StateWORKS Studio

Development Tools

User's Guide & Training Manual

Version 7.0

Contents

Figures and Tables

INTRODUCTION

The StateWORKS Studio [6] is a set of programs used to develop software for complex systems, such as control systems. That part of the software which controls the behaviour of the software is isolated from the numerical computation, and is treated as a set of state machines. In some cases, you could specify and test a control system without writing a single line of code.

This document is completed by tutorials [11][12][13] which teach you the details of using menu commands while working with StateWORKS Studio.

System specification

You specify the system using the StateWORKS Studio (SWStudio) which is a I.D.E. with two powerful editors: a State Machine Editor and a Project Manager. In addition, it contains SwLab and monitors (SWMon, SWQuickPro, SWQuick, SWTerm) to test the specified system. Note that no version contains all monitors. In the time of last updating the manual SWMon and SWQuickPro become standard monitors delivered in the installation package.

Project Manager

You organize your project with the Project Manager. The project contains files that describe state machines used by your system, IO-units, VFSM-templates and system configuration. All files are managed by the Project Manager that keeps track of all changes and assures the system consistency.

State Machine Editor

A control system is built of several state machines and I/O-units. The State Machine Editor is a graphical and text editor to specify the state machine behaviour. In addition, the State Machine Editor is used to specify VFSM-templates and I/O-units. The results of the specification can be built producing data files that are used by StateWORKS run-time system to perform the control task. The specification can be printed or exported to a text editor for documentation purposes. The specification is also available as XML files to be displayed in web browsers.

System Configuration

A system of state machines specified by the State Machine Studio must be adapted to the real control environment, i.e. all objects used by state machines must be determined, for instance:

- state machine inputs are linked with appropriate input signals,
- timer values are determined,
- switch-points for numerical (e.g. analog) inputs are determined, etc.
- The system is configured in the Project Manager.

System test

You test the system using the StateWORKS Lab and the StateWORKS monitors.

StateWORKS Lab

The StateWORKS Lab (SWLab) is a Window program that runs under Windows NT/2000/XP. SWLab is based on a real time data base (RTDB) with a state machine executor. The RTDB contains all object used in the control system. The SWLab simulates a small digital and analog input/output system.

SWLab is used to learn the VFSM concept. It can be used to test state machines and not too complex systems of state machines. It contains a fully functional VFSM execution environment: its limits result from the number and type of inputs and outputs. In a TCP/IP link the StateWORKS Lab is a Server.

StateWORKS Monitors

The StateWORKS Studio has different monitors to display and change RTDB objects, expecially: SWMon, SWQuickPro. In older versions you find also SWQuick and SWTerm.

SWMon

The StateWORKS Monitor (SWMon) is a debugging tool that cooperates with a VFSM execution environment based on TCP/IP link. Hence, it cooperates well with the StateWORKS Lab and is used to learn the VFSM concept. It gives you the true understanding of objects used in the control system. It can be used to test a system of state machines of any size.

SWQuickPro

The StateWORKS QuickPro (SWQuickPro) is a monitor which allows an access to a single object at a time. SWQuickPro creates automatically a log file that can be used as a command file. It has a quite elaborated test features that are described in [9]. Its simpler variant SWQuick does not have the testing facilities.

StateWORKS Terminal

The StateWORKS Terminal (**SWTerm**) is a debugging tool that cooperates with a VFSM execution environment using the TCO/IP link. It is a command line program. It allows the user to log all his activity (commands and system answers) and to use the log file as a command file. Hence, it is used to automate the tests.

Help Files and Other Documents.

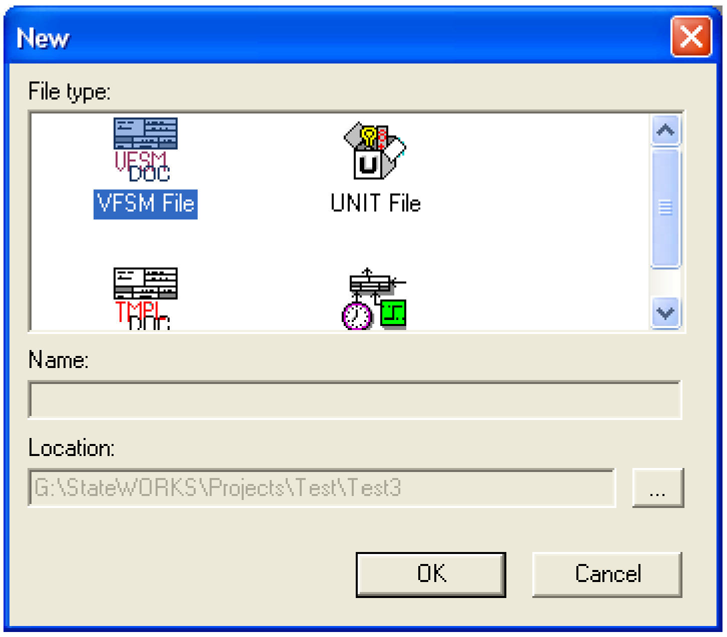
While reading this manual, or when working with StateWORKS Studio, please remember to use the StateWORKS Studio Help files, which provide a great deal of extra information and advice on various topics. Many other informative documents may be seen at the StateWORKS web site, and some are listed in the references.

GETTING STARTED

What is the StateWORKS Studio for?

The StateWORKS Studio is used to specify virtual finite state machines (VFSM). The VFSM [1][2] is a state machine that describes a pure behaviour of the control system. You achieve this by using names that describe states, input conditions and Actions. All names have a virtual character that is they are just names. You link these names to real signals by creating I/O objects that are later used to build a system configuration by the StateWORKS Project Manager.

Opening VFSM document


New

File type:

VFSM File

UNIT File

Name:

Location: G:\StateWORKS\Projects\Test\Test3

Figure 1 Choosing a file

If you want to specify a single VFSM you start the creation of a new VFSM Specification by

File / New

In the dialog window *New* you choose the **VFSM file** and click on the **OK**. The dialog window *Base template for this VFSM* opens.

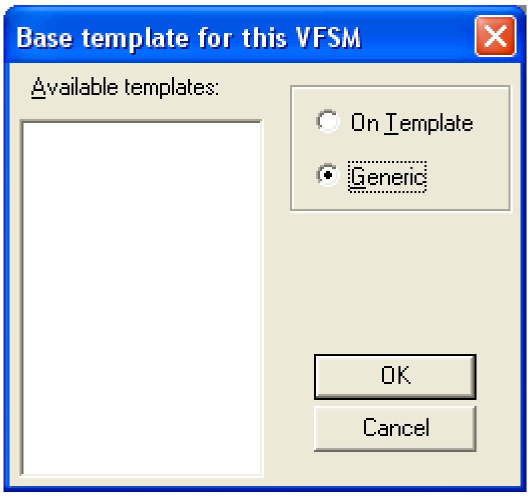


Figure 2 Creating a new VFSM

You choose **Generic** and click on the **OK.**

You are presented with a state transition diagram (ST diagram) of a VFSM with a default state *Init*. While saving or building you can change the VFSM name. You will be able to specify a VFSM state transition table using active menus and buttons. Before you start to elaborate your VFSM specification you will have to define several names that are used for the specification.

In addition, the initial state transition diagram displays an *Always* table. We will discuss it later.

Both elements: the *Init* state and the *Always* table cannot be deleted; each state machine contains them. You may rename the state *Init*.

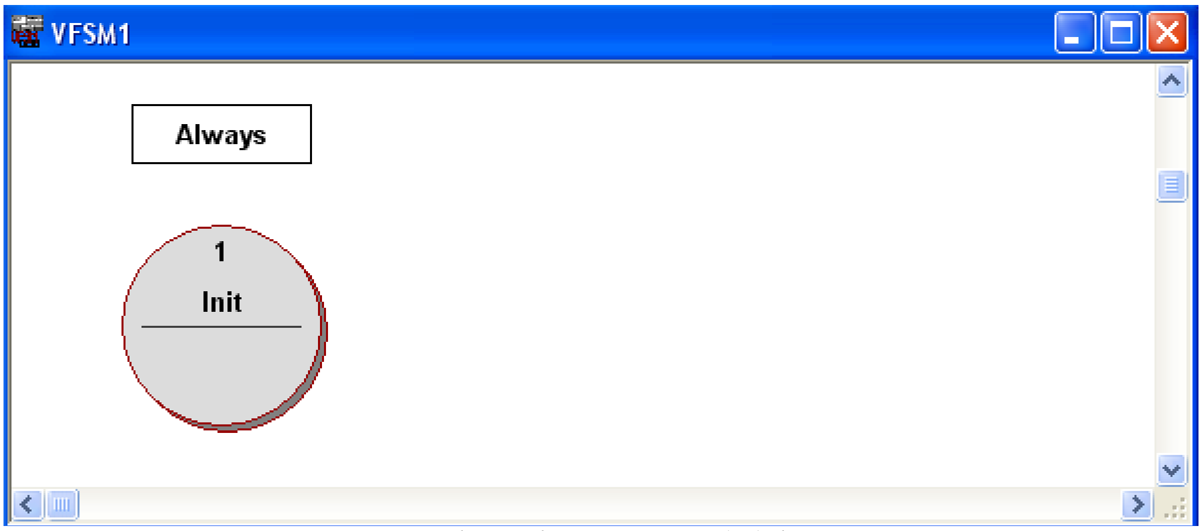


Figure 3 The inital state transition (ST) diagram

Example OnOff

The descriptions are supported by examples. The first example is a state machine OnOff:

It is a simple state machine that can be a basis of many complex state machines. It includes these basic I/O objects: Input Command (CMD-IN), Timer (TI), Alarm (AL), Digital Input (DI) and Digital Output (DO).

