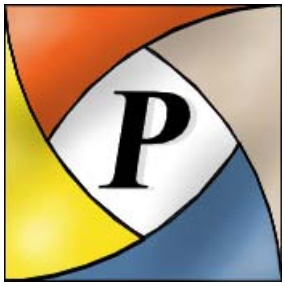
Institute for Energy Technology
OECD Halden Reactor Project



ProcSee

Graphical User Interface Management System

TUTORIAL

Microsoft Windows version

3.10

Institute for Energy Technology
OECD Halden Reactor Project

This document will be subjected to revisions in the future as the development of ProcSee continues. New versions will be issued at new releases of the ProcSee system.

Published by : Institute for Energy Technology, OECD Halden Reactor Project
Date : December 2015
Revision : 3.10

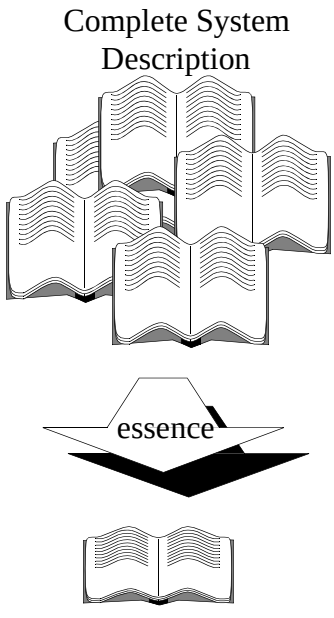
PROCSEE DOCUMENTATION

Table of Contents

1 Introduction

This chapter is a general description of this tutorial with a presentation of the contents and a description of the different conventions used in this document.

1.1 What is a Tutorial?



A tutorial, in our case, is a compressed documentation of how to create one or several complete applications using the ProcSee system. It is the intention to give the beginner a hands on experience of ProcSee and present terms and procedures to touch most of the different parts of the system. The tutorial will simplify the introduction to the system and it is recommended that the user works through it before retrieving more detailed information from the “User’s guide” and the “Reference manual”.

The tutorial is based on generating pictures to be used for process control applications. Note that ProcSee can be used to generate user interfaces for any type of application.

In most tutorials the goal is to work through one or more examples based on prepared files delivered with the system. These files are used to build a working application.

In addition to give a brief introduction to the system, it is also useful for the user to test out other more complex functionality by appending functions to the already working example. Using the working example as a template makes it efficient to test and debug added functionality.

The tutorial is divided into different chapters dealing with each of the modules and building blocks that the system consists of. The tutorial can therefore be used as a reference to get information about different ProcSee modules.

This tutorial is written on the assumption that the users are familiar with using applications on Microsoft Windows.

1.2 What to Expect from this Tutorial

This tutorial is dealing with most of the modules and utilities in the ProcSee delivery. The tutorial application, that will be the end product of working through this document, uses all the main building blocks for creating the required application files. The following list gives a short overview of what will be worked through in the tutorial:

creating directory The second chapter starts with creating a tutorial directory where all the files concerning this tutorial application will be located. Also the starting of the ProcSee system with the different modules will be described in this chapter. You will also find the procedures for copying the predefined files delivered with the system to this specific directory. These files will be used for start-up of the example application.

environment variables A reference to where the binary files for ProcSee is located and a help for the ProcSee system manoeuvring to the right resource files from a user defined place, is called **PROCSEE_DIR**. In addition the two Environment variables, **ARCH** and **PATH** must be defined. The first one is used by ProcSee to detect which machine architecture it is running on, and the second is to avoid typing the complete path name to the executable files.

starting the system The procedures for starting the complete ProcSee system requires only two command line inputs in a terminal window. The start-up order of the ProcSee processes is not fixed. Usually the *Run-Time Manager (RTM)* is started first. A name server called Control is already running. When the RTM is running, the *Graphics Editor (GED)* can be started.

default resources This chapter gives an overview of the default resources (colours, fonts etc.) in the ProcSee system and also the necessary tasks to be done for initial preparation.

introduction to GED Explanation of initial settings like background colour, world coordinates, snap etc.The drawing editor is used to draw a few objects and then add different attributes to these objects.

design of classes By using the GED two different ProcSee classes are constructed and placed in a library for later use. The class library can be thought of as a set of building blocks to be used to build pictures in GED.

design of picture After finished creating the different building blocks like configuration of the user's environment, classes, and programming the simulator, the process picture will be generated. The process picture consists of class instances, graphic primitives and application variables put together via the editor. After having completed the process picture, the functionality is tested by the editor test facilities.

the simulator A small simulator program, coded in standard C, is running as a stand alone process and is communicating with the RTM by use of standard ProcSee *Application Programmer's Interface (API)* functions. The task of this simulator is to periodically update the RTM with the variables that are used in the end picture.

running the application The application is expanded with a window and two pictures, two libraries, and a database. It is now a complete ProcSee application.

1.3 Conventions

The following conventions are used in this manual:

- courier bold is used for constants such as **.pctx**
- **%PROCSEE_DIR%** refers to the directory where ProcSee resides.
- <*appName*> means that *appName* is a user specified name.

- the information icon is used to highlight where more information about a specific topic can be found.

- the explanation mark icon highlights important points in the text.

1.4 How to Use this Manual

This is a document supporting the ProcSee features and capabilities by working through a complete application example based on a few system files delivered with the system.

- Start with page number one and follow the descriptions.
- Avoid implementing new features to the tutorial example until having finished the last page.

1.5 Definitions

application	See library, functions, windows, ...
attribute	An attribute is a variable local to the user interface, and does not have a corresponding variable in a user program. See section 9.9 in ProcSee User's Guide.
callback functions	User supplied functions that will be called by the API to inform the application code about changes in the connection with ProcSee.
class	A class is a reusable interface component. It is a picture, which may include attributes, functions, dialogues and graphic shapes, saved collectively as a class.
colour	A resource (see section 5.3 on page 27).
compile	Using the *pcc* program to generate a binary file from an ascii file.
font	A resource (see section 5.4 on page 28).
library	A container that can hold a number of graphic class definitions and resources.
pattern	A resource (see section 5.5 on page 29).
ProcSee connection	Mechanisms used to connect application code to the ProcSee RTM.
picture	The visible part of the user interface, displayed in windows.
resource	Colours, fonts and patterns, defined in a library or application.
toggle	Alternate between two states, e.g. on / off or open / closed.
update	Global update of picture in an application.

1.6 Requirements

Some prerequisites are required to be able to work through this tutorial. Check the list below and find out if there is something that your system is missing. Possible problems can be solved by your system administrator.

- The ProcSee system is properly installed according to the enclosed installation specifications.
- Your system is running a version of Windows NT which is supported by ProcSee.
- An editor for editing ascii files must be available
- Sufficient access privileges to the actual files and directories referenced in the tutorial.

Preparations

2

This chapter is dealing with a complete description of what is required before starting the ProcSee system; setting the ProcSee variables required, starting-up options, and copying the necessary files to a specified directory.

2.1 Configuration of a ProcSee Environment

In a *User Interface Management System (UIMS)* like ProcSee, the user need flexible possibilities to configure the system and interact with other systems according to his requirements.

The following subjects describe the preparation for starting a new and empty application based on predefined colours, fonts, and other standard graphics resources.

Environment Variables

On Windows NT, the **Environment variables** should already have been set to their correct values, if the system has been installed properly.

To make copying and system commands easier it is important to define a unique ProcSee **Environment variable** in the operating system. This variable is called **PROCSEE_DIR**, and is referred to by putting a % before and after the variable.

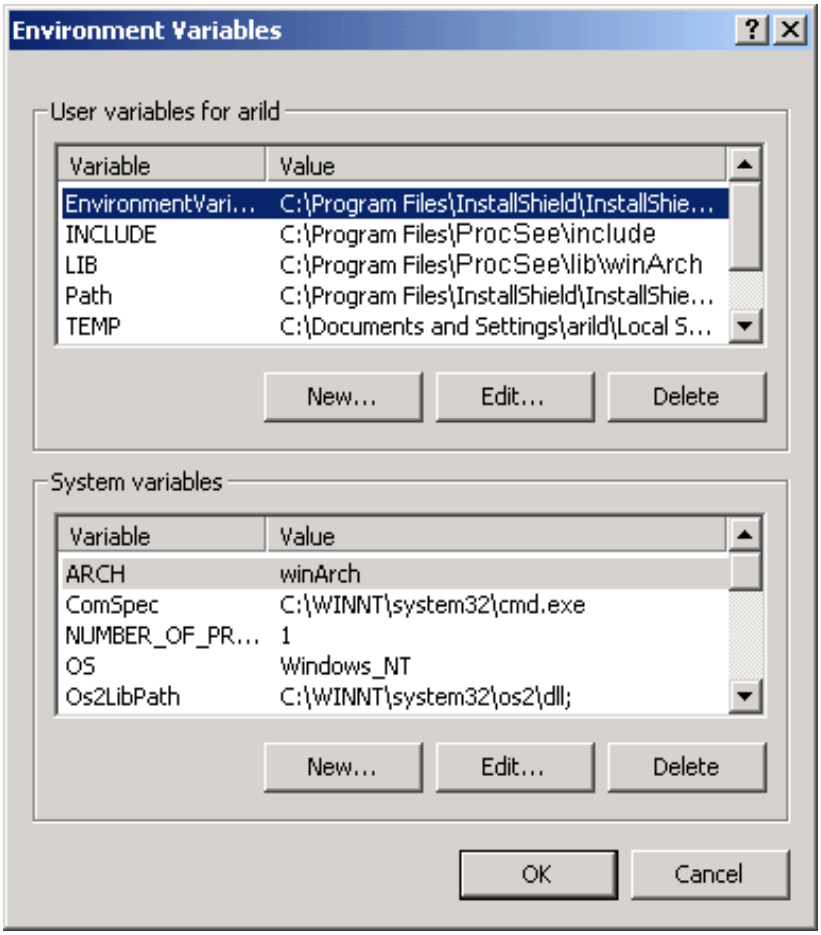
To set **Environment variables** on Windows NT, use the **Start** menu and chose: **Settings** and **Control Panel**. In the **Control Panel** window, select **System** and in the **System Properties** window, chose **Environment** or **Advanced** and then **Environment Variables...** (depending on the operation system on your computer) as shown in Figure 1 on page 8.

Figure 1

This is the System Properties window with Environment variables.

Important variables for ProcSee is: ARCH, Path and the PROCSEE_DIR.

When clicking on the different Variables, the variable Value is displayed in the Value window.



Copying Files

For convenience and separation of the different applications it is recommended to create a new directory for this tutorial. It is up to the user where to create this directory.

Use the file manager/explorer to create a new directory, and copy the file Tutorial.pctx from the ProcSee tutorial directory to your own directory.

- Change the permission from Read-only.

Preferably call the directory: **`testTutorial`**.

It is also recommended that files like pictures, libraries and data bases has their own directories. Go to the **testTutorial** folder.

Make a new folder and name it "**pictures**".

Make a new folder and name it "**libraries**".

Make a new folder and name it "**databases**".

3 Starting Up

This chapter describes what to be started in the ProcSee system and how it is done. The completed application picture will also be presented.

3.1 What to Start

There are two main programs in the ProcSee system that must be running to be able to make a new application, their names are abbreviated to:

- *RTM* Run-Time Manager
- *GED* Graphics Editor

When the system is installed, a system service called **control** is also installed

A graphics illustration of the connection between the two main programs and the control server in the ProcSee system is presented in Figure 2 on page 10.

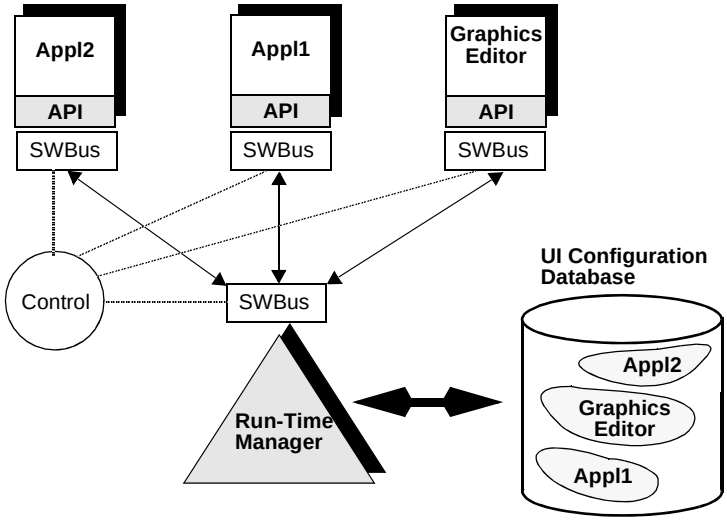
GED is itself an application just as the Appl1 and Appl2 applications. They are communicating through the Software Bus using the API library. To give some ideas about what such applications as Appl1 and Appl2 could be, refer to Chapter 8 *"The Simulator"* on page 51, describing a simulator application.

The container next to the RTM is illustrating that the RTM is storing and retrieving information about the user interface for the two applications in a database file. In addition, a server program called Control is used to set up the connection between the dif-

ferent programs, by holding the connection information like which host a process is running on, and which port on that host to use.

Figure 2

The figure illustrates that the Graphics Editor actually is a ProcSee application. The communication between the applications and the Run-Time Manager is taken care of by the Software Bus. The UI Configuration Database is where the applications user interface information reside. Control is started automatically by Windows .



3.2 How to Start the ProcSee System

In the previous chapter it was stated that it is required to start the two main ProcSee programs. Before trying to start these programs remember to define the three **Environment variables** as described in section 2.1.

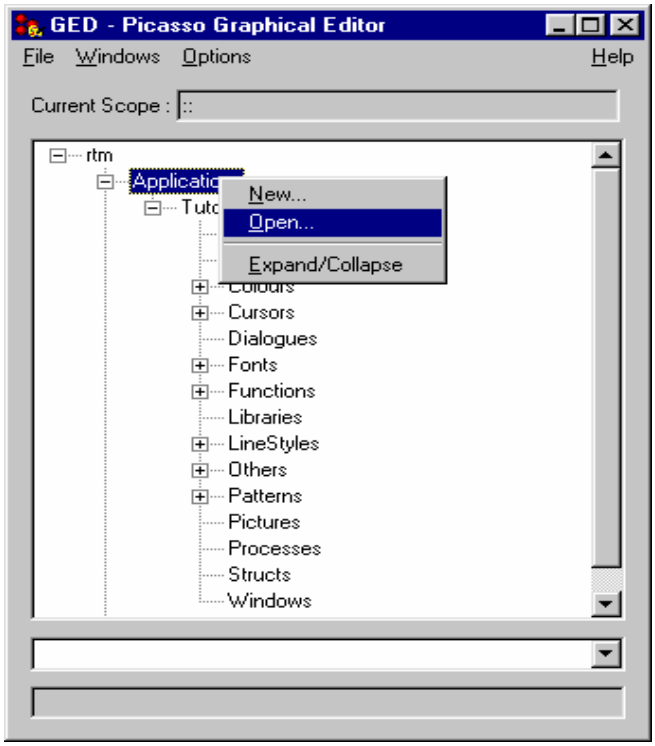
Start RTM and GED from the ProcSee group in the **Start** menu. Right click "**Applications**" node and choose "**Open...**".

Figure 3

This is how the Graphics Editor will look like the first time it appears on the screen.

This main editor offers editing possibilities on pictures and libraries by starting a drawing editor in another window.

By using the input field second from bottom, the user has direct access to the ProcSee language, pTALK, in the Run Time Manager. The bottom field is for output.



In the top of the main GED window there are pull-down menus indicated by the **File**, **Windows**, **Options** and **Help** menu. Under the menu is a tree view of the contents of the Run Time Manager.

In this tree view, you can select what to edit. A menu of available actions for the item that is selected will pop up when the right mouse button is clicked. This menu will change depending on what you have selected in the tree. The **File** menu in the menu bar changes to the same menu as this popup menu, but in addition also includes an Exit choice.

Examples of such menus are: By selecting the Applications node, you can create a new application, or open an application from file. If you select a named application, you can save the application to file, document the application, remove the application from the memory of the RTM, or print all the pictures displayed by the application at the moment. Nearly the same actions are available on pictures, in addition, you can open a **Picture editor** by choosing **Edit**.

The two fields at the bottom of the main window are for input and output purposes.

After you have choose **Open..**, a new window will pop up, the **Open window**. Go to where you have copy the **Tutorial.pctx** file, select it and click on **Open** button.

To get more detailed information on how to use GED refer to the ProcSee **User's Guide**.

3.3 Description of the Tutorial Application

The goal of this tutorial is to give the user the opportunity to work through and get an overview of the main features of the ProcSee system.

