

# ReaGeniX

## Programmer 2.0

A Real-time Application Generator

Tutorial

9.7.2002

OBP Research Oy

Teknologiantie 10 D, 90570 OULU, Finland

Tel. +358 8 551 5540

<http://www.obp.fi>

[obp@obp.fi](mailto:obp@obp.fi)

# ReaGeniX Programmer Tutorial

Copyright © 2002 OBP Research Oy

ReaGeniX is a trademark of OBP Research Oy.

Visio 2000 is trademark of Visio.

Windows and Visual C++ are trademarks of Microsoft Corporation.

OBP Research Oy does not guarantee merchantability or fitness of the ReaGeniX Programmer to any particular purpose. There is no warranty by OBP Research Oy or any other party or person that the functions contained in the software will meet your requirements or that the operation of the software will be uninterrupted or error-free. You assume all responsibility for the selection of the software to achieve your intended results, and for the installation, use and results obtained from it.

Our policy is continuous improvement. This can cause slight differences between appearances of actual symbols in the documentation and in the distributed software.

## Table of Contents

<b>1</b>	<b>INTRODUCTION</b>	<b>5</b>
	1.1 Purpose of tutorial	5
	1.2 Purpose of the ReaGeniX Programmer	5
	1.3 Scope of Application	5
	1.4 Components of ReaGeniX programmer for Visio	6
<b>2</b>	<b>WHAT YOU WILL LEARN</b>	<b>7</b>
<b>3</b>	<b>USING REAGENIX</b>	<b>8</b>
	3.1 Starting Visio	8
	3.2 Before Creating a New Project	9
	3.3 To open a ReaGeniX Template	9
	3.4 Selecting shapes from the stencil	12
<b>4</b>	<b>CREATE A STATE-TRANSITION DIAGRAM. AN EXAMPLE CASE.</b>	<b>13</b>
	4.1 Specifying a Stopwatch	13
	4.2 Creating the states	14
	4.3 Creating the transition symbols	16
	4.4 Create a timer	29
	4.5 Create local extended state variables	31
	4.6 Create the port symbols	32
	4.7 Changing the names of Actor Specification and Actor Body	34

<b>5</b>	<b>SAVING THE DIAGRAM</b>	<b>36</b>
	5.1 Name the Drawing Page	36
	5.2 Save a Drawing file	37
<b>6</b>	<b>RUNNING THE REAGENIX VISUAL PROGRAMMER</b>	<b>38</b>
	6.1 Generating the diagram	38
	6.2 Errors in the Diagram	39
<b>7</b>	<b>PUTTING IT TO LIFE</b>	<b>40</b>
	7.1 Running the Visual C++ project	40
	7.2 Run the Stopwatch	41

## 1 Introduction

### 1.1 Purpose of tutorial

The purpose of this tutorial is

- To familiarize the user with the ReaGeniX visual programmer interface and
- To give detailed instructions how to apply the ReaGeniX Visual Programmer.

In text following fonts are used to express function for the specific word:

**Rgx3w.vst**

**Bold is used for file names**

*Actor body*

*Italic is used for ReaGeniX symbol, stencil shape names*

"File / New"

Quotations is used for menu commands

### 1.2 Purpose of the ReaGeniX Programmer

The ReaGeniX Visual Programmer implements components of a real-time system. It compiles architecture and state transition diagrams to program modules written in ANSI/ISO C programming language.

### 1.3 Scope of Application

The ReaGeniX Visual Programmer can be used to produce code for hard real-time and embedded computer systems. The application areas include, but are not limited to control, measurement, sequence control, process simulation, and data communications.

ReaGeniX Visual Programmer can be used in quick prototyping, demonstration, validation of specifications and intermediate designs, and final implementation.

The code produced by the ReaGeniX Visual Programmer is independent of operating systems. So, it can be used on the top of most real-time operating systems.

## 1.4 Components of ReaGeniX programmer for Visio

The following explanations are for Microsoft Visio 2000.

In the following explanation following names are used:

- *Visiopath* is the directory where the Visio tool is installed
- *Rgxpath* is the directory where the ReaGeniX tool is installed

ReaGeniX Programmer for Visio consists of following parts:

*Visiopath*\Solutions\Reagenix\reagenix.vss - a stencil for ReaGeniX language.

*Visiopath*\Solutions\Reagenix\rgx3w.vst and **rgx4h.vst** - templates, which use reagenix.vss.

*Visiopath*\Solutions\Reagenix\reagenix.exe - a Visio add-on which reads the diagram information from Visio-tool using OLE2 interface into *pagename.vis* file. Reagenix.exe starts rgx.bat.exe after conversion. The reagenix should reside in one of Visio add-on directories.

*Visiopath*\Solutions\Reagenix\reagenix.hlp – a shape help for reagenix.vss.

*Rgxpath*\Visio\gener\rgxmsg.exe - a Windows executable for viewing the ReaGeniX warnings and error messages (file *pagename.err*). Opens the diagram page in Visio and shows the object the error message is referring by selecting it.

*Rgxpath*\Visio\gener\rgx.bat.exe - a Windows executable to control running of vis2rgx.exe and rgx2c.exe

*Rgxpath*\Visio\gener\vis2rgx.exe - a Windows console application to convert visio format diagram information into ReaGeniX (.rgx) format.

*Rgxpath*\Visio\gener\rgx2c.exe - a Windows console application to generate C-code from ReaGeniX (.rgx) format.

## 2 What You Will Learn

- To start Visio.
- To open a ReaGeniX template.
- To select shapes from stencil.
- To create and draw a state-transition diagram.
- To change the name of diagram / page.
- To save a diagram.
- To run the ReaGeniX code generator / to generate code from a diagram.
- To use the ReaGeniX Error Message log.
- To run Visual C++ project consisting the generated C-files.

### 3 Using ReaGeniX

The pictures of this chapter are from Visio 2000. ReaGeniX works also with newer Visio versions. The shape icons in stencil may look different in those versions. Also the symbols are under continuous improvement and actual symbols may differ from ones shown in these pictures.

#### 3.1 Starting Visio

This section is intended for system developers without previous exposure to the Visio-tool. To start Visio you have to select "Visio" from your Windows "Start / Programs" menu (Figure 1). If you have a Visio shortcut icon on the computer screen, Visio can also be opened by double clicking the icon.

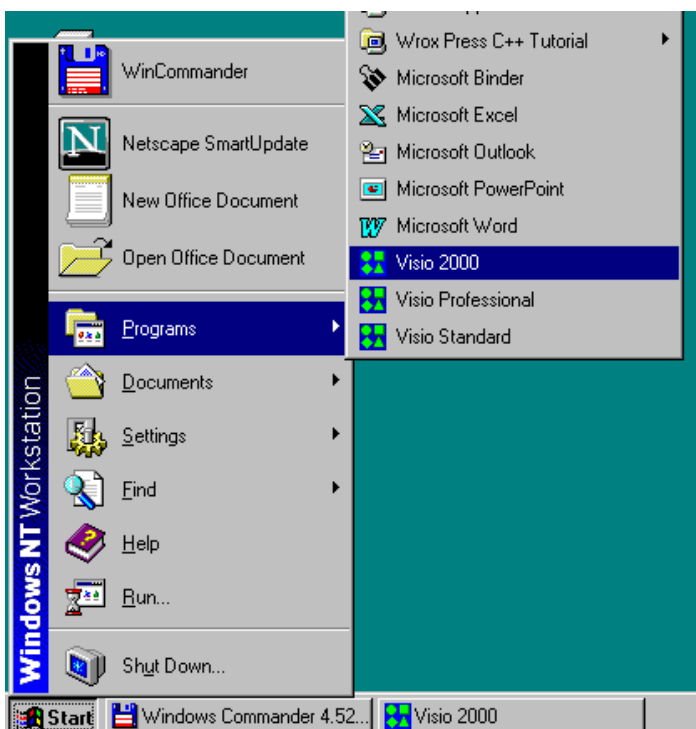


Figure 1: Selecting Visio from "Start/Programs" menu

### 3.2 Before Creating a New Project

To avoid mixing the diagrams and other files between projects it is a good idea to create an own folder (directory) for the tutorial project in this phase. Create project folder now.

### 3.3 To open a ReaGeniX Template

When you have started the Visio tool, a welcome window opens in the screen (Figure 2). Browse existing files.

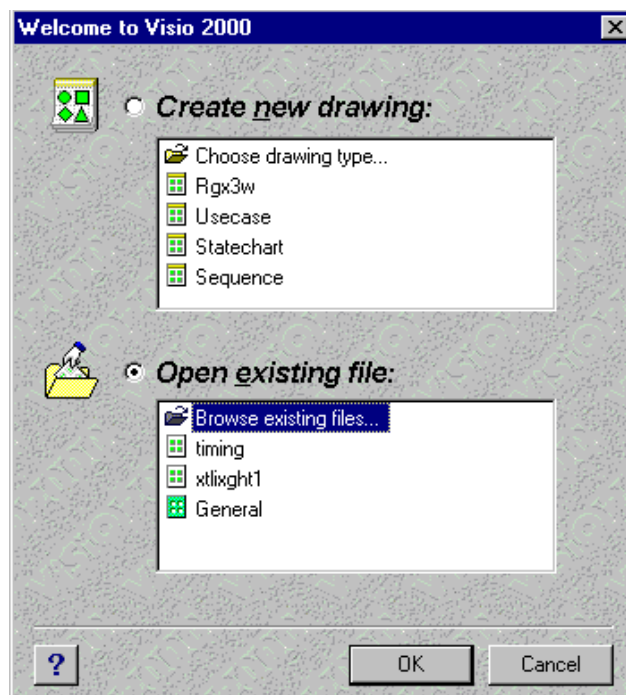


Figure 2: "Welcome to Visio 2000"

For this tutorial, you should open the template **Rgx3w.vst** (Figure3). You accept the selection by clicking the “Open” button. This opens a new diagram based on the selected template for editing (Figure 4). Other way to open a template is to open a “File / New” menu and select the template from there.