The behaviour of the state machine can be described in the following way: The state machine has two stable "done" states: *Off* and *On*. Normally, nothing is observed in a done state. The state machine enters a done state if the inputs signal that the controlled device has reached the required state. On entering the done state no Actions are carried out. Hence, the state machine does not wait for a reaction of the controlled device to a defined stimulus. In other words, a done state means that everything is OK.

In most cases, each done state is coupled with a busy state. In the example, the state machine has two busy states: *OffBusy* and *OnBusy*. On entering a busy state the state machine performs some Actions and expects that the controlled device reacts to them issuing some acknowledgements.

Usually, a busy state is guarded by a timer. If the acknowledgement does not come during a certain time the state machine must react to this situation. The simplest reaction is to generate an alarm.

The above principles are represented in the example. The state machine starts in a state *Off*. Receiving the command *CmdOn* it changes to a state *OnBusy*. Entering the state *OnBusy* the state machine switches on the device and starts a timer. If the device acknowledges its ON state by setting the input Di high the state machine goes to a state *On*. If the timer expires before the acknowledgment comes the state machine generates an alarm. On leaving the state *OnBusy* the state machine stops the timer as a timeout does not make sense in any state but *OnBusy*.

The state machine stays in the state *On*. Receiving the command *CmdOff* it changes to a state *OffBusy*. Entering the state *OffBusy* it switches off the device. If the device acknowledges its OFF state by setting the input Di LOW the state machine goes to the state Off. The state *OffBusy* is not guarded by the timer. Thus, if the Di does not change to LOW the state machine stays in this state and does not signal it, in this example.

Two additional transitions are possible: from *OnBusy* to *OffBusy* with the command *CmdOff* and from *OffBusy* to *OnBusy* with the command *CmdOn*.

Defining States

You are free to invent state names. Try to use names that "describe" the meaning of the states. „*MotorOn*" name carries more information than „*State_1*".

The description of the required control incorporates suggestion for state names. We accept them as the verbal description usually contains quite convincing and meaningful names.

As we mentioned already any state machine contains a state *Init*. The state *Init* is created automatically and cannot be removed from the State Name list but you can change the name of it. It is a convention that each state machine is in a state *Init* if the system is switched on. This convention allows you to specify the first transition, immediately, to a state that you consider as the real beginning state. As Entry Actions of the beginning state you can specify some Actions that you consider necessary for a safe start of the system.

The easiest way to define a new state is to double click on the ST diagram. It opens a state Name Dictionary by which you can add a state name to the list.

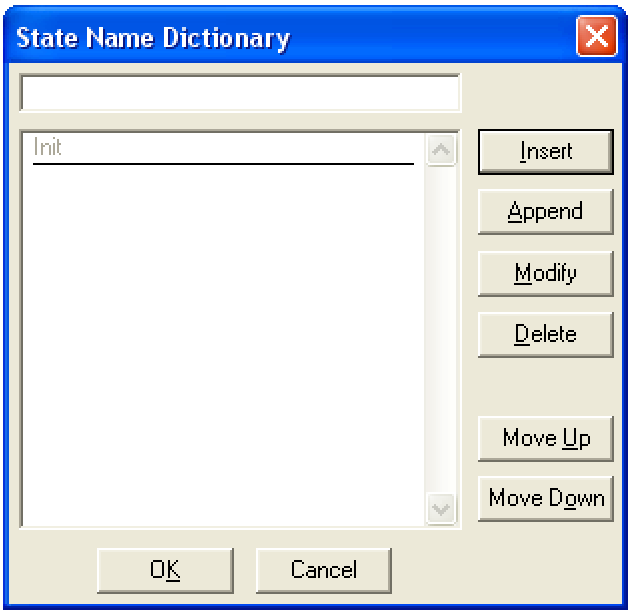


Figure 4 State Name Dictionary

You can manipulate the list adding, and removing states or changing the state sequence. The state symbol (a circle) on the ST diagram contains the state name and the state sequence number in the list. The list may be expanded and modified at any time. So, do not worry if you are not sure in the beginning what states you need. You may add several states in the state transition dictionary but when you close the dictionary only the currently selected state is created on the diagram.

Alternatively, you may open the ST diagram using menu command

Dictionary/State

and add state(s) to the list. Several new states can be "transferred" to the ST diagram one by one by double-clicking on the ST diagram, selecting the state in the list and closing the dictionary by OK.

You can change the sequence of states in the list at any time. The sequence has no influence on the VFSM specification or its behaviour. It is just used to express your preferences in the documentation.

When a state is created on the ST diagram a state transition table (ST table) for this state always opens (see Figure 5).

You can fill in the state transition table immediately, or at any time you find it appropriate. Usually, in the beginning you will ignore (just close) the state transition table as you have to define the virtual environment which you need for transition and Action specifications.

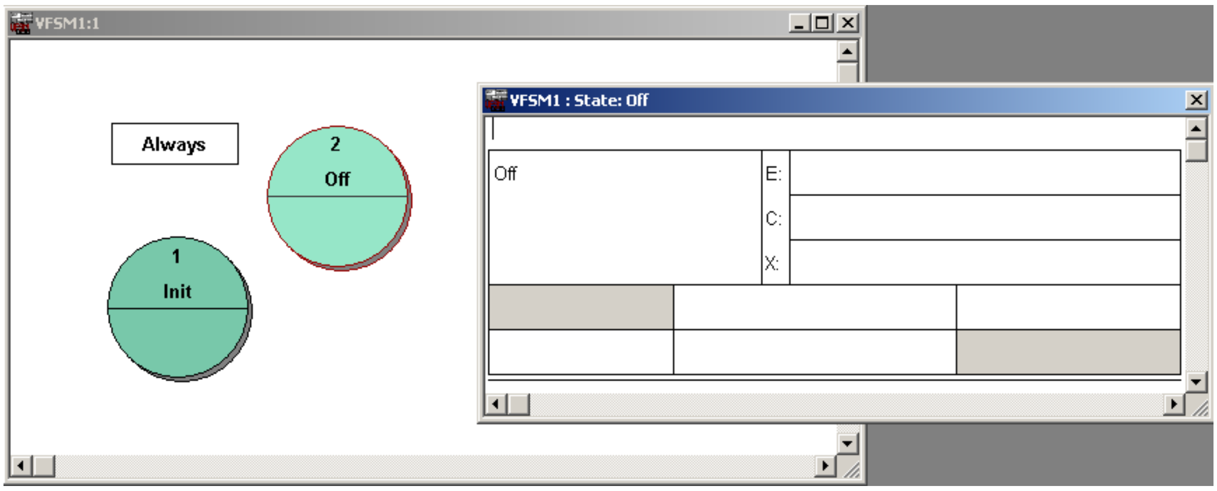


Figure 5 A state transition (ST) table opens when the state is created

Defining the Virtual Environment

The Virtual Environment [5] is an input/output space created by names invented by a designer to describing the behaviour of a system. The designer is free to choose the names that, from his point of view, are needed for the system specification.

Objects (variables) used in control systems store various values. An object that represents a digital input stores a Boolean value (true, false). An object that represents an analog input stores a number (for instance: float). A parameter stores a value that could be of any numerical type (integer, float, string, etc.). A timer is a counter, storing a number which changes when it runs.

In addition, objects have *Control Values*. A Control Value is a feature of an object that can be used for control purposes. The Control Value of a digital input corresponds to its value. The Control Value of an analog input is a set of value ranges (for instance: LOW, OK, HIGH). The Control Value of a parameter is the status (for instance: INIT, DEF, CHANGED). The Control Value of a timer is its state (for instance: RUN, STOP, OVER). In general Control Values must be defined separately for each object type.

The (virtual) input and output names are the only names that can be used to specify the control system. The Input Names are produced on Control Values of real input control signals. The Output Names are converted into values of real output control signals. The real/virtual conversion on inputs and virtual/real conversion on outputs are mostly done automatically by the real time data base (RTDB) in the VFSM Execution environment. In some cases, if required, the conversion process may be a program (e.g. a C function).

Before you start to define the Input and Output Names you have to decide what I/O objects will be used by the state machine.

Defining I/O Objects

The I/O Object Types identify sources (inputs) and targets (outputs) of control signals. The sources and targets of control signals are the real input and outputs in contrary to (virtual) Input and Output Names that are used to specify the behaviour of the finite state machine. The I/O Object Types can be grouped into following categories:

- other finite state machines: **VFSM**
- command: **CMD** (**CMD-IN**, **CMD-OUT**)
- alarms: **AL**
- internal counting devices:
  - timer **TI**
  - counter **CNT**
  - event counter **ECNT**
  - up-down counter **UDC**
- external digital signals: **DI**, **DO**
- external numerical (analog) signals: **NI**, **NO**,
- output demultiplexer **TAB**
- supervision:
  - switch-point: **SWIP**
  - string: **STR**
- data: **DAT**, **XDA**
- parameter: **PAR**
- use defined output function: **OFUN**
- programmer's I/O handler or output function interface **UNIT**

The I/O Objects are used to define virtual Input and Output Names. The virtual Input and Output Names plus State Names are the only names that can be used in the specification table. Some I/O Objects are pure inputs (e.g. digital input DI), some I/O Objects are pure outputs (e.g. digital output DO) and some I/O Objects are both, inputs and outputs (e.g. timer TI). CMD Objects are either input or output; therefore they appear as CMD-IN and CMD-OUT in the I/O Object list.