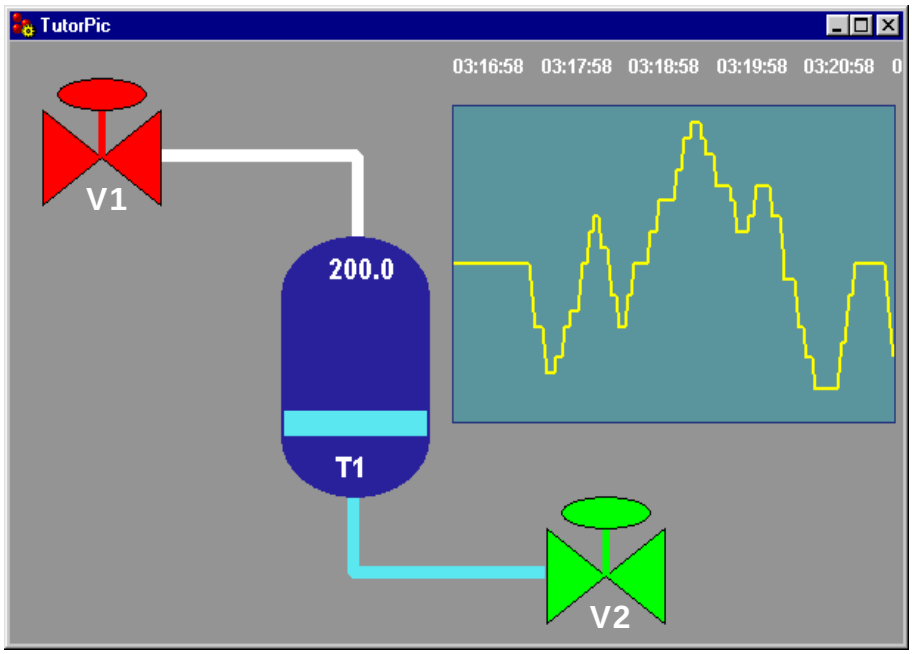
To start with the end product of this tutorial, a mimic picture will be produced presenting a part of a flow process with two valves and a tank connected to each other with pipes. See Figure 4

Figure 4

This figure illustrates the mimic picture for the tutorial application.

A part of a flow process where fluid is coming in to the upper left valve, flowing into a tank and out through the lower right valve.

The value at the top of the tank is indicating the fluid level in the tank. The two characters T1 is the name of the tank.



Fluid from the process is coming in through the valve **V1**, flowing into the tank **T1** and out through the valve **V2**. The pipelines are changing colours between light blue and white depending on whether there is fluid in the pipe or not. By clicking on the valves with the left mouse button, they will toggle between an open or closed state.

If the valve **V1** is open and **V2** is closed, the level in the tank will increase and the value (light blue rectangle) displayed in the tank, is updated accordingly. Closing **V1** and opening **V2**, will make the tank level decrease. Because of the same flow rate through the valves, the tank level will be stable if both valves are open.

The next chapters describe the procedures for obtaining the graphics and functionality for the tutorial picture described in the paragraph above.

4 GED

This chapter will describe the most important functions in the Drawing Editor and explain the design of some simple objects.

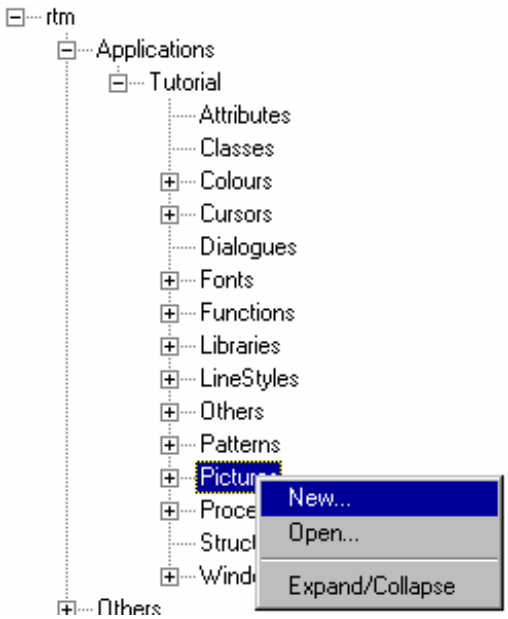
4.1 Creating a New Drawing

To create a new picture in the Tutorial Application, select the **Pictures** node in the tree, and right click on it to get the pop-up menu (see Figure 5).

Figure 5

The figure illustrates how to create a new picture in an application already running in the rtm.

Select Pictures in the tree and press the right mouse button while the cursor is focused on the "Pictures" node. Select New in the popup menu. When "Pictures" is selected, the same menu is also available in the File menu.

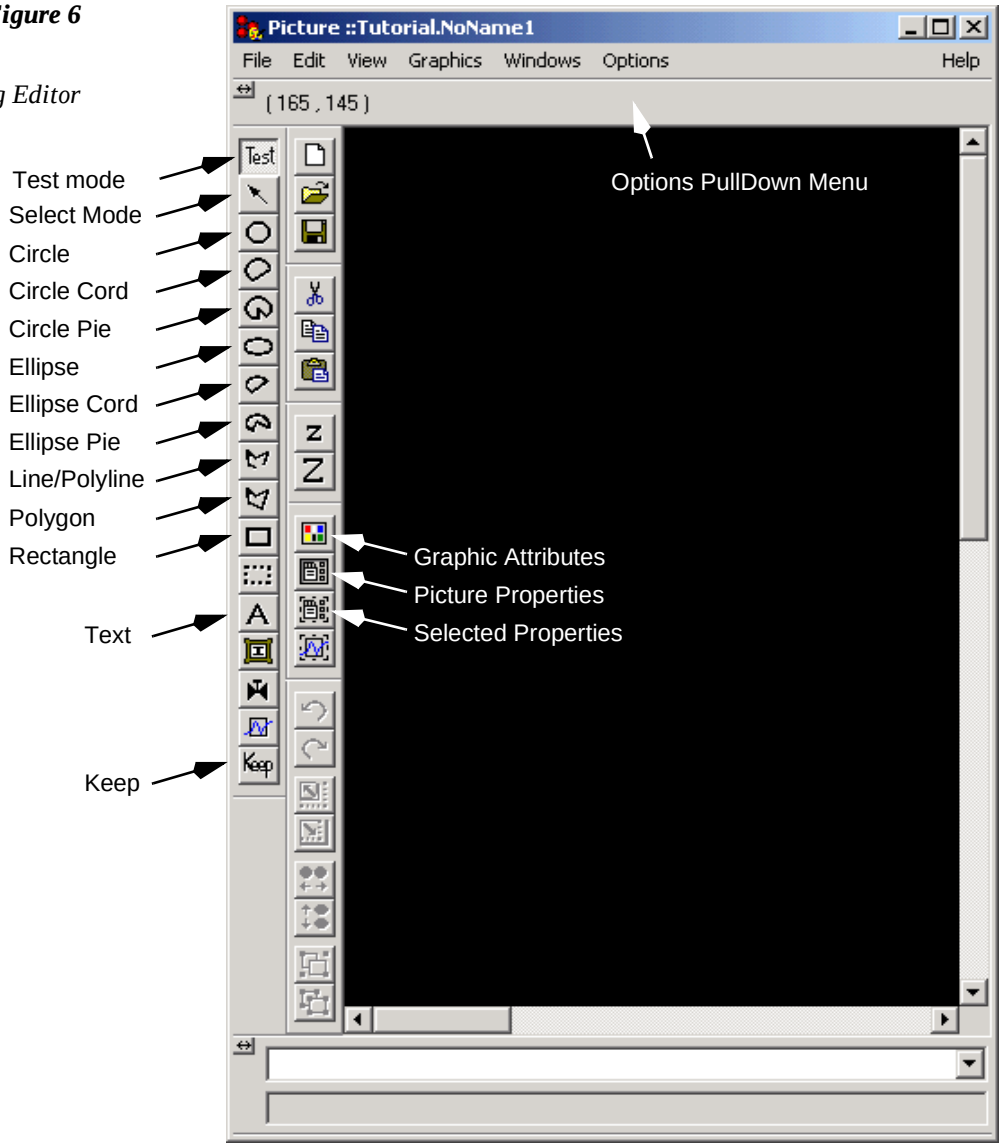


Select New on the popup menu.

The drawing editor will appear on the screen and a picture named **NoName1** will be created. (see Figure 6).

Figure 6

GED's Drawing Editor



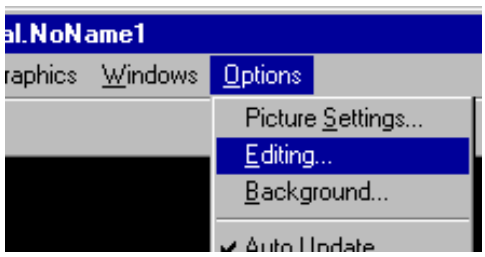
configurations

Before you start drawing, some editing options should be set, in order to make the drawing easier.

Chose the **Options** Pull Down menu and select **Editing...** . (see Figure 7)

Figure 7

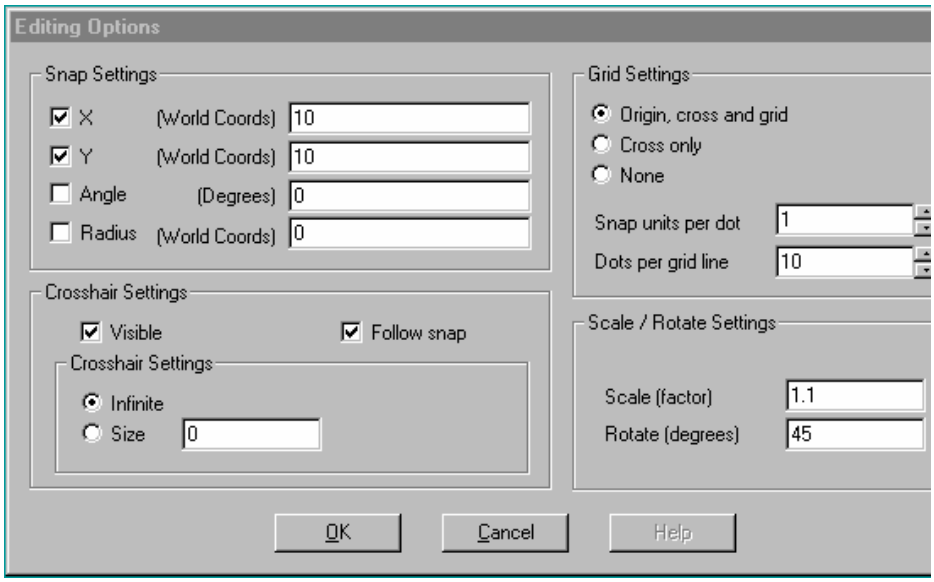
Options menu



The **Editing Options** window will appear on the screen as shown in Figure 8.

Figure 8

Editing Options



snap interval

The snap settings are done in order to make alignment of shapes more easy.

In **Snap Settings** (World Coords), set both the **X** and **Y** snap to **10**. Remember to activate the Toggle Buttons for **X** and **Y**. Click on the **OK** button to confirm the settings.

Background Colour

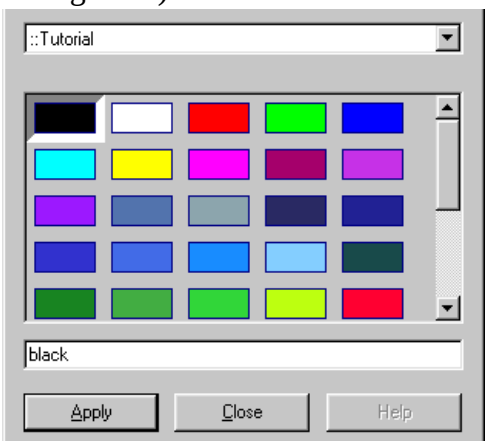
Default value for background colour is **black**. If you want to change the colour, do the following:

Chose the **Options** Pull Down menu and select **Background...** . The **Background** window will appear on the screen.

(see Figure 9)

Figure 9

Background Colours



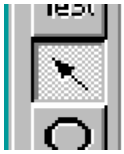
Chose a Colour (e.g. grey50) from the **Background** colour menu or write the colour name in the text field below. Confirm the text input by pressing the **Apply** button. The **Close** button will remove the Background Colour window.

4.2 Rectangle, circle, polygon, line/polyline and Text

This chapter will describe how to draw some simple shapes and how to equip the shapes with graphic attributes. It will also describe how to draw a line or polyline, how to write a text string and how to connect graphic attributes.

4.2.1 General

Select Mode



After terminating the drawing of a shape the editor will set itself into select mode, indicated by the **Select Mode** tool.

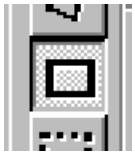
In select mode other shapes can be selected by clicking on the shape with the left mouse button. If the **Shift** key (keyboard) is pressed, the left mouse button will have a toggle function and more shapes may be selected or un selected. Shapes may also be selected by dragging a rectangle around them. The currently selected shape(s) will be marked with small rectangular handles.

Keep

If the **Keep** tool is selected, the chosen drawing tool will remain and more than one shape of the same type, can be drawn in sequence

4.2.2 Rectangle

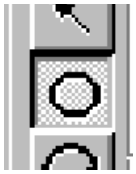
rectangle



Chose the **Rectangle tool** in the drawing editor's Toolbar. Select position of the first rectangle corner and press the left mouse button. Keep the button pressed and drag the cursor. Release mouse button when rectangle size is sufficient.

4.2.3 Circle

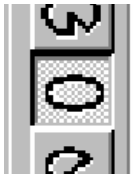
circle



Chose the **Circle tool** in the drawing editor's Toolbar. Select position of the circle centre, and press the left mouse button. Keep the button pressed and drag the cursor. Release mouse button when circle size is sufficient.

4.2.4 Ellipse

ellipse



Chose the **Ellipse tool** in the drawing editor's Toolbar. Select position of the ellipse centre, and press the left mouse button. Keep the button pressed and drag the cursor, up-down to change the height and left-right to change the width. Release mouse button when ellipse height and width is sufficient.

4.2.5 Polygon

polygon

Chose the **Polygon tool** in the drawing editor's Toolbar. Select position of the first corner, and press the left mouse button. Move the cursor and press the left button for each corner. Terminate the drawing by pressing the right mouse button

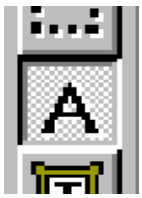
4.2.6 Line/Polyline

line/polyline

Chose the **Line tool** in the drawing editor's Toolbar. Select position of the first corner, and press the left mouse button. Move the cursor and press the left button for each corner. Terminate the drawing by pressing the right mouse button

4.2.7 Text

text



Chose the **Text tool** in the drawing editor's Toolbar. Select position and press the left mouse button. Write the text and press *"return"*.

4.2.8 Circle Arc

Circle Cord/Pie

A Circle **Cord** or **Pie** is drawn in the same way as the drawing of a Circle. The Circle Arc Generators are used to draw both type of objects. The consequence of this is that both a Circle **Cord** and Circle **Pie** object will be of the type Circle **Arc**.

start- & openingAngle

Default setting is: startAngle = 0 ° and openingAngle = 100°.

The startAngle and openingAngle may also be adjusted with the mouse by focusing the shape, pick up the adjusting point and drag to desired value.

4.2.9 Ellipse Arc

Ellipse Cord/Pie

An Ellipse **Cord** or **Pie** is drawn in the same way as the drawing of an Ellipse. The Ellipse Arc Generator is used to draw both type of objects. The consequence of this is that both Ellipse **Cord** and Ellipse **Pie** object will be of the type Ellipse **Arc**.

start- & openingAngle

Default setting is: startAngle = 0 ° and openingAngle = 100°.

The startAngle and openingAngle may also be adjusted with the mouse by focusing the shape, pick up the adjusting point and drag to desired value.

4.3 Attributes

This section will describe how to connect attributes to the shapes, lines and text strings.

4.3.1 Graphic attributes

Graphic Attributes

When you draw a shape, you can decide position, form and size. If you want to add attributes like colour, border, patterns etc., the **Graphic Attributes** is a tool for this purpose.

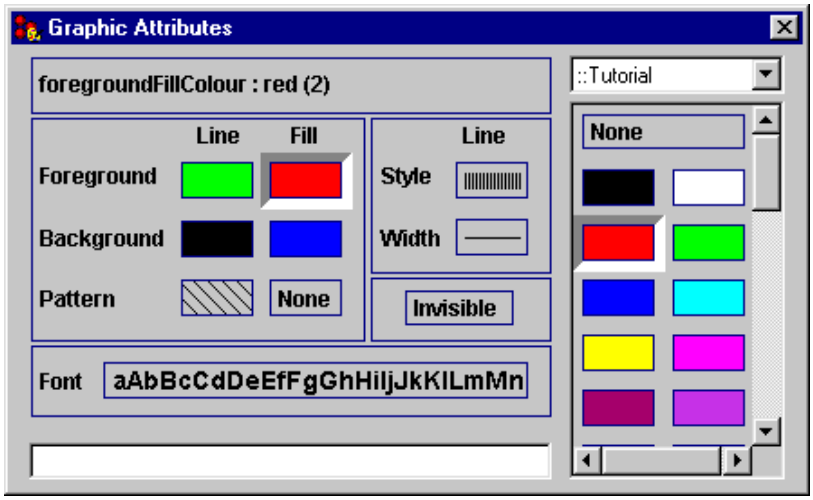
Select the shape you want to modify, and then click on the **Graphic Attributes tool** in the drawing editors Toolbar or click on the **View** menu and select the **Graphic Attributes...** from the pulldown menu.