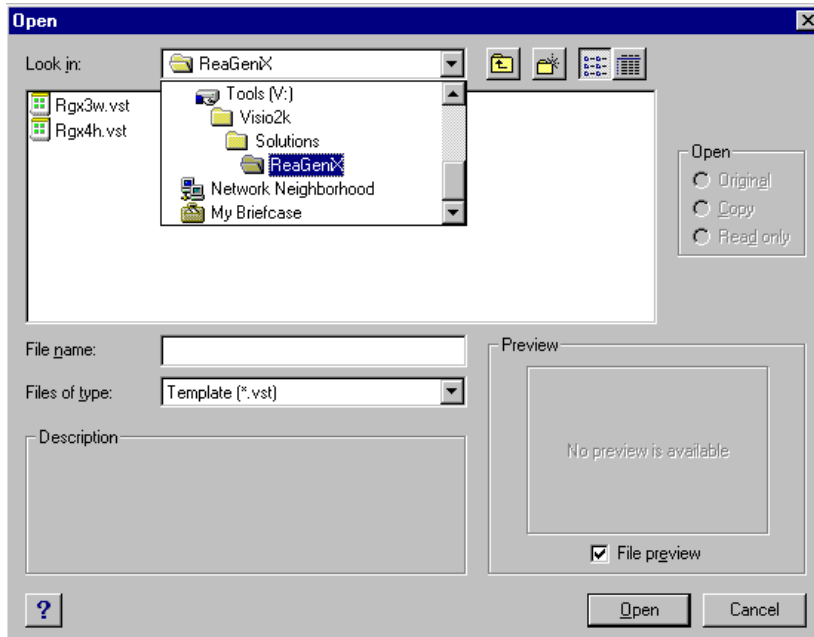


Figure 3. Select template Rgx3w.vst.

**NOTE:** Depending on the setup options, templates may be found deeper in the File menu hierarchy.

On the template there are an *actor body*, an *actor specification*, a *title*, and an *info* symbol (Figure 4). The *actor specification* and *actor body* symbols are necessary to the ReaGeniX diagrams. Generator can't understand diagrams without those symbols. So, if you want to create diagrams to a clean drawing page you have to put those necessary symbols in it by drag and drop method from the ReaGeniX stencil seen on the left side of the window.

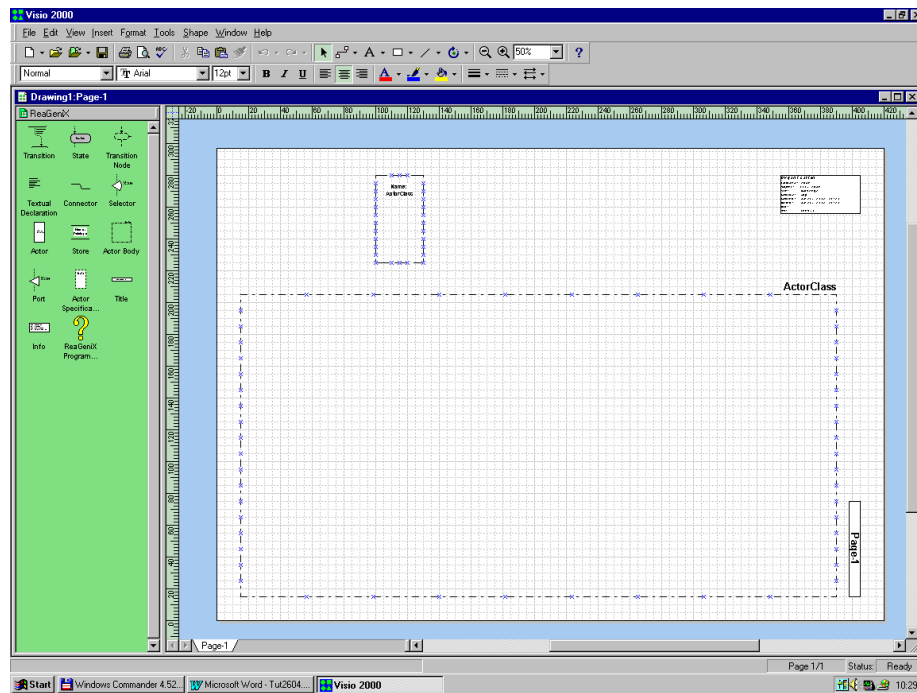


Figure 4: Visio screen with the ReaGeniX Rgx3w template and stencil

In the ReaGeniX stencil there are fourteen shapes (Figure 5) to be used for design diagrams

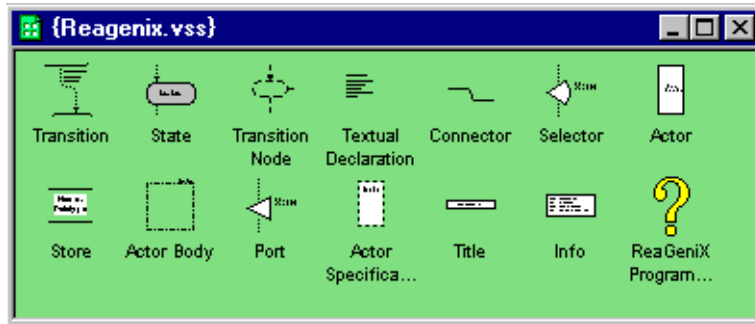


Figure 5: The shapes in the ReaGeniX stencil

You can get more information about each shape from Shape Help. The Shape Help opens by right-clicking the shape and selecting the "Shape Help" from the opened menu. It is also advised to familiarize yourself to *ReaGeniX programmer help*, which is opened the same way from the yellow question mark.

### 3.4 Selecting shapes from the stencil

The shapes are copied from the stencil to your diagrams using the "Drag and Drop" method. You point the shape by mouse arrow pointer and push the mouse left button down. Keep the button down and move the mouse arrow pointer on to the drawing page and loose the mouse button when the shape is on the wanted location. The Visio has excellent help and other documentation for drawing details.

## 4 Create a state-transition diagram. An Example Case.

In the later chapters we build an example application named "Stopwatch ". This example is done in Visio 2k. The Stopwatch calculates time from the system time. The simulator shows the passed time in hours, minutes, seconds, and seconds tenth and hundredth parts. The user can start or stop the time counting by pushing the "start/stop" button. When the time is running the simulator shows the time in tenth of a second accuracy. The user can also get an interval time by pushing the "interval/clear" button. The interval time stays in the display while the time is count in the background. When the user pushes the "interval/clear" button again in the Interval state the current time is shown again. Pushing "start/stop" button in Interval state causes stopwatch to show running time again. Counting is stopped by "start/stop" button and the final time stays in the display. "interval/clear" clears the count and "start/stop" resumes counting.

### 4.1 Specifying a Stopwatch

The Stopwatch application consists of one state-machine and an interface that shows the time running. The interface consists of a field that shows the passed time and three buttons named "start/stop", "interval/clear", and "exit". The exit button is to stop the application.

First we have to define the states the application needs. As it is noted before the time counting is standing or it is running. Time counting can also run, when the application shows an interval time. So we need three states. Let's call them *Time\_Standing*, *Time\_Running*, and *Interval*.

When the application is in the *Time\_Standing* state the timer is stopped and the time in time field is standing. When the application is in the *Time\_Running* state the timer is running and the time field shows the elapsed time in tenth-of-second accuracy. When the application is in the *Interval* state the timer is running but the time field shows the time when the interval button was pushed. Between the states we need *transitions* that handle the timer and the time outputs and show the *state* changes.

## 4.2 Creating the states

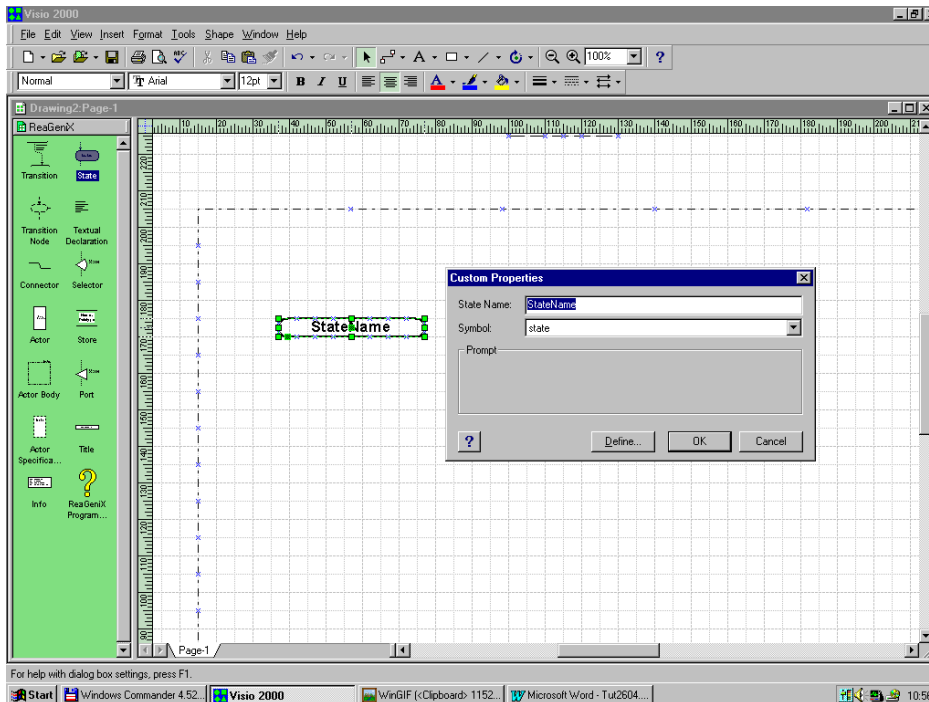


Figure 6: A new state and its "Custom Properties" window

To specify the *states* drag and drop the state symbols from the *stencil*. After dropping the *state* a "Custom Properties" window for the state name is opened (Figure 6). Use descriptive state names. Make the three *states* that you need in the diagram and name them as "*Time\_Standing*", "*Time\_Running*", and "*Interval*". (Figure 7)