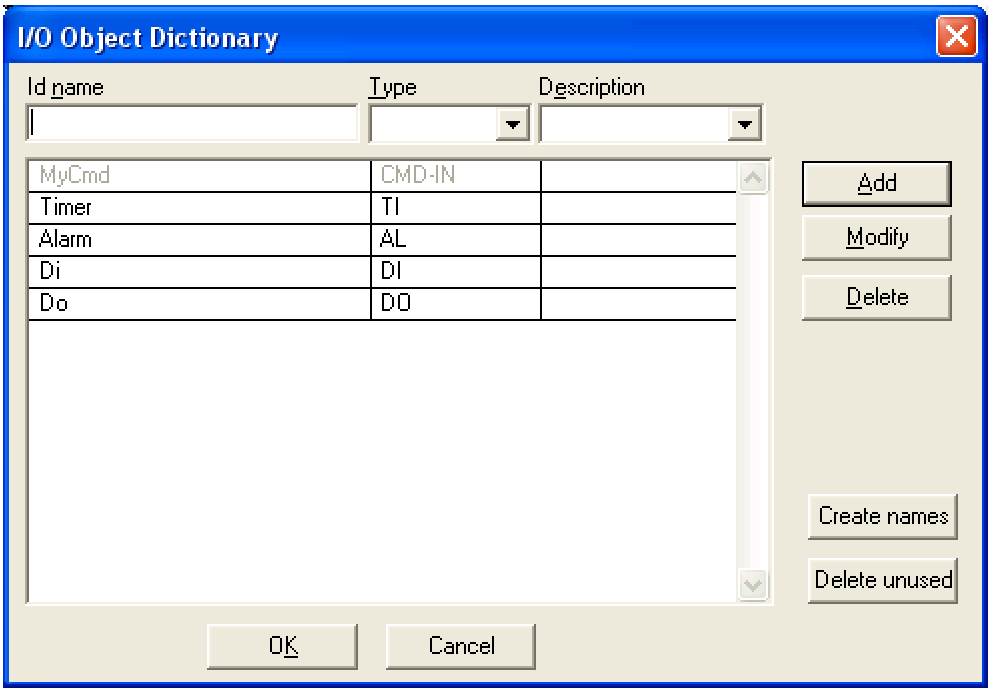
Open the I/O Object Dictionary(see Figure 6):

Dictionary / I/O Object ...

With the **Type** menu you choose the object type and give a name to the object in the column **Id name**. The column **Description** is not used for the objects considered in this example.

The MyCmd object is created automatically - this is an object that comprises commands that the state machine will "understand". Per default, each state machine has the MyCmd object. Even if you do not use it effectively the MyCmd object cannot be removed.

You have to choose and name four other objects: Di, Do, Timer and Alarm. You are free to invent names but we suggest you to use simple names that just describe the object type. The object names will be used by creating virtual names used by a VFSM specification. They will be used also by the project to create physical object names to configure the system. Naming is an important part of the documentation - the VFSM method supports it and gives you a chance to do both: specify the control and document it properly. We suggest you use simple names as in the following table (in the example we use: *MyCmd, Timer, Alarm, Di, Do*).



Id name	Type	Description
MyCmd	CMD-IN	
Timer	TI	
Alarm	AL	
Di	DI	
Do	DO	

Figure 6 I/O Object Dictionary

The buttons **Create names** and **Delete unused** can be used to create single objects for all object types. The button **Create names** composes names using object type and a sequence number beginning from 0: each click on the button creates a new set of objects. This feature is not very useful in designing a state machine (who needs all objects?) but are just provided for test purposes.

Note: The I/O objects you have now specified are real objects but are not yet configured. You can use them to make the VFSM specification. You will make them truly real, i.e. decide which Di or Timer is used, with the StateWORKS Project Manager when preparing a system configuration. While configuring them you will also specify their properties, for instance you will determine the timeout value for the Timer.

Defining Input Names

(Virtual) Input Names are names invented by the designer to specify the conditions for state transitions and Input Actions of a VFSM. Only Input Names may be used for these purposes in a VFSM Specification. A list of all Input Names is called a Virtual Input.

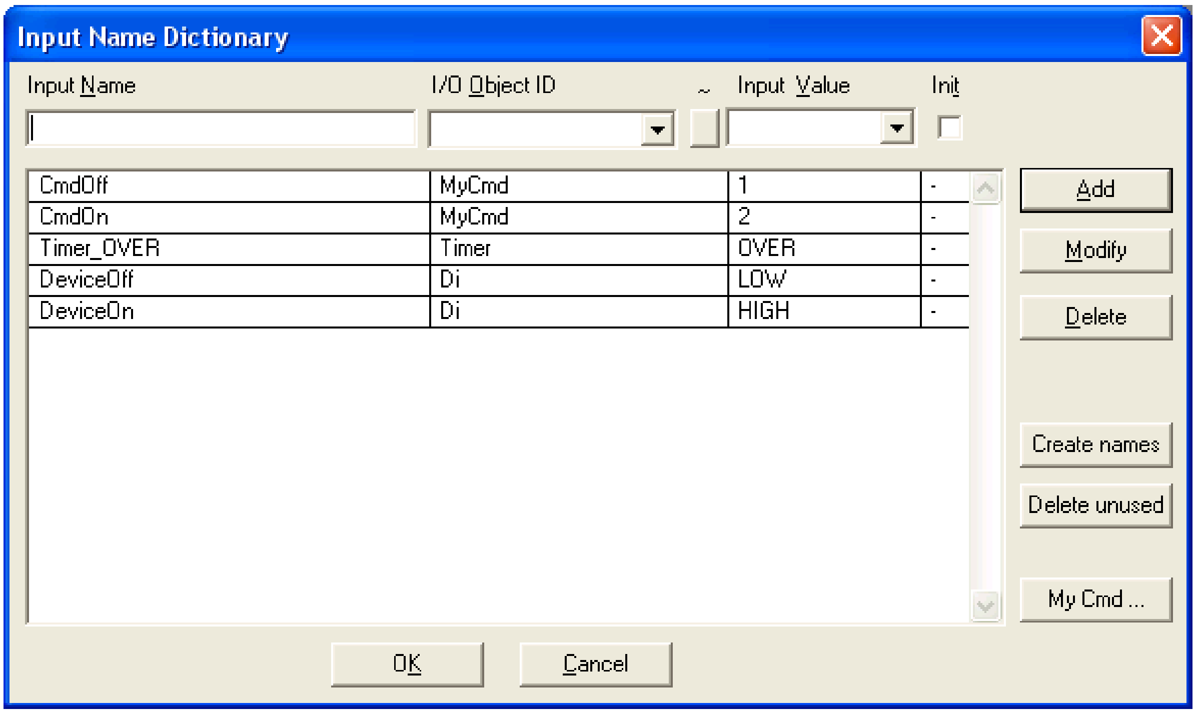
You are free in inventing Input Names. Anyway, also in this case, we will give you some suggestions. The editor helps you also by generating some proposals.

Open the Input Name Dictionary(Figure 7):

Dictionary / Input...

I/O Objects have various Control Values (i.e. can be in different states): Di can be *LOW*, *HIGH or UNKNOWN*, Timer can be in one of the states: *RESET*, *RUN, STOP, OVER, OVERSTOP*, etc. For each of these Control Values you can define an input name that describes its control meaning. If you choose the **I/O Object ID** and its **Input Value**, and click the **Add**-Key the editor produces for most object types automatically a name in the **Input Name** field. The editor does not propose you a name for object types whose Control

Value is a number (CMD, XDA, OFUN). If you like it you click again the **Add**-Key and the name will be inserted into the list of **Input Names**. The editor generates the name by combining the Object name with its (control) Value. Hence, in the example, if you choose the Object *Timer* and the Value *OVER* the editor will suggest the Name *Timer_OVER*.



Input Name	I/O Object ID	Input Value	Init
CmdOff	MyCmd	1	-
CmdOn	MyCmd	2	-
Timer_OVER	Timer	OVER	-
DeviceOff	Di	LOW	-
DeviceOn	Di	HIGH	-

Figure 7 Input Name Dictionary

You should not accept uncritically the editor's suggestions. In many cases, they might be OK, like for instance the name *Timer_OVER*. In contrary to this, the names: *Di_LOW* or *Di_HIGH* are not very meaningful names. Therefore, in the example, we have decided that names: *DeviceOn* and *DeviceOff* are better names than the neutral ones: *Di_LOW* or *Di_HIGH*.

A default name "a*lways*" is automatically added to the **Input Name** list. It exists always in the virtual input; it cannot be removed. It is not shown explicitly in this dialog window but will be available for specification of the state transition table.

The button ~ is used to define complement control values: see the discussion in the sub-section "Conditions as logical expressions" later.

The dialog window has a button **Create names** which generates automatically names for all objects using their values. The automatic generation applies to all object types except CMD_IN, XDA and OFUN whose values are integer numbers. The button *Create names* creates names on all true values if the button "~" is inactive and names on all complement values if that button is active.