The Graphic Attributes window will pop up on the screen. (see Figure 10). The window includes attribute values for line and fill colours, patterns, line style, line width and fonts.

Figure 10

This figure presents the Graphic Attributes dialogue window.

To set a new attribute value, select the actual attribute and change it in the presentation list at the right side of the window.



The attributes for shape colours are grouped in **Line** and **Fill. Line/Polyline** and **Text** shapes, have no **Fill** attributes. **Line/ Polyline** and **Shape Boarders**, do have an extra attribute for **Width,** and for **Line/Polyline** there is also an attribute for **Style**.

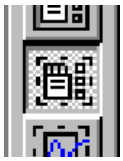
If the selected **Pattern** = None, the **Foreground** and **Background** colours are not used. The colour of a Pattern is equal to the Background colour selected.

Try to change the graphics attributes for some of the shapes, drawn in section 4.2 on page 18. Select the shape by choosing Select Mode and click on the shape. Chose from the different types of attributes.

Shapes with visibility = **Invisible**, will be visible only when **Test mode** is selected.

4.3.2 Selected Properties

Selected Properties



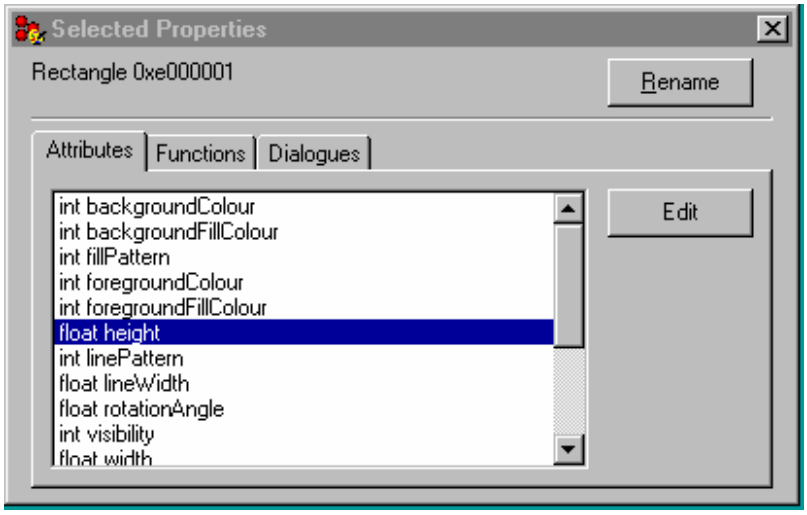
In addition to the attributes found in the **Graphic Attributes** window, the **Selected Properties window** includes dynamic attributes for adjustment of height, width, positioning, rotation etc. of the shapes.

Select the shape you want to modify, and then click on the **Selected Properties tool** in the drawing editors Toolbar.

The **Selected Properties window** will pop up on the screen as shown in Figure 11. The window includes editing facilities for Attributes, Functions and Dialogues.

Figure 11

This figure is illustrating the Selected Properties window for the selected shape.



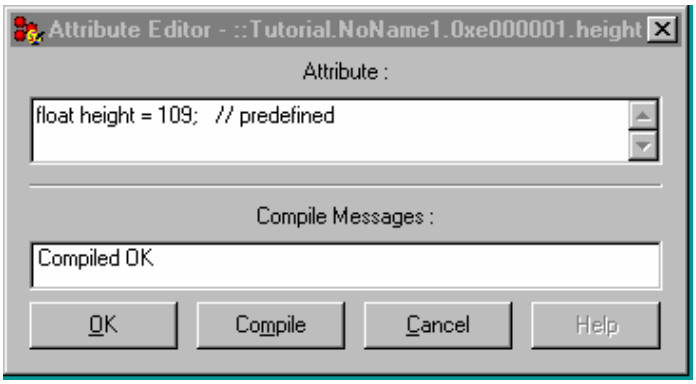
Select the earlier drawn rectangle. In the Attributes, chose **float height** and click on the **Edit** button.

The Attribute Editor window will pop up on the screen.

(see Figure 12 on page 22)

Figure 12

This window shows the Attribute Editor which is used for changing the selected attribute value.



Type the desired value for rectangle height (e.g. 109) and click on the **Compile** button. If compilation is OK, a message: "Compiled OK" will arrive in the Compile Message: window. Then click on the **OK** button and the rectangle will have the new height. (If you only click on the **OK** button, you will not be able to read the compile message if something went wrong).

4.3.3 Save and Document

save

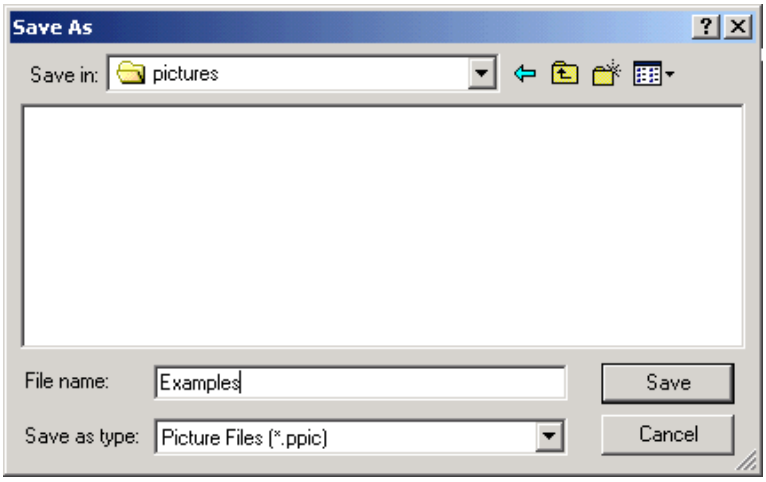
In order to store the picture for later use, it has to be named and saved.

Save the picture with a new name by pressing the **File** menu and chose the **Save As...** option. A standard file save window appears on the screen. (see Figure 13) Check that the directory is where

you want to save the picture (go to the **pictures** directory). Write the picture name in the **File name** text field and click on the Save button, to save the picture.

Figure 13

The Save As window..



document

Save the picture as an ascii file by pressing the **File** menu and chose the **Document** option.

5 Application Resources

This chapter gives an overview of the default resources (colours and fonts) in the ProcSee system and necessary tasks for initial preparation.

5.1 Database Definition

In order to make it possible for the *Graphics Editor* (GED) and the *Run Time Manager* (RTM) to operate without a program running, we declare the variables needed, in a database definition, called **Tutorial.pdat**. This text file will be read by ProcSee at start-up and contains the following definitions:

DATABASE DEFINITION

```
// .pdat

float t1_level = 0.0;
int32 v1_state = 0;
int32 v2_state = 0;
```

5.2 How to do this by means of GED.

We want the process database script from Chapter 8 *"The Simulator"* on page 51, to be included in our application. This is not strictly necessary, since the tutorial program creates the variables anyway, but we want to be able to edit our picture without starting this program[1]. For the application to read the database defi-

1 The use of the terms "application", "process" and "program" might be a bit confusing. With "application" and "process" we mean entities administered by the run-time manager. The "task", or the "program" is an external piece of code running independently, communicating with ProcSee through the API.

nition it has to be declared in the application resource file, the**.pctx** file that was originally copied from the **%PROCSEE_DIR%\tutorial\%ARCH%** directory.

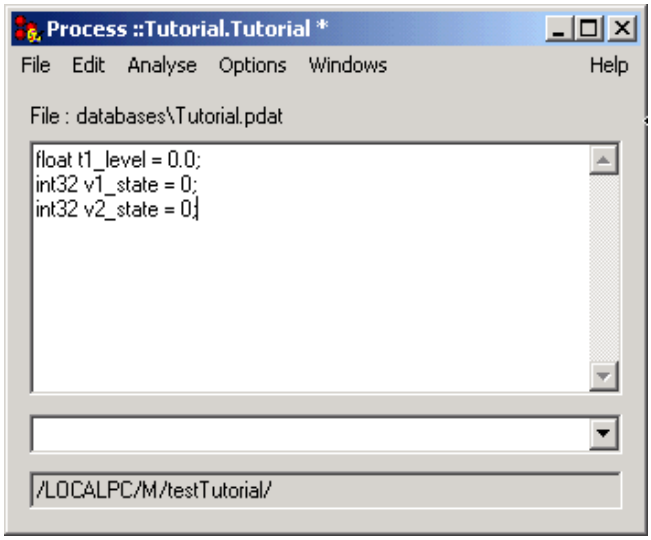
To do this from GED, select Processes under the Tutorial application in the tree view. Expand this node if possible, and select the Tutorial Process. Start the Process editor by choosing Edit from the File menu, or the menu on the right mouse button.

If the Tutorial process is not there, create it by selecting the Processes node, and choose **New...** on the File menu or the menu on the right mouse button. Type the name **Tutorial** for the name of the new process, and click on the **OK** button. The Process editor will then start.

If you have created the Tutorial.pdat file already, select open from the File menu in the Process editor (See Figure 14) and select the **Tutorial.pdat** file in the file selection window that is displayed.

Figure 14

The Process Editor.



If you are going to create this file now: select **New**. Enter text as shown in Figure 14 on page 26. Save the data base with **Save As...**, and name it **Tutor.pdat**. (Remember to save it in the "**databases**" directory).

Close the Process editor, and answer **Yes** to the questions about saving and installing the database files.

Now you should save the **Tutorial** Application, so the database declaration you have added is stored on disk.

This is done by selecting the **Tutorial** application, and **Save** in the popup menu that is displayed when right clicking on the library node.

5.3 Colours

A ProcSee delivery, includes a set of predefined colours. These colours may be edited, deleted and new colours may be added. The colours are described by **colour name**. A segment of the default colour palette is shown in Figure 15.

Figure 15

Segment of the predefined colour palette

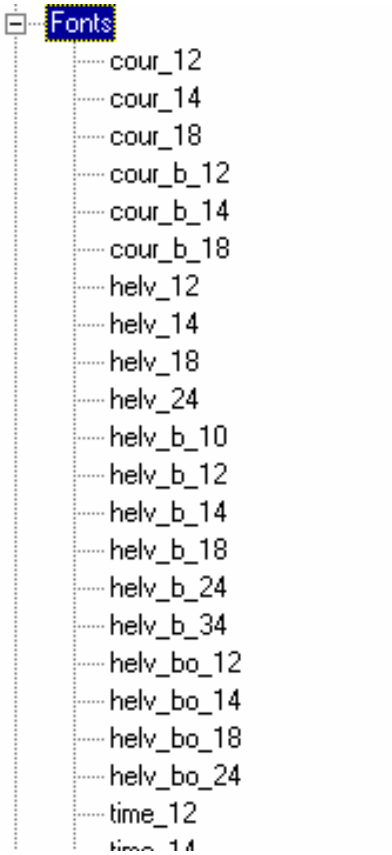
5.4 Fonts

A ProcSee delivery, also includes a set of predefined fonts. These fonts may be edited, deleted and new fonts may be added.

A segment of the predefined fonts is shown in Figure 16.

Figure 16

Segment of the predefined fonts



The name of the fonts has been chosen so that they describe the font- **name**, **angle/weight** and **size**.

font description

The first four letters describes the **font name**.

- aria = arial
- cour = courier
- helv = helvetica
- time = times

The intermediate item describes the **font angle** and **weight**.

- no item = normal font
- b = bold font
- i = italic font
- bi = bold and italic font

The numbers (last item) gives the **font size** in points.

If a font is not present on your system, the closest matching one will be used.

5.5 Patterns

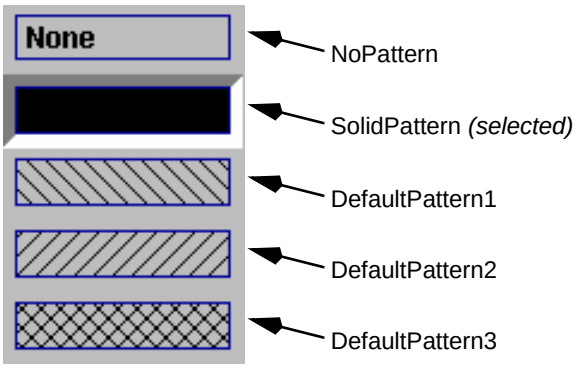
default patterns

In addition to **NoPattern** and **SolidPattern**, there is three default patterns (see Figure 17).

The patterns could be found in the **Graphic Attributes** window.

Figure 17

Default Patterns as seen in the Graphic Attributes window



More patterns are available. The User's Guide describes how to do this. (see User's Guide, page 79)

6 Design of Classes

This chapter will deal with the design of classes as in the ProcSee system. The classes are saved into libraries for later use when building the tutorial process picture.

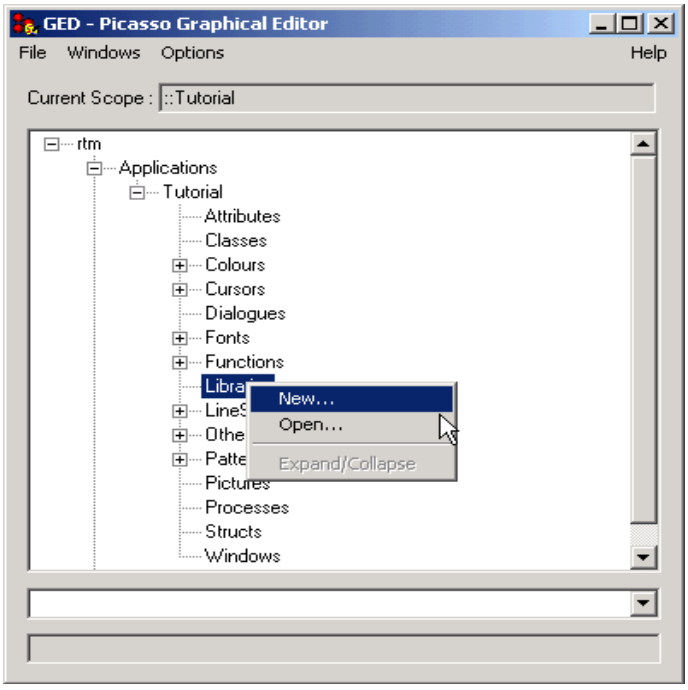
6.1 Making a New Library

To create a new library in the Tutorial Application, select the **Libraries** node in the tree, and right click on it to get the popup menu (see Figure 18).

Figure 18

The figure illustrates how a new library can be created in an application already running in the rtm.

Select Libraries in the chosen application, and press the right mouse button to get the popup menu and select New in the menu. The same menu is also available on the File menu in the menu bar when Libraries are selected.



Select **New** on the popup menu, and a library named **NoNameLib1** will be created.

When the library has been created, it is important to set a suitable name on it. This is done by selecting the **NoNameLib1** library, and selecting the **Save As** menu choice on the popup menu that is displayed when right clicking on the library node. Go to the library folder.

In the save window that is displayed, (see Figure 13 on page 23) type the name **TutorLib**, and click on the **Save** button.

You should also save the application now, so that the library inserted into the application is available next time the application is loaded. This is done by selecting the Tutorial application, and clicking on **Save** in the menu that pops up when the right mouse button is clicked, or in the File menu in the menu bar. (If failed, check the properties on the Tutorial.pctx file).

Drawing Editor

To create a new class in the library, bring up GED's ***Drawing Editor*** by: selecting the **Classes** node under the **TutorLib** library node in the tree, and clicking on **New** in the menu that pops up when clicking the right mouse button, or in the file menu in the menu bar.

