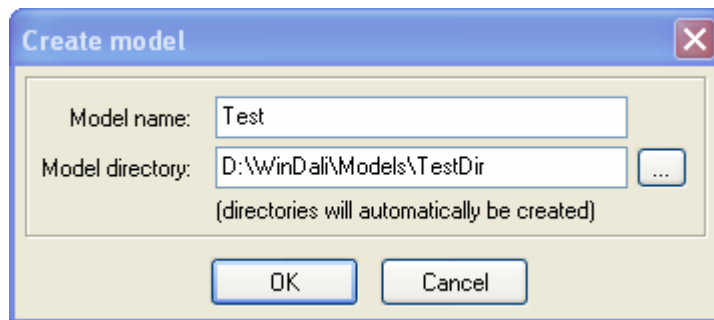


Figure 4. Create model dialog - continued

This will create a basic Model:

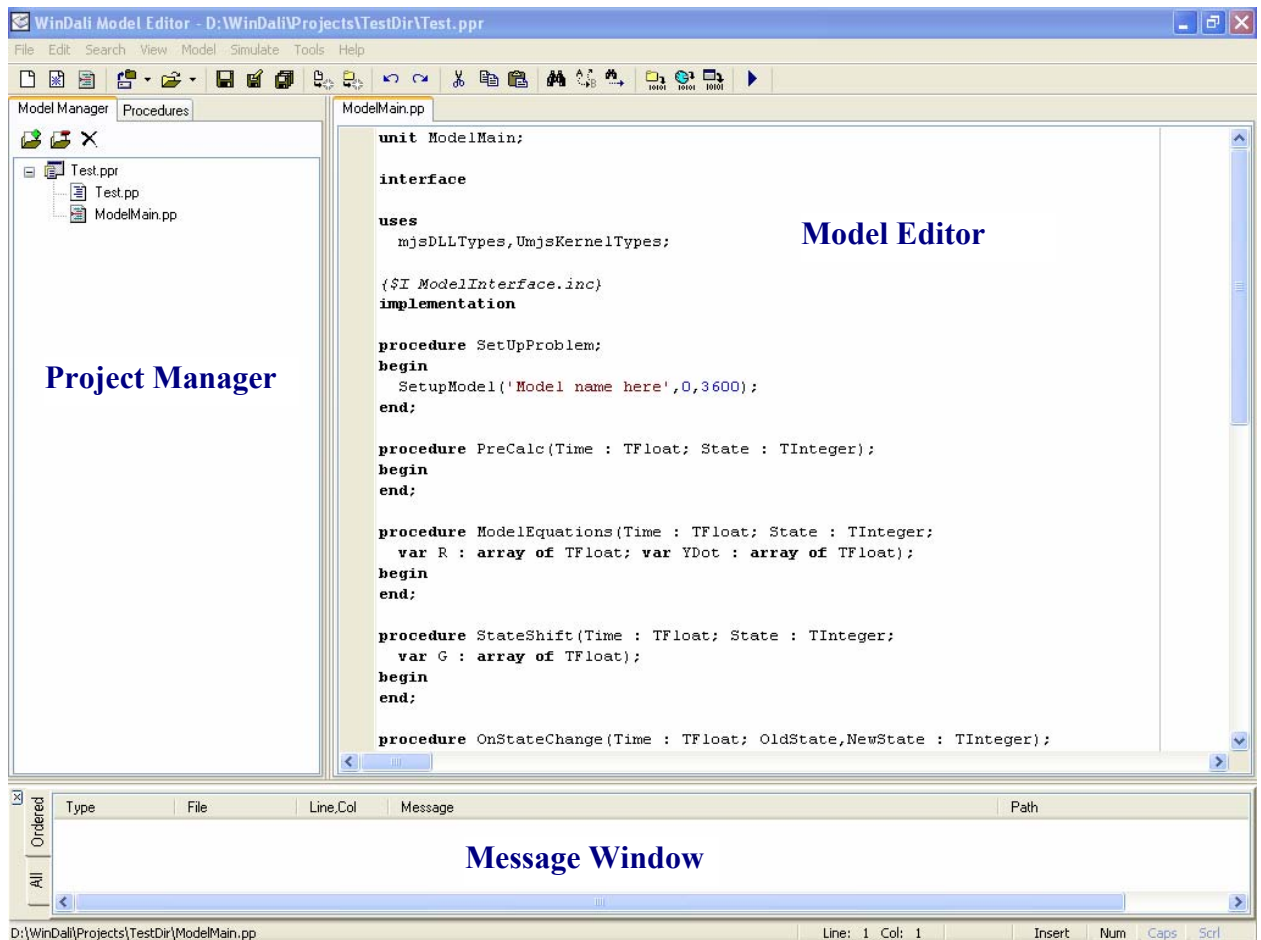


Figure 5. WinDali Model Editor

ModelMain.pp is the file where you specify your problem in the Model Editor. The file Test.pp (see the Model Manager) is the main file in your model, and should not be edited.

The ModelMain.pp file looks like this:

```
unit ModelMain;

interface

uses
  mjsDLLTypes, UmjsKernelTypes;

{$I ModelInterface.inc}
implementation

procedure SetupProblem;
begin
  SetupModel('Model name here', 0, 3600);
end;
procedure PreCalc(Time : TFloat; State : TInteger);
begin
end;
procedure ModelEquations(Time : TFloat; State : TInteger;
  var R : array of TFloat; var YDot : array of TFloat);
begin
end;
procedure StateShift(Time : TFloat; State : TInteger;
  var G : array of TFloat);
begin
end;
procedure OnStateChange(Time : TFloat; OldState, NewState : TInteger);
begin
end;
procedure OnSolution(Time : TFloat; State : TInteger);
begin
end;
procedure OnSample(Time : TFloat; var State : TInteger);
begin
end;
procedure EndCalc(Time : TFloat; State : TInteger);
begin
end;
procedure OnQuit;
begin
end;
procedure OnUIValueChange(UIType : TInteger; Num, State, Choice : TInteger;
  Value : TFloat);
begin
end;
procedure OnSaveSettings(FileName : PChar);
begin
end;
procedure OnLoadSettings(FileName : PChar);
begin
end;

end.
```

The task is now to fill out the 12 procedures defined in `ModelMain.pp`. The procedures and the functions you can call are described in detail in chapter 6. For the simple problem at hand you only have to worry about `SetupProblem`, `ModelEquations` and `EndCalc`.

First you have to define the variables and parameters so Free Pascal knows about them. This is done by writing the following right after **implementation**:

```
implementation
const
  Rho    = 8000;
  Cp     = 480;
  Lambda = 15;
  V      = 0.001;
  A      = 0.06;
  h      = 10;
  Ta     = 20;
var
  T : TFloat;
  Bi : TFloat;
```

5.1 SetupProblem

In the procedure **SetupProblem** you have to tell the system about the variables, parameters and states in your problem. This also specifies the user-interface of the Simulation Interface program.

You tell the system about your problem by calling a number of predefined functions. The ones you need to know about for this simple problem will be shortly explained (details are given in chapter 6).

```
SetupModel(Title : PChar; TStart,TEnd : TFloat);
```

This function specifies the title of your problem and the time you want to simulate. For this simple problem, let us say that you want to simulate 1000 seconds so you should call **SetupModel** like this:

```
SetupModel('Simple problem',0,1000);
```

```
AddDynamic(var Variable : TFloat; InitalValue : TFloat;
  Name,LongName : PChar);
```

This function is used to specify detailed information about each of the dynamic variables in the problem. The individual parameters are explained in detail in chapter 6. There is only one dynamic variable and lets say the initial value is 100 °C:

```
AddDynamic(T,100,'T','Temperature [°C]');
```

```
AddExplicit(var Variable : TFloat; Name : PChar; DoPlot : TBoolean);
```

Add variables you can calculate explicitly, but want to display the value of:

```
AddExplicit(Bi,'Biot number',False);
```

This completes the **SetupProblem** procedure.

5.2 ModelEquations

In **ModelEquations** you specify the equations. The heading looks like this:

```
procedure ModelEquations(Time : TFloat; State : TInteger;
  var R : array of TFloat; var YDot : array of TFloat);
```

This procedure is called every time the solver needs to do calculations on your model. The parameters in the procedure heading are:

- **Time** The current simulation time (in seconds)
- **State** The current state (as there is only one state in this example, this will always be equal to 1)
- **R** A zero-based vector where you should return a residual for each of the static equations in your model. As there are no static equations in this model, **R** can be ignored.
- **YDot** A zero-based vector where you should return the derivatives of the dynamic variables in your model.

A zero-based vector means that the first place in the vector is indexed 0, the second place 1 and so on. For the simple model there is only 1 dynamic variable and no static variables. Note that because of the registration of the dynamic variable (calling **AddDynamic**), the Pascal variable **T** will always have the correct and updated values when **ModelEquations** is called.

If equation (4.2) is arranged so the derivative is isolated, the programming **ModelEquations** of is straightforward:

$$\frac{dT}{dt} = \frac{hA}{\rho V c_p} (T_a - T) \quad (4.3)$$

```
procedure ModelEquations(Time : TFloat; State : TInteger;
  var R : array of TFloat; var YDot : array of TFloat);
begin
  YDot[0] := h*A / (Rho*V*Cp) * (Ta-T);
end;
```

This concludes the **ModelEquations** procedure.

5.3 EndCalc

EndCalc is called once at the end of the simulation. The only thing left to be calculated is the Biot number:

```
procedure EndCalc(Time : TFloat; State : TInteger);
begin
  Bi := h*V / (A*Lambda);
end;
```

In full the file ModelMain.pp looks like this:

```

unit ModelMain;

interface

uses
    mjsDLLTypes, UmjsKernelTypes;

    {$I ModelInterface.inc}
implementation
const
    Rho = 8000; Cp = 480; Lambda = 15; V = 0.001; A = 0.06; h = 10; Ta = 20;
var
    T, Bi : TFloat;

procedure SetupProblem;
begin
    SetupModel('Simple problem', 0, 1000);
    AddDynamic(T, 100, 'T', 'Temperature [°C]');
    AddExplicit(Bi, 'Biot number', False);
end;
procedure PreCalc(Time : TFloat; State : TInteger);
begin
end;
procedure ModelEquations(Time : TFloat; State : TInteger;
    var R : array of TFloat; var YDot : array of TFloat);
begin
    YDot[0] := h*A/(Rho*V*Cp)*(Ta-T);
end;
procedure StateShift(Time : TFloat; State : TInteger;
    var G : array of TFloat);
begin
end;
procedure OnStateChange(Time : TFloat; OldState, NewState : TInteger);
begin
end;
procedure OnSolution(Time : TFloat; State : TInteger);
begin
end;
procedure OnSample(Time : TFloat; var State : TInteger);
begin
end;
procedure EndCalc(Time : TFloat; State : TInteger);
begin
    Bi := h*V/(A*Lambda);
end;
procedure OnQuit;
begin
end;
procedure OnUIValueChange(UIType : TInteger; Num, State, Choice : TInteger;
    Value : TFloat);
begin
end;
procedure OnSaveSettings(FileName : PChar); begin end;
procedure OnLoadSettings(FileName : PChar); begin end;
end.

```

5.4 Compiling

Now it is time to compile the model. Select the Model|Compile menu and note if there is any error messages in the Message window. If you have made an error you can double-click it in the Message window and the error will be highlighted in your code. If no errors occurred you can run a simulation by selecting the Simulate|Run menu. The resulting model file will have the extension “.mdl”.

5.5 Simulation

When you select the Simulate|Run menu in the Model Editor you start the simulation program. If the model file was created exactly as described above, you will see the following screen:

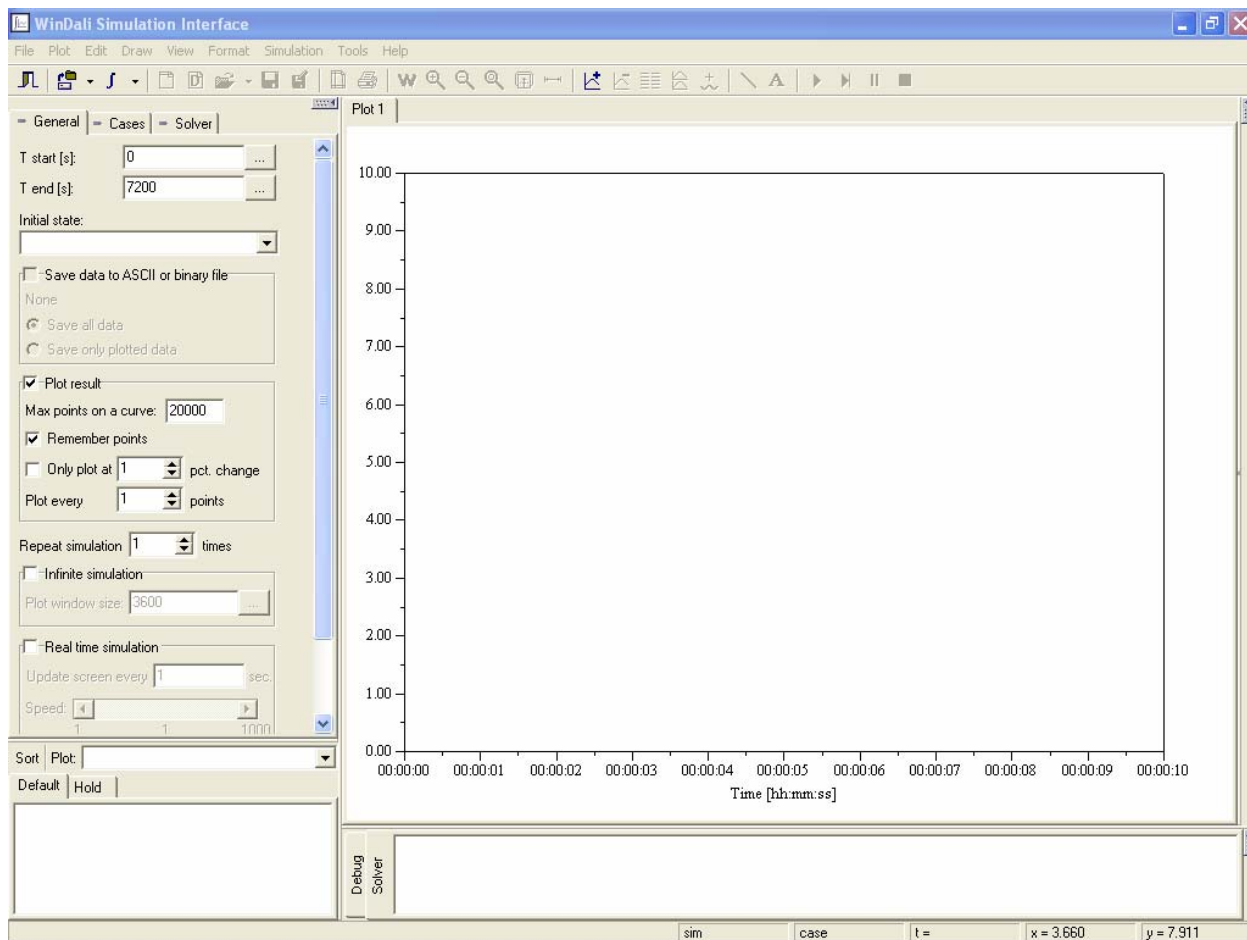


Figure 6. The Simulation Interface program

In the General page the simulation time can be set by the user (i.e. the values specified in the call to `SolverSettings` was only default values).

The rest of the values in the General, Cases and Solver pages will be explained in chapter 10.

To start the simulation, select the Simulation|Start menu or press the <Play> button on the toolbar. In either case the result should look like this:

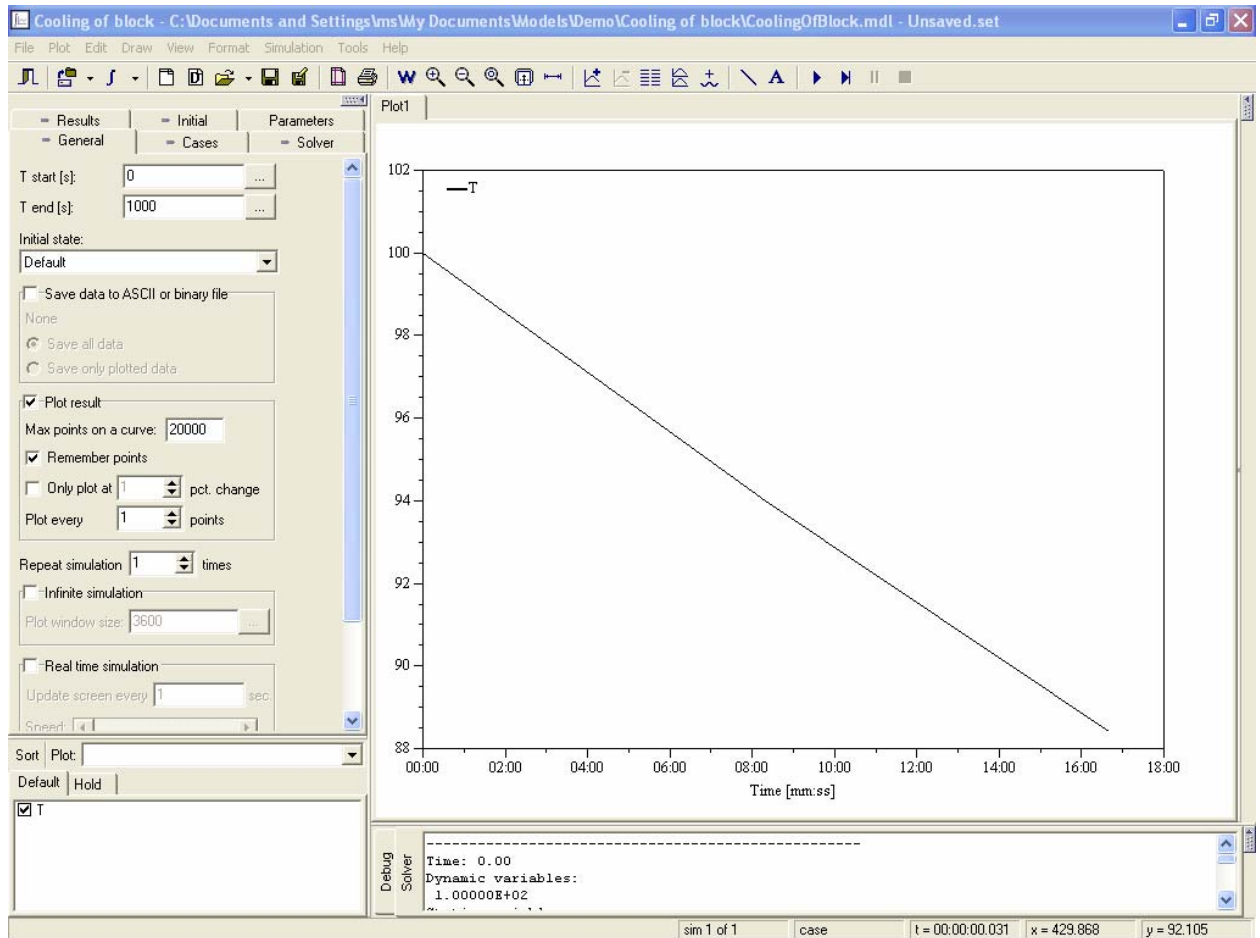


Figure 7. Simulation result.