Another button **Delete unused** deletes all names which are not used for specification in ST tables.

Initializing the virtual input

By setting an **Init** mark you can decide that a name is active when the state machine starts. This is an important point if a name is defined on a value that is not known at system start-up or on an initial object value.

For instance, after initialization names defined as digital values: *FALSE* and *TRUE* are not present in the virtual input as they are not known. The first change of the digital input will write one of them into the virtual input. If you know what the value of a digital input will be on start-up you can initialize the virtual input to the corresponding Control Value. If you do not know the value you had better set it as *UNKNOWN*.

Another example could be a Timer. Consider a state machine that resets a Timer in an Entry Action of the first state after state *Init*. You could not expect that at this moment the virtual input contains a name based on the *RESET* Control Value. The reason is the missing change of the object Control Value: the virtual input is actualized if the object Control Value changes. At system start-up, the Timer is initialized into the *RESET* state. Therefore, the Reset Action as an Entry Action does not change the Timer state. Hence, the virtual input will not be actualized. Initializing the virtual input to the name based on the *RESET* Timer Control Value would be the correct solution. Of course, you will only do this if you really need the *RESET* Control Value for the specification.

DI (Digital Input) Input Names

You can define names for three DI-object Control Values: *HIGH, LOW* and *UNKNOWN*.

In the example we use two Control Values: *HIGH* and *LOW* defining two names for them: *DeviceOn* and *DeviceOff*.

TI (Timer) Input Names

The Timer can have the following Control Values: *RESET, STOP, RUN, OVER, OVERSTOP*. For each Control Value you can define a name.

In the example we use only one timer Control Value: *OVER* which means that the Timer has elapsed. So, we have defined a name: *Timer_OVER*.

Note: After Initializing the value of a DI object is not known and the virtual input contains no names defined on digital inputs. The virtual input can be explicitly initialized to certain values by marking the Init flag in the Input Name Dictionary.

Defining Output Names

(Virtual) Output Names are names invented by a designer to specify Actions carried out by a VFSM when entering a state (Entry Action), when leaving a state (Exit Action) or when receiving a (virtual) input name (Input Action) without a state transition occurring.

You are free in inventing Output Names. The comments we have made for Input Names apply to Output Names too.

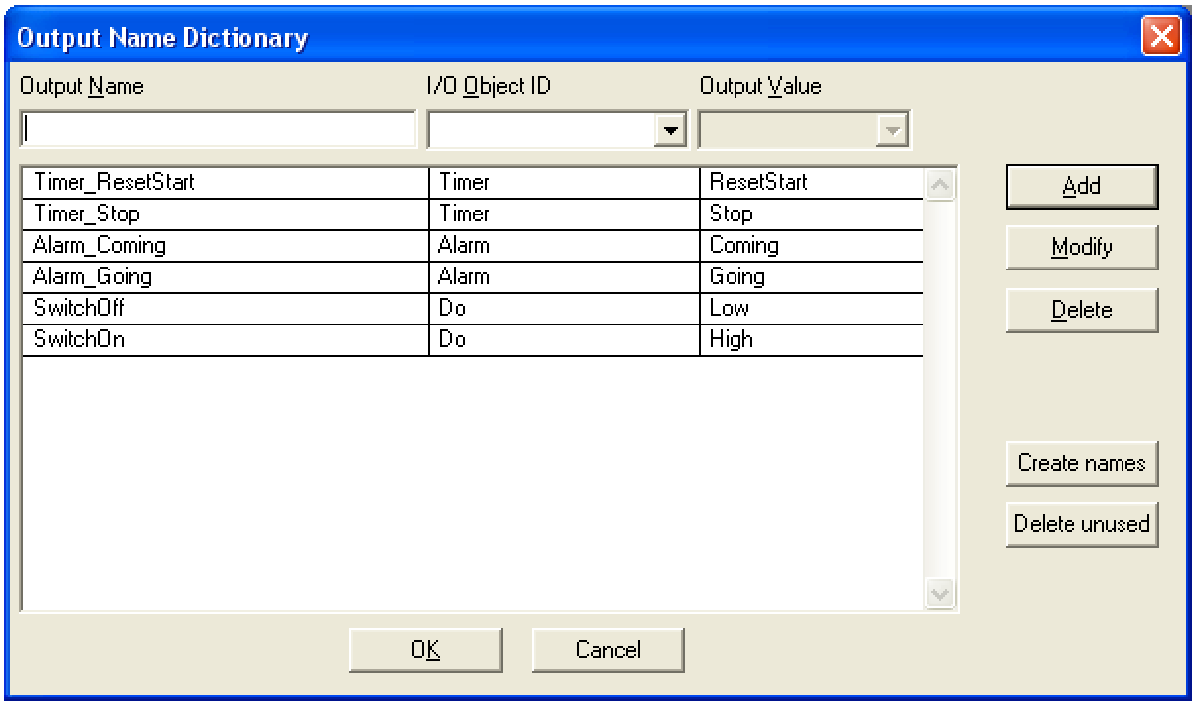
Open the Output Name Dictionary (see Figure 8):

Dictionary / Output...

Also in this case, we have not accepted names *Do_Low* and *Do_High* proposed by the editor. Instead, we have specified names: *SwitchOff* and *SwitchOn* that explain better what the control does. Note that if we have another device that is switched off by a *High* signal and switched on by a *Low* signal we just change the output value definition but not the VFSM specification. Another solution would be to change the value of the real digital output; we will discuss it later with the Project Manager. The VFSM specification specifies the behaviour of the state machine - we want to say that the device is to be switched on or off but we do not want to be more specific at this moment.

In other words, for the behaviour specification the details of switching on or off (*Low* or *High* signal or may be even another type of signal) are not defined; you should avoid defining them at this stage.

The dialog window has a button **Create names** which generates automatically names for all objects using their values. Another button **Delete unused** deletes all names which are not used for specification in ST tables.



Output Name	I/O Object ID	Output Value
Timer_ResetStart	Timer	ResetStart
Timer_Stop	Timer	Stop
Alarm_Coming	Alarm	Coming
Alarm_Going	Alarm	Going
SwitchOff	Do	Low
SwitchOn	Do	High

Figure 8 Output Name Dictionary

The automatic generation can be applied to all object types except those which values are integer numbers (XDA, TAB and OFUN).

DO (Digital Output) Output Names

You can define names for the two DO-object values: *High* and *Low*.
In the example we use both values, and define two names: *SwitchOn* and *SwitchOff*.

TI (Timer) Output Names

The following Actions can be carried out with a Timer: *Reset, Stop, Start, ResetStart*. You can define a name for each Action.

In the example we activate two timer Actions: we reset it and start at the same moment and we stop it. So, we have defined two names: *Timer_ResetStart* and *Timer_Stop*

AL (Alarm) Output Names

You can define Alarm names for three Alarm Values: *Coming*, *Going* and *Staying*.

Use *Coming* and *Going* values if the erroneous situation can be corrected by itself, for instance a too low temperature value could increase later and reach the correct value.

Use *Staying* value if the erroneous situation is an irreversible one, for instance it cannot be corrected without an operator intervention.

In the example we have defined two names: *Alarm_Coming* and *Alarm_Going*.

Filling the state transition table

Filling the state transition table requires a good knowledge of the state machine execution model [1][2] used by VFSM.

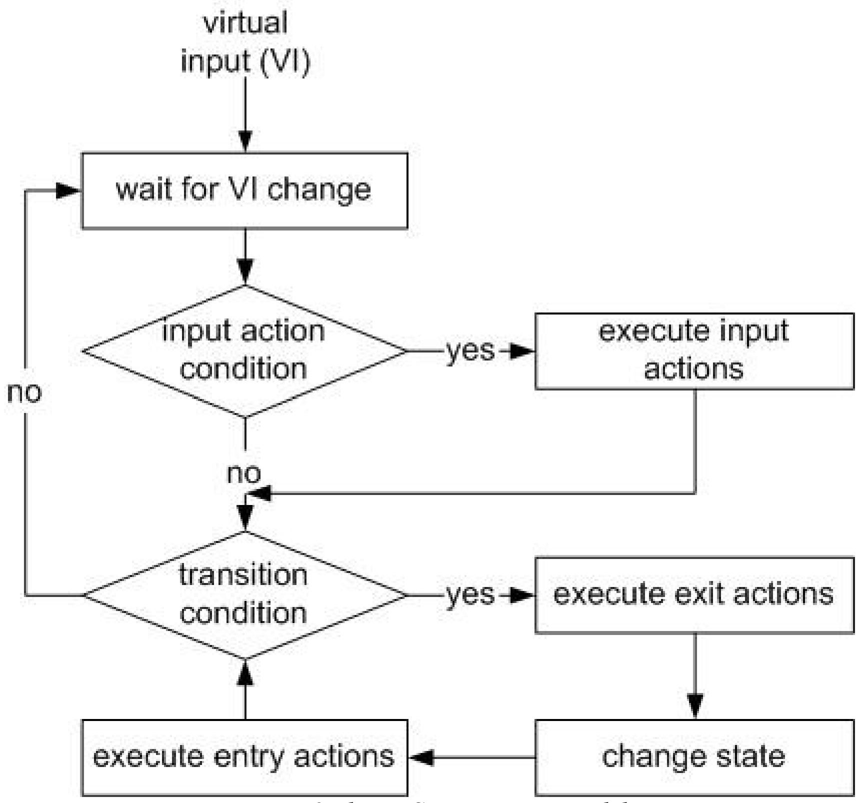


Figure 9 The VFSM execution model

VFSM is a finite state machine that describes behaviour of a system using a Virtual Environment. The VFSM execution model is a combination of Mealy and Moore types of automata, allowing Entry Actions, Exit Actions and Input Actions.