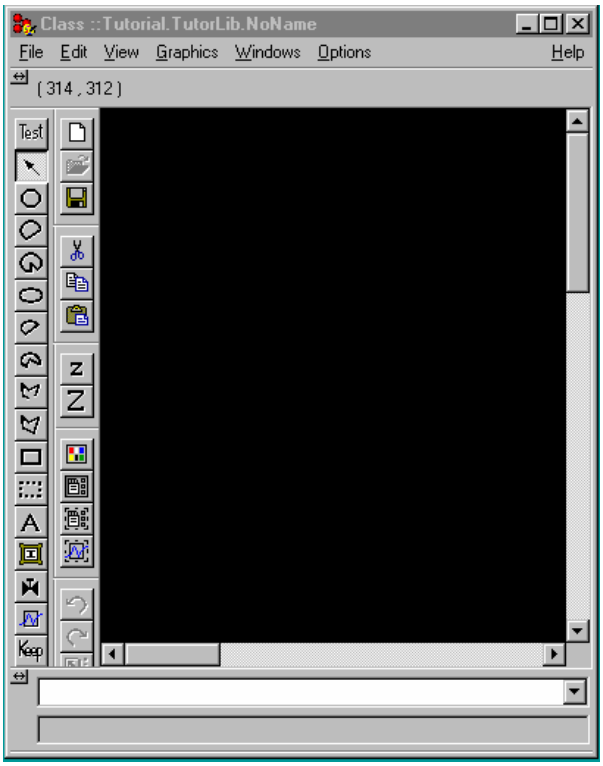
The drawing editor will appear on the screen as shown in Figure 19. The same drawing editor is used for designing both pictures and classes, see Figure 6 on page 16.

Figure 19

This is an illustration of GED's Drawing Editor.

The drawing editor is currently presenting a NoName drawing until saved as something else.

At the top there are pull-down menus, and at the left side there is a toolbar for drawing facilities. At the bottom there is an input and an output field.



6.2 The Valve Class

save the class

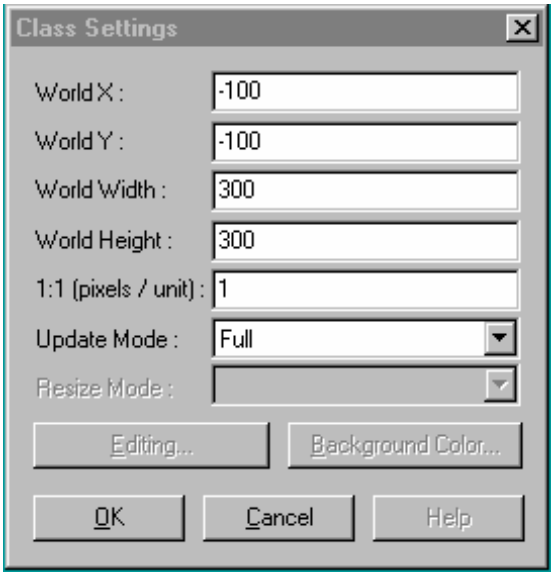
Start the design of the valve class by renaming the default **NoName** drawing. Click on the drawing editor's **File** menu and select the **Insert As...** option. Type the name: **Valve**, for the actual class in the *Insert as class:*. input field, and click on the **OK** button.

Note that this will only insert the class into the library, the class is not yet saved to file. In order to save the class, the library have to be saved.

configure the drawing It is mandatory to configure the drawing options before starting to work with it. This include setting the background colour, the snap interval, and the geometry for the drawing. Choose the **Options** menu and select the **Class Settings...** from the pull-down menu. This will bring up a new window for setting the coordinates of the drawing. Type in the settings illustrated in Figure 20.

Figure 20

Window for setting the geometry to the current drawing. World X and Y are the coordinates of the upper left corner of the drawing area.The origin in the coordinate system will represent the reference point of the class,



When you have typed the right values into the **World X**, **Y**, **Width**, and **Height** and selected **Update Mode** to Full, click on the **OK** button to confirm the changes.

Snap interval To set a snap interval, chose the Options menu, and select **Editing...** . (See Figure 8 on page 17.)

Background Colour Chose the Options menu, and select **Background...** . (See Figure 9 on page 17.)

Polygon Chose the **Polygon tool** as described in 4.2.5 on page 19. Place four corners of the polygon by referring to the **x,y** output in the information field below the **File** menu:

1) first point: x = -40, y = -30
2) second point: x= -40, y = 30
3) third point: x = 40, y = -30
4) fourth point: x = 40, y = 30

Then press the right mouse button to end the shape drawing.

Line

Continue drawing the valve's handle by pressing the **Line tool** (see 4.2.6 on page 19). Place two points the same way as with the polygon. Remember to terminate the line after two points by clicking on the right mouse button.

1) first point: x = 0, y = 0
2) second point: x = 0, y = -30

Ellipse

Chose the **Ellipse tool** as described in 4.2.4 on page 18. Press the left button down in position **x = 0**, **y = -40** and keep it down while dragging the cursor to position **x = 30, y = -30**.

Graphic Attributes

Select the **Graphic Attributes** button, as described in Figure 10 on page 20.

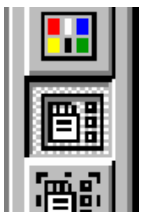
Select the ellipse shape in the drawing area.

Click on the **Foreground** attribute in the **Line** column, and select the black colour in the palette list at the right side of the window. Then click on the **LineWidth** attribute and select the width number two in the presented list. Click on the **Line-Pattern** attribute and set it to **Solid** (pattern number two from the top of the list).

Move the cursor to the drawing area and select the polygon shape. Do the same operation for this shape in the **Graphic Attributes** window.

Select the last **line** shape. Set the **LineWidth** attribute to width number five, and the **Line-Pattern** to **Solid** (pattern number two). To remove the graphic attribute window click on the same button that brought it up or click on the X-button on the graphic Attributes window.

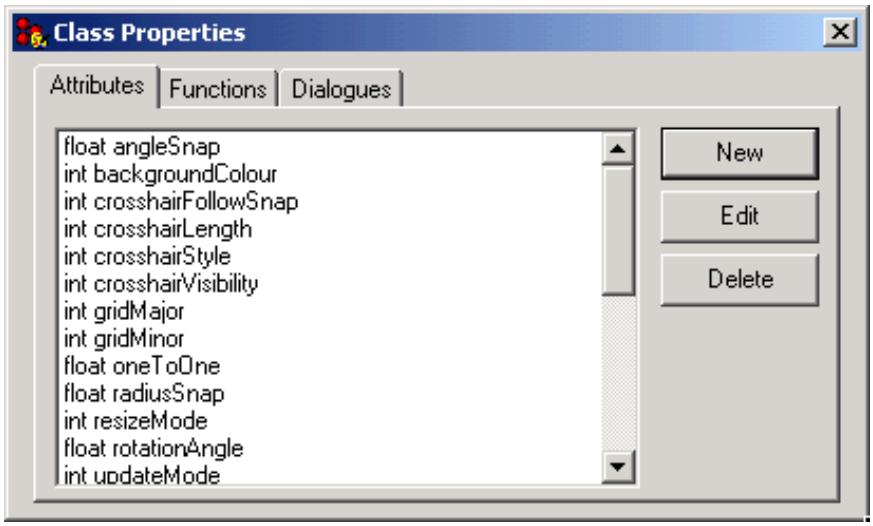
Class Properties



The graphics representation of the class is now defined and the next step is to define some attributes and functions for the valve class. This is done in the **Class Properties** window. Select the **Class/Picture Properties** button, or click on the **View** menu and select the **Class Properties** option from the pull-down menu.

The **Class Properties** window, illustrated in Figure 21, is used to create new, edit, and delete an attribute, a function, or a dialogue.

Figure 21

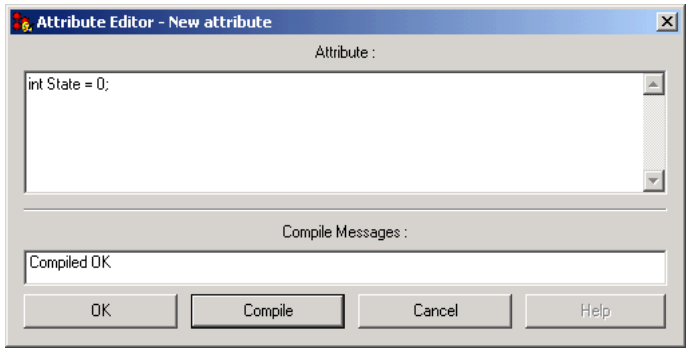


This figure is illustrating the Class Properties window.

The window is used to create new, edit, and delete different properties of the current class.

The first operation to do in this window is to create a new attribute for the valve class. Click on the **New** button and type in this pTALK[1] statement (as shown in Figure 22) in the **Attribute Editor** window and click on the **Compile** button to verify that it compiled successfully *(Compiled OK)*, before clicking on the **OK** button.

Figure 22



The Attribute Editor window.

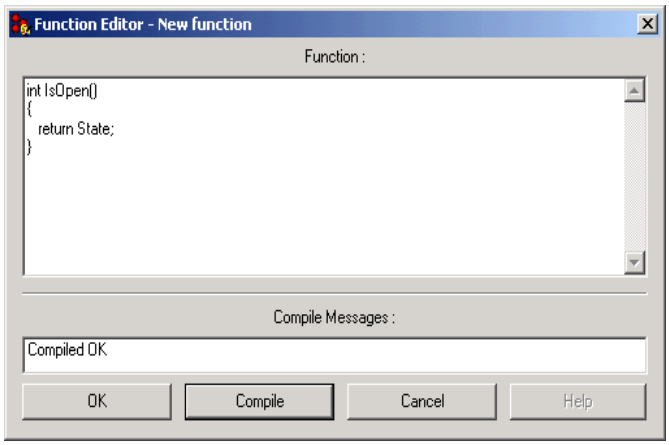
The statement is a declaration and initialization of the integer attribute. Later on we will set the State attribute to a process variable.

1 pTALK is the ProcSee programming language that has most of its syntax inspired from ANSI C, and can be compiled and evaluated at run-time by the RTM. For more information about pTALK, refer to chapter 8 in the ProcSee User's Manual

Select **Functions** in the **Class Properties** window and click on the **New** button. Type in the ***IsOpen*** function, (as shown in Figure 23) click on the **Compile** button. If the output field shows *Compiled OK*, click on the **OK** button to exit the window. The next

Figure 23

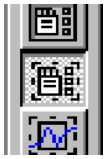
The Class Properties Function Editor window.



function to create, is for deciding the valve colour.

```
int ValveColour()
{
   if( IsOpen() )
     return green;
   else
     return red;
}
```

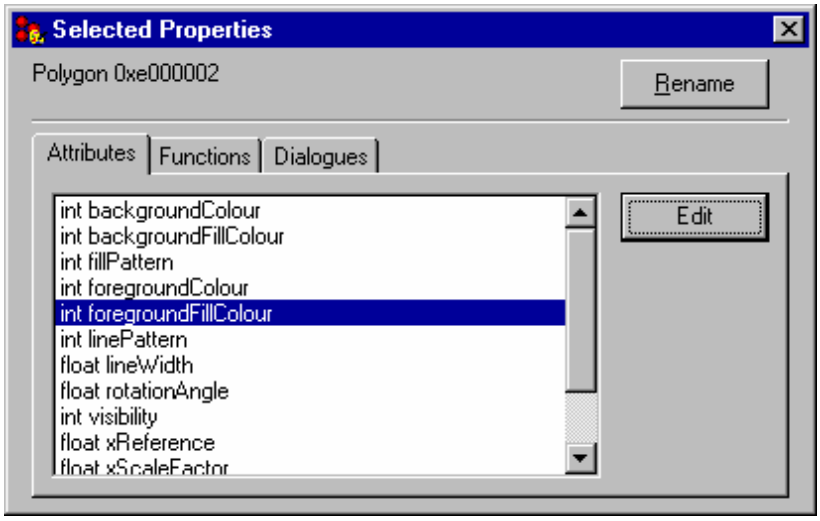
Selected Properties



To define dynamic shapes in the valve class, bring up the **Selected Properties** window either by pressing the Selected Properties button as illustrated in the left margin or select it from the **View** menu. This window is shown in Figure 24

Figure 24

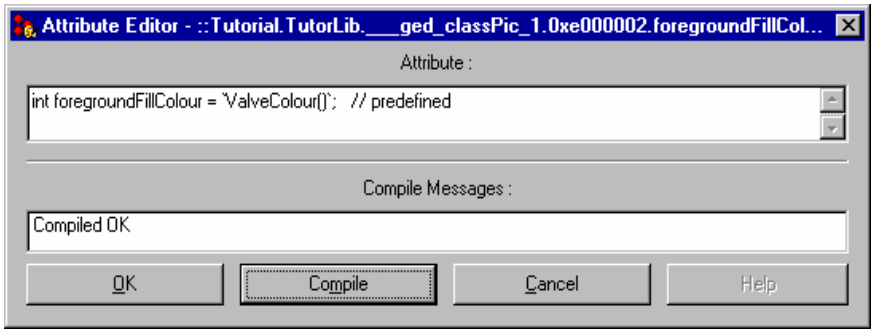
This figure is illustrating the Selected Properties window for the selected shape.



If the window is empty, select the polygon shape by pressing the left mouse button on the polygon shape. The properties of the polygon will be displayed in the window. Use the scrollbar for the attributes and select the **foregroundFillColour** as illustrated in Figure 24. Click on the **edit** button and change the predefined assignment as displayed in the **Attribute Editor** window as shown in Figure 25.

Figure 25

The Selected Properties Attribute Editor window



Click on the **Compile** button and if the *Compile Messages:* field shows *Compiled OK*, then click on the **OK** button to remove the window. Do the same operation for the **line** and **ellipse** shapes, but for the **line** shape select the **foregroundColour**. Click on the Selected Properties button or the X-button in the property window, to remove the Selected Properties window.

Insert the class in the **File** menu and close the drawing editor. Before going on with the **Tank** class, remember to save the **TutorLib** library first. Also do a Document of the file.

Dialogue

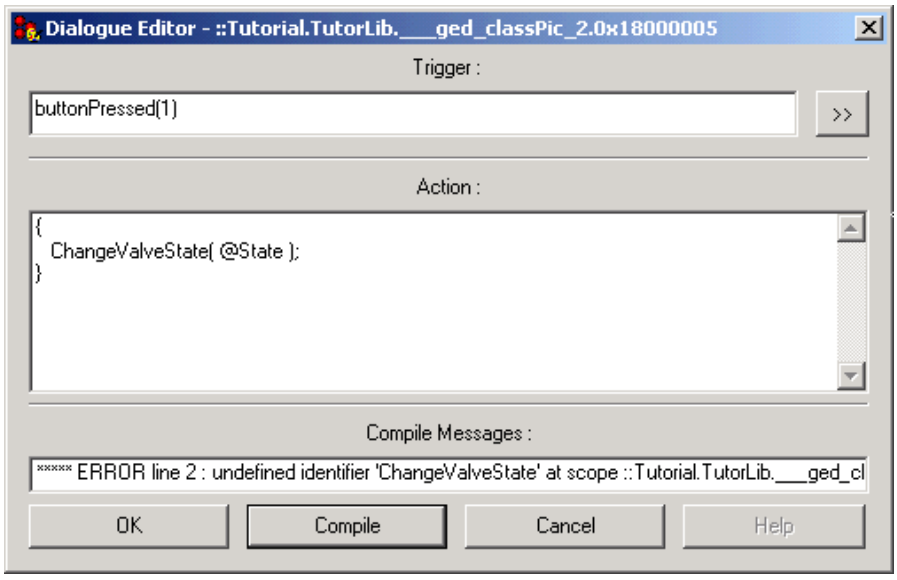
A dialogue in a graphics user interface defines how the user interacts with the system. To create a dialogue for the Valve class, select **Dialogues** in the **Class Properties** window and click on the **New** button, or do it directly from the menu tree as described for the Attributes.

The window appearing is separated into two input areas, one for the **Trigger** and one for the **Action**. Type the statement as listed below, in the **Trigger** area or click on the **Menu** Button (at the right side of the window) and chose from the **Trigger pop up** menu.

Then type the statement in the **Action** input field as shown in Figure 26.

Figure 26

The Dialogue Editor window.



The ***ChangeValveState*** is a **function** and is unknown for the **rtm** until it has been made (it is the next thing to do). Click on the **Compile** button. Remove the window with the **OK** button. An error window will pop up telling you that the compile failed, asking if the editor window shall be closed, answer **yes**. Click on the **Class Properties** button (as brought up the window) to remove the window.