This shows that the simulation time was too short. Try increasing the simulation end-time to for example 1 day (86400 seconds), and run the simulation again.

If you want to change for example the density ρ to 1000, then you have to go back to the Model Editor, change ρ and recompile the model. Instead you could install ρ as a parameter in the model (together with the rest of the constants you defined in the model). This is done by changing the model to the following:

```

implementation
var
    T,Bi          : TFloat;
    Rho,Cp,Lambda,V,A,h,Ta : TFloat;

procedure SetupProblem;
begin
    SetupModel('Simple problem',0,1000);
    SetParamPages('Parameters');
    AddDynamic(T,100,'T','Temperature [°C]');
    AddExplicit(Bi,'Biot number',False);
    AddFloatParam(Rho,8000,'Density [kg/m^3]',1);
    AddFloatParam(Cp,480,'Specific heat [J/kg K]',1);
    AddFloatParam(Lambda,15,'Conductivity [W/m K]',1);
    AddFloatParam(V,0.001,'Volume [m^3]',1);
    AddFloatParam(A,0.06,'Surface area [m^2]',1);
    AddFloatParam(h,10,'Heat transfer coefficient [W/m^2 K]',1);
    AddFloatParam(Ta,20,'Ambient temperature [°C]',1);
end;

```

Note that now **Rho,Cp** etc. are defined as Pascal variables instead of constants, and that a call to **SetParamPages** has been added.

When you compile the model and run it, you will see the following page in the Simulation Interface program:

General	Cases	Solver
Results	Initial	Parameters
Density [kg/m ³]:		8000
Specific heat [J/kg K]:		480
Conductivity [W/m K]:		15
Volume [m ³]:		0.001
Surface area [m ²]:		0.06
Heat transfer coefficient [W/m ² K]:		10
Ambient temperature [°C]:		20

Now you can change the parameters in the Simulation Interface program, without having to recompile the model.

6 Model file format

A model file has 12 procedures, which the user should fill out to specify the problem to be solved. The following figure shows the sequence in which the procedures are called from the simulation program:

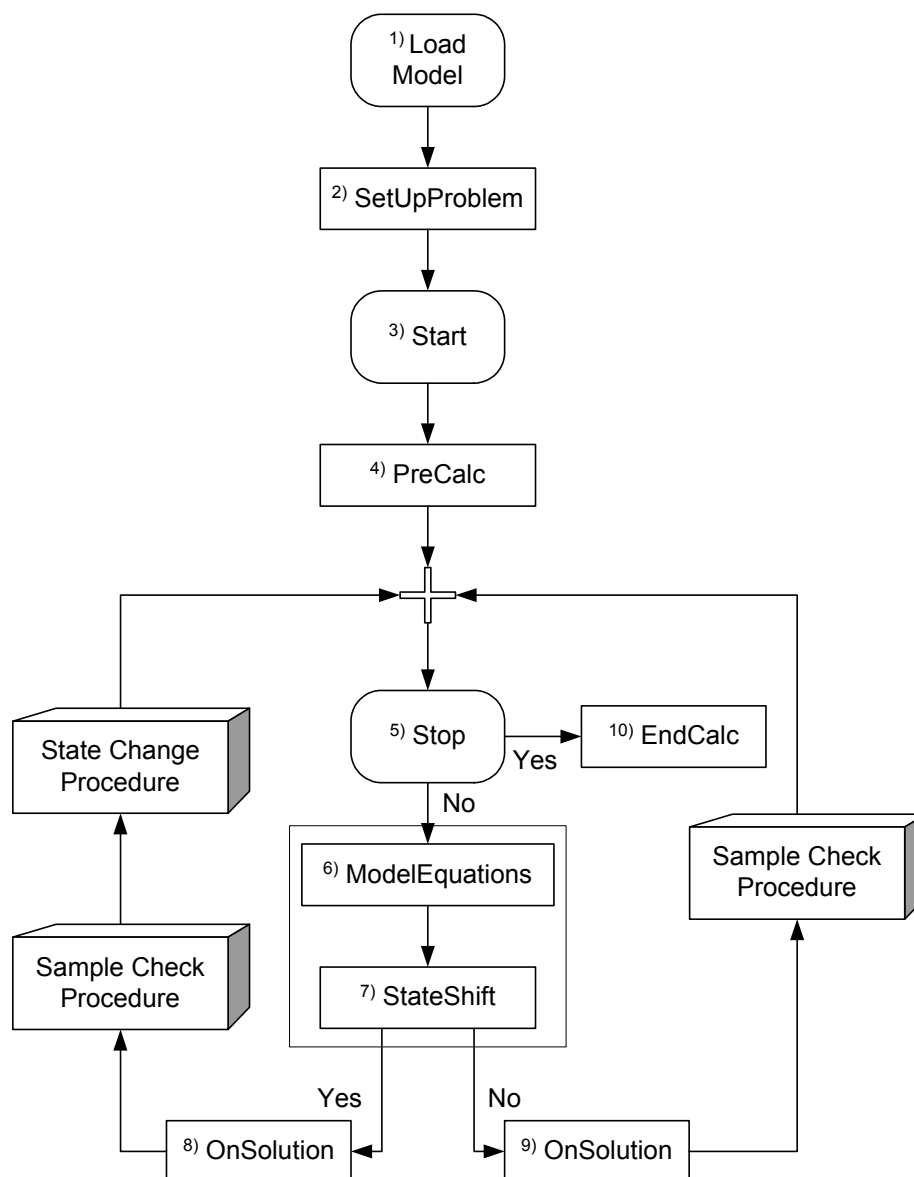


Figure 8. Calling sequence for the procedures in the model file.

The rounded rectangles in Figure 8 represent user-actions in the Simulation program. Note that the boxes 6 and 7 in general are called several times at each time step. This is because the solver iterates to find a solution. The procedure `OnQuit`, which is not represented in figure 8, is called when the user closes the simulation program or selects another model. The procedures `OnUIValueChange`, `OnSaveSettings` and `OnLoadSettings` all have to do with controlling the user-interface. They will be explained later.

The calling sequence with no state shift is straightforward (for now ignore the Sample Check Procedure regarding Sample time – it will be covered in chapter 6.8). But the calling sequence with a state shift needs some explanation.

The State Change Procedure looks like this:

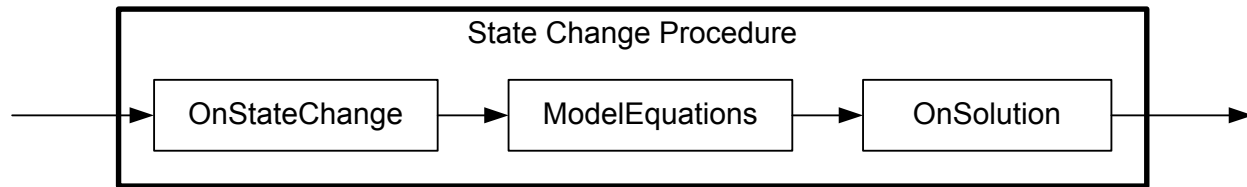


Figure 8a. State Change Procedure.

Lets say that we want to model a valve that is On when a temperature T is above 10°C , else it is Off. We want to record the temperature and the On-Off signal to the valve, and the initial temperature is 20°C . A plot of the temperature and the On-Off signal could look like this:

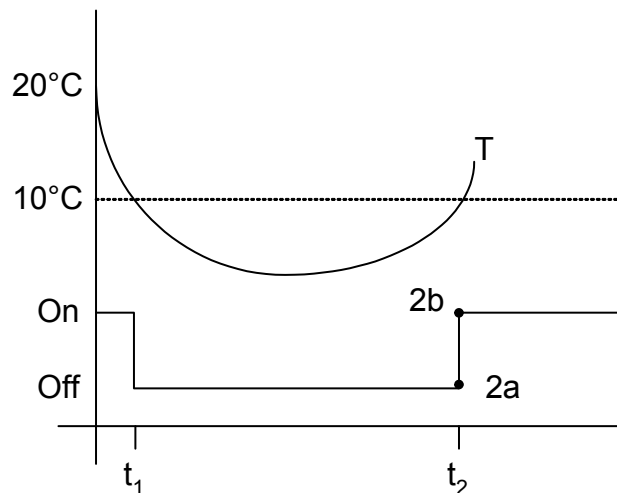


Figure 9. Temperature and On-Off signal.

When the solver reaches t_2 where the valve goes On, it first calls **OnSolution** (block 8) to inform the model file that a solution has been found. But at t_2 the equation system also shift state, which means that the solver has to start all over, and possibly with a new set of static equations.

The static equations have to be solved before the solver continues, and for this the solver needs guesses on the static variables.

In **SetupProblem** (as will be clear in chapter 6) default guesses on the static variables are given for each state, but if these guesses for some reason need to change, **OnStateChange** (see Figure 8a) is called. After **OnStateChange**, **ModelEquations** is called to solve the static equations in the new state, and before the solver continues, **OnSolution** is called to enable plotting of point 2b in Figure 9.

In the following the 12 procedures will be covered in detail.

Before any of the 12 procedures are filled out, all the variables and parameters in the model should be declared in standard Pascal fashion (see chapter 5). All Pascal floating-point variables should be in **TFloat** format.

6.1 Common parameters and datatypes

The custom datatypes used in WinDali are the following:

Datatype	Standard Pascal datatype
TFloat	Double
TInteger	Integer
TBoolean	WordBool

You can use the Standard Pascal datatypes if you prefer, but the meaning of **TFloat**, **TInteger** and **TBoolean** might change in future versions of WinDali.

The following parameter appears in several of the following function calls:

ALabel Label to add before or after the variable
 If first character in **ALabel** is
 “-“: Then the label is displayed before the variable and a line is
 added above the label.
 else : The label is displayed before the variable and no line is added.
 If **ALabel** equals “_” then a line will be drawn after the variable.

6.2 SetupProblem

In **SetupProblem** the model is specified so the solver knows the number of dynamic and static variables. But the user-interface in the Simulation Interface program is also created from the specifications.

To specify the model, a number of functions are available.

Most of the **Add...** functions have two versions: a simple and an extended. The extended functions give more control of the appearance of the user-interface in the Simulation Interface program. The extended functions end on **Ext** – for example:

AddDynamic (simple version)
AddDynamicExt (extended version)

The available functions are explained in the following sections.

6.2.1 SetupModel

Heading

```
procedure SetupModel(Title : PChar; TStart, TEnd : TFloat);
```

Parameters

Title	The title of the model.
TStart	Default start time of the simulation.
TEnd	Default end time of the simulation.

Example

```
SetupModel('Example', 0, 3600);
```

6.2.2 SetupState

Heading

```
procedure SetupState(ShowStartState : TBoolean; StartState : TInteger);
```

Parameters

ShowStartState	Let the user select the start state or not. If set to false then you can call SetStartState in PreCalc See 6.2.17.
StartState	Default initial state of the model.

Example

```
SetupState(True, 1);
```

6.2.3 SetTimeFactor

Heading

```
procedure SetTimeFactor(TimeFac : TFloat);
```

Parameters

TimeFac	Factor to divide time with on plots and in files. I.e. if you set TimeFac to 1000, you will internally in the model calculate in milliseconds. TStart and TEnd should be specified in the same units as the internal time. I.e. if TimeFac is 1000, then TEnd = 1000 means 1 second.
----------------	---

Example

```
SetTimeFactor(1);
```

6.2.4 SolverSettings

SolverSettings are kept for backward compatibility – use **SetupModel**, **SetupState** and **SetTimeFactor** instead.

6.2.5 Dynamic variables

AddDynamic adds information about a dynamic variable to the solver and the Simulation program.

Heading

```
function AddDynamic(var Variable : TFloat; InitalValue : TFloat;
  Name,LongName : PChar) : TInteger;
function AddDynamicExt(var Variable : TFloat; InitalValue : TFloat;
  Name,LongName : PChar; Min,Max : TFloat; Show : TBoolean;
  ALabel : PChar) : TInteger;
```

Parameters

Variable	The declared pascal variable, which represents the variable in the model.
InitalValue	Default initial value of the dynamic variable.
Name	Short name that appears on plots.
LongName	Long name, appears on the Initial page in the Simulation Interface program.
Min	Minimum value the user can set the variable to.
Max	Maximum value the user can set the variable to.
	If Min = Max = 0 then no limits are set.
Show	True: Display the dynamic variable in interface. Plot/save of this variable is available to the user. False: Do not display the dynamic variable in interface. Plot/save of this variable is not available to the user.

See also chapter 6.14.

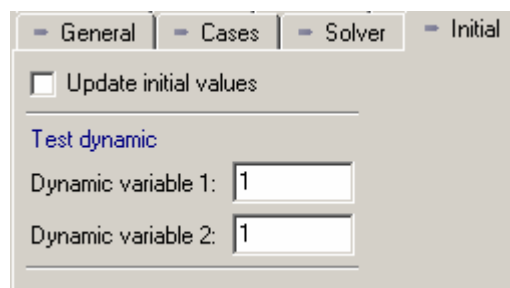
Return value

The number assigned to the dynamic variable. Can be used to identify the variable in calls to **SetInitial**.

Example

```
implementation
var
  Y1,Y2 : TFloat;
procedure SetupProblem;
begin
  AddDynamicExt(Y1,1,'DynVar1','Dynamic variable 1',1,10,True,
    '-Test dynamic');
  AddDynamicExt(Y2,1,'DynVar2','Dynamic variable 2',1,10,True,
    '_');
end;
```

Will produce the following output on the Initial page in the simulation program:



(Note that the checkbox “Update initial values” is automatically created).

6.2.6 States

SetStates defines the number and names of the states in the model.

Heading

```
procedure SetStates (Names : PChar);
```

Parameters

Names Comma separated string specifying the names of the states. If a state name contains spaces then enclose the name in `" "`.

Example

```
//Three states are created:
SetStates("Valve on",Off,"Valve Saturated");
```

6.2.7 Static variables

AddStatic adds information about a static variable in a state.

Heading

```
function AddStatic(State : TInteger; var Variable : TFloat;
  InitialGuess : TFloat; Name,LongName : PChar) : TInteger;
function AddStaticExt(State : TInteger; var Variable : TFloat;
  InitialGuess : TFloat; Name,LongName : PChar; Min,Max : TFloat;
  DoPlot : TBoolean; ALabel : PChar) : TInteger;
```

Parameters

State	The state the static variable belongs to.
Variable	The declared pascal variable, which represents the variable in the model.
InitialGuess	The default guess on the static variable.
Name	Short name that appears on plots.
LongName	Long name, appears on the Guesses page in the Simulation Interface program.
Min	Minimum value the user can set the variable to (and limit for the static equation solver).
Max	Maximum value the user can set the variable to (and limit for the static equation solver).
	If Min = Max = 0 then no limits are set.
DoPlot	True: Plot/save is available to the user for this variable False: Plot/save is not available to the user for this variable.

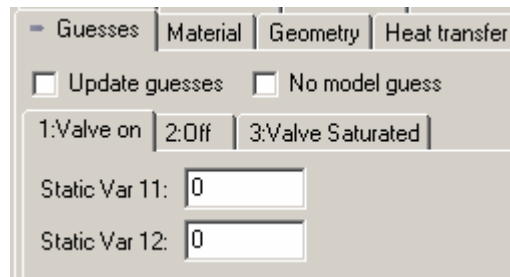
Return value

The number assigned to the static variable. Can be used to identify the variable in calls to **SetGuess**.