Figure 9 shows the execution model. While specifying the VFSM behaviour take into consideration the following features of the execution model:

- the Entry Actions are carried out only once on entering the state;
- the Exit Actions are carried out only once on leaving the state;
- the Input Actions are carried at any time the VFSM is triggered;

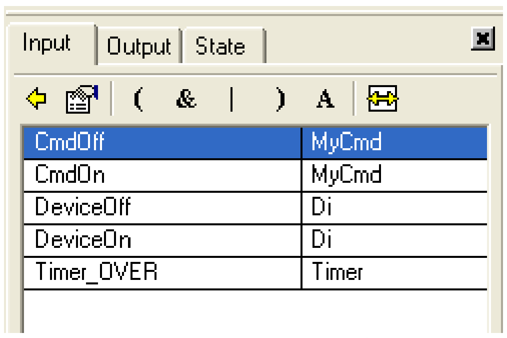
- VFSM execution is continued until there are no more transition conditions possible - this means that one input change might cause many state transitions.

Warning: All state machines are initialised to the state Init. In principle, state machines do not return to state Init. If you do specify a state machine which returns to the state Init, do not forget that it probably does not make sense to specify any Entry Action in the state Init because this Entry Action will be not performed on start-up.

The state machine specification consists of transition specifications and Action specifications. The transitions may be placed on the ST diagram by “drawing” a transition between two states. To achieve this you move the cursor over the start state where the transition is to be specified, click on the right mouse button and move the cursor to the transition destination state. During the movement a transition arc is displayed. Releasing the mouse over the destination state you get the required transition on the diagram and the ST table for the start state opens. The transition to the destination state is already entered in the ST table. You have now to fill in the transition conditions. A transition can be deleted at any time on the ST diagram by selecting it (click with the left mouse on the transition arc) and pressing the Delete key.

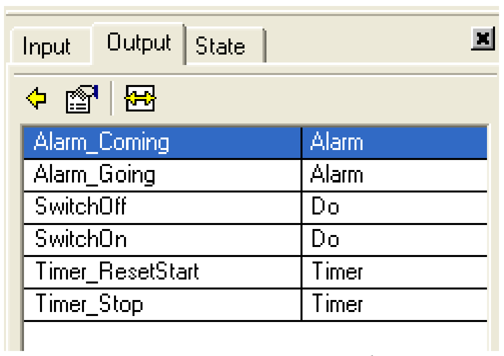
A transition can be also specified by opening a ST table (double click on a state in the ST diagram) and entering the transition. To update the ST diagram with the new state transition you just select the diagram again.

In other words, the ST diagram is not only a graphical representation of the state machine but it can be used to initiate the specification of transitions and Actions. The actual specification details are defined in the ST table.



CmdOff	MyCmd
CmdOn	MyCmd
DeviceOff	Di
DeviceOn	Di
Timer_OVER	Timer

Figure 10 Input Name list

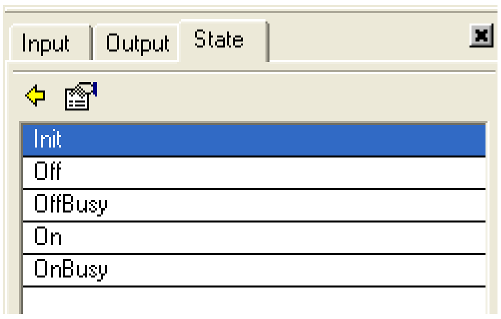


Alarm_Coming	Alarm
Alarm_Going	Alarm
SwitchOff	Do
SwitchOn	Do
Timer_ResetStart	Timer
Timer_Stop	Timer

Figure 11 Output Name list

You fill in the transition table either by writing the State, Input and Output Names or by copying them from the lists. Of course, we suggest the second approach which allows you to specify and modify the table quickly and effectively, with less risk of typing errors. The table fields are context sensitive. Having the cursor in a given field you can open a corresponding **Name** list by clicking the **right** mouse key.

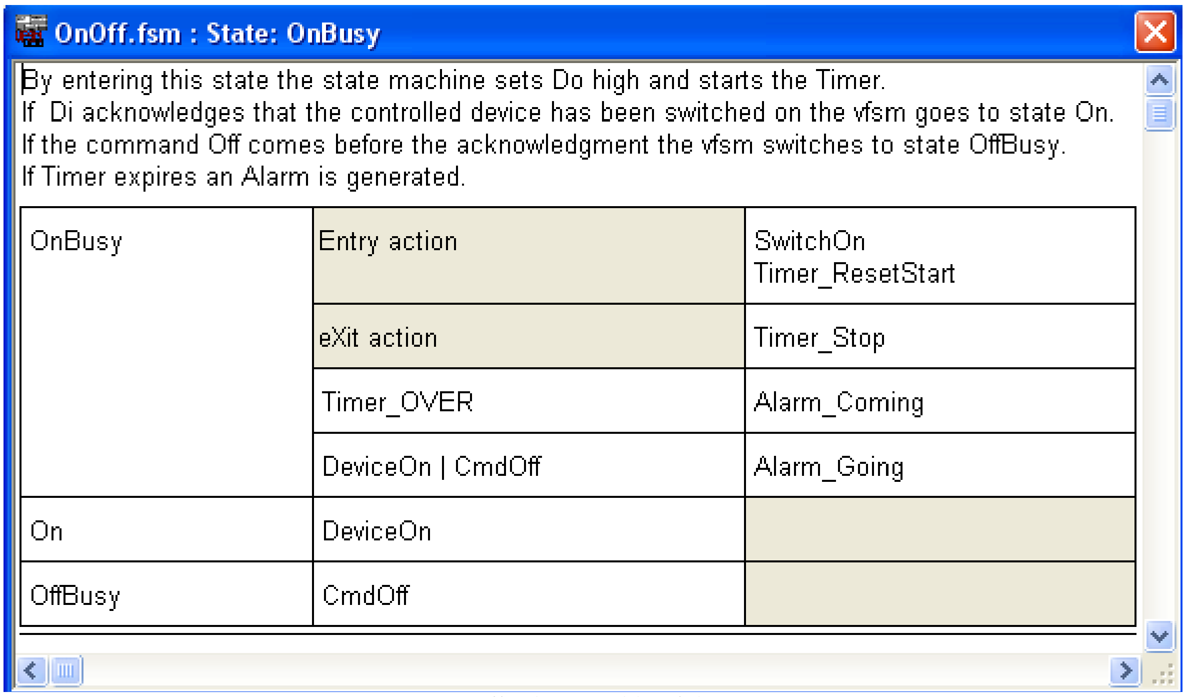
All (Input Name, Output Name and State Name) lists are available as tab windows (Figure 10, Figure 11, Figure 12) switching according to the cursor position in the fields of the ST table.



Init
Off
OffBusy
On
OnBusy

Figure 12 State Name list

Figure 13 shows one state of the OnOff state machine as specified by the StateWORKS Studio; in the project OnOff you will find the entire specification.



By entering this state the state machine sets Do high and starts the Timer.
If Di acknowledges that the controlled device has been switched on the vfsm goes to state On.
If the command Off comes before the acknowledgment the vfsm switches to state OffBusy.
If Timer expires an Alarm is generated.

OnBusy	Entry action	SwitchOn Timer_ResetStart
	eXit action	Timer_Stop
	Timer_OVER	Alarm_Coming
	DeviceOn | CmdOff	Alarm_Going
On	DeviceOn	
OffBusy	CmdOff	

Figure 13 OnOff: The ST table of the state OnBusy

Conditions as logical expressions

The Input Action condition and the transition conditions are in this example just single names. In general, you may use logical expressions similar to Boolean ones. The logical expressions may contain the two operators: AND (&) and OR (|) but there is no NOT operator. The VFSM concept does not use the NOT operator as the names are not Boolean values; they represent several Control Values of a given value (variable). For instance, a timer may have the following Control Values: RESET, STOP, RUN, OVER, OVERSTOP.

As equivalent for the NOT operator you may use the *complement* control values described in the next sub-section.

To see the consequences of this technique let's to look more closely at the DI object. In the VFSM technique it has three Control Values: FALSE, TRUE, UNKNOWN. Using the Control Values you may define three control names for it, for instance: DeviceOff, DeviceOn and DeviceStatusUnknown. As you see it is different from Boolean algebra where the variable always takes one of the two values: true or false. In effect, while specifying the conditions you may use three or more names and not just one Boolean value.

Considering the logical expressions, you may also use brackets if you find the form more readable. The expression may be then expanded to the standard form. For instance, you may specify an expression:

```
(DeviceOff | TimerOVER) & CmdOff
```

Using menu command:

Options / Expand expression

this expression will be transformed to:

```
DeviceOff  & CmdOff | TimerOVER & CmdOff
```

In the standard form the AND (&) operator has higher priority than the OR (|) operator.