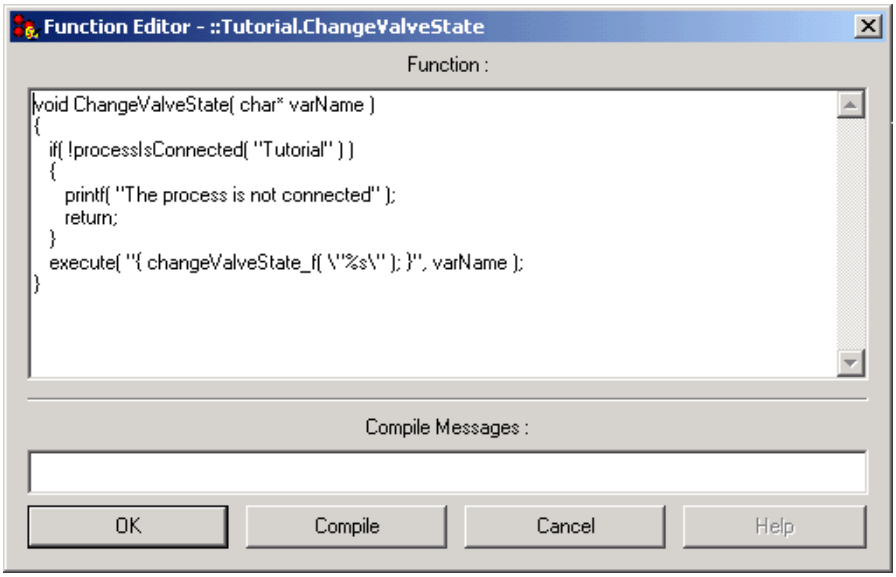
Application function

The ***ChangeValveState*** function is created selecting the **Function** node in the tree on the application level. Select **New...**Later on page 62 we will explain how to make the register function in the program. Putting the register function in an ***execute*** function makes the compile OK. This is done because the function is not

yet known in the rtm. It will first be known when the program is connected to the rtm. For more information about **execute** see the **Reference manual**.The function is shown in Figure 27.

Figure 27

The Function Editor window.



Click on the **Compile** button. If the output field shows *Compiled OK*, remove the window with the **OK** button.

6.3 The Tank Class

Start building a new class by pressing the **New** button. Configure the new drawing the same way as in section 6.2 on page 32 with the listed parameters:

- **World X** = -100, **World Y** = -100
- **World Width** = 750, **World Height** = 500
- **Snap Interval X** = 10, **Y** = 10
- **Background colour...** : ex. grey50 (see Figure 9 on page 17)

Save the tank by selecting the **Insert As...** option in the **File** menu and name it **Tank**.

Class Properties

Bring up the **Class Properties** window, see Figure 21 on page 35, and create two attributes for the tank class. Click on the **New** button and type the pTalk expression in the **Attribute Editor** win-

dow according to the input frame below: (see Figure 22 on page 35)

```
float MaxLevel = 1000;
```

Click on the **Compile** button and if *Compiled OK* appears in the output field, close the **Attribute Editor** window by pressing the **OK** button. Do the same operation with the second attribute according to the input frame below.

```
float Level = 0;
```

In the same **Class Properties** window, select **Functions** and click on the **New** button. In the **Function Editor** window, type in the pTALK expression according to the input frame below:

```
float WaterLevel()
{
   float L = Level;
   if( L > MaxLevel ) L = MaxLevel;
   if( L < 0 ) L = 0;
   return (L/MaxLevel)*91;
}
```

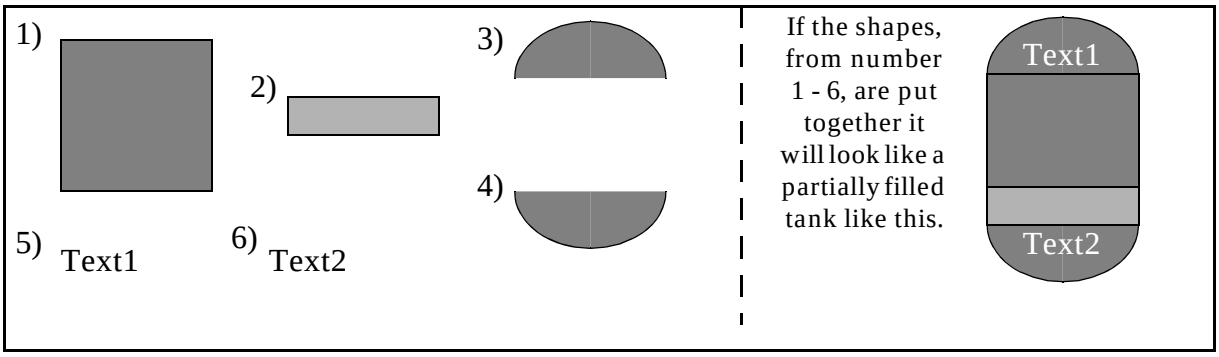
Click on the **Compile** button and if *Compiled OK* appears in the output field, close the **Function Editor** window by pressing the **OK** button.

Do the same operation with the second function according to the input frame below:

```
char *InstName()
{
   return name();      // returns the name of the picture object
}
```

Remove the **Class Properties** window by either pressing the **Class Properties** button, or pressing the **X-button in** the **Class Properties** window.

draw shapes Then draw six shapes roughly according to the numbered illustrations below; two rectangles, two opposite ellipse arcs, and two text instances. Put them together to form a tank. When drawing



elliptic arcs, be aware that the operation is separated into three parts. First you draw an ellipse with the wanted the size, then you pick and move the square "handles" on the ellipse to set the orientation and opening angle of the arc.

Graphic Attributes Bring up the **Graphic Attributes** window, see Figure 10 on page 20, and select the numbered shapes from 1 to 6.

1) Set **Line-Pattern** to *None,* and **Fill-Foreground** to colour medBlue (black is 0).
2) Set **Line-Pattern** to *None,* and **Fill-Foreground** to colour lightSkyBlue.
3) Set to same as **1)**.
4) Set to same as **1)**.
5) Set **Line-Foreground** to *white*, and **Font** to *helv_b_14*.
6) Set to same as **5)**.

Selected Properties Bring up the **Selected Properties** window, see Figure 24 on page 37, and select the numbered shapes from 1 to 6 over again. Edit the different shape attributes to get a tank. Use **float** in front of all the attributes, except for **theText** and **format** that uses **char***.

1) Edit to: `X = 0;` , `Y = 50;` , `width = 100;` , `height = 90;` .
2) Edit to:
```
X = 2;
Y = `50 + 91 - WaterLevel()`;
width = 96;
height = `WaterLevel()`;
```
3) Edit to: `X = 50;` , `Y = 50;` , `xRadius = 50;` , `yRadius = 37;` , `startAngle = 0;` , `openingAngle = 180;`

4) Edit to:
```
X = 50; , Y = 140; , xRadius = 50; ,
yRadius = 37; startAngle = 180; ,
openingAngle = 180;
```

5) Edit to:
```
X = 27; , Y = 40; , theText = 'Level'; ,
format = "%6.1f";
```

6) Edit to:
```
X = 37; , Y = 165; ,
theText = 'InstName()'; ,
format = "%s";
```

Insert the drawing and leave the drawing editor.

Save and **Document** the library in the main editor's **File** menu by selecting **TutorLib** with the right mouse button. With the left button, click on **Save** and **Document** in the pop-up menu.

7 The Picture

In this chapter a picture is built, using the classes defined in Chapter 6 "Design of Classes" and the database definition, created in Chapter 8 "The Simulator".

7.1 Database Definition

The picture that we shall create, will show three process variables: The tank and the two valves.

The three process variables will be created by the program created in chapter 8 *"The Simulator"* on page 51 and made known to ProcSee when the program is connected to the *Run Time Manager* (RTM). In order to make it possible for the *Graphics Editor* (GED) and RTM to operate without this program running, declare the variables in a database definition script, called **Tutorial.pdat**. See Figure 14 on page 26. This script will be read by ProcSee at start-up and contains the following definitions.

DATABASE DEFINITION

```
// .pdat

float t1_level = 0.0;
int32 v1_state = 0;
int32 v2_state = 0;
```

7.2 Instantiating

Now all the building blocks are ready and you can start building your applications picture.

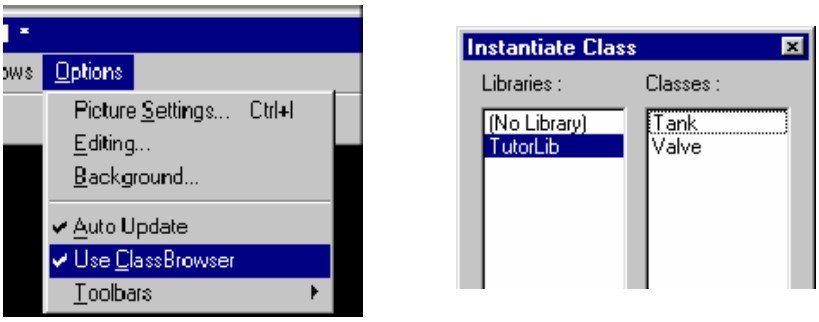
Select the **Pictures** node in the tree, and proceed as described in chapter 4 *"GED"* on page 15.

In the drawing editor, resize this window to a size that suits you before you start drawing, and set the background colour to a pleasant value, e.g grey50. The background colour can be set from the **Background**... option in the **Options** menu of the picture editor. You should also set the **X** and **Y** snap to 10 from the **Snap** option in the **Editing Options** window opened from the Options menu, all as described in Chapter 4 *"GED"*. Save the picture as "**pictures/TutorPic**" using **Save As...** in the **File** menu.

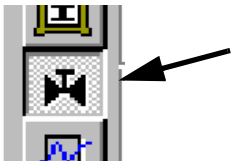
The classes may be presented in either an **Instantiate Class** window or in a **ClassBrowser** window. The selection is done in the **Options** pull-down menu. The **Instantiate Class** window will be used if **Use ClassBrowser** is *not* selected, see Figure 28. This selection must be done before the **Instance tool** is selected.

Figure 28

The Use ClassBrowseroption in the Options pull/down menu and the Instantiate Class window.



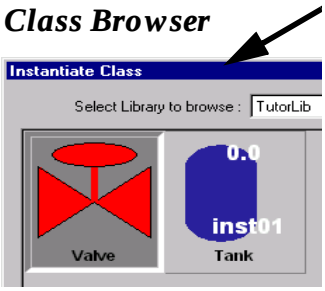
instances



Now you can start making instances of the Valve and Tank classes you created in chapter 6 *"Design of Classes"* on page 31.

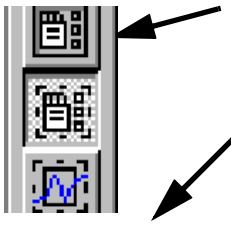
Select the **Instance tool** in the drawing editors Tools menu.

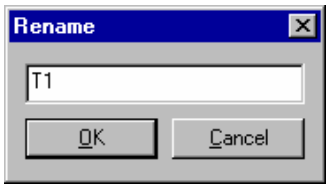
Class Browser



If **Use ClassBrowser** is selected, a browser with available libraries and classes appears. Select the library **TutorLib** first.Then select the class **Tank**. Move the cursor to approximately the middle of the picture, e.g coordinate **(180, 110)**, and click on the left mouse button. An instance of the tank will appear in the picture.

Selected Properties



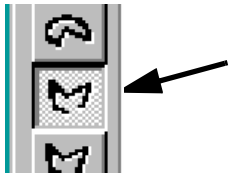


Select the **Selected Properties** from the *property window buttons*, located in the lower left corner of the drawing editor. The Selected Properties window appears. Select **Rename** in this window to give your instance a name. Place the cursor in the empty text field in the appearing dialogue box, and write the name **T1**.Then press **OK**.

Repeat the instantiating procedure for the valve class, creating a valve instance. Place this valve somewhere over and to the left of the tank, e.g coordinate **(60, 70)**. Enter the shape property window once more, and rename the instance **V1**.

Repeat this procedure, creating another valve in the lower right of the picture, e.g coordinate **(400, 340)**. Call it **V2**.

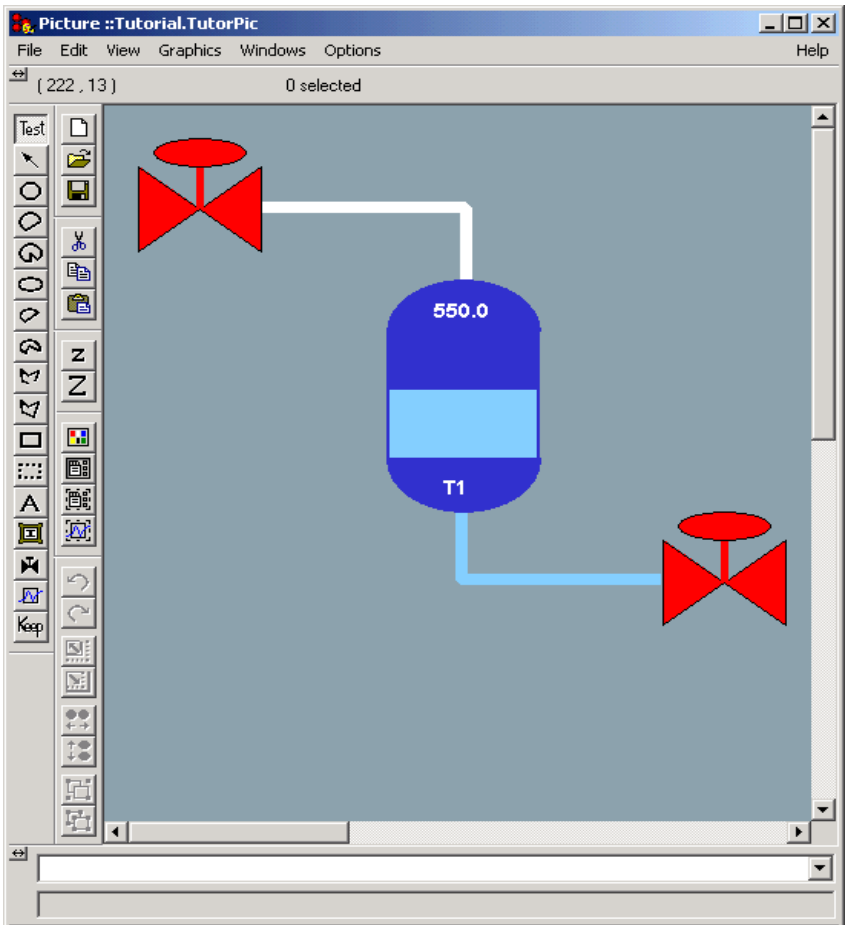
Line



Now connect the instances with a line. Select the **Line tool** in the Tools menu. Draw a line from the **V1** valve to the **T1** tank by pressing the left mouse button at the beginning, the corners, and the end points of the line you need to create the pipe between the valve and the tank. Finish the line by pressing the right mouse button. You can adjust the line afterwards with the handles if you are not satisfied with the coordinates. See Figure 29 on page 46. Draw a similar line from the tank **T1** to the valve **V2**.

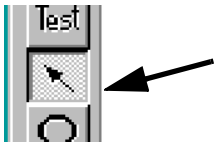
Figure 29

The picture. Two valves and a tank, interconnected with pipes.



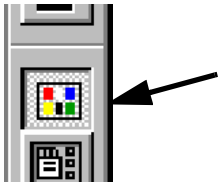
7.3 Setting Graphics Attributes

Select



Set the line width and colour from the Graphic Attributes window. Select the **Select too**l in the Tools menu. Then move the cursor to one of the lines and select it. The line is now the currently selected object on the screen and you can manipulate it through the graphic attributes window.

Graphic Attributes



The **Graphic Attribute window**, (see Figure 10 on page 20) pops up when you select the **Graphic Attributes** button from the property buttons. Select **LineWidth** among the displayed attributes and pick a line width in the list on the right side of the box, e.g number 9. Select a colour for the line in the same way. Select the second line in the picture, and repeat the procedure.

Now you should have a picture looking approximately like the one in Figure 29 on page 46.

7.4 Setting Dynamic Graphics Attributes

Now, connect the valve **V1** to the process variable **v1_state**. Grab the select tool once more and select the valve you named **V1**. To edit the state attribute of the valve, you must open the **Selected Properties** window. In this window you will find a list of attributes. Select the attribute "**int State**" and press the **Edit** button. The attribute editor window appears. Enter the window and set the value of the state attribute to the dynamic expression **`v1_state`**.

```
int State = `v1_state`;
```

Repeat this procedure for the valve **V2**, or if you prefer a faster approach, use the command line of the picture editor and write:

```
V2.State = `v2_state`
```

followed by a carriage return. If there is already something written in the command line, just select the old text and type the new one.

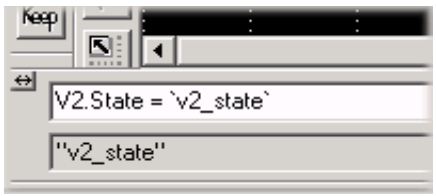
Connect the level of the tank to the process variable **t1_level** by writing in the command line:

```
T1.Level = `t1_level`
```

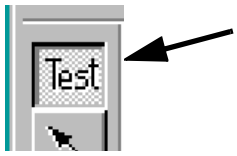
Be careful to include the right kind of quotes in these statements. The back quotes signifies a dynamic assignment. The tank level will change whenever the quoted statement changes value.

Figure 30

The drawing editors command line is situated below the drawing window.



Test mode



Before you continue, test the dialogue of the valves. Set the editor in test mode by selecting the **Test Mode** in the Tools menu. Change the states of the valves by write in the drawing editors command line:

```
v1_state = 1
v2_state = 1
```

The valves should switch colours between red and green, signifying closed and open.