Example

```
implementation
var
  X11,X12,X21,X22,X23,X31 : TFloat;
procedure SetupProblem;
begin
  AddStaticExt(1,X11,0,'Stat11','Static Var 11',0,0,True,'');
  AddStaticExt(1,X12,0,'Stat12','Static Var 12',0,0,True,'');
  AddStaticExt(2,X21,0,'Stat21','Static Var 21',0,0,True,'');
  AddStaticExt(2,X22,0,'Stat22','Static Var 22',0,0,True,'');
  AddStaticExt(2,X23,0,'Stat23','Static Var 23',0,0,True,'');
  AddStaticExt(3,X31,0,'Stat31','Static Var 31',0,0,True,'');
end;
```

If you follow the examples in 6.2.6 and this section, you will see the following result on the Guesses page in the simulation program:



If you have static variables that are common to several states, you can add the required information by calling `AddCommonStatic`:

Heading

```
procedure AddCommonStatic(var Variable : TFloat;
  InitialGuess : TFloat; Name,LongName : PChar; CommonStates : PChar);
procedure AddCommonStaticExt(var Variable : TFloat; InitialGuess : TFloat;
  Name,LongName : PChar; CommonStates : PChar; Min,Max : TFloat;
  DoPlot : TBoolean; ALabel : PChar);
```

Parameters

Variable	The declared pascal variable, which represents the variable in the model.
InitialGuess	The default guess on the static variable.
Name	Short name that appears on plots.
LongName	Long name, appear on Guess page in Simulation Interface program.
CommonStates	Comma separates string of the states the static variable exists in. If the variable exists in all states, CommonStates should be empty.
Min	Minimum value the user can set the variable to.
Max	Maximum value the user can set the variable to.
	If Min = Max = 0 then no limits are set.
DoPlot	True: Plot/save is available to the user for this variable False: Plot/save is not available to the user for this variable.

Example

```
implementation
var
  X13,XAll : TFloat;
procedure SetupProblem;
begin
  {add X13 to state 1 and 3:}
  AddCommonStaticExt(X13,0,'Stat13','Static Var 13','1,3',0,0,True,'');
  {add XAll to all states:}
  AddCommonStaticExt(XAll,0,'StatAll','Static Var All','',0,0,True,'');
end;
```

6.2.8 Parameter pages

SetParamPages sets the number and names of the parameter pages to create in the Simulation Interface program.

Heading

```
procedure SetParamPages(Names : PChar);
```

Parameters

Names Comma separated string specifying the names of the parameter pages. If a page name contains spaces then enclose it in "".

Example

```
SetParamPages('Parameters,Settings,"Control settings");
```

6.2.9 Initial Parameters

These kinds of parameters are special as they appear on the Initial page in the Simulation Interface program. Initial value parameters will be further discussed in chapter 6.14.

6.2.10 Floating point parameters

AddFloatParam adds information about a floating-point parameter.

Heading

```
function AddFloatParam(var Parameter : TFloat; DefaultValue : TFloat;  
    Name : PChar; ParamPage : TInteger) : TInteger;  
function AddFloatParamExt(var Parameter : TFloat; DefaultValue : TFloat;  
    Name : PChar; ParamPage : TInteger; Min,Max : TFloat;  
    ALabel : PChar) : TInteger;
```

Parameters

Parameter	The declared pascal variable, which represents the parameter in the model.
DefaultValue	The default value of the parameter.
Name	Text that appears in the Simulation Interface program.
ParamPage	The number of the parameter page to place the parameter on.
Min	Minimum value the user can set the parameter to.
Max	Maximum value the user can set the parameter to. If Min = Max = 0 then no limits are set.
ALabel	See 5.1.

Return value

The number assigned to the parameter. Can be used to identify the parameter in calls to **SetUIValue**.

Example

```

implementation
var
  Rho,Cp,Lambda,V,A,h,Ta : TFloat;
procedure SetupProblem;
begin
  SetParamPages('Material,Geometry,"Heat transfer"');
  AddFloatParamExt(Rho,8000,'Density [kg/m^3]',1,0,20000,'');
  AddFloatParamExt(Cp,480,'Specific heat [J/kg K]',1,0.001,5000,'');
  AddFloatParamExt(Lambda,15,'Conductivity [W/m K]',1,0.0001,5000,'');
  AddFloatParamExt(V,0.001,'Volume [m^3]',2,0.000001,1000,'');
  AddFloatParamExt(A,0.06,'Surface area [m^2]',2,0.000001,10000,'');
  AddFloatParamExt(h,10,'Heat transfer coefficient [W/m^2 K]',
    3,0.00001,100000,'');
  AddFloatParamExt(Ta,20,'Ambient temperature [°C]',3,0,0,'');
end;

```

Will produce the following result in the Simulation Interface program:

The figure shows three screenshots of the Simulation Interface program, each displaying a different tab in the parameter settings window. The tabs are 'Guesses', 'Material', 'Geometry', and 'Heat transfer'. The 'Material' tab shows 'Number of sections' (10), 'Specific heat [J/kg K]' (480), and 'Conductivity [W/m K]' (15). The 'Geometry' tab shows 'Volume [m^3]' (0.001) and 'Surface area [m^2]' (0.06). The 'Heat transfer' tab shows 'Heat transfer coefficient [W/m^2 K]' (10).

6.2.11 Integer parameters

Integer parameters differ from floating point parameter in that the user only is allowed to input Integer values.

AddIntParam adds information about an Integer parameter.

Heading

```
function AddIntParam(var Parameter : TInteger; DefaultValue : TInteger;  
    Name : PChar; ParamPage : TInteger) : TInteger;  
function AddIntParamExt(var Parameter : TInteger; DefaultValue : TInteger;  
    Name : PChar; ParamPage : TInteger; Min,Max : TInteger;  
    ALabel : PChar) : TInteger;
```

Parameters

Parameter	The declared pascal variable, which represents the parameter in the model.
DefaultValue	The default value of the parameter.
Name	Text that appears in the Simulation Interface program.
ParamPage	The number of the parameter page to place the parameter on.
Min	Minimum value the user can set the parameter to.
Max	Maximum value the user can set the parameter to. If Min = Max = 0 then no limits are set.
ALabel	See 5.1.

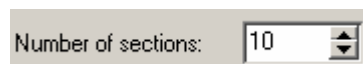
Return value

The number assigned to the parameter. Can be used to identify the parameter in calls to **SetUIValue**.

Example

```
implementation  
var  
    NSec : TInteger;  
procedure SetupProblem;  
begin  
    AddIntParamExt(NSec,10,'Number of sections',1,1,1,100,'');  
end;
```

This will add an Integer parameter that looks like this:



6.2.12 Boolean parameters

Boolean parameters are logical parameters that have the value True or False.

AddBoolParam adds information about a Boolean parameter.

Heading

```
function AddBoolParam(var Parameter : TBoolean; DefaultValue : TBoolean;
  Name : PChar; ParamPage : TInteger) : TInteger;
function AddBoolParamExt(var Parameter : TBoolean; DefaultValue : TBoolean;
  Name : PChar; ParamPage : TInteger; ALabel : PChar) : TInteger;
```

Parameters

Parameter	The declared pascal variable, which represents the parameter in the model.
DefaultValue	The default value of the parameter.
Name	Text that appears in the Simulation Interface program.
ParamPage	The number of the parameter page to place the parameter on.
ALabel	See 5.1.

Return value

The number assigned to the parameter. Can be used to identify the parameter in calls to **SetUIValue**.

Example

```
implementation
var
  TestBool : TBoolean;

procedure SetupProblem;
begin
  AddBoolParamExt(TestBool, False, 'TestBool', 3, '');
end;
```

This will add a Boolean parameter that looks like this:



6.2.13 List parameters

List parameters are parameters that can have one of several values displayed to the user in a listbox.

AddListParam adds information about a list parameter.

Heading

```
function AddListParam(var Parameter : TInteger; DefaultIndex : TInteger;  
    Items,Name : PChar; ParamPage : TInteger) : TInteger;  
function AddListParamExt(var Parameter : TInteger; DefaultIndex : TInteger;  
    Items,Name : PChar; ParamPage : TInteger; ALabel : PChar) : TInteger;
```

Parameters

Parameter	The declared pascal variable, which represents the parameter in the model.
DefaultIndex	Index of the default selected item. The first item has index 1.
Items	A comma separated string with the items the listbox should contain. If an item contains spaces it should be enclosed in "". When the simulation starts, Parameter will contain the index of the selected parameter.
Name	Text that appears in the Simulation Interface program.
ParamPage	The number of the parameter page to place the parameter on.
ALabel	See 5.1.

Return value

The number assigned to the parameter. Can be used to identify the parameter in calls to **SetUIValue**.

Example

```
implementation  
var  
    List : TInteger;  
  
procedure SetupProblem;  
begin  
    AddListParamExt(List,1,'"Item 1","Item 2","Item 3","Item 4"',  
        'TestList', 1, '');  
end;
```

This will add a list parameter that looks like this:



6.2.14 Enumerated parameters

Enumerated parameters are very similar to list parameters, except that the items will be displayed in a combobox instead of a listbox.

[AddEnumParam](#) adds information about an enumerated parameter.

Heading

```
function AddEnumParam(var Parameter : TInteger; DefaultIndex : TInteger;  
    Items,Name : PChar; ParamPage : TInteger) : TInteger;  
function AddEnumParamExt(var Parameter : TInteger; DefaultIndex : TInteger;  
    Items,Name : PChar; ParamPage : TInteger; ALabel : PChar) : TInteger;
```

Parameters

See list parameters.

Return value

The number assigned to the parameter. Can be used to identify the parameter in calls to [SetUIValue](#).

Example

```
implementation  
var  
    Enum : TInteger;  
  
procedure SetupProblem;  
begin  
    AddEnumParamExt(Enum,1,"Item 1","Item 2","Item 3","Item 4",  
        'TestEnum',1,'');  
end;
```

This will add a enumerated parameter that looks like this:



6.2.15 Enumerated choice parameters

Enumerated choice parameters are used to create structures with multiple choices, and where each choice has a different number of parameters. For example can the UA value of a heat exchanger be specified by:

1. An *UA*-value directly
2. Dimensioning values of Q and ΔT ($Q = UA \cdot \Delta T$).

AddEnumChoiceParam adds information about an enumerated choice parameter. Enumerated choice parameters are special, because besides **AddEnumChoiceParam**, another function has to be called to complete the specification. Enumerated choice parameters can be regarded as a combination of an enumerated parameter and a number of floating point parameters. This will be clear when an example is given below.

Heading

```
function AddEnumChoiceParam(var Parameter : TInteger;  
    DefaultIndex : TInteger; Items,Name : PChar;  
    ParamPage : TInteger) : TInteger;  
function AddEnumChoiceParam(var Parameter : TInteger;  
    DefaultIndex : TInteger; Items,Name : PChar; ParamPage : TInteger;  
    ALabel : PChar) : TInteger;
```

Parameters

Parameter	The declared pascal variable, which represents the parameter in the model.
DefaultIndex	Index of the default selected item. The first item has index 1.
Items	A comma separated string with the items the combobox should contain. If an item contains spaces it should be enclosed in "". When the simulation starts, Parameter will contain the index of the selected parameter.
Name	Text that appears in the Simulation Interface program.
ParamPage	The number of the parameter page to place the parameter on.
ALabel	See 5.1.

Return value

The number assigned to the parameter. Can be used to identify the parameter in calls to **SetUIValue**.

AddChoice is used to add information for each of the choices for each of the items **AddEnumChoiceParam**. Note that **AddChoice** has to be called immediately after **AddEnumChoiceParam**.

Heading

```
function AddChoice(ItemIndex : TInteger; var Parameter : TFloat;
  DefaultValue : TFloat; Name : PChar) : TInteger;
function AddChoiceExt(ItemIndex : TInteger; var Parameter : TFloat;
  DefaultValue : TFloat; Name : PChar; Min,Max : TFloat) : TInteger;
```

Parameters

ItemIndex	The index of the item the choice belongs to.
Parameter	The declared pascal variable, which represents the parameter in the model.
DefaultValue	The default value of the parameter.
Name	Text that appears in the Simulation Interface program.
Min	Minimum value the user can set the parameter to.
Max	Maximum value the user can set the parameter to.
	If Min = Max = 0 then no limits are set.

Return value

The number of the choice among the choices belonging to the item. Can be used to identify the choice in calls to **SetUIValue**.

Example

Taking the example where the UA value of a heat exchanger can be specified by:

1. An *UA*-value directly
2. Dimensioning values of Q and ΔT ($Q = UA \cdot \Delta T$).

```
implementation
var
  Choice      : TInteger;
  UA,QDim,DTDim : TFloat;
procedure SetupProblem;
begin
  AddEnumChoiceParamExt(Choice,1,'UA',"Q_Dim,DT_dim"',
    'Specify UA value by specifying',1,'');
  AddChoiceExt(1,UA,100,'UA value [W/K]',0,0);
  AddChoiceExt(2,QDim,1000,'Q_dim [W]',0,0);
  AddChoiceExt(2,DTDim,10,'DT_dim [K]',0,0);
end;
```

This will add an enumerated choice parameter that looks like this:

- a) When the user selects UA:

- b) When the user selects Q_dim,DT_dim:

6.2.16 Explicit variables

These variables are not part of the equation system; but they can be plotted and saved with the dynamic and static variables. In other words: explicit variables are variables that can be calculated explicitly.

AddExplicit adds information about an explicit variable.

Heading

```
function AddExplicit(var Variable : TFloat; Name : PChar;  
    DoPlot : TBoolean) : TInteger;  
function AddExplicitExt(var Variable : TFloat; Name : PChar;  
    DoPlot : TBoolean; ALabel : PChar) : TInteger;
```

Parameters

Variable The declared pascal variable, which represents the variable in the model.
Name Text that appears in the Simulation Interface program.
DoPlot True: The variable is plotted.
False: The variable is added to the Results page when calculation is done.

Return value

The number assigned to the variable. Can be used to identify the variable in calls to **SetUIValue**.

Example

```
AddExtraExt(Bi, 'Biot number', False, '');
```

6.2.17 Action buttons

Action buttons are buttons you can place on the user-interface. You are responsible for writing code, which responds when the user presses the button – this should be done in **OnUIValueChange** (see 6.11). You install buttons by calling **AddActionBtn**.

Heading

```
function AddActionBtn(Caption : PChar; ParamPage : TInteger) : TInteger;  
function AddActionBtnExt(Caption : PChar; ParamPage : TInteger;  
    ALabel : PChar) : TInteger;
```

Parameters

Caption The caption of the action button.
ParamPage The number of the parameter page to place the button on.
ALabel See 5.1.

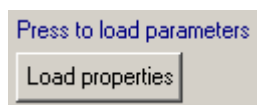
Return value

The number assigned to the button. Can be used to identify the button in calls to **SetUIValue**.

Example

```
AddActionBtnExt('Load properties', 1, 'Press to load parameters');
```

Will create a button, which looks like this:



6.2.18 Info Labels

Info labels are labels you can place on the user-interface, displaying some information to the user. You install info labels by calling `AddInfoLabel`.

Heading

```
procedure AddInfoLabel(Caption : PChar; ParamPage : TInteger) : TInteger;
procedure AddInfoLabelExt(Caption : PChar; ParamPage : TInteger;
    ALabel : PChar) : TInteger;
```

Parameters

<code>Caption</code>	The caption of the info label.
<code>ParamPage</code>	The number of the parameter page to place the info label on.
<code>ALabel</code>	See 5.1.

Return value

The number assigned to the label. Can be used to identify the label in calls to `SetUIValue`.

Example

```
AddInfoLabelExt('This is a message', 1, '');
```

6.2.19 HideSampleTime

`HideSampleTime` can be used to hide information about fixed sample in the simulation program from the user.

On the Solver page in the simulation program the user can select to run the simulation with fixed sample time. If you do not want the user to have this possibility you should call `HideSampleTime`. This could for example be the case if you want the user to input the sample time together with other parameters for a controller and not on the Solver page. In this case you should also call `SetSampleTime` in `PreCalc` (see chapter 6.3 and 6.8).

Heading

```
procedure HideSampleTime;
```

Parameters

None

Example

```
HideSampleTime;
```

6.2.20 Model help file

As of version 1.37 it is possible to specify a help file for the model. The help file can be in any format as the simulation program executes the application, which is associated with the help file.

The help file should be located in the same directory as the model. If the help file consists of several files then you should manually include files other than the installed help file when creating a distributable copy (see chapter 13).

The help file is installed by calling the following procedure within **SetupProblem**:

Heading

```
procedure AddHelpFile(FileName : PChar);
```

Parameters

FileName Name of the help file (without directory information).

Example

```
AddHelpFile('Example.html');
```

6.3 PreCalc

PreCalc is called just before the simulation is started. This procedure can be used to initiate variables, allocating memory etc. Furthermore PreCalc is used to specify initial values for dynamic variables based on initial value parameters, and to add static variables. This is treated in chapter 6.14.

If you set the **ShowStartState** parameter in a call to **SetupState** to false, you can specify the initial state in **PreCalc**. This is done by calling **SetStartState**.

You can also add explicit variables in **PreCalc** by calling **AddExplicitVar** and control the sample time by calling **SetSampleTime**.

6.3.1 SetStartState

Heading

```
procedure SetStartState(Value : TInteger);
```

Parameters

Value Number of the initial state.

Example

```
SetStartState(1);
```

6.3.2 AddExplicitVar

Heading

```
function AddExplicitVar(var Variable : TFloat; Name : PChar;
  DoPlot : TBoolean) : TInteger;
function AddExplicitVarExt(var Variable : TFloat; Name : PChar; DoPlot :
  TBoolean; ALabel : PChar) : TInteger;
```

Parameters

Variable	The declared pascal variable, which represents the explicit variable in the model.
Name	Name of explicit variable.
DoPlot	True: The variable is plotted. False: The variable is added to the Results page when calculation is done.