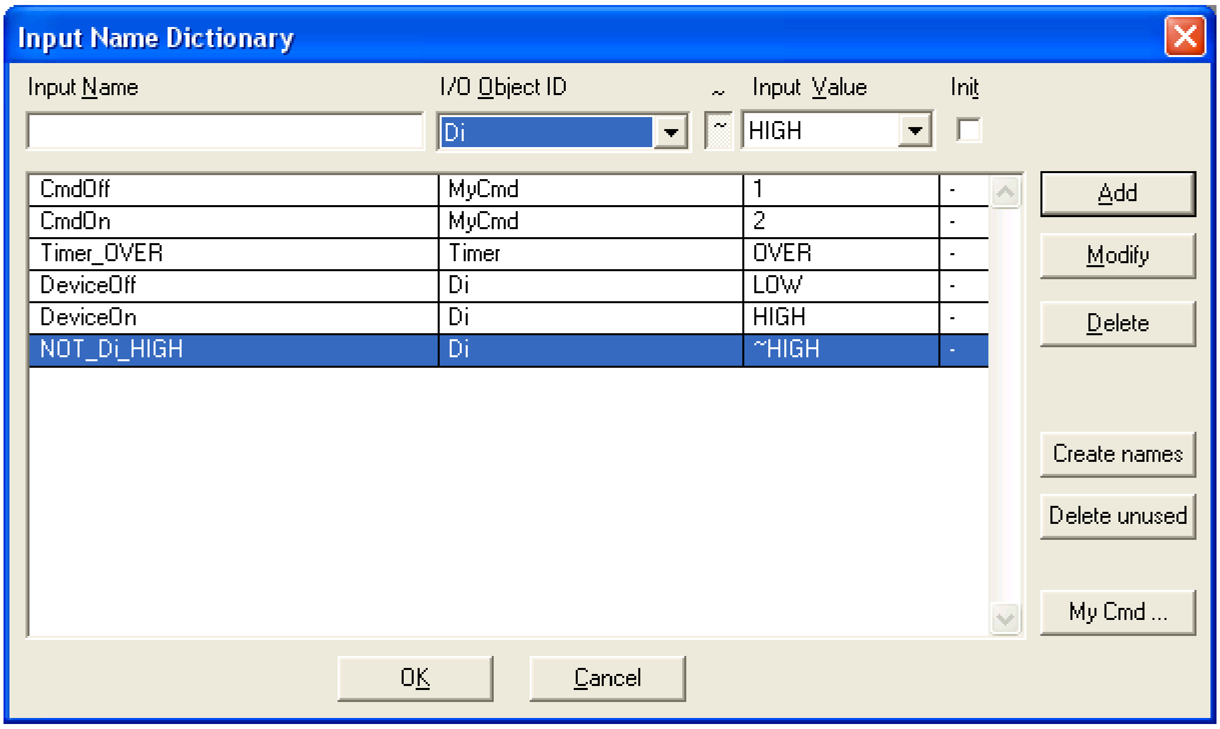
Warning: Once expanded the expression cannot be reduced to the original form with brackets.

Complement value

In the specification we can use two forms of the control value: *true* and *complement* [10]. A negated control value - its complement - is denoted by the prefix ~ and means any other value, for that object. For instance, for the above discussed DI object the ~TRUE means any other value: FALSE or UNKNOWN.

The use of complements within a specification is simple. In the Input Name Dictionary dialogue window we see the button labelled as "~". This button is used to complement the control values. If the button is not pushed (as in Figure 7) the values used are true (not complemented).

Pushing this button (as in Figure 14) it presents the caption "~"and the values used are complemented. By default the input names generated on complement values receive the name with a prefix NOT_ (of course we may rename the default value with any string). We may use the button "~" when adding (button *Add*) or modifying (button *Modify*) input names. While modifying a name the prefix NOT_ will be added if changing to a complement value or it will be removed if changing to a true value.



Di ~ HIGH

CmdOff	MyCmd	1	-
CmdOn	MyCmd	2	-
Timer_OVER	Timer	OVER	-
DeviceOff	Di	LOW	-
DeviceOn	Di	HIGH	-
NOT_Di_HIGH	Di	~HIGH	-

Figure 14: Input Name Dictionary with activated button Compliment

Actions as outputs

The same technique is used for outputs. Let’s discuss the DO object which has two Actions: *Low* and *High*. You define two names for these Actions, for instance: SwitchOff and SwitchOn. The presence of the Action defined by the name SwitchOff means that the real digital output corresponding to the DO object will be set to Low. Using the Action defined by the name SwitchOn means that the real digital output will be set to High. As you see it is essentially different from the usage of Boolean variables which value defines the output. VFSM technique uses two different Actions for this purpose.

We have already mentioned that the (input or output) name might not express the true value, i.e. for instance the name SwitchOff may be defined for the Low or High object value of a DO. The name SwitchOff should describe the effect – what happens on the output and not which value causes this effect. There is also the possibility of using *Invert* on any input/output when specifying the object Properties in the Project, which will need to be taken into account.

„Always“ Table

The VFSM specification may use the Always table. The Always table specifies Input Actions that are carried out in all states. You deal (Insert, Append and Delete) with them exactly as in the other state tables.

Of course, the Always table has nothing in common with the “always” input name - it is just a similar naming. The Always table is used to define state independent Input Actions, i.e. actions performed in each state.

Saving VFSM

Saving a VFSM the first time you will be asked about the prefix. The editor will suggest a default prefix using the first three letters of the file name. Normally, you should accept this.

The prefix is used in the *.h and *.iod files. All names get a prefix that must be unique in the VFSM project.

Building a VFSM

The menu command:

File / Build

generates files (*.str, *.iod, *.h) that are used by the VFSM Executor.

When building, the Studio signals whether the specification is correctly done. If the specification table contains errors the Build is stopped, the first encountered error is signalled on the status bar at the bottom of the Studio window and the cursor is positioned in the field with the error.

You can test the VFSM using the StateWORKS Lab which is a kind of a simulator that allows you to debug your control logic without having the controlled device.

Before you can test your VFSM you have to create a project to configure the system, especially the I/O devices.

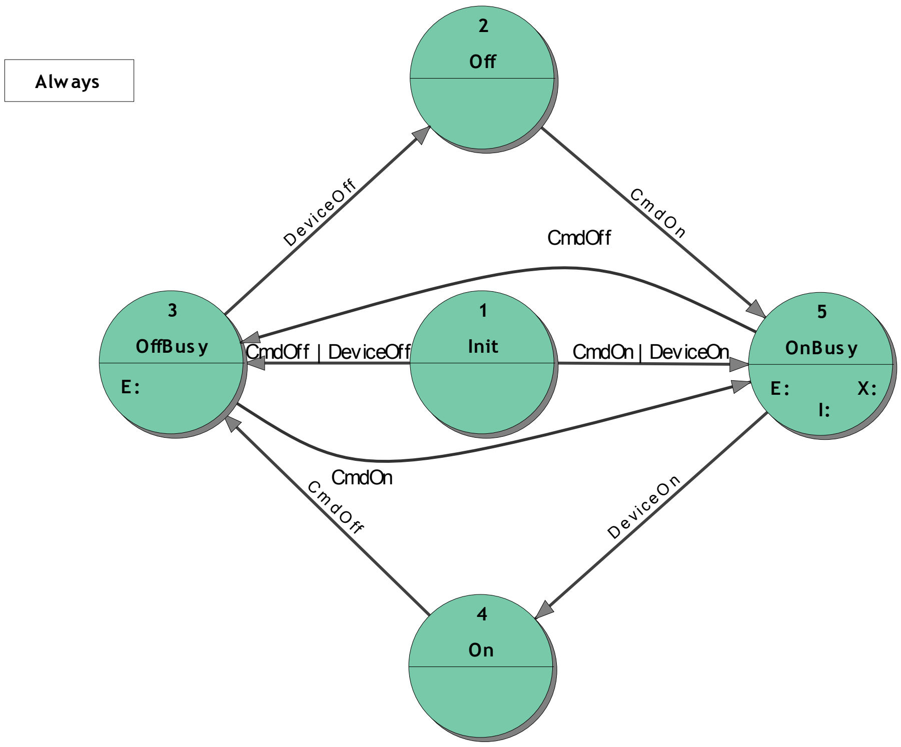


Figure 15 OnOff: the ST diagram

Example OnOff

When the specification of the OnOff example is completed the ST diagram shown in Figure 15 will be displayed. The ST diagram represents general information about the ST diagram: the states and transitions. In addition, the symbols E:, X:, I: indicates what kind of Actions are specified in states. Full information is displayed if you place the mouse cursor over a given symbol.

System Configuration

Figure 16 shows what we have done so far with the example OnOff. We have specified the behaviour of the VFSM using symbolic names taken from three lists: Input, Output and State. A state is an internal variable of the system and does not need any translation into the real world. In contrary to the state, input and output names are just descriptions of some conditions and Actions in the controlled device or system resources. As the names are not real signals but only representations of control properties of the real signals we call them virtual inputs and outputs.