Also test if the dynamics on the tank is working. In the drawing editors command line write:

```
t1_level = 500.0
```

Then press *Return*. A light blue rectangle should now cover half of the tank.

Let us add some more dynamics in the picture.

Chose the **Select Mode** and select the line connecting **V1** and **T1**. Enter the graphic attributes window. Select **Line-Foreground**. In the text edit field at the bottom of the window, write:

```
V1.IsOpen() ? lightSkyBlue: white
```

When typing is finished, press *Return* to insert this dynamic into the selected shape.

Select the line connecting **T1** with **V2**. Repeat the procedure, but this time write:

```
V1.IsOpen() || T1.Level > 0 ? lightSkyBlue : white
```

Set the editor in **Test Mode** and test the new dynamics by opening and closing the valves as described above using v1_state and v2_state.

Save your picture. Select **Save** in the drawing editors **File** menu.

7.5 Creating a Window

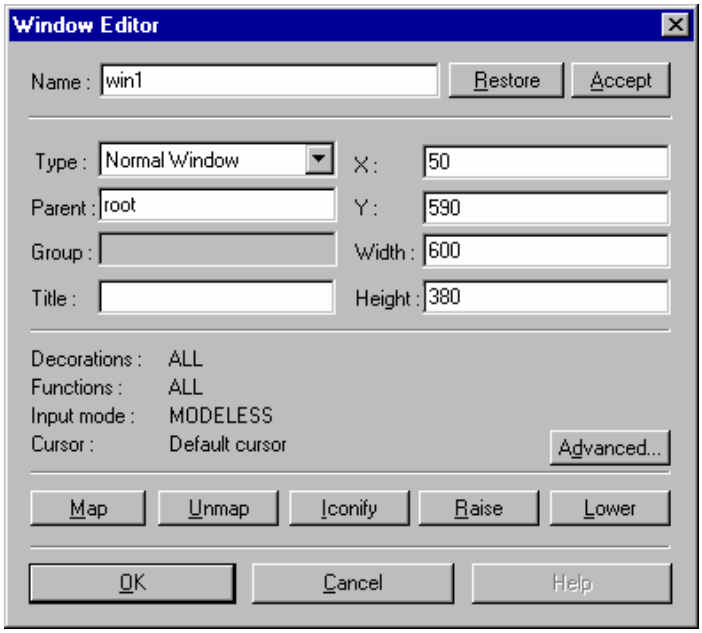
Your picture is now finished, but still you cannot access it from outside of the graphics editor. To use the picture directly in your application, the application needs a window of its own.

You use the Window editor in GED to create a window for your application.

Open the Window editor by selecting **Windows** under the Tutorial application in the tree view, and then select **New** on the File menu, or the menu that pops up when the right mouse button is pressed.

Figure 31

The window editor.

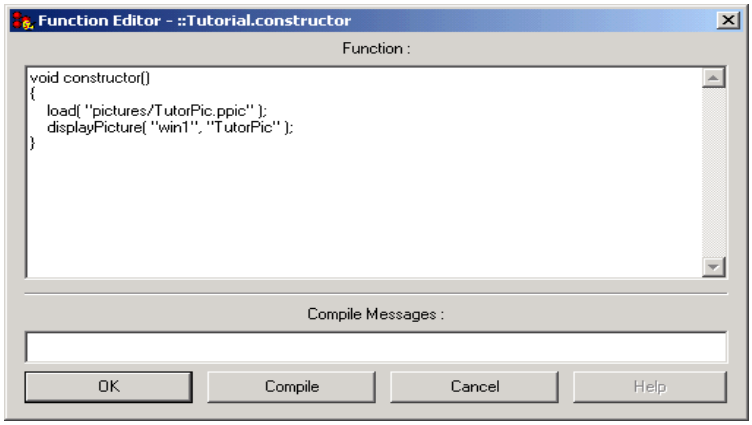


In the **Window editor** that is displayed (see Figure 31), change the name of the window to **win1**, and change the width to 600, and the height to 380. Click on the **Accept** button to set the values into the window. You can click on the **Map** button to have a look at the window. Click on the **OK** button when finished.

This will supply your application with a window. To display the picture in the window, add the following constructor function to your application, by selecting **Functions** under the Application level. Type the function text as displayed in Figure 32. Then

Figure 32

The Function Editor window showing the constructor function.



Click on the **Compile** and **OK** buttons and save the **application** to file.

To test the new window, leave the graphics editor (make sure you saved your picture, library, and application), and shut down the RTM.

Stop the RTM by closing the Output Window of the RTM. Answer *Yes* to the question *"Do you really want to quit?"*.

Restart the RTM by double clicking on the **`Tutorial.pctx`** file in the Windows Explorer.

When the RTM starts up it will automatically load the application, open the window and display your picture.

7.6 Using Tdoc as Source

The application and library and picture that we have created using GED, could also have been created without GED, by using a text editor and the ProcSee Code Compiler PCC.

PCC converts ProcSee.Tdoc files to binary files for applications, libraries, pictures, etc. The RTM produces .Tdoc files, when Document is chosen from the menus in GED.

In order to document a picture, select the **File** menu and click on **Document** to create an ascii file (**TutorPic.Tdoc**) of the TutorPic.

At the moment not everything possible to do in the ProcSee system, is possible to do from GED, like adding trend logger information. In these cases you have to use a text editor to add these things to the system.

The sequence here are that you produce the .Tdoc file, use the text editor to change the .Tdoc file, then you run PCC on the file to create the binary file that you read into the system with the Open menu options in GED, or by restarting the RTM.

One situation where you need to convert all the files to .Tdoc files, except for documentation, is if you want to store the files in a source code control system, which only handles text files. In this situation, you will also want to create a makefile for your system, that compiles all your files (Both .c and .Tdoc). an example of such a makefile is shown on page 48.

To compile the files, select the **Tutorial.Tdoc** in Windows Explore, right click on it and choose *"Compile"* from the popup menu. Use the same procedure to compile the **TutorLib.Tdoc** and **TutorPic.Tdoc** files.

8 The Simulator

In this chapter we introduce a small program, coded in C, that interacts with our user interface. The program is using the ProcSee *Application Programmer's Interface (API) to update values of database variables.*

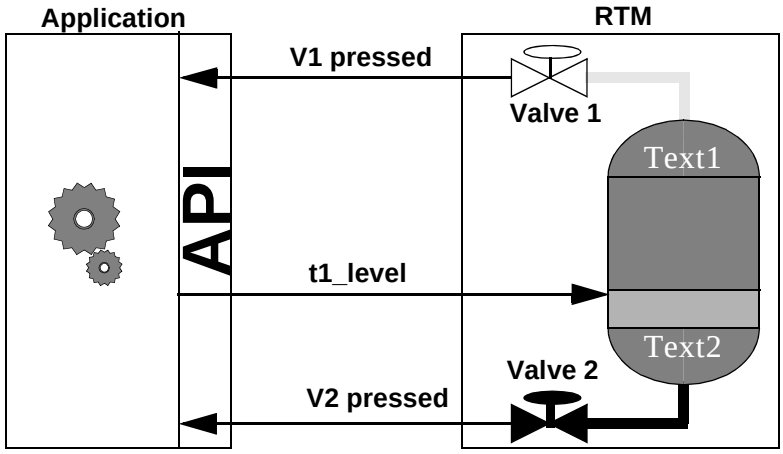
8.1 Task

Imagine a process consisting of two valves with a tank in between connected by pipes. Water flows through the first valve into the tank and out through the second valve.

We will make a small program that changes the level of the tank according to the state of the valves. If valve no. 1 is open, the level of the tank should increase by a steady rate. If valve no. 2 is open, the level should decrease accordingly.

Figure 33

The level of the tank is calculated by the external application according to the state of the two valves. The valves are controlled by an operator .



The application program will communicate with ProcSee through three process variables and a register function, expressing the state of the two valves and the level of the tank.

8.2 Source

To create this program, a source file: **Tutorial.c** has to be created. If you don't want to do this now, all files for the tutorial can be found at %**PROCSEE_DIR%\tutorial\winArch\result**, and you can copy some of the files you need from this directory.

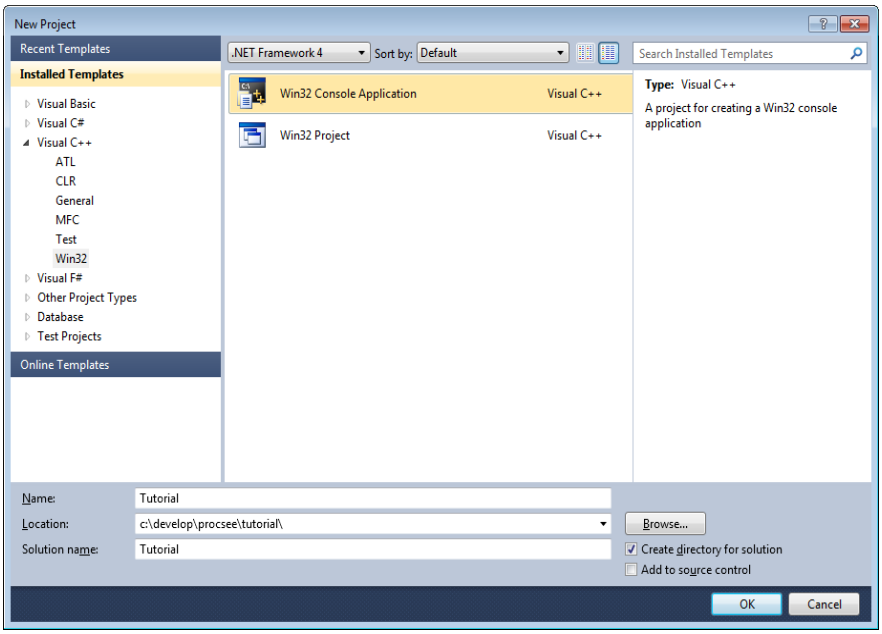
Visual Studio

This section shows how you can use **Microsoft Visual Studio** to follow the Tutorial. We start with a new solution and project

Choose "**New...**" from the file menu and select "**Win32 Console Application**" from the "**Projects**" tab. (see Figure 34)

Figure 34

New project



Give the project a name, e.g.: **Tutorial**. **Location** is where you want your program file to be. Then click on the **OK** button.

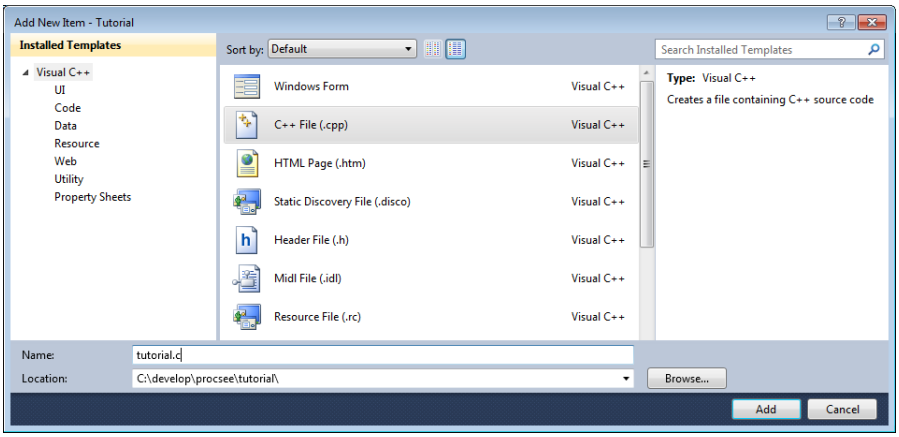
Select the "**An Empty Project**" radio button in the "**Application wizard**" and press "Finish".

Now, you can either create an empty file called **Tutorial.c**, or copy the file from %**PROCSEE_DIR%\tutorial\winArch\result**.

To create a new file, select "Add New Item" from the Project menu and select the "C++ File(.cpp)" template from the list. Enter the name "**Tutorial.c**" in the "Name" text box and press "**OK**".

Figure 35

New file



Now you have a text file where you can enter your program code for the tutorial

If you do not want to enter all the text in Tutorial.c, you can insert the file into your Visual Studio project by right-clicking on the *"Source files"* folder in your project, and chose *"Add - Existing Item"* from the popup menu.

Go to the **%PROCSEE_DIR%\tutorial\winArch\result**, select the **Tutorial.c** file and click on the **OK** button. The file will now be inserted into your project.

The project need to use the ProcSee and **ws2_32** libraries, the ProcSee header file include path, and set the **winArch** define.

Detailed instructions on how this is done, can be found in the *Reference Manual* under *"Part 3 - Manual Pages - Application Programmers Interface Manual pages - Compiling and Linking"*

Since this tutorial will configure a Visual Studio 2010 project with the debug target we will use the following settings.

Additional Library Directories is set to "$(PROCSEE_DIR)\lib\winArch\vc10"

Additional Dependencies adds **ws2_32.lib**, and **ProcSeeApiC_MDd.lib**

Under C/C++ settings we define **winArch** and set **Additional include directories** to be "$(PROCSEE_DIR)\include"

No Visual Studio

If you are not using the Visual Studio, you need to create a working directory where you create the files: **Tutorial.c** and **makefil**e. The content of these files are described in section 8.3 - 8.6.

The **tutorial.exe** file is also needed if no compiler is available.

Copy the **tutorial.exe** from:

%**PROCSEE_DIR%\ tutorial\winArch\result**,

8.3 Requirements

The program, **Tutorial.c**, contains some definitions and variable declarations. Three functions that will be set up as callback functions (user supplied functions that will be called by the API), and a main program. In ProcSee's Application Programmer's Interface (API) a callback function is used to update the actual variable from both the application program and the RTM.

Constant definitions, type definitions, and function declarations required to use the API, are found in the **api.h** header file.

PROGRAM

```
/* c */

#include <stdio.h>
#include <time.h>
#include <api/api.h>
```

In order to control the flow into and out of the tank, we define some constants representing the flow through the valves, and the maximum and minimum allowed level of the tank

PROGRAM (*continued*)

```
/* The max and min levels of the tank */

#define MAX_LEVEL 1000.0
#define MIN_LEVEL 0.0

/* The flowrates of the valves */

#define V1_FLOWRATE 50
#define V2_FLOWRATE 50
```

We declare a Boolean variable to keep track of the ProcSee connection.

PROGRAM (*continued*)

```
bool isConnected = 0;
```

For more information about the connection of the application code to the RTM, see section 19.1 *"A Small Example"* in the ProcSee User's Guide.

The three essential variables of the application have an internal representation in the program.

PROGRAM (*continued*)

```
/* The application variables */

float  t1_level = MAX_LEVEL;
int    v1_state = 0;
int    v2_state = 0;
```

In addition the variables have three identifiers used by the API when updating the ProcSee representation.

PROGRAM (*continued*)

```
/* Id's for the variables */

int t1_id;
int v1_id;
int v2_id;
```

Three functions are called in different situations. A repetitive function is called to change the water level continuously. Another function is called when the program connects, the ***up()*** function, and a third is called if the connection is broken, the ***down()*** function. In addition one register functions ***changeValveState()*** must be declared.

PROGRAM (*continued*)

```
int32 repeater( int32 );    /* This function is called repeatedly     */
void  up( int32, char* );    /* Functions called at connection events */
void down( int32, char* );   /* Functions called at connection events */
int changeValveState( int32, void* ); /* Register function */
```

8.4 Initializing the Application

The main routine of your program should initialize the application, set up a process handler and start a ***main*** loop.

PROGRAM (*continued*)

```
int main()
{
```

To get contact with ProcSee, the API needs to know on which host to find the communication server and the name of the ProcSee run-time manager. In addition we have to identify our program with an application name and a process name.

The actual initialization of the API is done with the API function ***PfInitialize***. Note that all the API functions have the prefix ***Pf.*** The ***Pf*** is an abbreviation for ProcSee **function**.

PROGRAM (*continued*)

```
PfInitialize( "Tutorial", "Tutorial", "rtm", NULL, 0, 0, up, down );
```

PfInitialize takes eight parameters:

applName The name of the application. Our application is called "Tutorial". Several tasks can be running under the same application with different process names.

processName The name of the process. Our process is called "Tutorial". Several tasks can connect to ProcSee with the same application name, but the combination of application and process names must be unique. If NULL, the name entered in ***applName*** is used.

rtmName The name of the ProcSee run-time manager. If NULL, the default name of the RTM, which is "**rtm**", is used.

controlHostName Name of the host where Control is running. If NULL, the environment variable **$CONTROLHOST** is used.

cacheSize Unused for the moment, use 0 as default value.

master Unused for the moment, use 0 as default value.

up The function to be called when our task connects to ProcSee.

down The function to be called if the ProcSee connection is broken.

If something went wrong in the initialization routine, this will be registered in the **apiError** variable and can be displayed through **PfPrintSystemError.** The same procedure can be used after calling other API functions.