Example

```
AddExplicitVar(W, 'Compressor work', True);
```

6.3.3 SetSampleTime

Heading

```
procedure SetSampleTime(IsFixed : Boolean; Value : TFloat);
```

Parameters

IsFixed	Run with fixed sample time? If true the sample time specified in value is used; else fixed sample time is not used.
Value	The value of the fixed sample time in seconds.

Example

```
SetSampleTime(True, 10);
```

6.4 ModelEquations

In the **ModelEquations** procedure the equations are written. **ModelEquations** is called every time the Solver needs to evaluate the model.

Suppose the problem from chapter 5 is extended, so that instead of just cooling the block, it is required that the temperature is held at $50\text{ °C} \pm 2\text{ °C}$. To accomplish this, the block is put into an air chamber, with the possibility to add air at 70 °C or air at 20 °C :

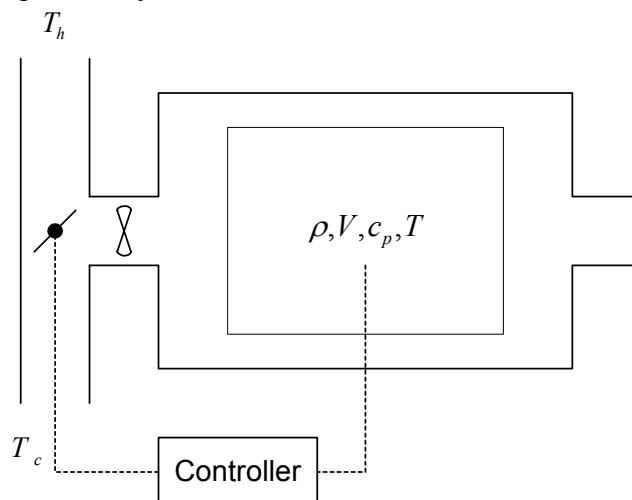


Figure 10. Controlling the temperature.

Now the system has two states: either hot air supply or cold air supply.

The equations for the two states are:

$$\begin{aligned}\rho V c_p \frac{dT}{dt} &= h A (T_c - T) , Cold \\ \rho V c_p \frac{dT}{dt} &= h A (T_h - T) , Hot\end{aligned}\tag{5.1}$$

The **SetupProblem** procedure for this case will look like this:

```
procedure SetupProblem;
begin
  SetupModel('Cooling of block 2',0,20000);
  SetStates('Cold,Hot');
  SetParamPages('Material,Geometry,"Heat transfer"');
  AddDynamic(T,100,'T','Temperature [°C]');
  AddFloatParamExt(Rho,8000,'Density [kg/m^3]',1,0,20000,'');
  AddFloatParamExt(Cp0,480,'Specific heat [J/kg K]',1,0.001,5000,'');
  AddFloatParamExt(Lambda,15,'Conductivity [W/m K]',1,0.0001,5000,'');
  AddFloatParamExt(V,0.001,'Volume [m^3]',2,0.000001,1000,'');
  AddFloatParamExt(A,0.06,'Surface area [m^2]',2,0.000001,10000,'');
  AddFloatParamExt(h,10,'Heat transfer coefficient [W/m^2 K]',
    3,0.00001,100000,'');
  AddFloatParamExt(Tc,20,'Cold temperature [°C]',3,0,0,'');
  AddFloatParamExt(Th,70,'Hot temperature [°C]',3,0,0,'');
  AddExplicit(Bi,'Biot number',False);
end;
```

and the **ModelEquations** procedure will look like this:

```
procedure ModelEquations(Time : TFloat; State : TInteger;
  var R : array of TFloat; var YDot : array of TFloat);
begin
  if State = 1 then
    YDot[0] := h*A/(Rho*V*Cp)*(Tc-T)
  else
    YDot[0] := h*A/(Rho*V*Cp)*(Th-T);
end;
```

It is also necessary to fill out **StatesShift** but this will be treated in chapter 6.5.

To show how static equations are formulated assume that the specific heat of the material, no longer is constant; but can be calculated as:

$$c_p = c_{p0} \sqrt{\frac{c_p - T}{c_p}}\tag{5.2}$$

Note that this equation is pure fictional. It has no physical meaning and only serves to illustrate static variables. c_{p0} is a parameter provided by the user.

In **SetupProblem** the following line:

```
AddFloatParamExt(Cp,480,'Specific heat [J/kg K]',1,0.001,5000,'');
```

Is changed to:

```
AddFloatParamExt(Cp0,480,'Specific heat [J/kg K]',1, 0.001,5000,'');
```

And the following line is added to **SetupProblem**:

```
AddCommonStaticExt(Cp,480,'Cp','Specific heat [J/kg K]','',0,0,True,'');
```

The c_p equation also has to be added to **ModelEquations**. This is done by adding the line:

```
R[0] := Cp-Cp0*Sqrt((Cp-T)/Cp);
```

See also the included "Cooling of Block 2" demo.

6.5 StateShift

StateShift is used to specify when the logical state is changed. The heading of the **StateShift** function looks like this:

```
procedure StateShift(Time : TFloat; State : TInteger;
  var G : array of TFloat);
```

The G -array is zero-indexed, and has the same length as the number of states. A shift to another state is done when the corresponding item in the G array becomes negative. The **State** parameter holds the number of the current state.

In the example from chapter 6.4 there was two states:

- 1: Cold corresponds to $G[0]$
- 2: Hot corresponds to $G[1]$

The temperature should be held at $50\text{ °C} \pm 2\text{ °C}$. That is when the system is in state 1 it should shift to state 2 when the temperature drops below 48 °C . And when the system is in state 2 it should shift to state 1 when the temperature is above 52 °C :

```
procedure StateShift(Time : TFloat; State : TInteger;
  var G : array of TFloat);
begin
  case State of
    1 : G[1] := T-48;
    2 : G[0] := 52-T;
  end;
end;
```

What the code states is that when the current state is 1 (Cold) then $G[1]$ becomes negative when T is less than 48 °C and the Solver will then shift to state 2. When the current state is 2 (Hot) then $G[0]$ will become negative when T is larger than 52 °C , and the Solver will then shift to state 1.

The procedure **ShiftToState** can be used to perform the same as the code above, but it makes it a bit more readable:

```
procedure StateShift(Time : TFloat; State : TInteger;
  var G : array of TFloat);
begin
  case State of
    1 : SwitchToState(2, T-48, G);
    2 : SwitchToState(1, 52-T, G);
  end;
end;
```

Read the call `SwitchToState(2, T-48, G)` as "Switch to state 2 when T-48 becomes negative".

Remember to supply G as the last parameter in the call to **SwitchToState**.

SwitchToState has the following implementation:

```
procedure SwitchToState(StateNum : TInteger; SignChange : TFloat;
  var G : array of TFloat);
begin
  G[StateNum-1] := SignChange;
end;
```

6.6 OnStateChange

OnStateChange is seldom modified. It can be used to supply guesses (different from the default) on the static variables or to change the “initial” value of a dynamic variable before continuing in to a new state. An example showing this is included in the Bouncing Ball demo

The solver supplied with WinDali automatically updates guesses on static variables. That is it uses the supplied guesses the first time it enters a state. If it enters the same state a second time, the solution from the first time is used as guesses.

6.7 OnSolution

Called every time a solution has been found. Can be used to do intermediate calculations.

6.8 OnSample

The procedure **OnSample** is called if you choose to run the simulation with a fixed sample time.

If for example the sample time is 10 seconds it forces the equation solver to produce a solution at every 10 seconds, but it might also produce solutions in between each sample.

When a solution is found at a sample time, then first **OnSolution** is called then **OnSample** and finally **OnSolution** again. The reason for this is that you want to plot both the solution just before **OnSample** is called, and if some values are changed in **OnSample**, then the changes should also be plotted with the same timestamp.

If you for example implement a controller, which operates with a fixed sample time of 10 seconds – let us say it controls the speed of a compressor – then at time 100 a solution is found

which is plotted. This solution is then used in the controller when **OnSample** is called to change the speed, and finally **OnSolution** is called to reflect the change in the speed.

You should note that **OnSample** will not be called at the initial time. If your simulation runs from 0 seconds and you specify a sample time of 10 seconds then **OnSample** will be called the first time at Time = 10 seconds. This is because the results of the calculations in **OnSample** might be used in **ModelEquations** and the static and dynamic variables might be used in **OnSample**. At the initial time the static variables only have guess-values, so to solve the equations in **ModelEquations** the results from **OnSample** has to be known. In other words: you have to specify initial values for the values calculated in **OnSample**, which are used in **ModelEquations**.

Typically a controller will also control which state the system is in (for example if the compressor is On or Off). Therefore you can force the state to change by changing the supplied State parameter in **OnSample**:

```
procedure OnSample(Time : TFloat; var State : TInteger);
```

Note that **State** is a var parameter, which means that you can change **State** within **OnSample**.

The calling sequence for the procedures in figure 8 was:

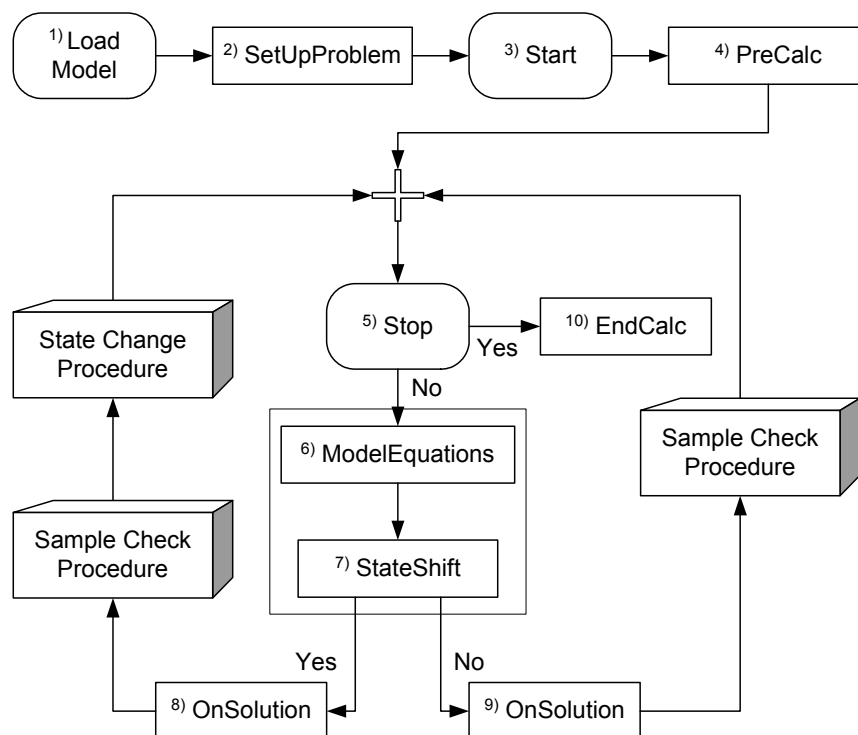


Figure 8. Calling sequence for the procedures in the model file.

And the State Change Procedure called the following procedures:

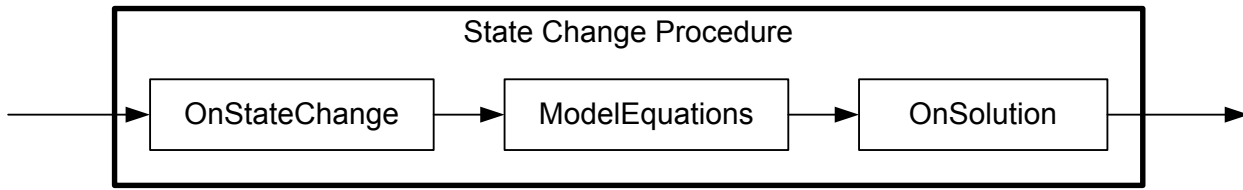


Figure 8a. State Change Procedure.

The Sample Check Procedure calls the following procedures in the Model file:

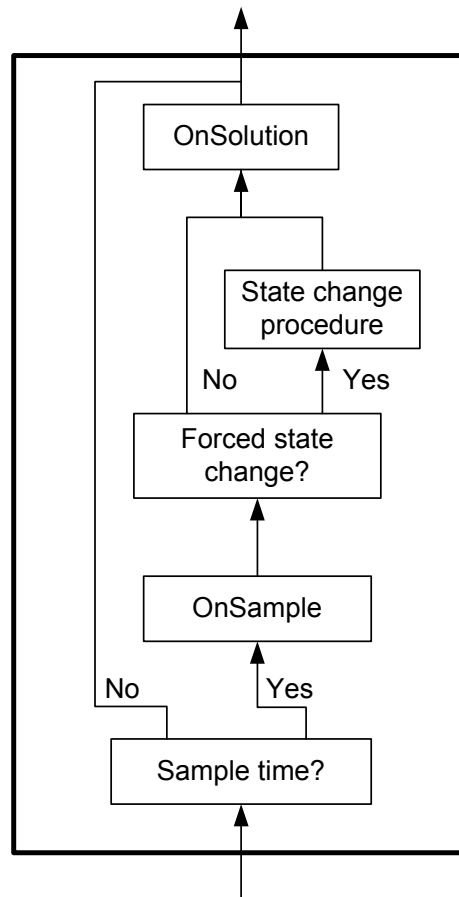


Figure 8c. Sample Check Procedure.

Note that the State Change Procedure is called within the Sample Check Procedure if the state is forced to change in the `OnSample` procedure.

6.9 EndCalc

Called once when the simulation stops. Can be used to free memory allocated in `PreCalc`, and to do some final calculations on explicit variables.

6.10 OnQuit

Is called when the user exits the Simulation Program or changes the model. Can be used to do some final clean up.

6.11 OnUIValueChange

OnUIValueChange (On User Interface Value Change) is called whenever the user changes a value in the Simulation Interface program (including when the user presses an Action button).

The heading looks like this:

```
procedure OnUIValueChange(UIType : TInteger; Num,State,Choice : TInteger;
    Value : TFloat);
```

UIType can have one of the following values:

0	ptNone:	No value changed (you will never get this value)
1	ptDynamic:	An initial value for a dynamic variable has changed
2	ptStatic:	A guess value on static variable was changed
3	ptInitial:	An initial parameter was changed
4	ptExplicit:	An explicit variable has changed
5	ptInput:	An input variable has changed
6	ptFloat:	A floating point parameter was changed
7	ptInteger:	An Integer parameter was changed
8	ptBoolean:	A boolean parameter was changed
9	ptList:	A list parameter was changed
0	ptEnum:	An enum parameter was changed
11	ptEnumChoice:	An enum choice parameter was changed
12	ptChoice:	A choice in a Enum Choice parameter was changed
13	ptActionBtn:	An action button was pressed
14	ptInfoLabel:	An info label was changed
15	ptDynDynamic:	An initial value for a dynamic variable created calling AddDynVar has changed
16	ptDynStatic:	A guess value on static variable created calling AddStatVar has changed
17	ptDynExplicit:	An explicit variable created calling AddExplicitVar has changed
18	ptDynInput:	An input variable created calling AddInputVar has changed

The values **ptNone**, **ptInput** and **ptDynInput** are reserved for future use, and you will never see these values.

Num Is the number of the variable/parameter/button/info label that was changed.

State

- if **UIType** is 2 then **State** is the state, the static variable with number **Num** belongs to
- if **UIType** is 12 then **State** is the **ItemIndex** of the enumerated choice parameter with number **Num**
- if **UIType** is none of the above then **State** is ignored.

Choice

- if **UIType** is 12 then **Choice** is the **ParamIndex** of the **ItemIndex** (= **state**) of the enumerated choice parameter with number **Num**
- else **Choice** is ignored.

Value is the numerical value of the value in the user interface that was changed.

You can call two procedures as respond to a change in a value:

```
procedure SetUIValue(UIType : TInteger; Num, State, Choice : TInteger;  
  Value : TFloat; StrValue : PChar);
```

which sets a value in the user interface, and

```
function GetUIValue(UIType : TInteger; Num, State, Choice : TInteger;  
  StrValue : PChar) : TFloat;
```

which gets a value from the user interface.

Three notes should be made:

- the Pascal variables you have declared does not reflect the values in the user-interface when **OnUIValueChange** is called (the variables are only updated just before **PreCalc** is called)
- The only type you can use for getting and setting user interface values is **TFloat**. This is also true if you want to get/set a Boolean parameter. If it's a Boolean parameter then if the value:
 - = 0 it's False
 - = 1 it's True
- The **StrValue** parameter is used if the item in question is an Action button or an Info label.

The parameters in **SetUIValue** and **GetUIValue** have the same meaning as for **OnUIValueChange**.