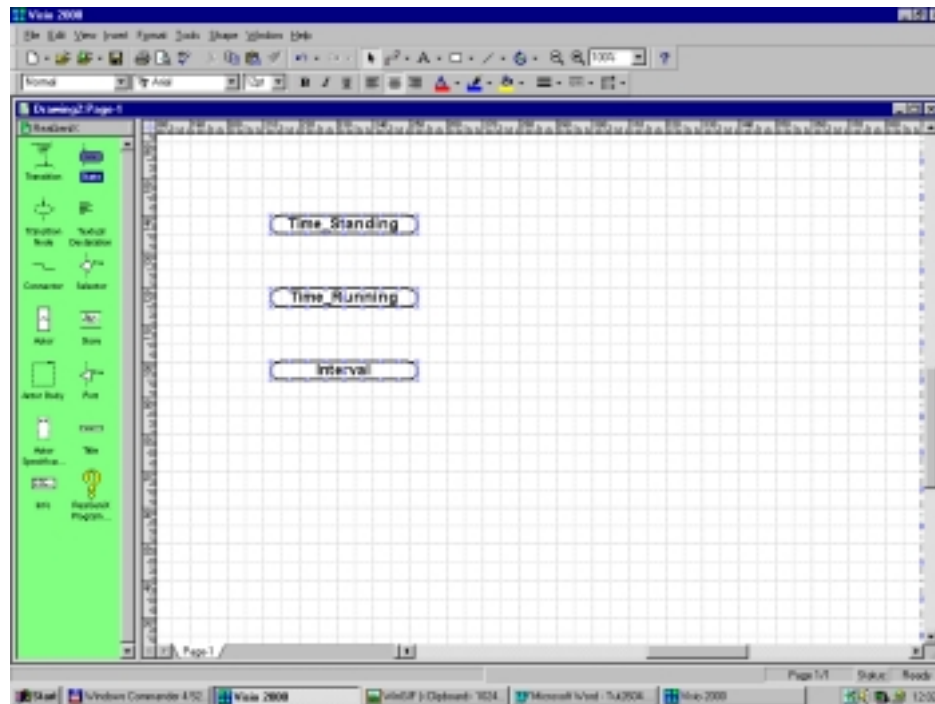


Figure 7: The three states of the Stopwatch

### 4.3 Creating the transition symbols

Drag and drop a *transition* to the drawing page. After dropping a *transition* symbol only lies on the drawing page. You must connect it to the *states*. Before connecting the color of the end points of a *transition* symbol is green (Figure 8).

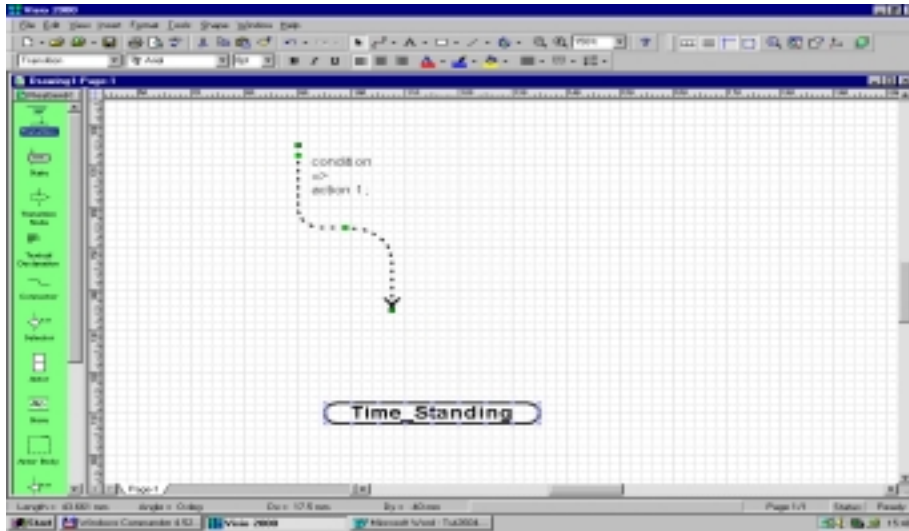


Figure 8: An initial transition before connecting

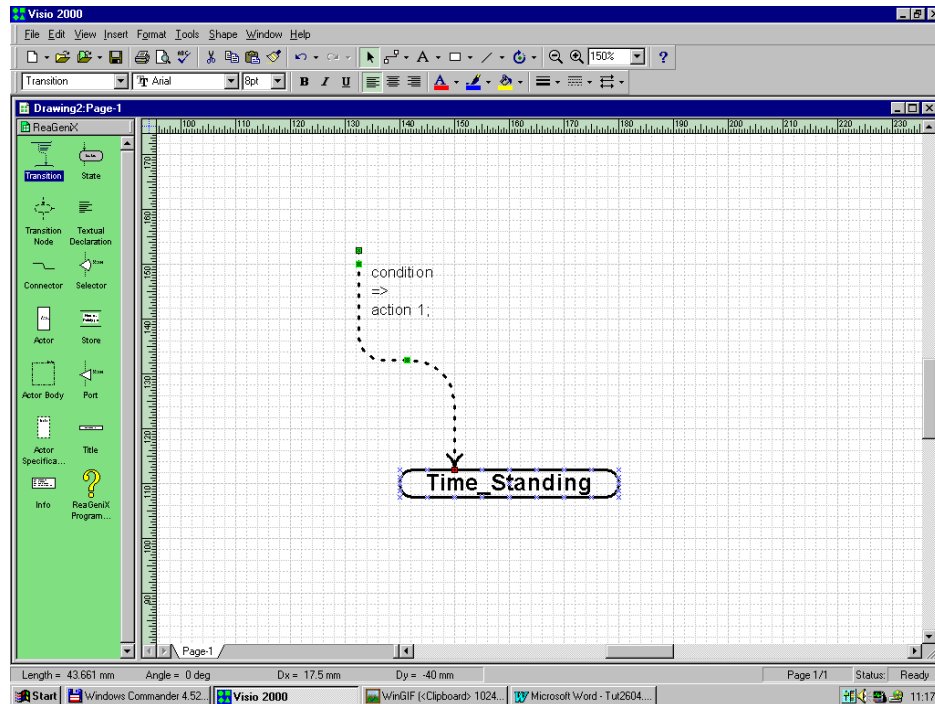


Figure 9: The initial transition glued to the Time\_Standing state

Drag the end point of the *transition* symbol to a connection point of the first *state*. The color of end point of the *transition* changes to red to indicate that it is glued to the connection point (Figure 9). The third green point near begin point of *transition* is for moving the text box. The fourth green point is for adjusting the shape of the *transition*.

The *condition* and the *action* are written to the text box of the *transition* symbol. Doing a double click on the *transition* symbol (Figure 10) activates the text field of a *transition*.

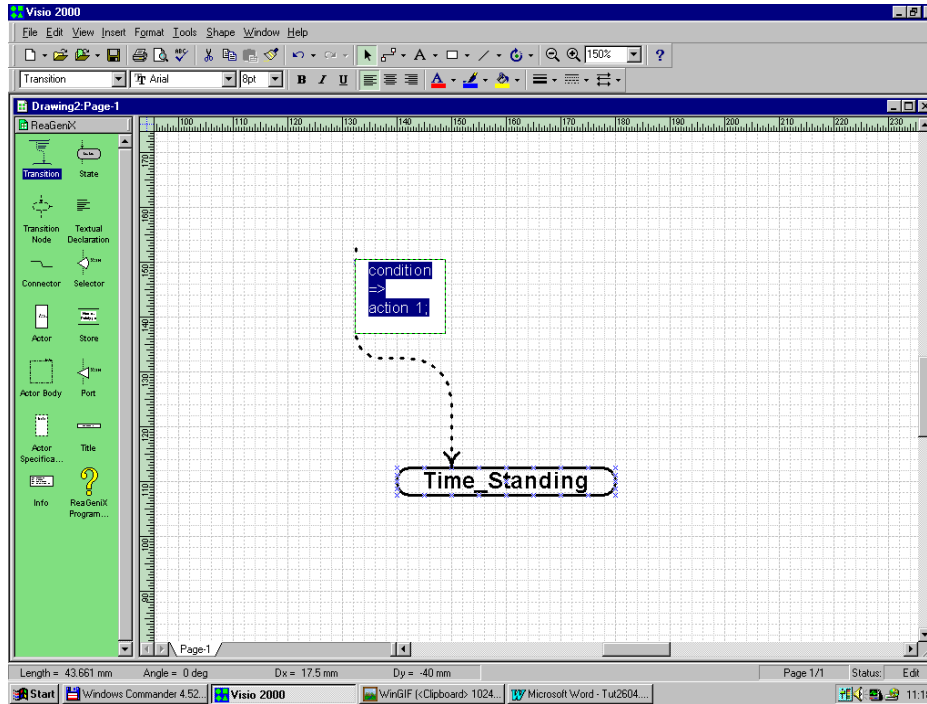


Figure 10: An activated text field of a transition

Replace the default text with the following (Figure 11):

#### Transition text

=>

```
v(timed) = 0;
```

```
emit(timed);
```

#### Explanation

Initial transition does not need a condition.

Condition / action separator

Clears the value of "timed" port

Sends an event to possible receivers of "timed"

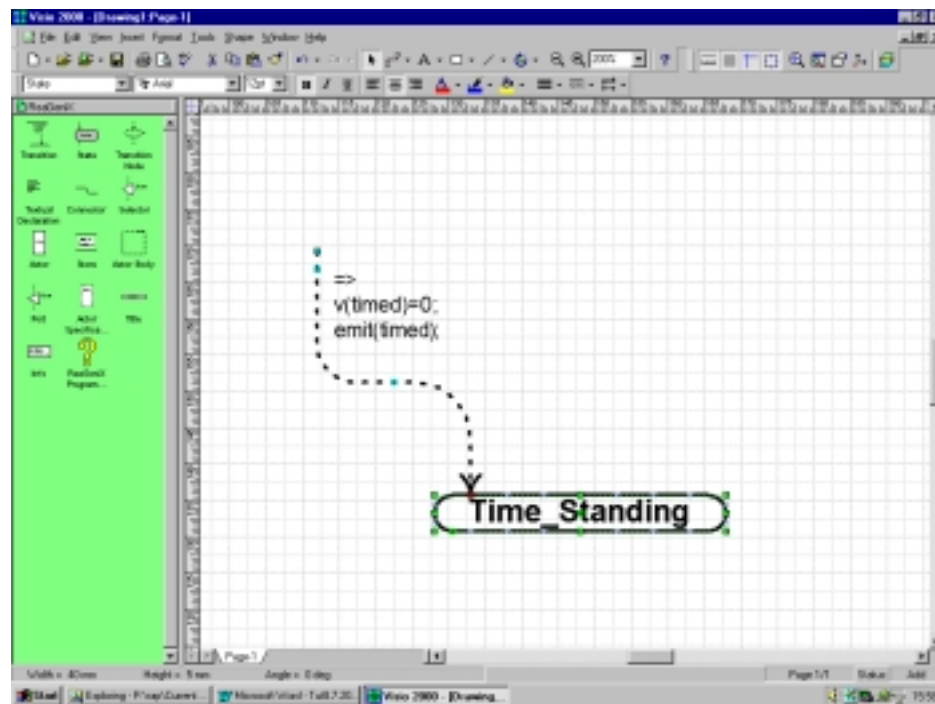


Figure 11: Actions written to the textbox of a transition

When you have typed the actions, point by the mouse pointer to other part of the diagram and do one left mouse click. It deactivates the text field of *transition*.