PROGRAM (*continued*)

```
if( apiError != OK )
{
    printf( "Could not init API...\n" );
    PfPrintSystemError( apiError );
}
```

Note that ***PfInitialize*** doesn't actually initialize the ProcSee run-time manager. It initializes a routine that tries to get in touch with the RTM. When it succeeds the callback function registered in the **up** parameter of ***PfInitialize*** is called. Your program's initialization should be placed in this function, not after the return from ***PfInitialize***.

We want the program to update the process variables at a steady interval. We can accomplish this by using a "Process handler" - a function called at regular intervals. We call our process handler "repeater" and register it with ***PfSetProcessHandler***.

PROGRAM (*continued*)

```
PfSetProcessHandler( repeater,1000 );/*repeater called every 1000 ms*/
```

The function repeater should be called every second.

8.5 Program Flow

All preparations are finished and the program can enter its main loop. All connection callbacks and process handlers will be executed in the ***PfMainloop***.

PROGRAM (*continued*)

```
printf("Enter Pf loop\n");
if( PfMainLoop() != OK )
{
    printf( "Could not start Pf loop ...\n" );
    PfPrintSystemError( apiError );
}
```

The main loop is normally not terminated. If it terminates anyway, we close our ProcSee connection and exits the program.

PROGRAM (*continued*)

```
if( PfClose() != OK )
{
    printf( "Could not close Pf ...\n" );
    PfPrintSystemError( apiError );
}
printf( "Exit\n" );
return 0;
}                   /* End of main program */
```

When the program connects to ProcSee the connection callback "***up***", is called. This is the natural place to create register functions and process variables.

PROGRAM (*continued*)

```
void up( int32 status, char* msg )
{
```

This callback is called each time the ProcSee connection is established. The register function need two parameters.

PROGRAM (*continued*)

```
int32 numArgs = 1;
 PfTArg args[] = { PfCUnsignedChar, 0 };
```

To know whether to create the register function and variables or not, we check the status parameter.

PROGRAM (*continued*)

```
if( status & PfCrtmResume )
{
    printf( "Connection re-established \n" );
    return;
}
printf( "Connection established \n" );
```

If the **PfCrtmResume** bit is set, this routine has already been executed and ProcSee remembers it. If not we go on creating ***PfRegisterFunction***.

PROGRAM (*continued*)

```
PfRegisterFunction( "changeValveState_f", changeValveState, numArgs,
                    args);
if( apiError != OK )
    printf( "Failed to register changeValveState.\n" );
```

The parameters in the ***PfRegisterFunction*** is the **name** of the function, the **function** itself, the **number of arguments** and the **argument description**. The ***changeValveState*** function have 1 argument, the name of the variable to be toggle. Continue to create the variables.

PROGRAM (*continued*)

```
v1_id = PfCreateVar( "v1_state", PfCInt, NULL, 1, &v1_state );
if( apiError == OK )
    printf( "variable v1_state added \n" );
```

We create our variables with a name, a type, a Record type name, a cache fix parameter (unused) and a value pointer.The variable name is identical to the one we use in the definition of our picture and in the database definition script. We have chosen integer as a representation for the state of the valves in our picture. In this context, the integer is defined **PfCInt**. Since the type is not a record, you should enter NULL for the record type name.

In the last parameter we enter the address to the internal representation of the variable. If the variable is changed in one of the pictures it will be reflected in this variable.

PROGRAM (*continued*)

```
v2_id = PfCreateVar( "v2_state", PfCInt, NULL, 1, &v2_state );
if( apiError == OK )
    printf( "variable v2_state added\n" );

t1_id = PfCreateVar( "t1_level", PfCFloat, NULL, 1, &t1_level );
if( apiError == OK )
    printf( "variable t1_level added\n" );
```

We add two more variables representing the state of a second valve and the level of a tank. The tank level is represented as a float; **PfCFloat**.

In order to increase speed when creating variables, the function ***PfCreateVar*** is storing the variable information in a local buffer, and relies on the user to call the API function ***PfFlushCreateVar*** when variable creation is finished. When ***PfFlushCreateVar*** is

called, all the buffered variables are created in the RTM at once. Before leaving the connection callback, a flag is set to remember that the ProcSee connection is working.

PROGRAM (*continued*)

```
    PfFlushCreateVar();
    isConnected = 1;

}                   /* End up */
```

If the ProcSee connection is broken the callback function "***down***" will be called. In this function you don't have to do much. Just print a message and reset the flag.

PROGRAM (*continued*)

```
void down( int32 status, char* msg )
{

    printf( "Lost contact with  ProcSee (`%s`)\n", msg );
    isConnected = 0;

}                   /* End down */
```

Every second the ***repeater*** function will be called

PROGRAM (*continued*)

```
int32 repeater( int32 i )  /* This function is called repeatedly */
{
```

Here we check the state of valve number 1. If the valve is open

PROGRAM (*continued*)

```
    if ( v1_state != 0 )  /* Valve V1 is open */
        t1_level += V1_FLOWRATE;
```

the level in the tank should increase according to the flowrate of the valve.

If valve number 2 is open the level of the tank should decrease according to the flowrate of this valve.

PROGRAM (*continued*)

```
    if ( v2_state != 0 )  /* Valve V2 is open */
        t1_level -= V2_FLOWRATE;
```

Before updating the ProcSee variable, we check the tank level against the limits. At this point it could be appropriate to execute some kind of alarm action, but we are satisfied with just stopping the tank level from increasing / decreasing.

PROGRAM (*continued*)

```
    if ( t1_level < MIN_LEVEL )  /* Tank is empty */
        t1_level = MIN_LEVEL;

    if ( t1_level > MAX_LEVEL )  /* Tank is full */
        t1_level = MAX_LEVEL;
```

Finally we tell ProcSee about the change of the value. We use the flag from the connection callbacks to determine if the ProcSee connection is working. If the connection is ok, the value is put into the send buffer with the function **PfSend**.

PfSend uses the id of the variable, that was returned by **PfCreateVar** in the ***up*** function. When all the values has been put into the send buffer, the buffer is transmitted to the rtm with the **PfFlush** function.

PROGRAM (*continued*)

```
    if ( isConnected )
    {
        PfSend( t1_id );  /* Make the changes known to ProcSee */
        PfSend( v1_id );
        PfSend( v2_id );
        PfFlush();
    }
    return OK;
}                         /* End of repeater */
```

The function ***changeValveState*** will be called every time one of the valve is clicked on with the left mouse button.

PROGRAM (*continued*)

```
int changeValveState( int32 numArgs, void* data )
{
    int32 size, type, id;
    void *theData;
    int *value;

    if( numArgs != 1 )
       return !OK;

    theData = PfGetFuncArg( &data, &type, &size );
    if( type != PfCUnsignedChar || size > 64 )
       return !OK;

    id = PfId( PfLocalProcess, (char*)theData );
    if( apiError == OK )
    {
       value = (int*) PfData( id ):
       *value = !*value;
    }

    return OK;
}                    /* End of changeValveState */
```

This function toggle the state of the two valves in the process picture.

8.6 Compiling

In **Visual Studio**, Select **Build Tutorial.exe** from the "Build" menu.

For more information on compiling, see "Application Programmers Interface Manual pages - Compiling and Linking - Windows" in the ProcSee Reference Manual.

8.7 Testing Dynamics

Now its time to start the program you wrote in chapter 8 *"The Simulator"* on page 51. This program is a console application, so it will run in a Command Prompt window. The program can either be started by double clicking on it in the NT explorer, or if using Visual Studio to run it. You can also start it from a Command Prompt.

At your shells command line, write the program name:

```
Tutorial
```

or run your program from **Visual Studio.** Either by selecting "Start Debugging" or "Start without Debugging" from the "Debug" menu, or by pressing **F5** or **Ctrl-F5**.

If the program is working correctly it will reply with the following messages in the terminal window:

```
Enter Pf loop

variable v1_state added
variable t1_level added
variable v2_state added
```

If you are loading the picture in GED, set the editor in test mode.

Open valve **V1**. Close valve **V2**. Watch the level of tank **T1** increase.

Close valve **V1**. The tank level stops increasing.

Open valve **V2**. Watch the level of tank **T1** decrease.

Open valve **V2**. The level of tank **T1** stabilizes.

9 Historic Trend

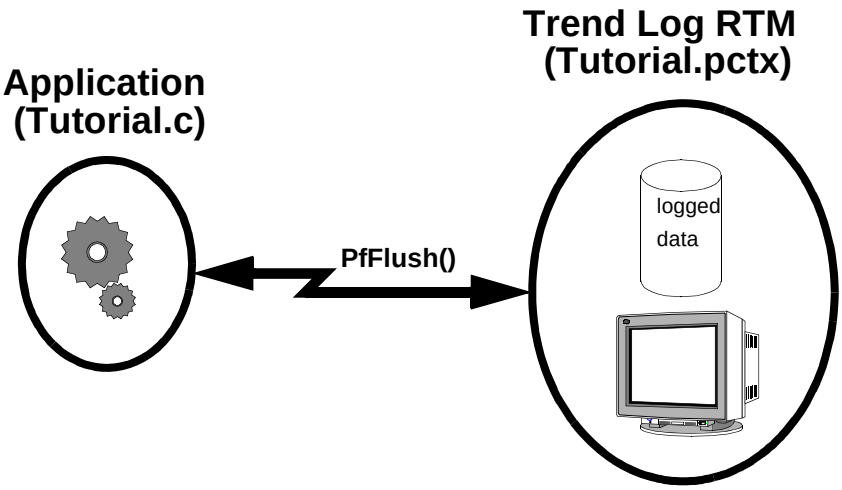
The aim of this chapter is to give a brief description of the ProcSee trend logging system. Historical data can be displayed as curves in trend diagrams, referred to as Trend Display. Trend logging in our terms means the process of collecting and administrating data.

9.1 Internal Trend Logger

Many applications need to store and display historical data. The number of variables to be trended vary a lot. Thus the design of the ProcSee trend logger had to be as flexible as possible to meet a wide range of requirements.

In our tutorial, there are only a few variables that are to be trended. In this case, the trend logger can easily run within the RTM without affecting performance of the system. This is referred to as an internal trend logger.

Figure 36 *The application developed in this tutorial is connected to an internal trend logger in Tutorial.pctx.*



9.2 Trending Variables

Let us say we want to see how the level of the tank is changing over a period of time. The way this is done in ProcSee is by setting up a trend logging system. This system consists of two parts. Part one says which variables to trend, the log medium (disk or memory), how often they should be logged etc. Part two is concerned with the physical presentation or display of the historic data, e.g. colour of the curve.

Unfortunately, GED cannot be used to specify how variables should be trended in this version of ProcSee. So we have to use the combination of a text editor and pcc to do the job.

Edit Tutorial.Tdoc

Document the application, by using the menus in GED, to get a **Tutorial.Tdoc** file. Open the file **Tutorial.Tdoc** in your text editor. Try adding the following lines within the scope of application:

Declaration of a trend logger which is internal to the display RTM

```
trendLog myLog                          // name of the logger
{
    timeTickIntvl= 1;                    // 'ticked' every second
    timeMaster   = 2;                    // 'ticks' sent by application
    timeVariable =     "myGlobalTime"; // variable for time stamping
    trendVariableFile = "myVars.ptrv"; // where to find trend variables
}
```

This declaration says that we will have a trend logger that is named ***myLog*** and it will be ticked every second. The **timeMaster** variable has a value of 2, this indicates that an external application is responsible for ticking the trend logger at the specified interval. In our case this application is the simulator we have written.

The trend logger must have a notion of time. Here this is done by letting the application set and advance time by changing the value of a process variable named **myGlobalTime**.

Save Tutorial.Tdoc

Further on, we want to declare the trend variables in a separate file. The reason for keeping them separate, is that the number of trend variables tend to be very large in most trend logging systems.

pcc Tutorial.Tdoc

After adding the lines in the application, save the file and run it through **pcc** to get a new **Tutorial.pctx** file. This is done by clicking on the file in the Windows explorer with right mouse button and chose **Compile** in the pop up menu. This file will be loaded into the RTM making the trend logger to start.

Edit Tutorial.c

The next step is to add the **myGlobalTime** variable to the simulator program. Add the time variable as described below:

Add variable to the simulator program

PROGRAM

```
/* The application variables */

float  t1_level     = MAX_LEVEL;
int    v1_state     = 0;
int    v2_state     = 0;
int    myGlobalTime = 0;          // telling trend logger current time

/* Id's for the variables */

int t1_id;
int v1_id;
int v2_id;
int time_id;                      // id for time added
```

The variable must then be created:

Initialize and create the time variable in the API

PROGRAM

```
    myGlobalTime = time(NULL);
    time_id = PfCreateVar("myGlobalTime", PfCInt, NULL, 1, &myGlobalTime);
    if(apiError == OK)
        printf("variable myGlobalTime added\n");
```

PfSend

We have now created the variables necessary to setting time in the trend logger. In the first version of the simulator, we only updated the level of the tank, **t1_level**. This time we also need to update the value of **myGlobalTime**. This is done by calling the function ***PfSend*** again with **time_id** as parameter. The ***repeater*** function will now look like:

The repeater() function now updating more than one variable

```
PROGRAM

int32 repeater(int32 i)
{
    if ( v1_state != 0 )
        t1_level += V1_FLOWRATE;

    if ( v2_state != 0 )
        t1_level -= V2_FLOWRATE;

    if ( t1_level < MIN_LEVEL )
        t1_level = MIN_LEVEL;

    if ( t1_level > MAX_LEVEL )
        t1_level = MAX_LEVEL;

    if ( isConnected )
    {
    myGlobalTime++;                 // increment time by one second
    PfSend( t1_id   );              // new value for tank level in RTM
    PfSend( v1_id );                // the state of V1
    PfSend( v2_id );                // the state of V2
    PfSend( time_id );              // new value for time variable in RTM
    PfFlush();                      // update variables, tick trend log
                                     // and update screen
  }
  return OK;
}
```

make

Next step is to compile and link your simulator program. This is done by issuing the **make** command at your working directory.

Edit Tutorial.pdat The variables will also have to be declared in the **Tutorial.pdat** file:

Showing the Tutorial.pdat file

DATABASE DEFINITION

```
// .pdat

float t1_level   = 0.0;
int32 v1_state   = 0;
int32 v2_state   = 0;
int myGlobalTime = 5;       // time variable must be defined here
```

The last thing that remains to be done in order to configure the trend logger, is to create the file containing trend variables, i.e. to specify which of the variables declared by **PfCreateVar** that is to be trended.

Edit myVars.Tdoc Using a texteditor create and open a new file called **myVars.Tdoc**. Place the file in the same directory as the Tutorial.pctx file. The content of this file should look like:

Trend variable configuration file

```
trendVarConfig myVars
{
    trendVar t1_level
    {
        lCycle=1;
        hist=10800;
        type=5;
        lo=-9999999;
        hi=9999999;
    }
}
```

The **lCycle** attribute of the trend variable **t1_level** says that it is to be logged every second.The **hist** attribute is the time span of which values are logged. (10800 / 1 gives 10800 values to be logged in a ring buffer). Attribute **type** has a value of 5, which means the actual value of the variable is logged. **lo** and **hi** reflects the lower and upper range of the trended value.

pcc myVars.Tdoc This file must now be run through pcc to get the **myVars.ptrv** file. This is done by clicking on the file in the Windows explorer with right mouse button and chose **Compile** in the pop up menu.The application **Tutorial.pctx** can now be reloaded from GED. Click on the **Application** chose **Open...** and select Tutorial.pctx in the Open window coming up. After selected the Tutorial.pctx file press the **Open** button.

9.3 Trend Curves

Up to now we have been looking at how to define an internal trend logger. Now it is time to present data in a ProcSee picture. To define a trend, select the trend tool in the Tools menu in the Drawing editor, and move the cursor to position (300,40) and press the left mouse button. The trend graphic shape will appear in the window with a default width and height. As default one time label shape is placed above the trend diagram.

Now it's time to change the width and height of the trend. Select the Selected Properties menu from the Tools menu and choose the width and the height attributes and set the geometry to 300 and 200 respectively. Set the **foregroundColour** and **foreground-FillColour** to *medBlue* and *steelBlue*.

To get the connection to the trend logger we just have defined, the default attribute **trLog** in the Trend shape is set to *myLog*. Recall that this trend logger name is the one specified in the Tdoc file in the application scope with the keyword **trendLog**. Another important attribute in the trend shape is **timespan**. This attribute is used for specifying the number of seconds of the curve which is visible in the trend diagram. As default this attribute is set to 240, change it to 5 minutes by setting **timespan** to 300.

To define a trend curve select the **Selected Trend** window in the **Tools** menu. (Keep the Selected Properties window open). A window will appear which is used for defining all shapes a trend may consist of. From this **Selected Trend** window it is possible to create a **Presentation** (trend presentation), **Time Labels**, **Ruler** and **Grid**.