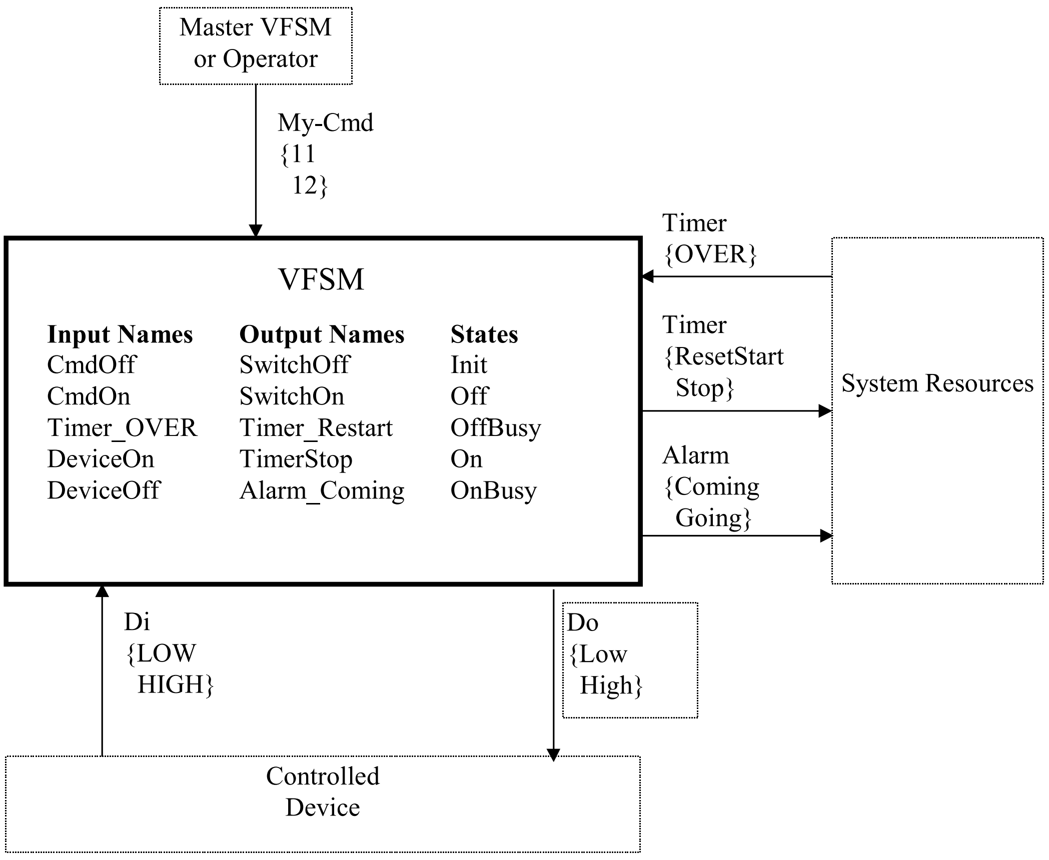


Figure 16 The elements of the virtual specification

The Input and Output Names describe Control Values and Actions of the commands (MyCmd), the digital input (Di), the digital output (Do), the timer (Timer) and the alarm (Alarm). The command comes from another state machine or is generated by a system, operator, etc. in a form of a number. The digital input comes from the controlled device. The digital output goes to the controlled device. The timer belongs to system resources, normally, it is a part of the software. The alarm is usually a part of the software and produces for instance some text on a display.

The system configuration is prepared by using the StateWORKS Project Manager. For the OnOff example you have to configure the following I/O objects:

- the Timer and its timeout value
- the Alarm and its text
- the Cmd (where it comes from)
- the origin of Di
- the origin of Do

Opening the Project Manager

In order to configure the system you have to create a project:
Project / New.

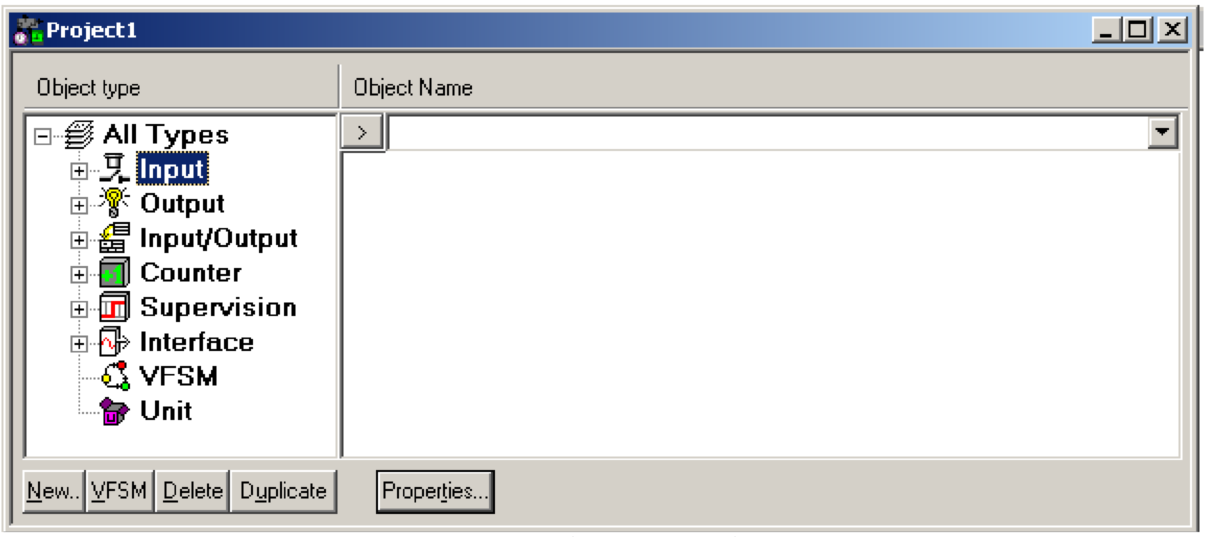


Figure 17 The Project window

The project window has two panes. The first pane **Object Type** contains a list of all object types available in the project. This is the object type list we have used when creating the I/O Dictionary. The object types are organized in a directory structure. By opening the Project Manager, the Object Type pane displays only the highest directory which shows the basic groups: Input, Output, etc. You may display a specific object type by clicking on the corresponding directory name.

The second project pane **Object Name** is in the beginning empty; it will include objects you configure in the system.

Defining the system of state machines

You define the state machine types that will be used in configuration of the system by adding each state machine to the project. In our first example you have only one state machine - OnOff. Add it to the project:

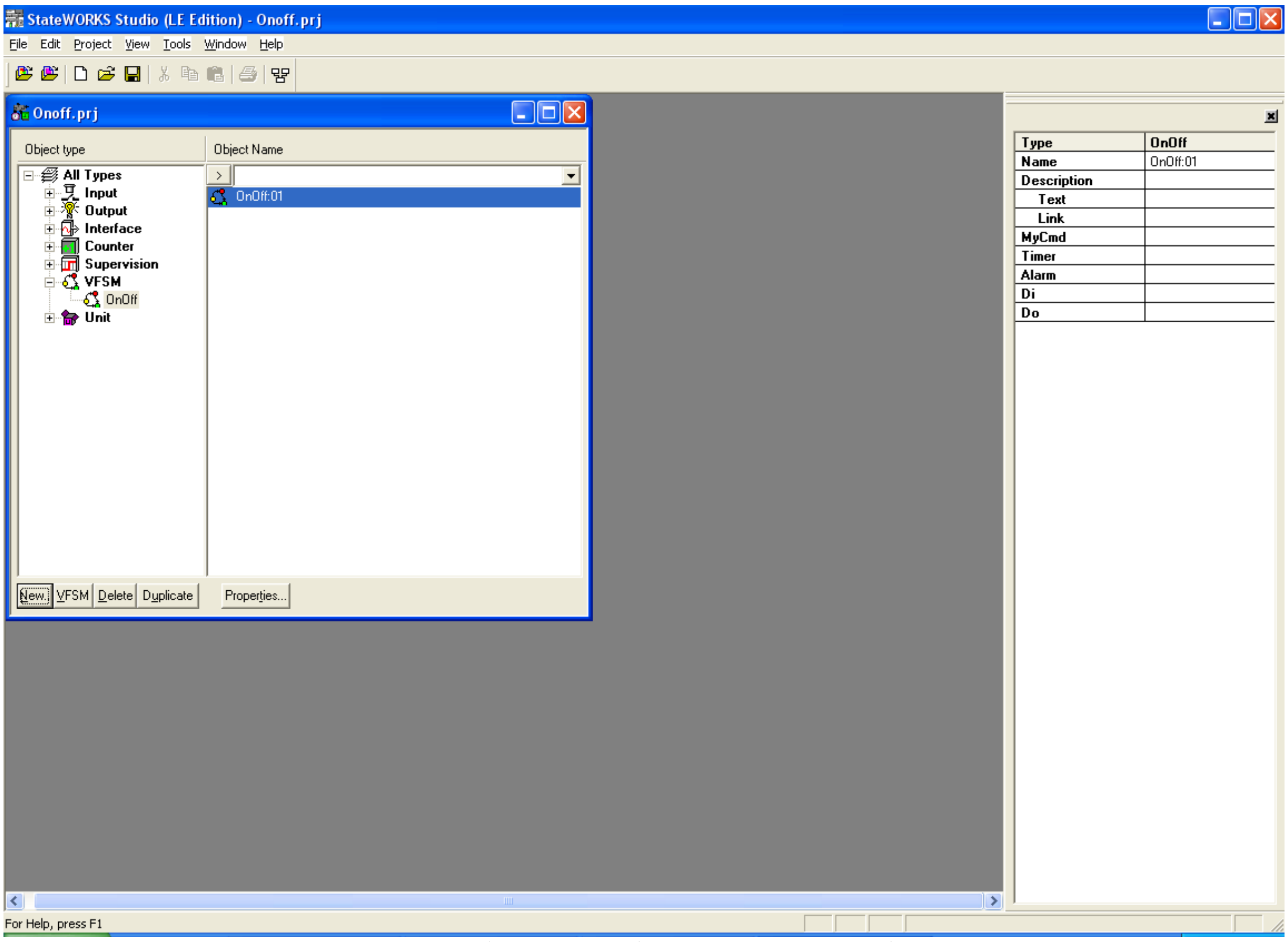
Project / Edit / Add.