**NOTE:** that capital letters make a difference in transition texts.

Next you have to make a *transition* that changes the *state* of application, if the "start/stop" button is pushed in the user interface. Drag and drop a new *transition* to the diagram. Glue the begin point to a connection point of the *Time\_Standing state* and the end point to a connection point of the *Time\_Running state*. (Figure 12)

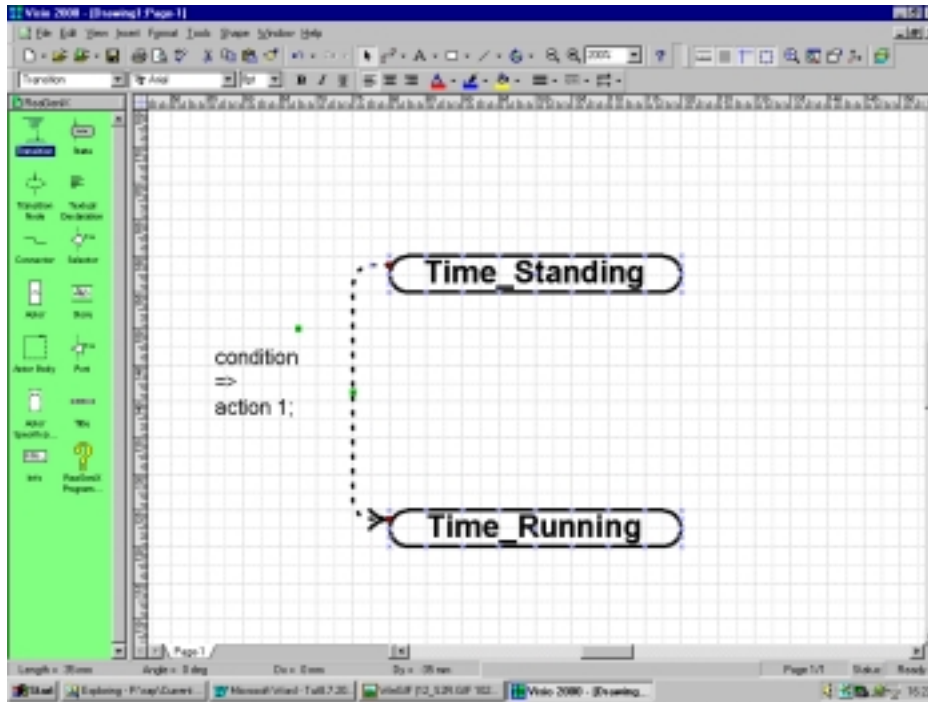


Figure 12: A glued transition from Time\_Standing to Time\_Running

Activate the transition from *Time\_Standing* to *Time\_Running* and replace the default text with the following (Figure 12):

<b>Transition text</b>	<b>Explanation</b>
<code>on(start_stop)</code>	Fires the transition, when the signal <code>start_stop</code> arrives
<code>=&gt;</code>	Condition / action separator
<code>ov(timer)=ov(store_timer);</code>	Needed for exact timing

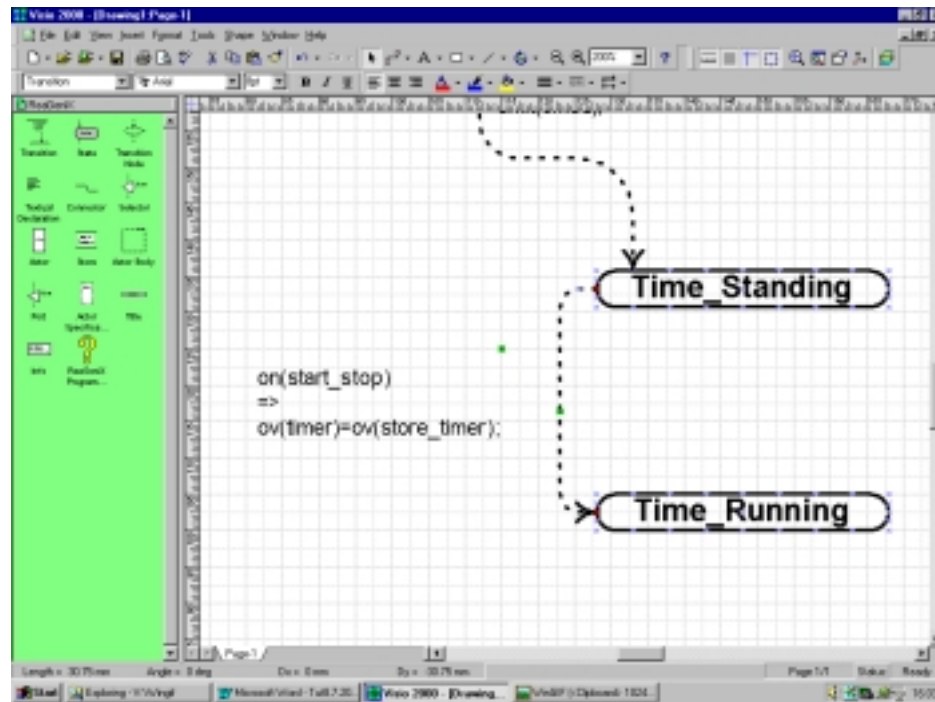


Figure 13: The transition to start the stopwatch

Next you need to make a *transition* that calculates the time in the *Time\_Running* state. Drag and drop a *transition* to the diagram and glue the begin point and the end point to the *Time\_Running* state. Activate the *transition* text field and replace the default text with the following (Figure 14).

<b>Transition text</b>	<b>Explanation</b>
<code>when(timeout(timer))</code>	Fires the transition, when the timer reaches zero
<code>=&gt;</code>	Condition / action separator
<code>ov(timing)=ov(timing)</code>	Increments time count
<code>+Second/10;</code>	
<code>v(timed)=ov(timing);</code>	Updates the port value
<code>emit(timed);</code>	Notifies the receivers of the new value
<code>ov(timer)=Second/10;</code>	Resets the timer to 0.1 s

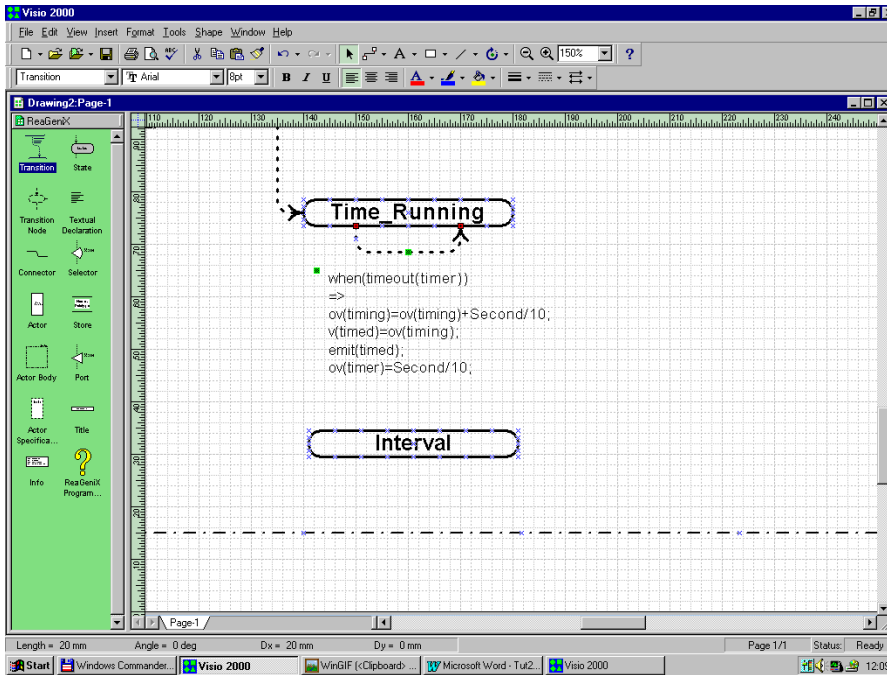


Figure 14: A transition to update tenths of a second in the display

Make next a *transition* that portrays the event when time is running and the user push the "start/stop" button. Glue its begin point to the *Time\_Running* state and the end point to the *Time\_Standing* state.

Activate the text field of the new transition and replace the default text with the following (Figure 14).