Example:

Suppose you have defined an action button, a boolean parameter and an enumerated choice parameter in **SetupProblem**:

```
implementation  
var  
  UACond          : TInteger;  
  UA_c, QDim_c, DTDim_c : TFloat;  
  TestBool        : TBoolean;  
procedure SetupProblem;  
begin  
  AddEnumChoiceParam(UACond, 1, 'UA-value', 'Q_Dim, DT_dim'',  
    'Specify UA value by specifying', 1);  
  AddChoice(1, UA_c, 1000, 'UA value [W/K]');  
  AddChoice(2, QDim_c, 8000, 'Q_dim [W]');  
  AddChoice(2, DTDim_c, 8, 'DT_dim [K]');  
  AddActionBtn('Update DTDim_c', 1);  
  AddBoolParam(TestBool, True, 'TestBool', 1);  
end;
```

In **OnUIValueChange** you want the following to happen:

When the user press the button and "Q_Dim, DT_dim" is selected in the enumerated choice parameter and **QDim_c** is larger than 6000 then **DTDim_c** is set to 6, and the boolean parameter is set to false.

```
procedure OnUIValueChange(UIType : TUserinterfaceType;
                           Num,State,Choice : TInteger;
                           Value : TFloat); stdcall;

begin
  //Only perform if button number 1 is pressed
  if (TParameterType(UIType) = ptActionBtn) and (Num = 1) then
    begin
      {Only perform if item number 2 (i.e. "Q_Dim, DT_dim") is selected in
       enumerated choice parameter 1. Rounding is done to avoid problems
       comparing a floating point number with an integer}
      if Round(GetUIValue(Ord(ptEnumChoice),1,0,0,nil)) = 2 then
        begin
          {Only perform if choice number 1 in item 2 in enumerated
           parameter 1 is larger than 6000}
          if GetUIValue(Ord(ptChoice),1,2,1,nil) > 6000 then
            begin
              SetUIValue(Ord(ptChoice),1,2,2,6,''); //DTDim_c is set equal 6
              SetUIValue(Ord(ptBoolean),1,0,0,0,''); //TestBool is set false
            end;
          end;
        end;
      end;
    end;
  end;
```

See also the Action btn demo.

6.11.1 Running simulations from the model

You can call three procedures from the model to run simulations without user-interaction. This could for example be useful if you want to investigate the frequency response of a model. In this case you could display a button to the user, and when it is pressed several simulations are run and finally some resulting plots are drawn (for example Bode-plots, Polar plots etc.).

The three procedures that can be called are:

Heading

```
procedure RunSimulation(TStart,TEnd,hMax : TFloat);
```

Parameters

TStart Start time for the simulation.
TEnd End time for the simulation
HMax Reserved for future use

Example

```
RunSimulation(0,3600,0);
```

RunSimulation starts a simulation and control is returned back to the model when the simulation is done.

Heading

```
procedure CreatePlot(var PlotNum : TInteger; XAxisType : TInteger;  
    XLabel : PChar; YAxisType : TInteger; YLabel : PChar);
```

Parameters

PlotNum	Number of the plot created (this value is used in calls to AddCurveToPlot – see below)
XAxisType	Type of the X-axis. If 0 then linear axis. If 1 then logarithmic.
XLabel	Label on X-axis.
YAxisType	Type of the Y-axis. If 0 then linear axis. If 1 then logarithmic.
YLabel	Label on Y-axis

Example

```
var  
    PlotNum : TInteger;  
  
CreatePlot(PlotNum, 0, 'X-Axis', 0, 'Y-Axis');
```

Heading

```
procedure AddCurveToPlot(PlotNum : TInteger; CurveName : PChar;  
    XData, YData : array of TFloat);
```

Parameters

PlotNum	Number returned from call to CreatePlot
CurveName	Name of the curve.
XData	Array with x-coordinates of the curve.
YData	Array with y-coordinates of the curve.

Shortcut

Write **curve** and press <Ctrl>+J.

Example

```
var  
    X, Y : array[1..20] of TFloat;  
  
AddCurveToPlot(PlotNum, 'TestCurve', X, Y);
```

An example showing how to use the procedures is included in the demos.

6.12 OnSaveSettings

This procedure is called every time the user selects the File|Save menu in the simulation program. The name of the file the settings are saved to is passed in the parameter **FileName**.

If you want to save additional information together with the settings the Simulation program saves, you can do it in this procedure. You should not write directly to file with the name passed as **FileName** – instead you could save your custom setting in a file with the same name but with a different extension.

6.13 OnLoadSettings

OnLoadSetting is called every time the user selects the File|Open menu in the Simulation program. The name of the file the user opens is passed in the parameter **FileName**.

6.14 Using Initial parameters

Initial parameters are used to change the default behavior of the installed dynamic variables. Normally, installing a dynamic variable will create an Edit box on the Initial page in the Simulation program, where the user can enter an initial value. But there might be situations where you want to avoid this.

Imagine for example you have a model where the dynamic variable is internal energy. Installing the dynamic variable would then require the user to enter an initial value for the internal energy, which can be difficult to acquire knowledge about. Instead it might be preferred to let the user enter for example temperature and pressure as initial values, and then calculate the initial value for internal energy before the simulation starts (requires that you know that the internal energy can be expressed as a function of Temperature and Pressure – this is not the case if you have two-phase flow).

To make this work you have to:

1. Use Initial parameters for the temperature and pressure. This is done by calling **AddInitialParam** as described below.
2. Avoid the default Edit box for the internal energy to show in the user interface. This is done by setting the **Show** parameter in **AddDynamicExt** to **False** for the internal energy dynamic variable. Note that you still have to call **AddDynamicExt** for the internal energy variable (see 6.2.5). Note also that setting **Show** to **False** prevents the internal energy from appearing in plots and from saving it. If you still want to plot and/or save the internal energy, you have to add an explicit variable, which you just set equal to the internal energy.
3. Calculate and set the initial value for the internal energy before the simulation starts. This can be done in **PreCalc** (see 6.2.17) by calling **SetInitial** (see 6.14.1) as described below.

AddInitialParam adds information about an initial parameter.

Heading

```
function AddInitialParam(var Parameter : TFloat; DefaultValue : TFloat;
    Name : PChar) : TInteger;
function AddInitialParamExt(var Parameter : TFloat; DefaultValue : TFloat;
    Name : PChar; Min,Max : TFloat; ALabel : PChar) : TInteger;
```

Parameters

Parameter	The declared pascal variable, which represents the parameter in the model.
DefaultValue	The default value of the parameter.
Name	Text that appears on the Initial page in the Model & Solver settings window.
Min	Minimum value the user can set the parameter to.
Max	Maximum value the user can set the parameter to.
	If Min = Max = 0 then no limits are set.
ALabel	See 5.1.

Return value

The number assigned to the parameter. Can be used to identify the parameter in calls to **SetUIValue**.

Example

```
implementation
var
  U,P,T : TFloat; {Internal Energy, Pressure and Temperature}
  UIndex : TInteger;
procedure SetupProblem;
begin
  :
  UIndex := AddDynamicExt(U,0,'U','Internal Energy',0,0,False,'');
  AddInitialParamExt(P,1E5,'Pressure [Pa]',0,0,'');
  AddInitialParamExt(T,20,'Temperature [°C]',0,0,'');
  :
end;
procedure PreCalc;
var
  IntEnergy : TFloat;
begin
  :
  IntEnergy := CalculateInternalEnergy(T,P);
  SetInitial(UIndex,IntEnergy);
  :
end;
```

Another example where you could use Initial parameters is when you want the user to be able to change the number of dynamic and/or static equations. This situation arises if you have made a model of for example a pipe, which is divided in to several sections, and you want the user to decide on the number of sections to be used. In this case you don't have any dynamic variables to install in **SetupProblem** (if the pipe is all you want to model), as the number of dynamic variables depends on the number of sections the user want to divide the pipe in. Instead you add an initial parameter, which allows the user to set the number of divisions, and you may also add an initial parameter to allow the user to set the initial value for the dynamic variables.

When the user changes the number of pipe divisions, the number of static variables might also change (depending on the model). The number of dynamic and static variables that the resulting model has when the user starts the simulation is specified in **PreCalc** by calling **AddDynVar** and **AddStatVar** (see 6.14.3 and 6.14.4).

A model of a pipe with selectable number of divisions could be defined as this:

```
implementation
const
  MaxDiv    = 20;
var
  T : array [1..MaxDiv] of TFloat; {Temperature is dynamic variable}
  N : TFloat;                      {Number of divisions}
  NDiv : TInteger;                 {An initial parameter is always
                                   floating point - regardless of the
                                   DataType. NDiv is the TInteger version
                                   of N}

  procedure SetupProblem;
  begin
    :
    {Note that input is limited between 1 and MaxDiv: }
    AddInitialParamExt(N,10,'Number of divisions',1,MaxDiv,'');
    :
  end;
  procedure PreCalc;
  var
    i,      : TInteger;
    AName   : string;
  begin
    :
    NDiv := Round(N);
    for i:=1 to Ndiv do
      begin
        AName := 'T_'+IntToStr(i)
          {Note the typecast from string to PChar}
        AddDynVar(T[i],1,PChar(AName),PChar(AName));
      end;
    :
  end;
```

6.14.1 SetInitial

SetInitial is used to change the initial value of a dynamic variable just before the simulation begins (i.e. after the user starts the simulation, but before **ModelEquations** is called the first time). **SetInitial** is normally called in **PreCalc**.

Heading

```
procedure SetInitial(Index : TInteger; Value : TFloat);
```

Parameters

Index Number of the dynamic variable to set the initial value of.

Value The new initial value.

Example

```
SetInitial(1,10.25);
```

6.14.2 SetGuess

SetGuess is used to change the guess value of a static variable just before the simulation begins (i.e. after the user starts the simulation, but before **ModelEquations** is called the first time).

SetGuess is normally called in **PreCalc**.

Heading

```
procedure SetGuess(State, Index : TInteger; Value : TFloat);
```

Parameters

State The state the static variable belongs to
Index Number of the static variable to set the guess value of.
Value The new guess value.

Example

```
SetGuess(1, 1, 10.25);
```

6.14.3 AddDynVar

AddDynVar is used to add dynamic variables just before the simulation begins (i.e. after the user starts the simulation, but before **ModelEquations** is called the first time). **AddDynVar** is normally called in **PreCalc**.

Heading

```
function AddDynVar(var Variable : TFloat;  
  InitalValue : TFloat; Name, LongName : PChar) : TInteger;  
function AddDynVarExt(var Variable : TFloat; InitalValue : TFloat;  
  Name, LongName : PChar; Min, Max : TFloat; Show : TBoolean;  
  ALabel : PChar) : TInteger;
```

Parameters

Variable The declared pascal variable, which represents the variable in the model.
InitalValue Default initial value of the dynamic variable.
Name Short name that appears on plots.
LongName Long name, appears Initial page in the Simulation Interface program.
Min Minimum value the user can set the variable to.
Max Maximum value the user can set the variable to.
If **Min = Max = 0** then no limits are set.
Show True: Display the dynamic variable in interface.
Plot/save of this variable is available to the user.
False: Do not display the dynamic variable in interface.
Plot/save of this variable is not available to the user.
See also chapter 6.14.

Return value

The number assigned to the dynamic variable. Can be used to identify the variable in calls to **SetInitial**.

Example

```

implementation
var
  T : array [1..10] of TFloat;
  P : TFloat;

procedure PreCalc;
begin
  P := 1;
  T[1] := 10;
  T[2] := 10;
  ...
  AddDynVar(T[1], 10, 'T1', 'Temperature 1');
  AddDynVar(T[2], 10, 'T2', 'Temperature 2');
  ...
  AddDynVar(P, 1, 'P', 'Pressure');
end;

```

6.14.4 AddStatVar

AddStatVar is used to add static variables just before the simulation begins (i.e. after the user starts the simulation, but before **ModelEquations** is called the first time). **AddStatVar** is normally called in **PreCalc**.

Heading

```

function AddStatVar(State : TInteger; var Variable : TFloat;
  InitialGuess : TFloat; Name, LongName : PChar) : TInteger;
function AddStatVarExt(State : TInteger; var Variable : TFloat;
  InitialGuess : TFloat; Name, LongName : PChar; Min, Max : TFloat;
  DoPlot : TBoolean; ALabel : PChar) : TInteger;

```

Parameters

State	The state the static variable belongs to.
Variable	The declared pascal variable, which represents the variable in the model.
InitialGuess	The default guess on the static variable.
Name	Short name that appears on plots.
LongName	Long name, appears on the Guesses page in the Solver & Model settings window in the Simulation program.
Min	Minimum value the user can set the variable to (and limit for the static equation solver).
Max	Maximum value the user can set the variable to (and limit for the static equation solver).
	If Min = Max = 0 then no limits are set.
DoPlot	True: Plot/save is available to the user for this variable False: Plot/save is not available to the user for this variable.

Return value

The number assigned to the static variable. Can be used to identify the variable in calls to **SetGuess**.

Example

```
implementation
var
  T : array [1..10] of TFloat;

procedure PreCalc;
begin
  ...
  AddStatVar(1,T[1],10,'T1','Static Var 1');
  AddStatVar(1,T[2],10,'T2','Static Var 2');
  ...
end;
```

6.15 Mathematical text

WinDali includes possibilities to make text appear in the Simulation Interface program with Greek letters and super- and subscript.

To use this facility, you have to specify the **Name/LongName** parameter in calls to **Add...** functions using the following special characters:

- ; Start/end special character section.
- _ Move text down. Creates subscript or moves text back to normal after a superscript.
- ^ Move text up . Creates superscript or moves text back to normal after a subscript.
- A string specifying a Greek letter (see below).
Writing the string in lowercase creates the corresponding Greek letter in lowercase.
Writing the string in uppercase creates the corresponding Greek letter in uppercase.

Examples:

'c;_;p;^; [kJ/kg-K]' will display as: c_p [kJ/kg-K]:

'alpha;_;i;^; [W/m;^;2;_;K]' will display as: α_i [W/m²K]:

The following table shows the strings to include Greek letters:

String	Greek lowercase	Greek uppercase
'alpha'	α	A
'beta'	β	B
'gamma'	γ	Γ
'delta'	δ	Δ
'epsilon'	ϵ	E
'zeta'	ζ	Z
'eta'	η	H
'theta'	θ	Θ
'iota'	ι	I
'kappa'	κ	K
'lambda'	λ	Λ
'mu'	μ	M
'nu'	ν	N
'xi'	ξ	Ξ
'omicron'	\omicron	O
'pi'	π	Π
'rho'	ρ	P
'sigma'	σ	Σ
'tau'	τ	T
'upsilon'	υ	Y
'phi'	ϕ	Φ
'chi'	χ	X
'psi'	ψ	Ψ
'omega'	ω	Ω

6.16 Debugging

For debugging purposes you can call the procedure **DebugMsg**:

Heading

procedure DebugMsg (Msg : PChar; Num : TFloat)

Parameters

Msg Message to display
Num An optional number to display (if Num is larger than 1E299 then the number is not displayed).

Shortcut

Write **dbg** and press **<Ctrl>+J**.

Example

```
DebugMsg ('Pressure', P);           {will display for ex. 'Pressure = 1.2'}
DebugMsg ('In ModelEq', 1E300);    {will display 'In ModelEq'}
```

The messages you write with **DebugMsg** will appear on the Debug page in the Message Window in the Simulation Interface Program.

7 Common problems

When compiling a model in WinDali the following problems can occur:

Problem	Solution
Problems compiling a model and thereafter running it	When you open a model be sure to use File Open Model. Models are always saved as projects (not in single files).
Problem compiling a model (can't create model file)	Remember to close the Simulation program before compiling the model currently loaded in the Simulation program. If you always run the Simulation program from the Model Editor this problem should not occur...

8 Using refrigerant equations

With WinDali comes a package with equations for the thermodynamic and thermophysical properties of 45 refrigerants. These equations can be directly used in your models.

To include one of the refrigerants do the following:

1. Include **CRefrigWrapper** in your uses clause
2. Create a **TRefrigerantWrapper** object
3. Use the equations
4. Free the object.

All these things will automatically be included if you select File|New|New Refrigeration Model when you create your model.

You can inspect **CRefrigWrapper.pp** in the **\lib** directory to see which functions you can call with a **TRefrigerantWrapper** object, browse the documentation installed with WinDali, or just use the Procedures tab in the Model Manager in Model Editor.

9 WinDali Model Editor

WinDali Model Editor is a text editor with capabilities to interact with Free Pascal Compiler. Free Pascal Compiler (FPC) is a freeware 32-bit compiler, compatible with Borland Turbo Pascal™, and partially compatible with Borland Delphi™.

FPC is available from the following web-address: <http://www.freepascal.org/>

FPC is an ongoing project, so you should check for updates at the address above.

Documentation and licensing information about FPC can be found at the above web-address.