The first we will do is to create a **Presentation** shape. Select the **Presentation** button from the options menu which will enable the **New** button. Press this **New** button and select the *TrendPres1* shape in the list. The attributes that are available for the **Presentation** will appear in the **Selected Properties** window. If this window isn't open then open it by choosing it from the **Tools** menu.

The trend curve just created is by default called *TrendPres1*. Select the *TrendPres1*. Change its name by selecting the rename button in the **Selected Properties** window. Rename it to *myCurve*. It is very important to set some of the attributes in the **Presentation** shape to another value than system default, otherwise no trend curve will appear on the display. The essential attributes are:

- **variable**
- **logFrequency**
- **lowerLimit**
- **upperLimit**

The attribute **variable** must be set to *"t1_level"* which was added to the trend variable file **myVars.Tdoc**. The name of the trend variable file where set up in the **trendLog** part within the application scope. Change the attribute to *"t1_level"*. This variable is used as an indication of the amount of liquid in the tank. By changing the state of the valves the trend curve *myCurve* will reflect the amount of liquid in the tank for the last 300 seconds.

A trend variable can be logged at different time intervals. As an example, *t1_level* can be logged at intervals of 1, 5 and 10 seconds. The **logFrequency** attribute can be used to specify the desired log interval for the current **Presentation** shape. A value of -1 (default) indicates that the best possible resolution for this trend variable will be supplied by the trend logger.

lowerLimit and **upperLimit** is set to the minimum and maximum value the variable *t1_level* may get. In our tutorial the liquid in the tank will never exceed 1000.0 nor will it be lower than 0. Set **upperLimit** and **lowerLimit** to 1000.0 and 0 respectively.

The default colour of the trend curve is white. To change the foreground colour select the attribute **foregroundColour** in the list and set it to the value ‘yellow‘. The **LineWidth** attribute should be set to 2.

Select a suitable font by choosing the **Time Labels** in the **Selected Trend** window and the attribute **theFont** from **Selected Properties** window.Look in the Graphic Attributes window to see which fonts to select from. You can let it be 0.

The definition of the trend shape is now finished. Remember to save (and document) the picture.

Make a Frame Work 10

In this chapter we will make a start picture using a button from the example libraries Buttons.plib, start and quit the process from a button and add a quit when removing the window.

10.1 Make a start picture

Copy the **Buttons** library found in **examples/libraries/Buttons/winArch** to the **libraries** directory. Select the **Libraries** node in the tree, and right click on it to get the popup menu. Select **Open** on the popup menu. Select then **Buttons.plib** in the Open popup window and click on the **Open** button. Save the application.

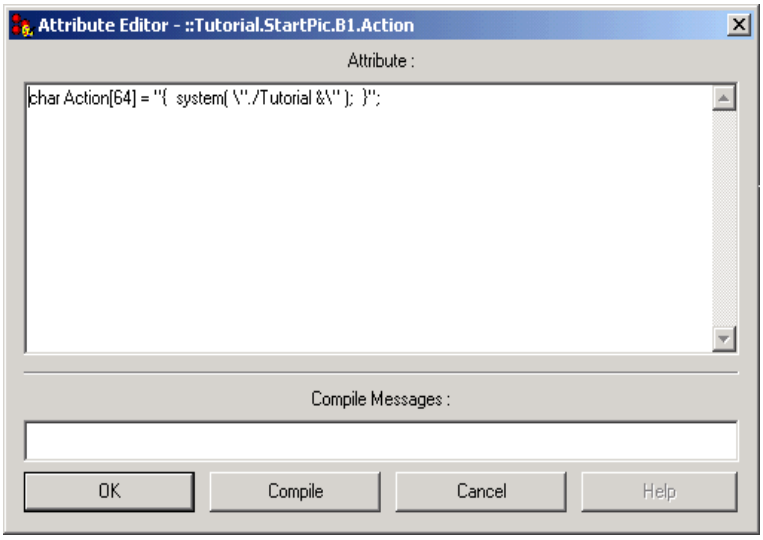
Select the **Pictures** node in the tree, and select **New...**. Set the size and background colour as in the **TutorPic** picture. In the **File** menu do **Save As...** and name the picture **StartPic**.

10.2 Use the librarie Buttons

Select the **Instance tool** in the drawing editors tools menu. Choose Buttons. Select the **PushButton** in the class browser, and place it in the picture, e.g coordinate **(260, 220)**. Select the **Selected Properties** from the property window buttons. Click on the **Rename** button and write "**B1**". In the command line to the Editor type **B1.MakeDynamic()** to set some of the attributes to the button dynamic. Select the attribute "**char Action[64]**" and press the **Edit** button.Type in the following statement:(see Figure 37). Set the **Attributes Height** to 40 and **Width** to 120.

Figure 37

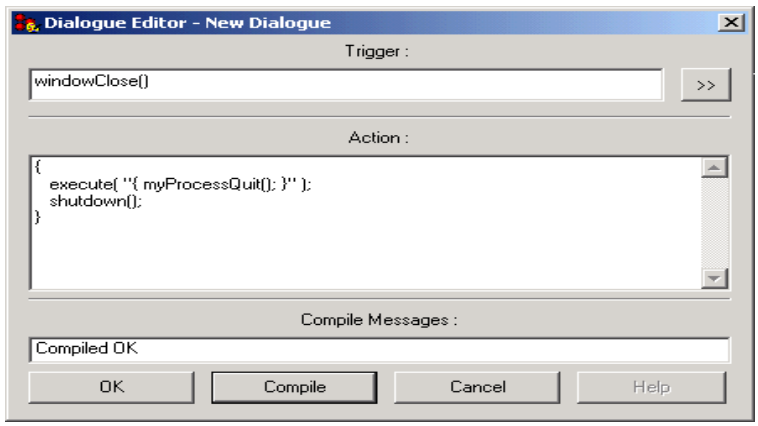
The Attribute Editor window.



Click on the **Compile** button to verify that it compiled successfully, before clicking on the **OK** button. Select the attribute **char LabelText[32]** and type "**Start Process**". Do the **Compile** and **OK**. Then draw a **Text** shape in e.g (**130,170**). In the attribute **char* theText**, type "**Simulator Not Running**". Select font "**helv_b_34**" from the **Graphic Attributes** window. In the **Picture Properties** window choose the **Dialogue** and click on the **New** button. Type the following code as in Figure 38: Do the **Compile** and **OK**. **Save**

Figure 38

The Dialogue Editor window.



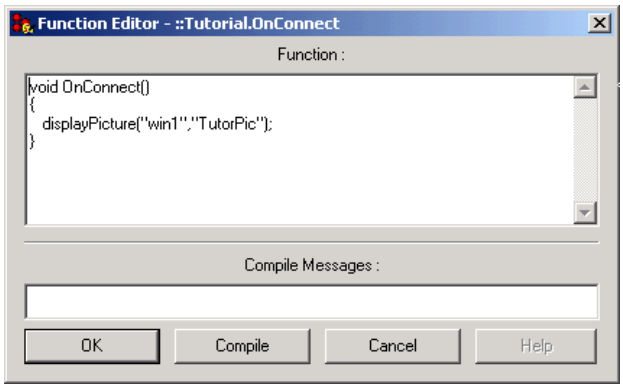
and **Document** this picture. Select the **Function** node in the Tutorial tree and choose the **constructor** function. Right click on it and click on **Edit**. Change the content as follow: Do the **Compile** and **OK.**

```
void constructor()
{
  load( "pictures/TutorPic.ppic" );
  load( "pictures/StartPic.ppic" );
  displayPicture("win1","StartPic" );
}
```

Then select **Function** and **New** node in the tree. Type the function text as displayed in Figure 39: Do the **Compile** and **OK**. Se-

Figure 39

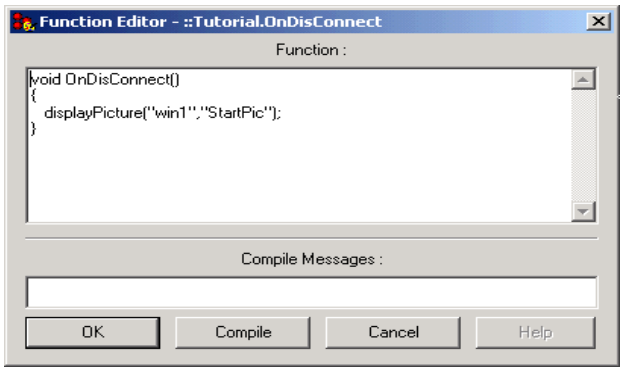
The Function Editor window.



lect **Function** and **New** node in the tree. Type the function text as displayed in Figure 40: Do the **Compile** and **OK.**This two func-

Figure 40

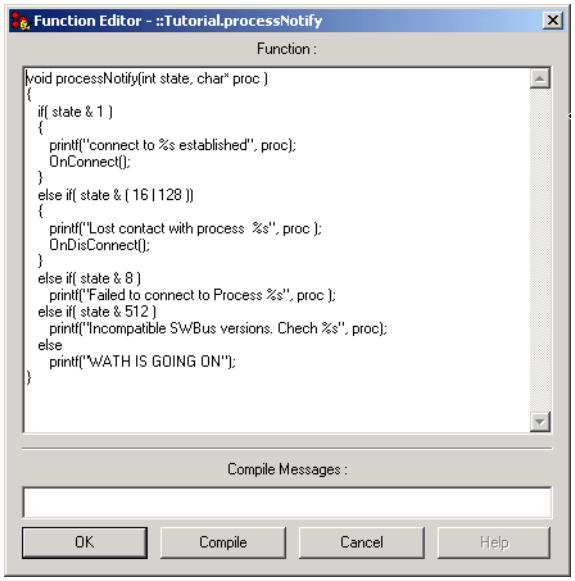
The Function Editor window.



tions are called from a function called ***processNotify*** which is a user defined callback function. Select **Function** and **New.** Type the function text as displayed in Figure 41: Do the **Compile** and

Figure 41

The Function Editor window.



OK. Do a **Save** and **Document** on the **Tutorial** in the main ged tree.

10.3 Extend the TutorPic

Open the **TutorPic** and add from the library **Buttons** the **Push-Button** in e.g coordinate **(540,300)**. In the attribute **char Action[64]** type the following:

```
"{ myProcessQuit(); }"
```

Do the **Compile** and **OK**. In the attribute **char LabelText[32]** type the following:

```
"Quit Process"
```

Do the **Compile** and **OK**.

Add the same **dialogue** on the picture level as in Figure 38 on page 74. **Save** and **document** the picture.

10.4 Extend the program

In the **Tutorial.c** add the following:

Add the declaration of the myProcessQuit to the simulator program

PROGRAM

```
int changeValveState( int32, void* );
int myProcessQuit( int32, void* ); // Registerfunction called from rtm
```

In the **up()** function do the following:

Register myProcessQuit function to the simulator program

PROGRAM

```
PfRegisterFunction( "changeValveState_f", changeValveState, numArgs, args
);
if( apiError != OK )
    printf( "Failed to register changeValveState\n" );

numArgs = 0;
PfRegisterFunction( "myProcessQuit",myProcessQuit, numArgs, args );
if( apiError != OK )
    printf( "Failed to register myProcessQuit\n" );
.
.
.
PfFlushCreateVar()
PfExecute( "TutorPic", "{ Trend1.panAbs( myGlobalTime ); }" );
/* set the trend to current time */
```

Type then the function itself:

Add the register function myProcessQuit to the simulator program

PROGRAM

```
int myProcessQuit( int32 numArgs, void* data )
{
    PfEndLoop();
    return OK;
}
```

Compile and build the program. When you are sure that all the files are saved and documented, Shut down Ged, rtm and Visual Studio. Double click on the **Tutorial.pctx** in the Windows Explorer and the start picture should appear. Pressing the **Start Process** button the program should start. The tutorial picture should switch place with the start picture. You can now open and close the valves. Shut down the program with the **Quit Process** button. The start picture will return.

11 Further Enhancements

In this chapter we present some ideas to how you can further extend your application.

11.1 Picture update mode

Make sure that the picture updates without any flickering.

Clue Use the update Mode attribute in the trend and the picture.

11.2 Window title

Change to a better text in the window frame.

Clue Use Window.title().

11.3 Trend extensions

Add grid in trend. Show upper and lower limit for the curve in the diagram.

Add a ruler and text showing the ruler value and the ruler time.

Clue Use TrendRuler, dialogue trigger cursorMoved, getRulerValue / getRulerTime.

Make it possible to pan back and forward in time outside what is the time span in the trend shape.

Clue Use panAbs and panRel.

Let the trend curve change colour when crossing a limit.

Clue Use the tr() function.

11.4 Plot the picture

Plot the picture using a button.

Clue Use the pTALK function exportPostScript().

11.5 Run two RTM's on the same Tutorial

Two RTM's running the same Tutorial and using the same simulator.

Clue Use -n option to the RTM, more call on PfInitialize, or perhaps use PfInit and PTALK function connectToProcess.

11.6 Use a data configuration file

Let the simulator read data from a configuration file instead of coding the variables in the simulator.

Clue Use the PfReadScript().

11.7 Performance

Measure the time RTM use to update the process picture.

Clue Use the standard library for the performance test, or use the PTALK function msecsToday and the dialogue triggers beforePictureUpdate and afterPictureUpdate.

11.8 More Process Units

Add some more classes to your library. Choose some representation for a pump, a safety valve, a regulation valve etc. How many different states should they have? Open, Closed, Running, Not Running, Starting, Closing, Half Way open etc. How should the components look like in their different states?

Draw the classes in the drawing editor. Try out different graphic attributes to represent the different states. Give the classes attributes to set the states.

Clue Use the visibility attribute to swap between different representations.

11.9 Extend the Picture

Save your class library and reopen the picture. Place instances of the new classes in the picture. i.e. a safety valve on the T1 tank.

11.10More Process Variables

Extend the database definition script with some more variables. The flowrate of valves, pump speed etc. Create and change the contents of the variables from the Tutorial program. When the tank level exceed some limit, open the safety valve.

11.11Scales

Create new classes to visualize the process variables. Scales measuring the flowrate through valves and in/out of tanks. Show pump speed as speedometer

Clue Draw a filled circular arc slice, set the opening angle equal to some attribute.

11.12Functions

Create a function that evaluates an opening angle from a flow-rate. Take into consideration the maximum and minimum flow-rate.

Clue Re-edit the water level function.

11.13Text Fields

Add an editable text field to enter the opening value of valves.

Clue Let the text field work directly on some process variable read by the Tutorial program.

12 Summary

Summary of what is done throughout this document.

12.1 What is Covered of the ProcSee System

- Chapter 1 *"Introduction"*: A general introduction to this document and the layout.
- Chapter 2 *"Preparations"*: Configuration required before starting the **ProcSee** system.
  Required environment variables:

  - **PROCSEE_DIR**
  - **ARCH**
  - **PATH**

- Chapter 3 *"Starting Up"*: The two main programs in the **ProcSee** system, used when building an application:

  - **RTM** the *Run-Time Manager*
  - **GED** the *Graphics Editor*

  The related commands used are; **rtm**, and **ged**.

- Chapter 4 *"GED"*: Description of the most important functions in the Drawing Editor. Drawing of simple objects and adding different types of attributes to these objects.
- Chapter 5 *"Application Resources"*: An overview of initial resources like colours, fonts, patterns etc. was given. Initial preparations like declaration of variables needed for operations without an application program running, was done. The data base definition script is called **Tutorial.pdat**.
- Chapter 6 *"Design of Classes"*: Design of a new class library with a valve class and a tank class. The classes were designed with shapes, attributes, and functions.
- Chapter 7 *"The Picture"*: The tutorial picture was built using the classes defined in Chapter 6 *"Design of Classes"*. The three database variables were assigned to the class attributes to get the dynamic behaviour. The pipelines connecting the tank and the valves were coded, using pTALK syntax, to reflect the state of the tank and valve instances.

- Chapter 8 *"The Simulator"*: Coding of a small simulator program. The program was written in C language, using the ProcSee *Application Programmer's Interface (API)*. The purpose of this simulator was to update the database variables to the RTM used in the tutorial picture.
- Chapter 9 *"Historic Trend"* An internal trend logger were added. The level of the tank was logged and displayed in a trend diagram as a function of time.
- Chapter 10 *"Make a Frame Work"*:A start picture was made. Quit process button and shut down dialogues was added.
- Chapter 11 *"Further Enhancements"*: Some ideas on how to build further on your application. A list of subjects with more detailed description of the different expansions possibilities is included.

12.2 Where to go From Here

When you are going to build your own application, this document will work as a good start reference. Use the different building blocks made in the tutorial as a basis, i.e the simulator program could be expanded with more variables, and the class library is also suited for reuse and expansion.

To get more detailed information on the ProcSee system, refer to the two other manuals delivered with the system: *The ProcSee User's Guide* and *The ProcSee Reference Manual*.

Index

E

F

G

P

R

S

T

U

V

W