Transition text	Explanation
<code>on(start_stop)</code>	Fires the transition, when the start_stop signal arrives
<code>=&gt;</code>	Condition / action separator
<code>v(timed)=ov(timing)- (ov(timer)-Second/10);</code>	Transmits the passed time
<code>emit(timed);</code>	Notifies the receivers of the new value
<code>ov(store_timer)= ov(timer);</code>	Internal timer runs even when in Time_Standing state

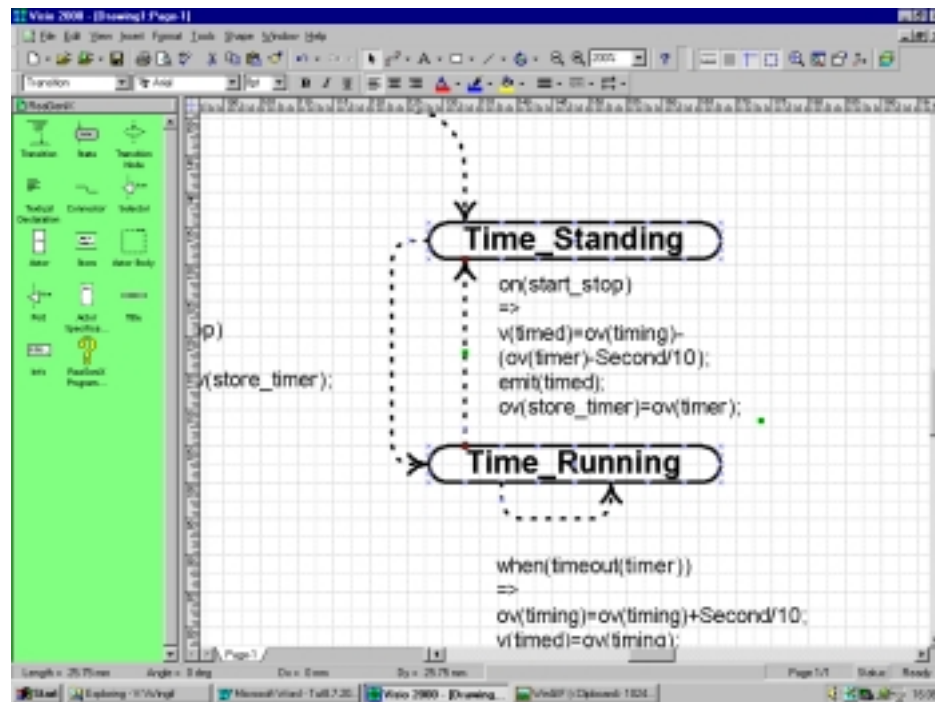


Figure 15: The transition to stop the stopwatch

Next you have to make *transition* that portrays the event where the user push the "interval/clear" button when the time is standing. Glue its begin point and the end point to the *Time\_Standing* state.

Activate the text field of the new transition and replace the default text with the following (Figure 15).

Transition text	Explanation
<code>on(interval_clear)</code>	Fires the transition, when the interval_clear signal arrives
<code>=&gt;</code>	Condition / action separator
<code>ov(timing) = 0;</code>	Clears internal time counter
<code>v(timed) = 0;</code>	Clears the output
<code>emit(timed);</code>	Notifies the receivers of the new value
<code>ov(timer)=Second/10;</code>	Resets the timer

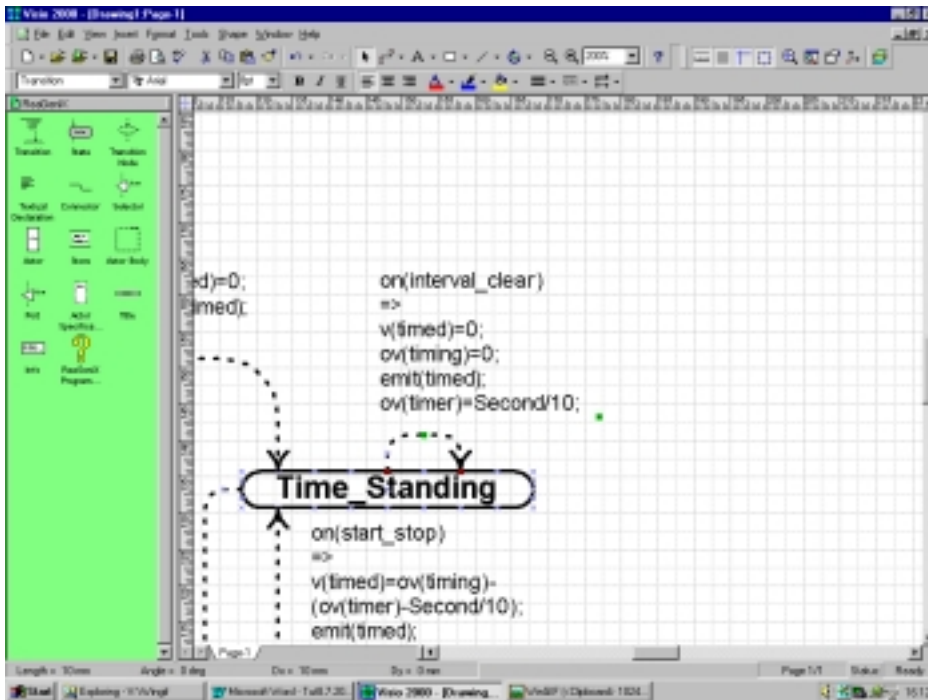


Figure 16: This transition clears the Stopwatch display and memory

The interval time is shown when application is in the *Time\_Running* state and "interval/clear" button has pushed. You need to make a *transition* for that from the *Time\_Running* state to the *Interval* state.

The transition text for the new transition is: (Figure 17).

Transition text	Explanation
<code>on(interval_clear)</code>	Fires the transition, when the interval_clear signal arrives
<code>=&gt;</code>	Condition / action separator
<code>v(timed)=ov(timing)- (ov(timer)-Second/10);</code>	Transmits the passed time
<code>emit(timed);</code>	Notifies the receivers of the new value

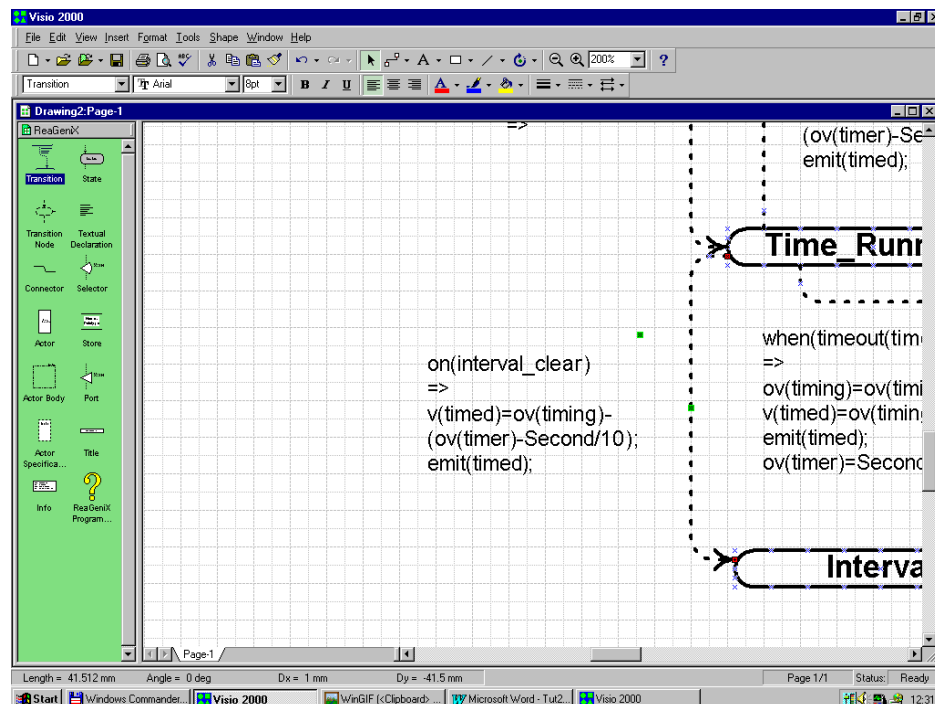


Figure 17: Freezing of the interval time to the display

When showing the interval time the time calculation must go on. That's why you need to make a *transition* that calculates the time but does not send it to the interface.

The transition text for the new transition is: (Figure 18).

<b>Transition text</b>	<b>Explanation</b>
<code>when(timeout(timer))</code>	Fires the transition, when the timer reaches zero
<code>=&gt;</code>	Condition / action separator
<code>ov(timing)=ov(timing)</code>	Increments the time count
<code>+Second/10;</code>	
<code>ov(timer) = Second/10;</code>	Sets timer to 0.1 s

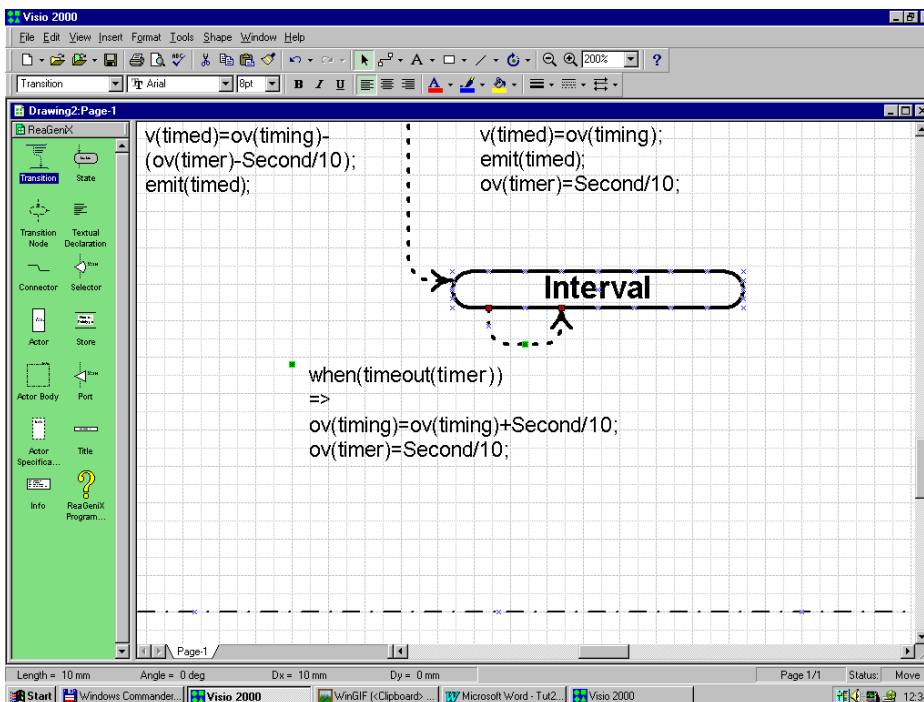


Figure 18: Hidden time counting in the Interval state

When showing the interval time the time, pushing "interval/clear" button fires a *transition* that updates the display to show the current time running.

The transition text for the new transition is: (Figure 19).