When you start Model Editor and select File|New|New Model you get the following screen:

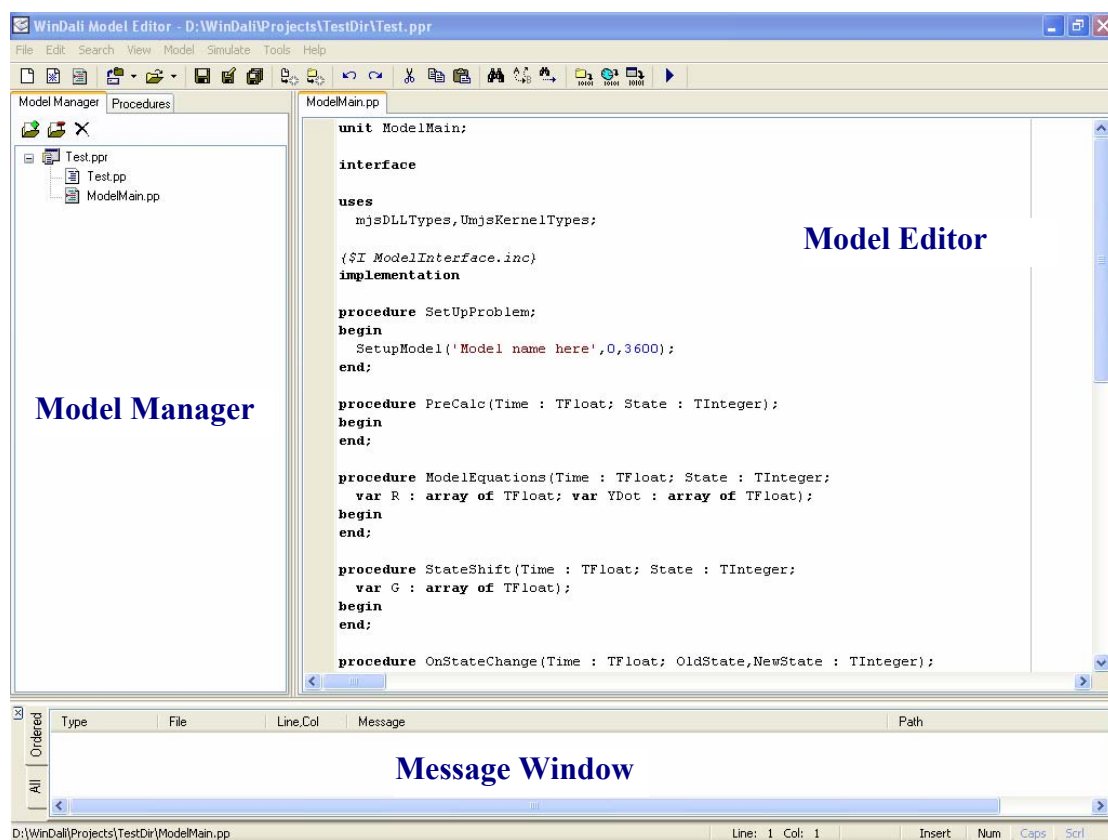


Figure 11. WinDali Model Editor.

- **Model Manager** is an overview of the files in your current model. When you compile your model, only the files listed in the Model Manager and the libraries (units) used by these files will be compiled (i.e. other files you may have opened in the Model Editor will NOT be compiled). When you double-click on a file in Model Manager, the file will open in the Model Editor.
- **Model Editor** is where you edit the files in your model. The Model Editor can be customized in numerous ways by selecting the Tools|Environment Options menu.

- **Message Window** will display messages from the compiler. If an error occurs you can double-click on the description of the error in the Message Window, and you will be lead to the place in the code that caused the problem.

Most menu items in the Model Editor should be self-explaining, but a few require some comments.

Menu	Item	Explanation
File	Save File as Template	The current file is saved as a template
Edit	Copy mode	Lets you select if the text copied in the editor should be in RTF (Rich Text Format), HTML or normal text. If a text is copied in RTF or HTML format to a word processing program, then the font and syntax highlighting is preserved.
Search	Insert/Goto Bookmark	You can define up to 10 bookmarks in your code. These bookmarks are not saved with the file.
Model	Save Model as Template	Allow you to save the current Model as a template. The template will be selectable from the File New New From Template menu.
	View Source	Opens the Model source as read only. This is only for inspection.
	Options	Displays the Compiler Options dialog – see later.
	Compile, Build, Build all	"Compile" recompiles files that have been changed since last compile. "Build" recompiles all files even though they haven't changed since last compilation. Compile and Build are direct calls to FPC; but as Build does not always produce the expected result, Build All has been added to the compile options. Build All ensures that all binary files are deleted before the compiler starts compiling. This forces the compiler to recompile all files the model depends on.
Tools	Environment Options	See later
	Find Compiler	If the program for some reason could not find the Free Pascal Compiler when it started, you can manually start a new search by selecting this menu.
	Macro	Press <Ctrl>+<Shift>+R to start recording a macro. When you have finished recording press <Ctrl>+<Shift>+R again. To play the macro press <Ctrl>+<Shift>+P. You can only have one macro at a time, and it can not be saved.
	Configure Tools	Allow you to add items to the Tools menu; where an item starts another program. When you add a tool you can use the following macros: %EXEPATH Path to Model Editor (default <code>c:\WinDali</code>) %EDITNAME Name of the current file in the Code Editor %EXENAME Name of the model file – i.e. the name of the resulting file when the model is compiled

9.1 Compiler Options

Compiler options apply only to the model you are currently working with. Options can be saved and loaded and you can change the default options.

When you select the Model|Options menu you will see the following dialog:

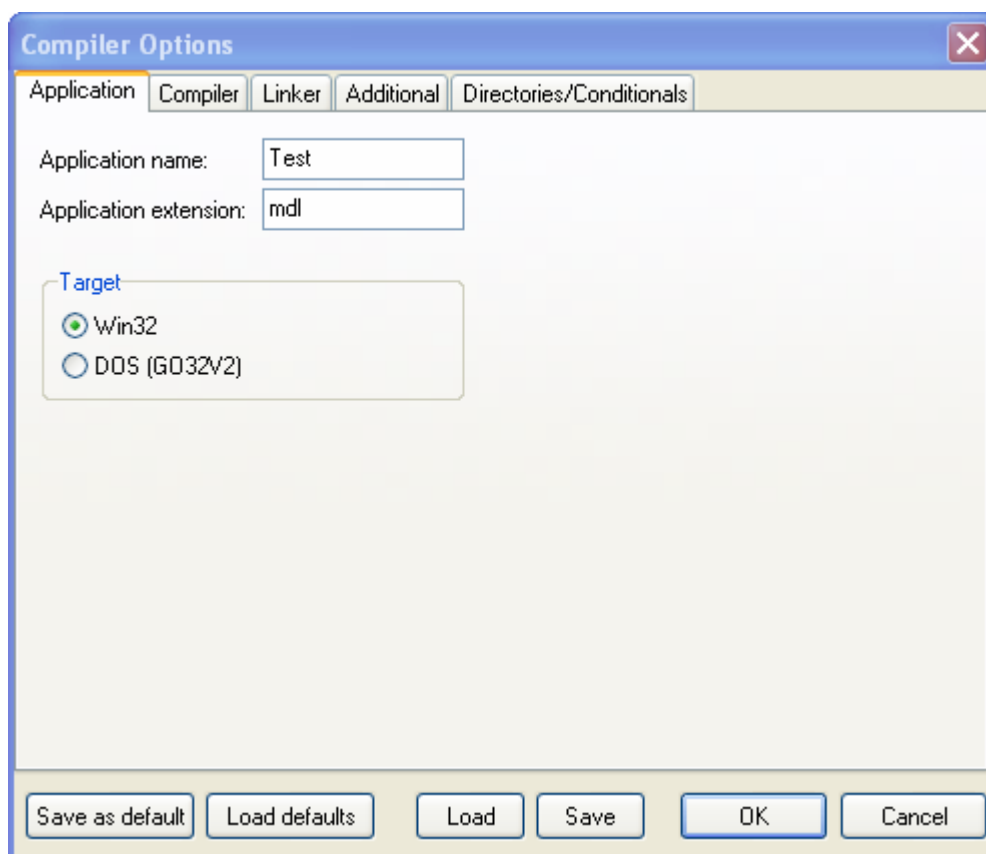


Figure 12. Compiler options.

The dialog contains a number of possible settings that mostly has to do with FPC. The most important settings will be explained shortly below. For more information see the FPC documentation.

Application

Application name	Name of the resulting file, when the model is compiled.
Application extension	Extension of the resulting file, when the model is compiled.
Target	
Win32	Build Win32 application (or DLL)
DOS	Build DOS application

NOTE: to use both targets, you need the Win32 and the DOS version of FPC. WinDali only comes with the Win32 version.

Compiler

Code generation	
Optimize for speed	Optimize code for speed (default)
Optimize for size	Optimize code for size of compiled file
Level 1 optimizations (quick optimizations)	Only simple optimizations, but fast compilation.
Level 2 optimizations (Level 1 + some slower optimizations)	More time consuming and extensive optimizations (default)
Level 3 optimizations (Level 2 + uncertain optimizations)	Should be used with caution. The optimizations could cause erroneous behaviour of your code (read the FPC documentation).
Runtime Errors	
Range checking	Checks that array and string indexes are within bounds (default on).
I/O checking	Checks for I/O errors after every I/O call (file operations) (default on).
Overflow checking	Checks for numerical overflow in TInteger operations (default on).
Debugging	
Generate browse info	Generates information that can be used by a code browser/debugger.
Include local	Includes local symbols in the browser information.
Syntax Options	
Delphi compatible	Compatible with Delphi (default on).
TP 7.0 compatible	Compatible with Turbo Pascal.
Delphi 2 extensions	Some extensions specific for Delphi version 2 are supported.
Support C-style operators (*=, +=, /= and -=)	Support expressions like <code>x += 1</code> (which in standard Pascal has to be written as <code>x := x+1</code>)
Gpc (Gnu Pascal Compiler) compatible	Compatible with GNU Pascal Compiler
Support label and goto commands	Allow use of label and goto as in standard Pascal (default on).
Messages	
Show hints	Show the hints the compiler might return (default on).
Show warnings	Show the warnings the compiler might return (default on).
Show notes	Show the notes the compiler might return (default on).
Show general information	Show some general information about the compilation (default on).

Linker

Debugging	
Generate dbg info for use with GDB	Generate information that can be used with GNU debugger
Generate dbg info for use with DBX	Generate information that can be used with DBX
Use the heaptrc unit	Use the heap-trace unit
Exe and DLL options	
Generate Console application	Create console application (like old time dos applications – but 32 bit)
Include .reloc section	Includes .reloc section in the executable. Default on for DLL's.
Misc	
Strip symbols from executable	Strips all symbols from an executable. Default off for a DLL.
Generate profiler code for gprof	Generate code that can be used with GNU profiler.
Omit linking	Skip the linking stage. This can then be done manually after compilation.
Linker output	
Do not delete generated assembler files	Prevents deletion of assembler files created by the compiler. You can select between several types of assembler.
Memory	
Size of reserved heap space	Size of heap space the programs reserve at startup (default 8000000 bytes)
Stack size	Maximum size of the stack (default 1048576).
Image base	Preferred load address of the compiled image (default \$10000000)

Additional

Additional options – read FPC documentation.

Directories/Conditionals

Output directory	Directory where executable/DLL should be placed.
Unit output directory	Directory where compiled units should be placed.
Unit directory	Directory where compiler should look for unit source files.
Library directory	Directory where compiler should look for compiled unit files.
Include directory	Directory where compiler should look for files included with {\$I filename} compiler directive.
Object directory	Directory where compiler should look for object files included with {\$L filename} compiler directive.
Conditional defines	Compiler conditional defines that should be enabled for all files in the model.
Conditional undefines	Compiler conditional defines that should be disabled for all files in the model.

9.2 Environment Options

When you select the Tools|Environment Options menu you will see the following dialog:

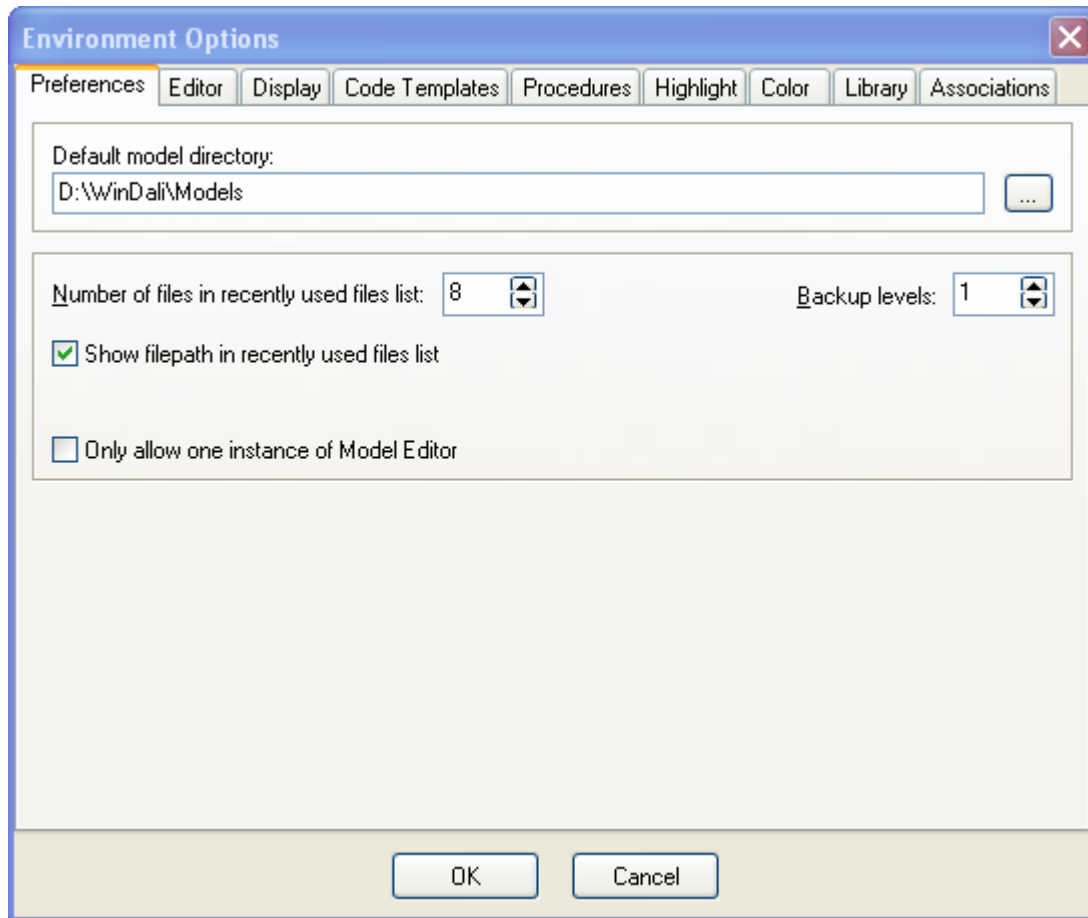


Figure 13. Environment options.

Preferences

Default model directory	Where you want Model Editor to look for your models as default
Number of files in recently used file list	Number of files the Model Editor keeps in the recently used file list.
Backup levels	Number of backup files to keep. One backup level creates backup files names filename.~ext. two backup levels keeps two backup files named filename.~ext and filename.~ext_1, etc.
Only allow one instance of Model Editor	If on then double-clicking on a .pp file in a filemanager will open files in the Model Editor already open and not open a new instance of the Model Editor program.

Editor

Editor Speedsetting	Can be Default, which selects standard windows shortcuts for common menu items, or IDE classic, which selects shortcuts as known from Turbo Pascal™.
Undo limit	Number of undo actions to remember.
Block indent	Number of spaces to move when moving blocks
Extensions in Open/Save dialog	Which file types you want to include in the Open and Save dialog box (installed highlighter extensions are added automatically).

Display

Visible right margin	Show a gray line to indicate the right margin
Right margin	Number of characters before right margin line is drawn
Visible gutter	Show visible gray area to the left. This area is used to display line numbers and bookmarks.
Gutter width	Width of the gutter in pixels.
Show line numbers	Shows line numbers in the gutter.
Editor font and size	The font to use in the editor.
Editor Window	
Tab style	Style of tabs, which are used to select the files in the editor.
Multiline tabs	Allow the tabs to stretch over several lines.
Hottrack tabs	Highlights the caption on a tab when the mouse is moved over it.
Ragged right	Specifies whether rows of tabs stretch to fill the width of the control.
Toolbar images	
Type	Can be Default: Default windows icons Blue: More colorful icons

Code Templates

Code templates are pieces of code you can insert into your code by writing only part of the code and then press <Ctrl>+J. Try writing “**be**” (without quotes) in the editor and press <Ctrl>+J – you will see it expands into:

```
begin
```

```
end;
```

You can add your own pieces of code by pressing Add and fill in a Shortcut (i.e. the shortcut you write in the editor, for example **be** in the example above) and a description (that will help you remember what the shortcut does). The actual code you want the shortcut to generate should be written into the editor on the Code Templates page. You specify where you want the cursor to be after you have applied the template by writing a “|” into the editor. Inspect the templates that already in the program to see how it is done.

If you use Borland Delphi™ you can use the templates from Delphi by copying the file Delphi32.dci (located in `\Borland\Delphi\bin` directory) to CodeTpl.dci.

Highlight

Here you install which syntax highlighters you want Model Editor to use – and which extensions you want to associate with each highlighter. To add a highlighter you check it in the list box, and perhaps change the Filter name and extensions for that highlighter.

The Filter name is the text you see in the Open dialog box and the extensions is a list of extensions separated with semicolons, which you want to the highlighter to work on.

You can make your own (primitive) highlighter by using the General highlighter. For the General highlighter you can specify keywords, comment style and string delimiters.

Color

Enables you to specify the color and font style of different code elements for each highlighter.

Library

Here you specify directories that apply for all models:

Unit directory	Default directory where compiler should look for unit source files.
Library directory	Default directory where compiler should look for compiled unit files.
Include directory	Default directory where compiler should look for files included with {I filename} compiler directive.
Object directory	Default directory where compiler should look for object files included with {L filename} compiler directive.

Associations

In this page you can specify the file extensions that Model Editor should be associated to. Note that if you uninstall WinDali, these associations will still exist in the registry. Before you uninstall WinDali you should deselect any associations you have created. This will delete the corresponding keys in the registry.