Transition text	Explanation
<code>on(interval_clear)</code>	Fires the transition, when interval_clear signal arrives
<code>=&gt;</code>	Condition / action separator
<code>v(timed)=ov(timing)- (ov(timer)-Second/10);</code>	Transmits the passed time
<code>emit(timed);</code>	Notifies the receivers of the new value

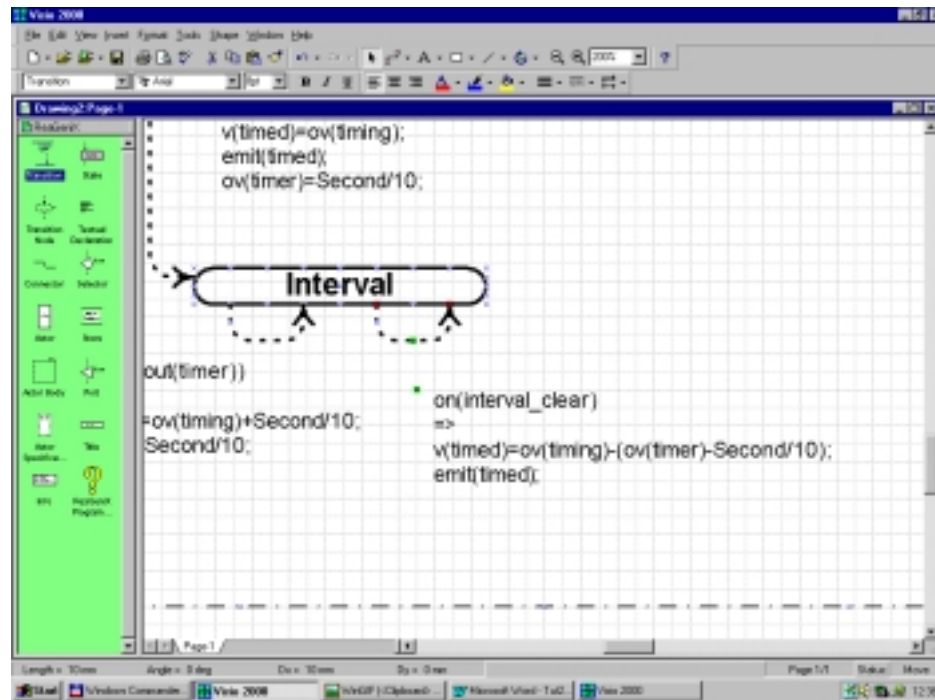


Figure 19: Update the display to show current time

The application needs a way to go back to the *Time\_Running state*. That's why you need to make a new *transition* that does the change from the *Interval state* to the *Time\_Running state*. Glue the begin point of the new *transition* to the *Interval state* and the end point to the *Time\_Running state*.

The transition text for the new transition is: (Figure 20).

<code>on(start_stop)</code>	Fires the transition, when the start_stop signal arrives
<code>=&gt;</code>	Condition / action separator
	No actions are needed

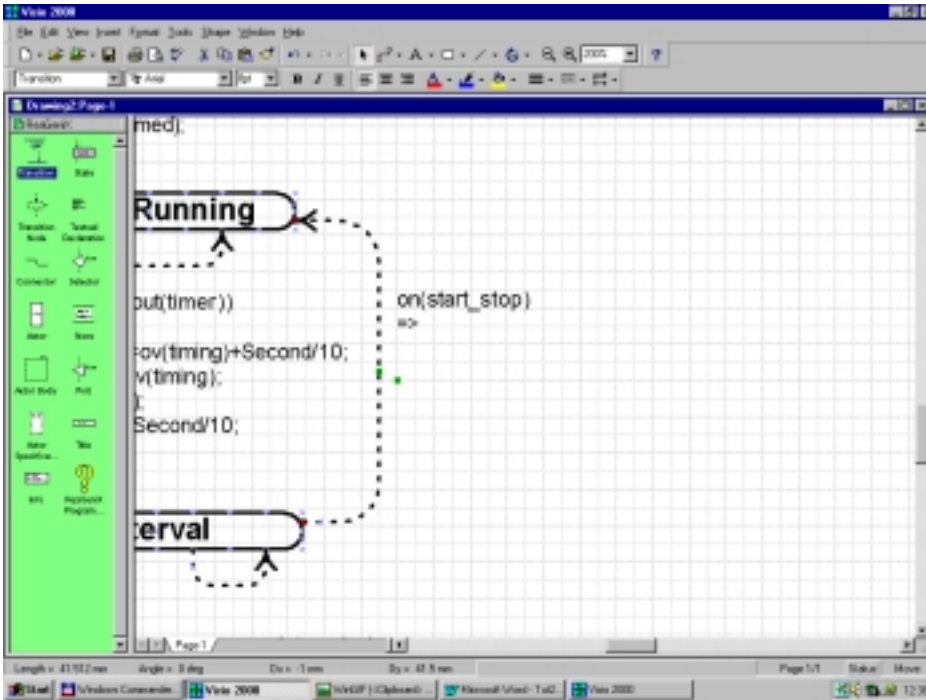


Figure 20: Return from interval display to running time count

#### 4.4 Create a timer

Next we need a *timer*. Drag and drop a *textual declaration* to the diagram. Dropping the shape opens a *Custom Properties* window. Select “*timer*” from “Decl.Class” menu (Figure 19) in the window. In this example we have a *timer* called “*timer*”. Type the name “*timer*” into the Declaration field (Figure 20). Figure 21 shows the ready timer.

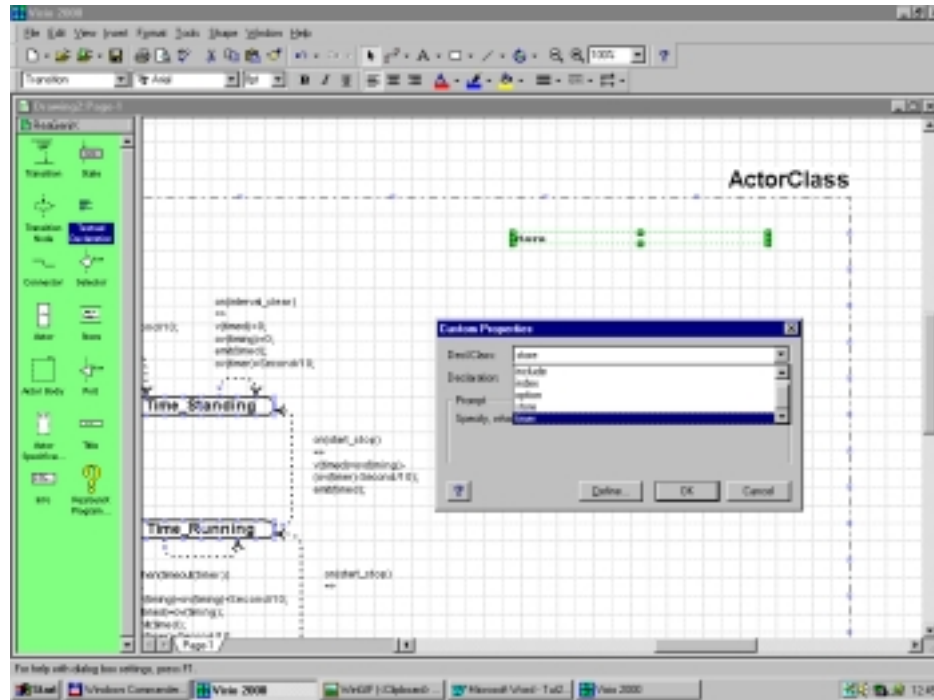


Figure 21: Timer selected from “Decl.Class” menu

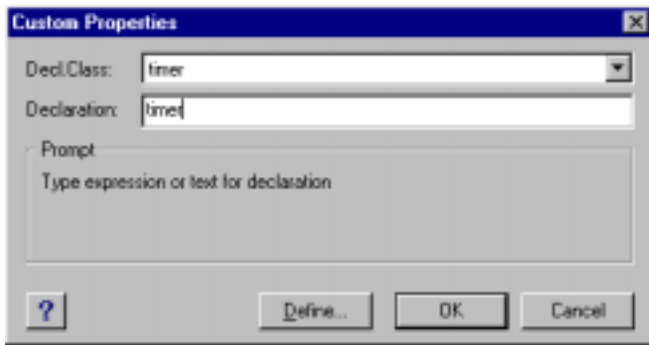


Figure 22: Name "timer" into the "Declaration" field

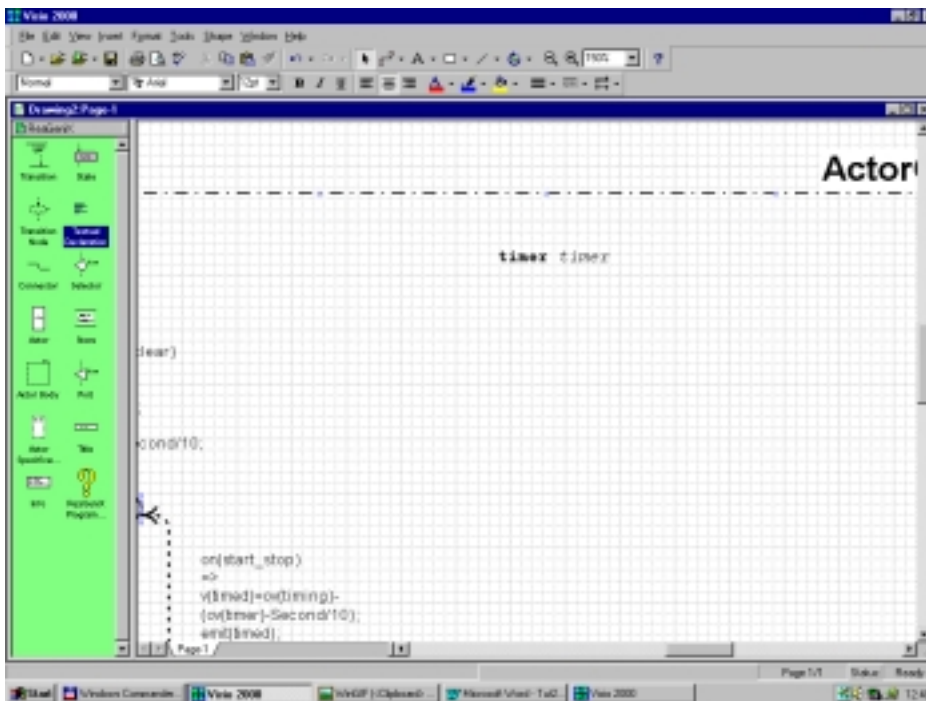


Figure 23: A timer definition in the diagram

## 4.5 Create local extended state variables

In the transition texts you used variable *timing*. You need so-called extended state variable to retain a value from an event to another. Extended state variable is defined using a *textual declaration* shape. Drag and drop a *textual declaration* into the diagram.

After dropping the Custom Properties window opens. The default value of the declaration class is “*store*”, which means an extended state variable. Type into the declaration field the text “*timing: int:=0*”, this text means that the data type of variable *timing* is integer and its initial value is zero (Figure 22).

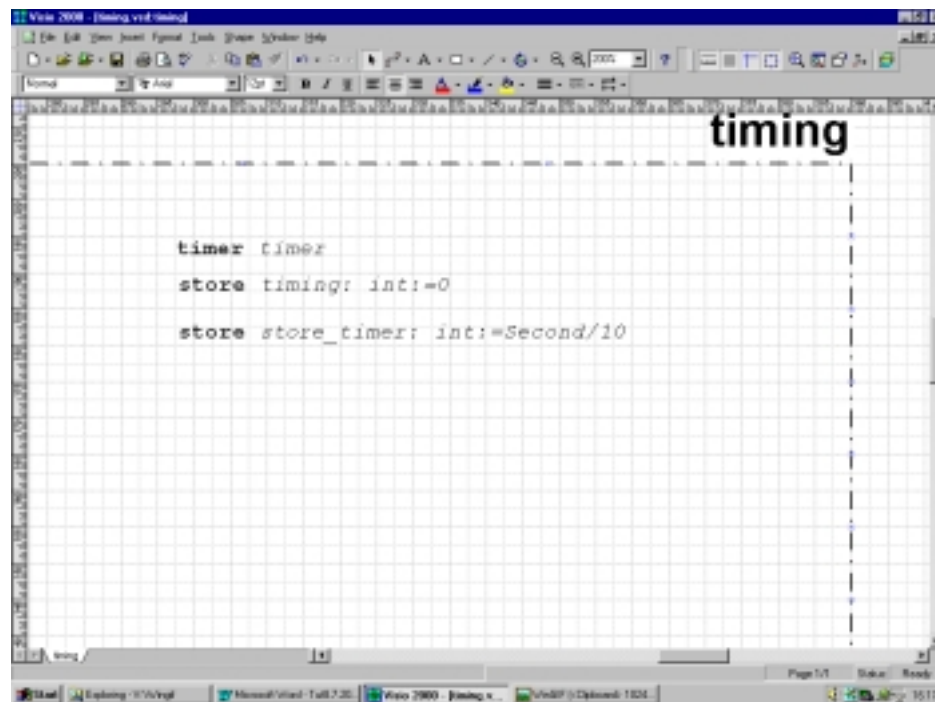


Figure 24: Defined local extended state variable

## 4.6 Create the port symbols

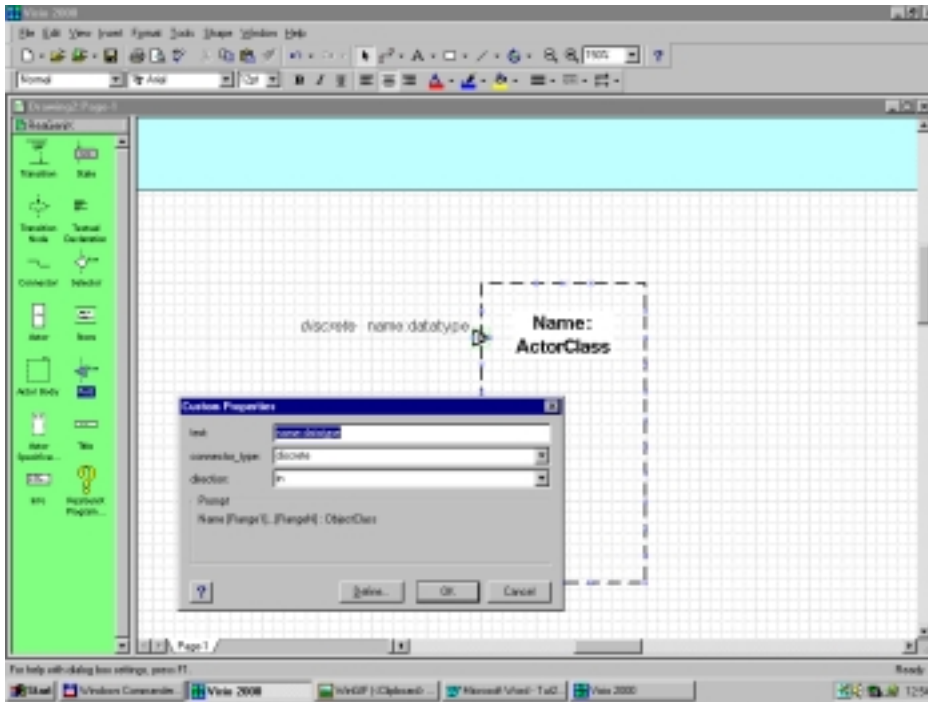


Figure 25: A port symbol glued to the actor specification symbol

Glue a new port symbol to an actor specification symbol (Figure 25). Dropping a port symbol opens Custom Properties window. Type the port name “start\_stop” to the “text” field. From connector\_type menu you select the type “signal”. Select “in” for the direction.

Make two more port symbols.

- 1     text:                   interval\_clear  
       connector\_type:       signal  
       direction:            in
- 2     text:                   timed: int  
       connector\_type:       continuous  
       direction:            out

The text “timed: int” means that the name of the port is “timed” and the data type passed by the port is “integer”. (Figure 26)

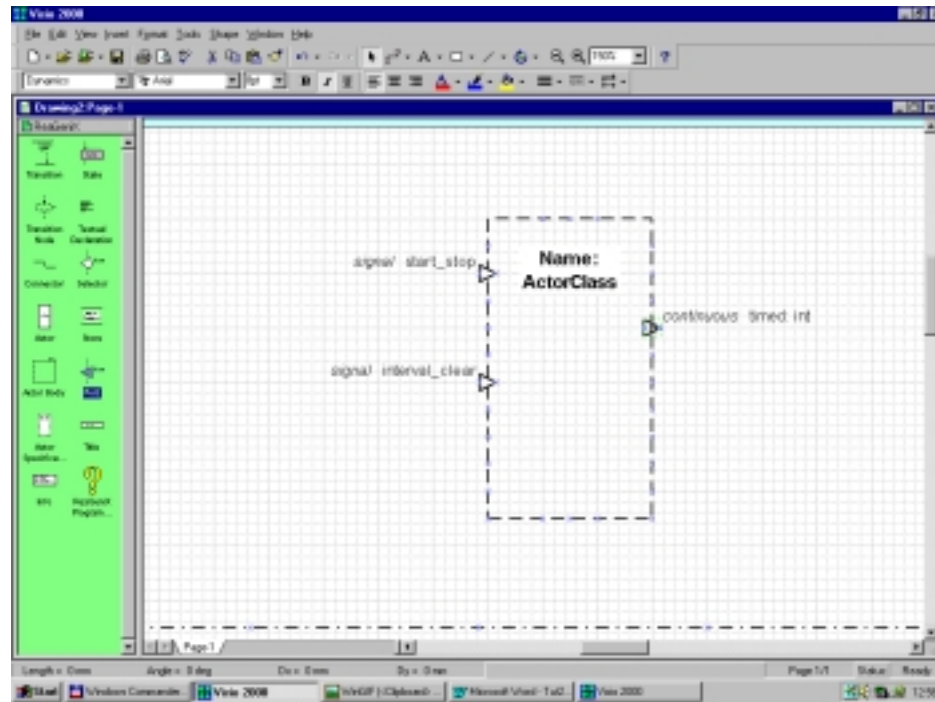


Figure 26: Ports attached to the actor specification symbol

## 4.7 Changing the names of Actor Specification and Actor Body

An *actor specification* specifies the interface of a component type; an *actor body* specifies its internal structure or behavior. The name of the *actor specification* and the *actor body* defines the name of the component *type*.

To define the name point to an edge or the name of the *actor specification* symbol by mouse pointer, so that the pointer arrows color changes to white. Do then a mouse right click. So the Custom Properties window opens. Type the name "timing" to the Text field and push OK button (Figure 27). Figure 28 shows the renamed actor specification.

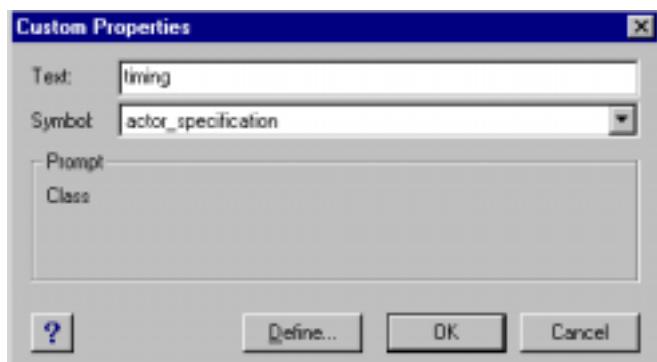


Figure 27: Typing the name of the component type to the "Text" field

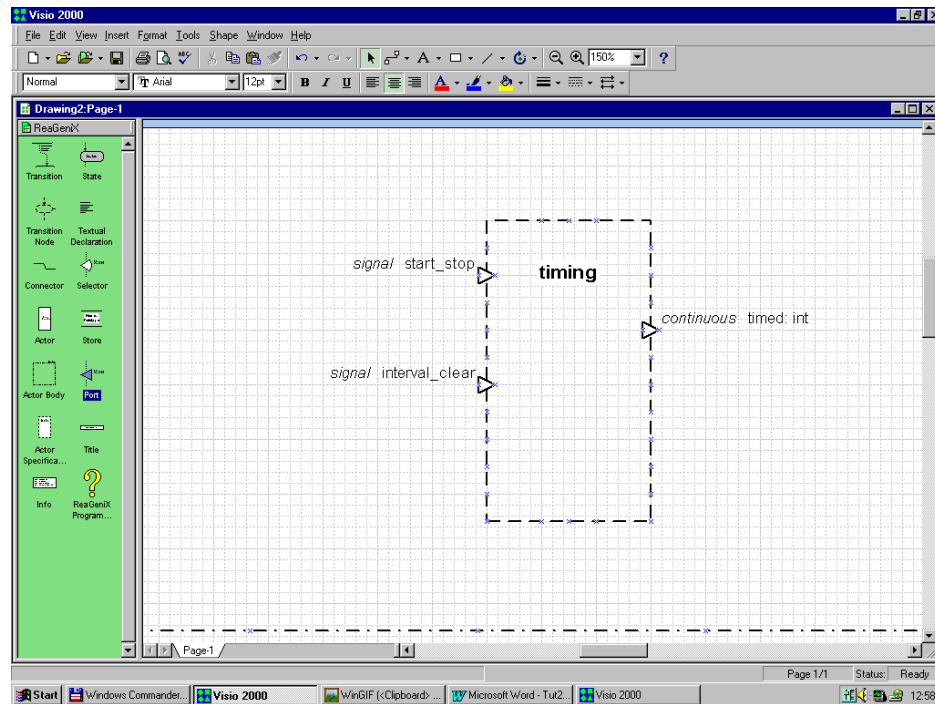


Figure 28: Actor specification "timing"

Do the same to the actor body symbol. Open the Custom Properties window and type into the Text field a text "timing" and push OK button.