10 WinDali Simulation Interface

When you start the simulation program you will see the following screen (not all windows will be expanded):

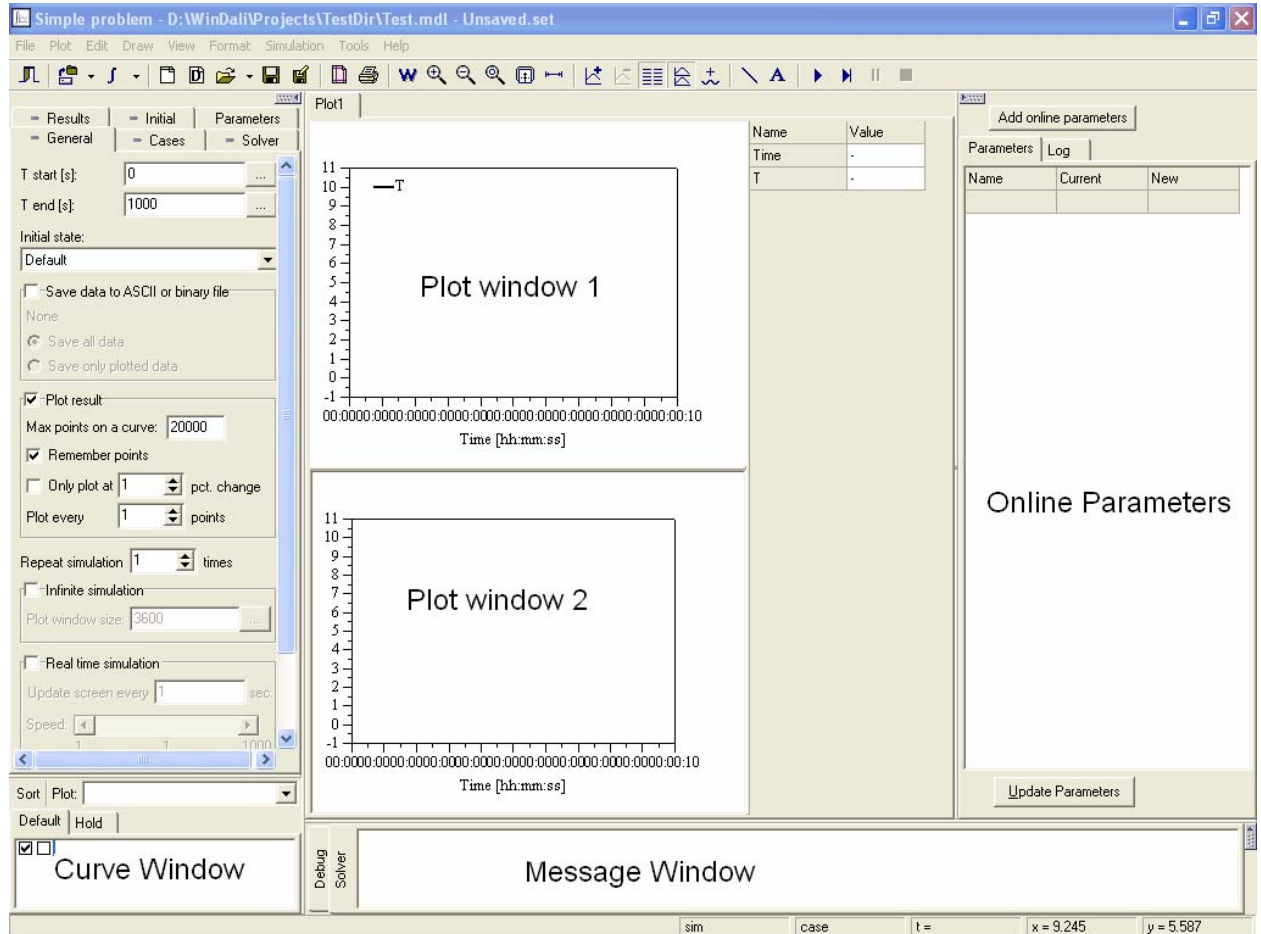




Figure 14. Simulation Program.

The items in the program are:

- Plot Window 1 and 2, where results are plotted. You can add as many plots as you want. Normally only Plot Window 1 is open, but you can display number 2 by pressing .
- Curve Window, where the variables you can plot will be displayed. If both plot 1 and 2 are visible, then the first column checkboxes select curves in plot 1 and the second column selects curves in plot 2.
- Message Window, which has two pages:
 - Solver, where messages from the solver are displayed.
 - Debug, where debug messages from the model are displayed.
- Data window where you can inspect the numerical values of the selected variables. You bring up the data window by pressing .
- Online parameters where you can change parameters during the simulation (see chapter 10.2).

The look of the Simulation Interface program will change according to the model that is loaded. As default the following pages will be created:

General

TStart	Start time for simulation
Tend	End time for simulation
Initial state	The state the simulation should start in
Save data to ASCII or binary file	Saves data to an ASCII or a binary file. When you start the simulation, a dialog where you can select what to save will appear. If you save to a binary file you can use the Post Process application to display the data when you have finished – see chapter 12
Save All Data	Saves all data
Only plotted data	Saves only data which is plotted (see below)
Plot result	If not selected no data will be plotted. If you save data to disk, and you are not interested in the plots, this will speed up the simulation.
Max points on a curve	Decides how many points a curve can contain. Each point on a curve is stored in memory during simulation. If you work with very long simulations you could eventually fill up the computer memory, if this number is too large. When this number is reached, the program starts to delete old points when a new point arrives. Note that if you do not save data to disk you will lose old data when this number is reached.
Remember points	If unchecked the points on curves you are not watching (i.e. unchecked curves in the curve window) will not be stored.
Only plot at X pct. change	If checked then points are only changed if the slope of the curve changes more that the percentage specified.
Plot every X point	If for example 10 then only every tenth point will be plottet
Repeat simulation	Specifies the number of times the simulation should be repeated. Together with Update initial values (see later) you can use this to get a periodic stationary solution.
Infinite simulation	If checked then the simulation runs until you press Stop. The plot window size defines the time range shown in the plot windows.
Real time simulation	If checked then the simulation will be run in real time. You can also specify how often the screen (plots and data) should be updated. When running real time simulations, you can control the speed of the simulation by adjusting the slider. The maximum speed depends on the screen update value you specify (it equals 1000·the screen update value). A speed of 1 means that one simulated second will take 1 second in real time. A speed of 1000 means that 1000 simulated seconds will take 1 second in real time. The max speed will also depend on how fast the solver is able to actually solve the equations, so the value of the speed slider is not always accurate.
Messages	Decides how to display messages from the solver (debug messages will always be added to the Message Window). <ul style="list-style-type: none">• None, no messages are displayed (if you after a simulation change Messages from None to Display you will be able to see debug messages).• Screen, messages are shown on screen• File, messages are saved to a file you specify.

Solver

Use external static...	Use an external static equations solver to solve the static equations at the initial point, and to solve the static equations the first time after a state shift. This functionality is included because the solver included in WinDali has a fast but not always stable static equations solver. Sometimes it is necessary to solve the static equations the first time by using the more stable (but also slower) external equation solver. After the first solution is obtained, the internal static equation solver is normally sufficient.
External solver	A list of external solvers that can be used
Use fixed sample time	If checked the solver will be forced to create a solution at the sample time you specify – i.e. if you specify a sample time of 10 seconds, then the solver will make a solution at Time=10, Time=20 etc. but it might also produce solutions in between the samples.

The contents of the Solver page also depends on the solver you have loaded (see 10.4).

Initial

This page will only be created if the model has dynamic variables. The page allows you to change the default initial values for the dynamic variables.

The program automatically creates a checkbox called Update initial values, that when checked:

- Updates the initial values to the last solution point in the simulation when the simulation ends.
- Updates the initial state to the state the model was in, when the simulation ended.

Guesses

The Guesses page will automatically be created if the model contains static variables. The page allows you to change the default guesses for the static variables in the different states.

The program automatically creates a checkbox called Update guesses, that when checked updates the guesses on the static variables in all states. The guesses in a state are updated to the first solution found when the model was in that state.

The page also has a checkbox called “No model guess”, which if checked prevent the loaded model from providing any guesses it might want to set (i.e. calls to [SetGuess](#) from the model are prevented).

Results



















If the model includes extra variables that are not plotted, their value will be displayed on this page when the simulation stops.

Cases

- see chapter 10.3

10.1 Menu commands






File menu

Item	Explanation	Shortcut/toolbar
Open Model	Open a new model. The icon on the toolbar keeps a list over previously opened models.	 or 
Open Solver	Loads a new Solver.	 or 
New	Clears the filename that the user previously has saved the settings under.	Ctrl+N,  or 
Default	Load the settings from the model-file and ignore if a default settings-file exists (see below)	 or 
Open	Open a settings file. A settings file includes all information of the parameters the user can change in the Simulation program. Settings files have extension .set	Ctrl+O,  or 
Save	Save settings.	Ctrl+S,  or 
Save as	Save settings with a new file name.	 or 
Print Report	Print report of a simulation. The report can contain settings and plots. The print will contain the current date and time.	 or 
Exit	Exit the simulation program.	 or 





If you save a settings file with the same name as the model file (but with extension **.set**) this file will be loaded as default when the model is loaded.

The Plot, Edit, Draw, View and Format menus are explained in the accompanying help file (select the Help|Plot menu), except for the following items in the plot menu:

Plot menu

Item	Explanation	Shortcut/toolbar
Add plot	Add a new plot page	
Delete plot	Delete a plot page	
Data window	Show/hide data window	
Second plot	Show/hide the second plot on a page	
Create curve	Create a curve by combining two existing curves (phase plot)	
Next plot	Move to next plot	F9
Previous plot	Move to previous plot	F10

Simulation menu

Item	Explanation	Shortcut/toolbar
Start	Start simulation	F2 / 
Step	Make one step	F3 / 
Pause	Pause the simulation	F4 / 
Stop	Stop simulation	F5 / 

Tools menu

Environment options	See below
Create distributable copy	See chapter 13
Configure tools	Allow you to add program items to the Tools menu.

The environments options dialog allows you to customize the Simulation Program.

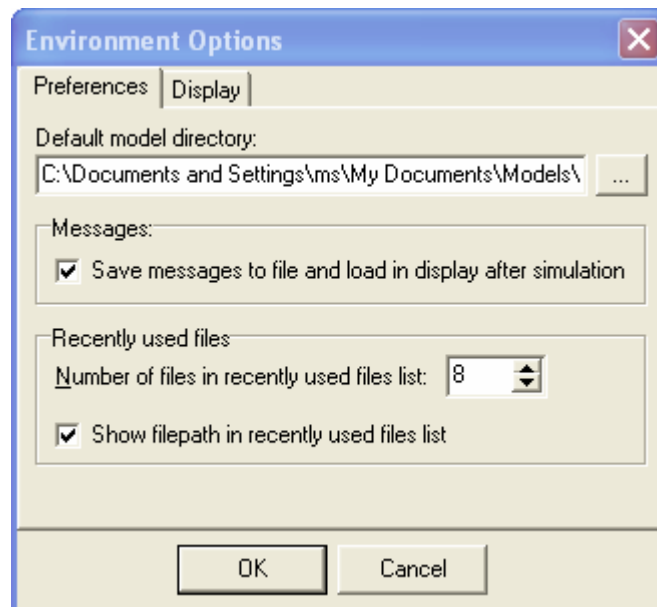


Figure 15. Environment Options.

Preferences

Default model directory	Default location of your models
Save messages to file and load in display after simulation	Makes the simulation run faster. If the solver (Dali.sol) encounters problems it will react by outputting a lot of information. If you choose to display messages while the simulation runs (option unchecked) then the simulation might be very slow.

Display

Colored checkboxes	Makes the checkboxes in the Curve Window the same color as the curves.
Toolbar images	Can be: Default: Default windows icons Blue: More colorful icons

10.2 Online parameters

Changes to parameters while the simulation is running will not affect the parameters in the model (at least not until the simulation is restarted). If you want to experiment with parameter changes while the simulation is running you need to do it through the Online parameters window.

To change parameters you first have to select the parameters into the window by pressing Add online parameters. This will bring up a dialog where you select the parameters you want to vary. The type of parameters that can be varied are integer, float and values in enumerated choice parameters.

When the simulation is running you enter new values for the parameters in the New column and press Update parameters when ready. This will change the columns so that the column containing the new parameters will become the Current column, and the old Current column will become the new New column. This enables you to keep track of the old parameter values while you enter the new values.

Every time you press the Update parameters button, the changes will be recorded together with the time the changes was effected and a line will be drawn in all plots. The log can be printed together with plots and settings when you select File|Print report or you can print/save it directly from the Log tab in the Online parameters window.

10.3 Varying parameters

If you press the Vary button on the Cases page you can vary one or more of the floating point or integer parameters. This gives possibility to easily perform parameter studies on the model. If the model has a parameter called Pressure and you want to vary it form 1 to 10 with a step of 1, the simulation will automatically run 10 times with Pressure having the value 1,2,3...10. The results from the simulations are saved to one or more files.

When you press the Vary button you will see the following dialog:

Group	Name	Vary	From	To	Step
Material	Density [kg/m ³]	<input type="checkbox"/>	0	0	0
	Specific heat [J/kg K]	<input type="checkbox"/>	0	0	0
	Conductivity [W/m K]	<input type="checkbox"/>	0	0	0
Geometry	Volume [m ³]	<input checked="" type="checkbox"/>	0.001	0.005	0.001
	Surface area [m ²]	<input type="checkbox"/>	0	0	0
Heat transfer	Heat transfer coefficient [W/m ² K]	<input type="checkbox"/>	0	0	0
	Cold temperature [°C]	<input type="checkbox"/>	0	0	0
	Hot temperature [°C]	<input type="checkbox"/>	0	0	0

Select dependent variables

☒ T

☐ Cp

Clear all OK Cancel

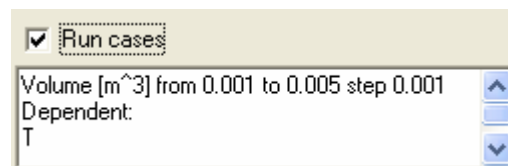
Figure 16. Varying parameters.

In the dialog Volume is varied from 0.001 to 0.005 with step 0.001, i.e. the simulation has to run 5 times or there are 5 **cases**. The dependent column allows you to select, which variables you want to save in the resulting files. You can specify to vary one or more parameters. If you for example also selected to vary Cold temperature from 20 to 25 with step 1, you would generate 30 cases (and thereby 30 simulations):

Volume	Ambient Temperature
0.001	20
0.002	20
:	:
0.005	20
0.001	21
0.002	21
:	:
0.005	21
:	:
:	:
0.001	25
0.002	25
:	:
0.005	25

When you vary more than one parameter, then the cases are created by varying the parameters in the order they appear in the dialog (Volume is varied before Cold temperature, because Volume appears before Cold temperature in the dialog).

When you select OK, you will see the following information in the Cases page:

**Figure 17.** Info on Cases page.

To actually run through the cases you have to check Run cases.

When the simulation is started you will be asked how to save the data generated from the simulations:

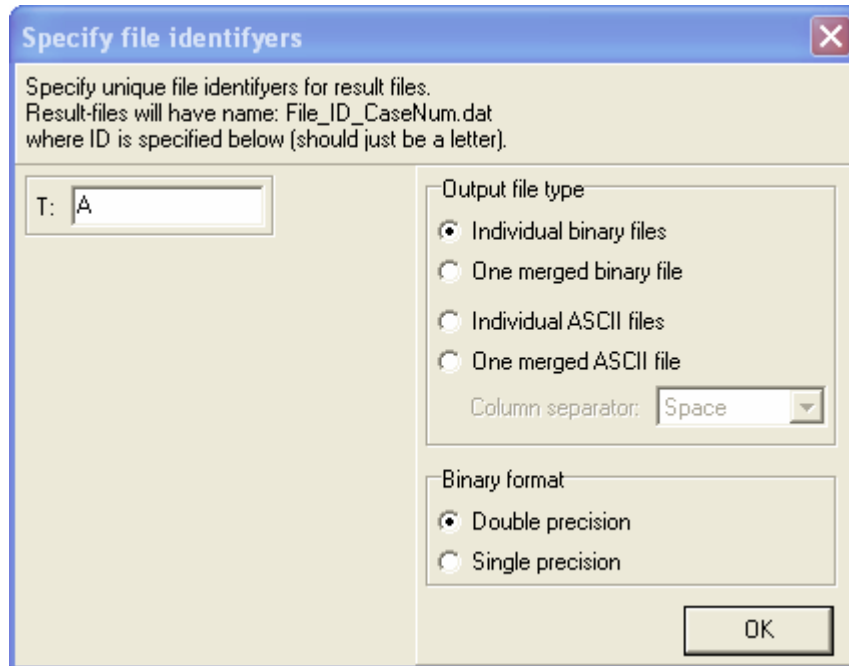


Figure 18. File identifiers and file type.

For each of the dependent variables, you have specified, there will be created one or more files. For each dependent variable, you have to specify a unique identifier – in the dialog T has been assigned the identifier A. The file(s) created will have the name(s) File_ID_CaseNum.dat. If you select individual files (binary or ASCII), and follow the example from the dialogs, the files created will have the names: File_A_1.dat, File_A_2.dat, File_A_3.dat, File_A_4.dat and File_A_5.dat.