## 5 Saving the diagram

### 5.1 Name the Drawing Page

The name of drawing page defines the names of the generated .c- and .h-files. When you are going to save a diagram you have to name it first. Select a "Page Setup" from "File" menu. This command opens a Page Setup window. Select "Page Properties" from the Page Setup window and type a text "timing" into the Name text field. When the name is ready accept it by pushing the OK button. Don't do other changes (Figure 29). This defines the generated files to be "**timing.c**" and "**timing.h**".

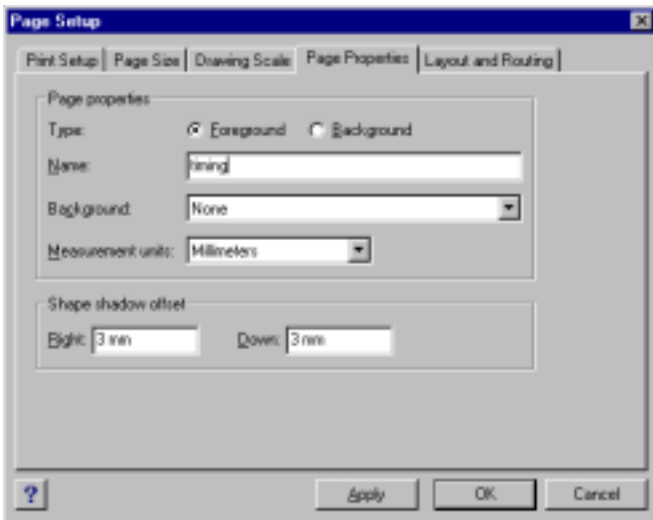


Figure 29: Page name "timing" in "Page Setup" window

## 5.2 Save a Drawing file

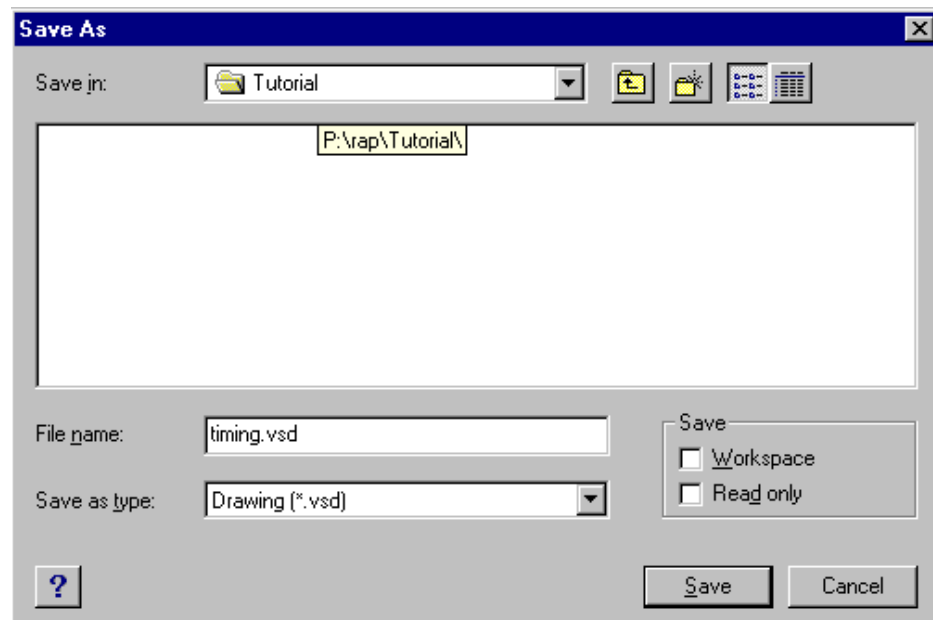


Figure 30: "Save As" dialog

To save a drawing file you have to select "Save As" from "File" menu. After selecting, there opens a "Save As" window. Into the "Save As" window you have to type the name of file into the "File name" text field. Make sure you have selected "Drawing (\*.vsd)" as type. Type there a text "timing". If you haven't created an empty folder for this exercise, do it now. Select the folder where you want to save the diagram from the Save in menu. After typing the name and selecting the folder accept them by clicking the "Save" button. (Figure 30)



## 6.2 Errors in the Diagram

You can test the behavior of generator by putting errors in the diagram. For example move one *port symbol* off on the *actor specification*, change other *port symbol* to a *selector symbol*, and add an *actor symbol* to the diagram. Save the diagram and run the generator. The ReaGeniX Visual Programmer message log window opens after generation is done. In the window the generator lists errors and warnings that it finds. ReaGeniX Visual Programmer message log shows the position of the error by selecting the problematic shape in the diagram, if you click on the message. (Figure 32)

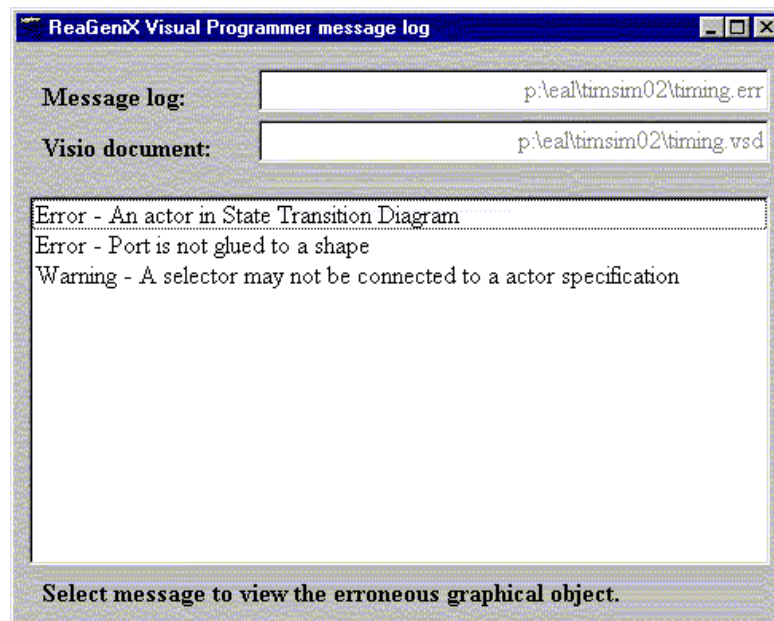


Figure 32: ReaGeniX Visual Programmer message log

## 7 Putting it to Life

### 7.1 Running the Visual C++ project

1. Run **Tutorial.exe** located in "Start menu/Programs/ReaGeniX/Tutorial/Visual C++ Project" (or download from <http://www.reagenix.com/Tutorial.exe> which contains Visual C++ 6.0 project and interface technology required to run timing tutorial. Extract files to your tutorial project folder you made at the beginning of the tutorial.
2. Make sure that **timing.c** and **timing.h** files are in the same folder with the project file **timing.dsp**.
3. Run **timing.dsp** by "double clicking it". Execute the project by pressing "CTRL+F5" when Visual c++ has started.

If no misspellings are made stopwatch runs as it was specified in the chapter 4.1.

## 7.2 Run the Stopwatch

Select Stopwatch from the Start menu of the Timing Simulator window. The application Stopwatch starts. Figure 33 shows the Stopwatch user interface. You can start several stopwatch instances by selecting repeatedly Start / Stopwatch.

Pressing the START/STOP button sends the `start_stop` signal to the generated component. Pressing the INTERVAL/CLEAR button sends the `interval_clear` signal to the component. The numeric window shows the value of the `timed` output in hours, minutes, seconds, and hundredths of a second. EXIT button closes the stopwatch instance.

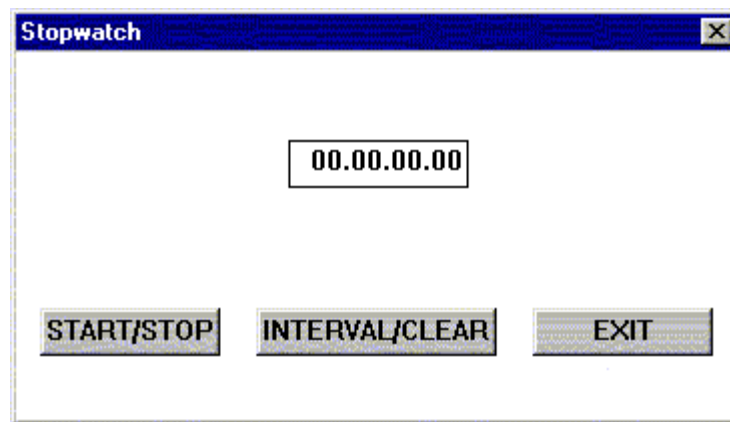


Figure 33: The Stopwatch user interface.

!!! If any problem occur, feel free to contact us by e-mail.

obp@obp.fi