You can select from these file types:

Individual ASCII files

Data will be saved to individual ASCII files, one file for each case, and one file for each dependent variable. The files has the following structure:

```
Time1;Dep
Time2;Dep
...
TimeN;Dep
```

Where **Dep** is the dependent variable for that file, and ';' represents the selected column separator. In the example above 5 files would be created, and one file would contain:

```
Time1;T_1
Time2;T_2
...
TimeN;T_N
```

One merged ASCII file

Data for the different cases will be saved to one merged ASCII file for each dependent variable. One file has the following structure:

```
Time1;InDep1;InDep2;...InDepM;Dep_case1
Time2;...
...
TimeN;...
Time1;InDep1;InDep2;...InDepM;Dep_case2
...
...
TimeN;InDep1;InDep2;...InDepM;Dep_caseL
```

Where **InDep1...InDepM** is the independent variables (i.e. the parameters that are varied).

In the example above the file would contain:

```
Time1;0.001;T_1
Time2;0.001;T_2
...
TimeN;0.001;T_N
Time1;0.002;T_1
...
...
TimeN;0.005;T_N
```

If Cold temperature also was varied the file would contain:

```
Time1;0.001;20;T_1
Time2;0.001;20;T_2
...
TimeN;0.001;20;T_N
Time1;0.002;20;T_1
...
...
TimeN;0.005;20;T_N
Time1;0.001;21;T_1
...
...
TimeN;0.005;25;T_N
```

Individual binary files

Data is organized exactly as for individual ASCII files, except that the binary files do not contain line breaks or column separators.

One merged binary file

Data is organized exactly as for one merged ASCII file, except that the binary files do not contain line shifts or column separators.

Binary files can be saved in Double or in Single format. Each value in the file takes up 8 bytes of space in Double format, while each value in the file takes up 4 bytes of space in Single format. All though Single format saves space, it also has fewer significant digits.

When you select OK in the dialog in Figure 18, you will be asked to specify a directory where you want to save the data. When the simulations end, an information file called CaseInfo.txt is written to this directory. It contains information about the created data files.

10.4 Dali solver

The default solver used by the simulation program, is a slightly modified version of the solver described in [1]. The solver has the following characteristics:

- Differential equation solver: 3 step, 3. order semi-implicit Runge Kutta (NT1 developed by Nørsett & Thomsen). Specially suited for stiff problems.
- Algebraic equation solver: Modified Newton iteration, which includes
 - Convergence control (method for keeping the same Jacobian in several iterations)
 - Divergence control (extrapolation method which extrapolates the variables into the convergence area at beginning divergence).
- Interpolation with 2. order splines, which is used for:
 - Guessing static variables in next step (by extrapolation).
- The location of discontinuities is found using a secant-method.

The following parameters can be set for the solver:

Parameter	Meaning
Write	Select between the following <ul style="list-style-type: none"> • Start, End and Disc. points: writes the solution at Start, End and Discontinuity points. • All: writes the solution at all points. • Debug: writes Jacobians, information on iterations, etc. This generates very extensive information.
Max iterations	Maximum number of iterations in solution of static equations.
Max Jacobians	Maximum number of Jacobians the solver is allowed to calculate each time the static equation set is solved.
Relative error	Convergence criterion.
Max step size	The maximum step size the solver is allowed to use.
Min step size	The minimum step size the solver is allowed to use.
Max number of rejected steps	How many times the solver is allowed to reject a step, change the step size a try another step.
Use extern static solver	The build in static equation solver is fast, but can also cause problems. More stable (but also slower) solvers can be selected.

If you encounter problems solving the equations, a good idea is to try one or more of the following:

- Decrease the maximum step size
- Increase/decrease the Relative error
- Try an extern static equation solver
- Increase the number of iterations and/or the number of Jacobians.

When an extern static equation solver is used, the maximum number of iterations should be increased. The maximum number of Jacobians has no influence when an extern static equation solver is used.

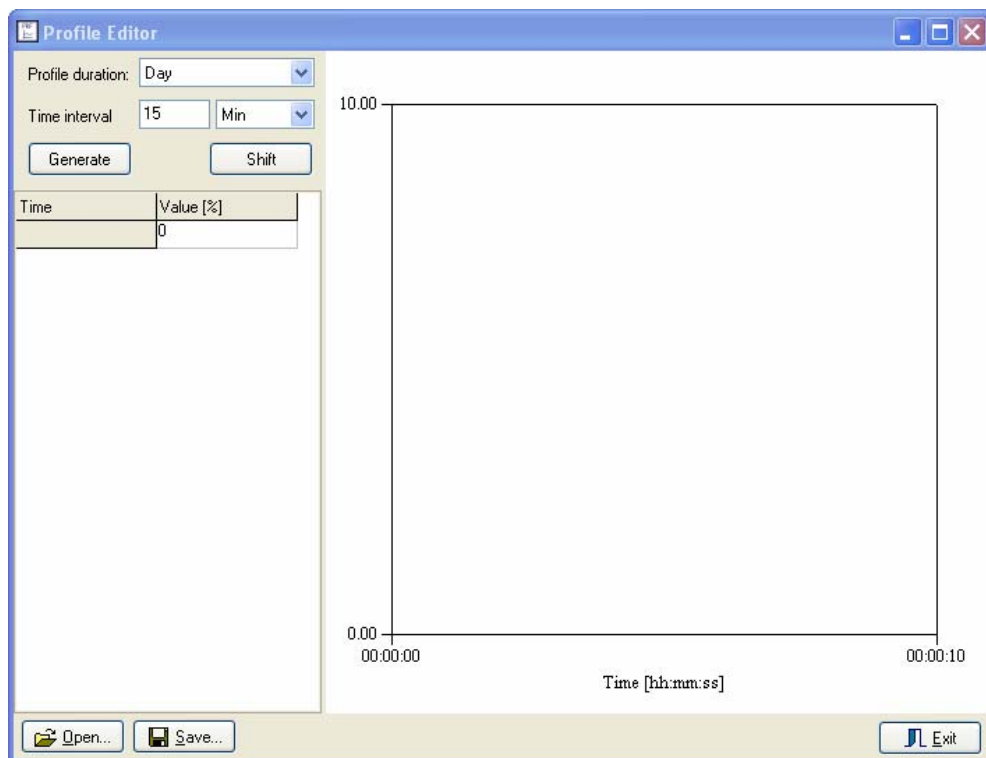
11 Using Profiles in models

When you create model it often happens that parameters in the model are not constant, but instead are in the form of time-series. This could for example be a load profile on a display case in a supermarket.

To easily be able to include such profiles in your model, WinDali has a helper-tool called Profile Editor you can use to create profiles, and a corresponding Pascal unit called **CmjsProfile.pas** you can include in your model to use the profiles.

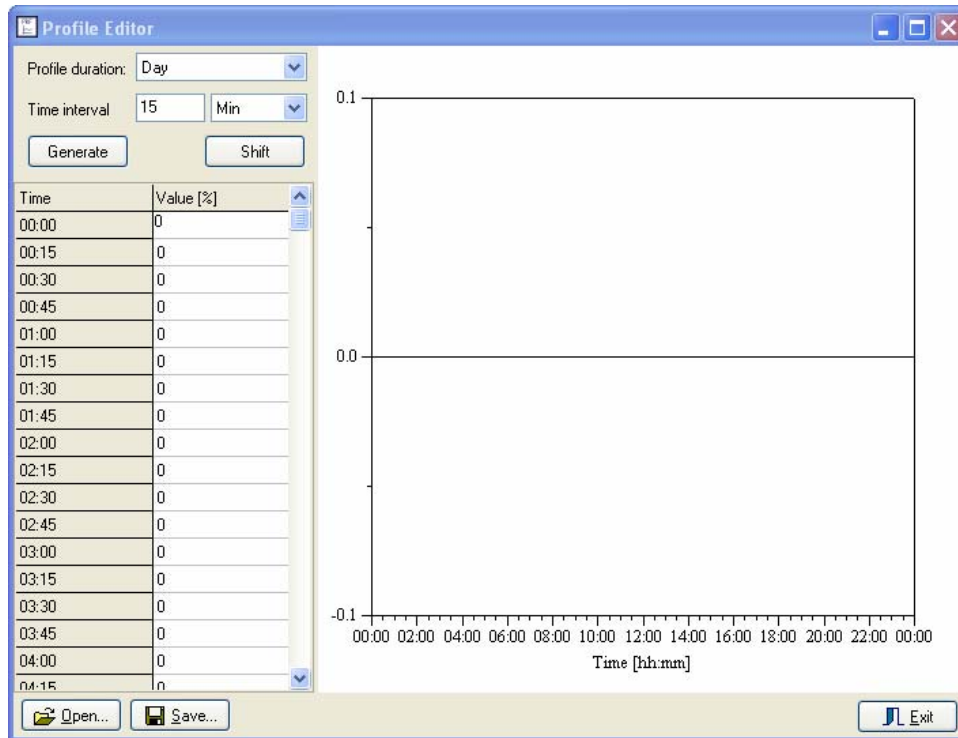
11.1 Generating profiles

When you execute the Profile Editor application, the following window appears:



You start by selecting the duration of the profile (when you simulate the profile will be looped) and by specifying the interval you want to provide data for.

When you press <Generate> you will see the following:



Note that now a table is created where you can enter values for the profile (in percent of a value you specify when you include the profile in a model), and that the profile will automatically be shown in the plot.

When you have entered the values, press <Save> to save the profile.

The format of a profile file is a simple ASCII file with the four first lines specifying the profile duration and the time interval:

The first line is equal to:

- 0: If duration is one minute
- 1: If duration is one hour
- 2: If duration is one day
- 3: If duration is one week
- 4: If duration is one month
- 5: If duration is one year

The second line is always equal to 1 (reserved for future use)

The third line is equal to

- 0: If time interval is given in seconds
- 1: If time interval is given in minutes
- 2: If time interval is given in hours
- 3: If time interval is given in days
- 4: If time interval is given in weeks
- 5: If time interval is given in months

The fourth line is equal to the specified time interval.

The rest of the file is just the entered percent-values – one value on each line.

11.2 Using profiles in a model

Included in the demos is a modeified example of the "Cooling of Block" example, where a profile is used for the ambient temperature. This demo illustrates several points:

- Using an Action button to open a file from your model
- Changing the caption of an Info-label
- Loading and using a profile
- Using `OnSaveSettings` and `OnLoadSettings` to store information in files created from within the model.

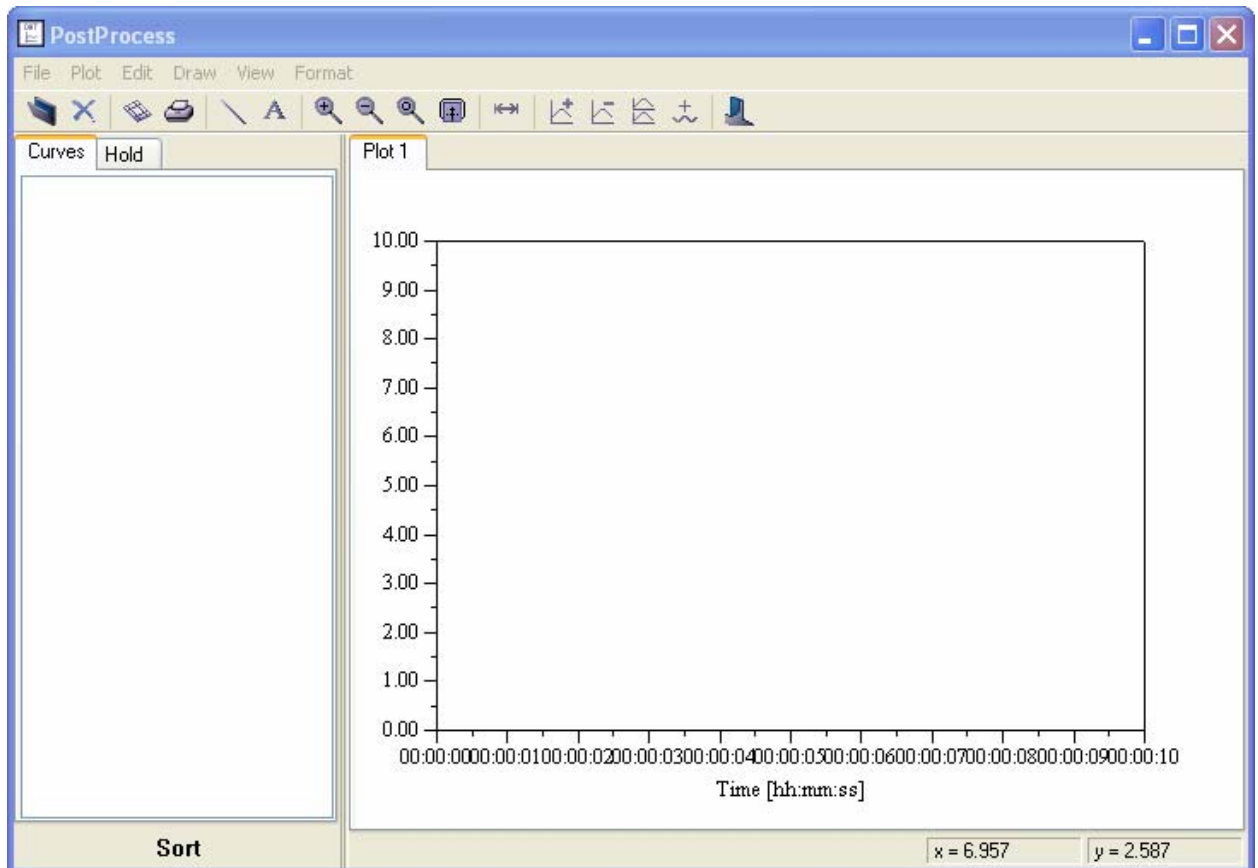
When you use profiles then remember to:

1. Create the profile object in `SetupProblem`
2. Initialize the profile in `PreCalc`

The profile-object uses a linear interpolation to find profile values at each time step. If you want to improve this the change the code in `CmjsProfile.pas` located in the `\Lib` directory.

12 Using Post Process

Post Process is a simple application you can use to display a binary data file. The program looks somewhat like the Simulation Interface program without the parameter pages:



The files you can open in Post Process must be files created when you select "Save data to ASCII or binary file" in the Simulation Interface program, and thereafter select a binary file.

13 Distributing models

You can easily distribute your models by going through the following steps:

- In the Simulation Program go to the Tools menu and select Create distributable copy.
- You will be prompted to select a directory where the files should be placed.
- All necessary files will be copied to that directory – including the current loaded model.
- To install the model on a different PC, just copy the files to a directory of your choice.
- To run the model, start Simulation.exe and load the `.mdl` file.
- If you want to distribute several models, just include the `.mdl` and `.set` files for those models.

The current settings file will also be included with the files. Remember that if you save a settings file with the same name as the model file (but with extension `.set`) this file will be loaded as default when the model is loaded.

The files that should (minimum) be in the distribution are:

- `Simulation.exe` Main program
- `Dali.sol` The default solver
- `NonLin.dll` DLL with external static equation solvers
- `RefrigFPC.dll` DLL containing refrigerant equations
- `WDMModelUtils.dll` DLL used by some models
- `SimIntPlt.hlp` Help file for main program
- `SimIntPlt.cnt` Used by help file.
- `' .mdl '` and `' .set '` files.

14 References

- [1] Askjær K.A. – *Numeriske metoder anvendt i DALI. En Differential-Algebraisk ligningsløser*; Master thesis; Refrigeration Laboratory, Technical University of Denmark, November 1985.
- [2] Free Pascal Compiler, <http://www.freepascal.org/>
- [3] Borland® Delphi™ 5 – *Object Pascal Language guide*; Inprise Corporation. 1999.

Appendix A Used file types

The following list shows the file types used in WinDali:

Name / Extension	Content
<code>*.ppr</code>	A model file. Contains among other things a list of the pascal source-code files included in a model.
<code>*.pp, *.pas</code>	Pascal source-code files. These files contains the model and other code. The model file binds these files together.
<code>*.inc</code>	Include files used by the <code>*.pp</code> and <code>*.pas</code> files.
<code>*.ppw, *.ow, *.cfg, *.err</code>	Files created during compilation. These files can safely be deleted.
<code>*.mdl</code>	Compiled model file. Ready to use in the simulation program.
<code>*.sol</code>	File containing a solver, which can be used by WinDali.
<code>*.set</code>	Settings file created by the Simulation program. Is used when settings and parameters for a simulation are saved.
<code>nonlinerr.txt</code>	File created if you use the external static equation solver and an error occurs. You can safely delete this file.

Appendix B Files and directories created during installation

All programs in WinDali are enabled to run in multi-user environments – and also on PC's where the user is not logged in as administrator. This means that all files/directories where the user should have write access are placed in the user's profile.

All files in the basic installation directory (default `c:\WinDali`) can be regarded as read-only.

During installation WinDali creates the following directories:

```
All Users\Application Data\IPU\WinDali\Lib
All Users\Application Data\IPU\WinDali\Templates
```

where " `All Users\Application Data`" is the path to the users application data directory, normally something like `c:\Documents and Settings\All Users\Application Data`

Note that this directory is hidden as default, so if you want to browse it, you should enable "Show hidden files and folders" in Windows Explorer, and press <Apply to All Folders> (Select Tools|Folder Options menu and go to the View tab within Windows Explorer).

As default new models are created in the `\CurrentUser\My Documents\Models` folder and the demos are placed in the `All users\Documents\Models\Demo` folder (during installation a link to the demo folder will be placed in the `\CurrentUser\My Documents\Models` folder).

You can easily change the default model folder, in the Model Editor and the Simulation Interface program (in both cases go to the Tools|Environment Options menu).

When you run WinDali, all internally used setting files will be placed in the `\CurrentUser\Application Data\IPU\WinDali\` folder.