State machines that are in the project appear at the end of the object type list before the Unit. A state machine is an object like any other object.

You need the following object types you are going to use in the system configuration: CMD (MyCmd), TI (Timer), AL (Alarm), DI (Di), DO (Do) and the state machine VFSM (OnOff).

Configuring the system

With the key **New** in the Project Window (Figure 18) you can create an object of a given type. The sequence of creation is irrelevant. You can at any time add and delete objects or change their properties.


StateWORKS Studio (LE Edition) - Onoff.prj
File Edit Project View Tools Window Help
Onoff.prj
Object type
Object Name
All Types
Input
Output
Interface
Counter
Supervision
VFSM
OnOff
Unit
OnOff:01
New.. VFSM Delete Duplicate Properties...
For Help, press F1

Type	OnOff
Name	OnOff:01
Description	
Text	
Link	
MyCmd	
Timer	
Alarm	
Di	
Do	

Figure 18 The Project and VFSM Properties windows

Create first the VFSM OnOff. In the **Object name** window, you see the object with a default name OnOff0. With the key **Properties** or by double click on the object name, open the **OnOff properties** window. The window comprises a list of all I/O objects that are needed by the VFSM OnOff. The list begins with the Name and Description fields (Text, Link). The following positions in the left column are just names of all I/O objects used by the VFSM specification. The right column is still empty as we have not defined yet any other object.

First of all, we do not like the name OnOff0. So, we change it to OnOff - we have only one state machine in the project and we do not see any reason to introduce another name as the name of the VFSM type.

Now, you have to create the objects that belong to this state machine. A click on the button **New** creates an object of a type selected in the **Object type** pane.

If you click now on the top directory name AllType in the Object Type window a list of all objects defined so far will be displayed in the Object Name window. Alternatively, you may display only a subset of the objects by selecting a specific group type or specific object type.

Specifying the object properties

To specify the properties of any I/O object select the object in the **Object Name** list which opens the Properties window of this object.

VFSM (OnOff) Properties

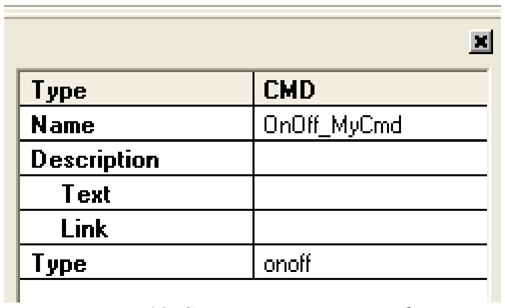
The specification of properties of the state machine OnOff is fairly obvious: those are the names of the I/O objects that belong to the state machine. For the example OnOff we have used the following I/O names (you are free to use any names you like):

- OnOff_MyCmd
- OnOff_Timer
- OnOff_Alarm
- OnOff_Di
- OnOff_Do

CMD (Command) Properties

Creating object of type CMD you see in the **Object Name** window the default **Name** of the created object. You may change it if you do not like it. By clicking on the CMD object *OnOff_MyCmd* in the **Object Name** window you open the **CMD properties** window.

The only missing property is the command **Type** (you ignore in this moment the **Description** fields). If you write there the name of the VFSM - *OnOff* the Project Manager will pass this information to the data file used by the VFSM Executor and the VFSM Executor will use the symbolic command names defined (indirectly) during the VFSM specification.



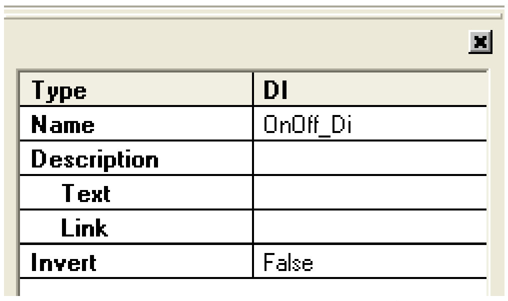
Type	CMD
Name	OnOff_MyCmd
Description	
Text	
Link	
Type	onoff

Figure 19 CMD properties window

If you do not write the type the VFSM Executor will not know the names and will react to command values (numbers) specified in the Input Name Dictionary.

DI (Digital Input) Properties

In addition to the name itself the DI type objects have one specific property: **Invert**. If the **Invert** check box is not marked the digital value is passed as is. Marking the **Invert** check box inverts the digital input value.



Type	DI
Name	OnOff_Di
Description	
Text	
Link	
Invert	False

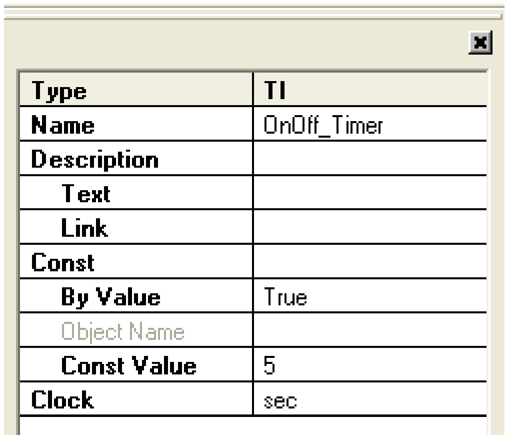
Figure 20 DI properties window

DO (Digital Output) Properties

The DO type objects are like DI objects. We do not mark the **Invert** property which means the values is passed as is.

TI (Timer) Properties

Specifying the TI object you see in the **Object Name** window the name Ti:01 which you may change to a more expressive **OnOff_Timer**. Open the **TI Properties** window. You have to specify the timer's features: timeout **Const**(ant) (choose for instance 5) and timer **Clock** (choose *sec*). The timeout **Const(**ant) could be a parameter if you remove the mark **By value**. We will try this later.



Type	**TI**
Name	OnOff_Timer
Description	
Text	
Link	
Const	
By Value	True
Object Name	
Const Value	5
Clock	sec

Figure 21 TI properties window

AL (Alarm) Properties

Create an AL object giving it a name *OnOff_Alarm*. In the **Al Properties** window for OnOff_Alarm specify an alarm **Text**. Do not bother about the **Category** at this stage – give it for instance the value 2 (we will discuss this property later).

Introducing I/O Units

Several objects such as: Cmd, Alarm and Timer could be considered as internal system resources created and managed by the software.

Digital inputs and outputs are signals that belong to the controlled device, i.e. they are external, physical signals. Usually, they would be organized in groups that are managed by I/O drivers. StateWORKS software is based on the assumption that I/O signals received and sent from/to controlled devices exist in units. For simulation purposes the StateWORKS Lab software recognizes four predefined standard units types: DI16, DO16, NI4 and NO4 that must be included in projects that are tested by the StateWORKS Lab. In your case you add DI16 and DO16 unit types to the project:

Project / Edit / Add.

Having the units in the project, create a digital input unit with the name *DI16_Unit1* and a digital output unit with the name *DO16_Unit3*.

The properties of the *DI16_Unit1* are:

- Physical Addr = 1
- Di0 = OnOff_Di.

The properties of the *DO16_Unit3* are:

- Physical Addr = 3
- Do0 = OnOff_Do.

The Physical Address property of the units must not be changed if you want to test the system with the StateWORKS Lab. The value of the Comm Port property is irrelevant in that case.

Creating the system configuration

If all I/O Units are defined and all their properties are specified you can create the system configuration:

Project / Build configuration

Alternatively, you can create all system files (VFSM and configuration files) by

Project / Build All

The results of Build are displayed in the Configuration Error Messages window. The messages describe fully the completeness of the configuration. There are three kinds of messages

- Some of the messages are just warnings, such as: a digital input present in the DI unit has been not used in the system.
- Other messages say that some properties are just default values: you have to decide whether you have accepted these intentionally or whether you have forgotten to set the required values.
- Eventually, there are messages that signal a significant error such as a missing property.

By clicking on the message you enter the Properties window with the missing property. Irrespective of the messages we always allow the system to generate the ProjectName.SWD file. We believe that it is important to have the chance to make a test with an incompletely configured system even if the results might sometimes not be quite predictable.

The ProjectName.SWD file can be used by the StateWORKS Lab to test the system.

Testing the system

When the system is built it may be tested using SWStudio. SWStudio allows starting the SWLab which is a specific RTDB application. SWLab reads the specification results and simulates the system behavior being triggered by inputs. The system can be monitored using SWMon, SWQuickPro, SWQuick or SWTerm. SWMon is a program with GUI which displays all objects and allows changing of their properties. SWMon allows access to all attributes of a single object at a time. SWTerm is a similar program but in a form of a console client with the command line interface.

All programs required for testing the system can be started in the menu Tools. Helps for those programs are available under menu Help.

Starting the StateWORKS Lab (SWLab)

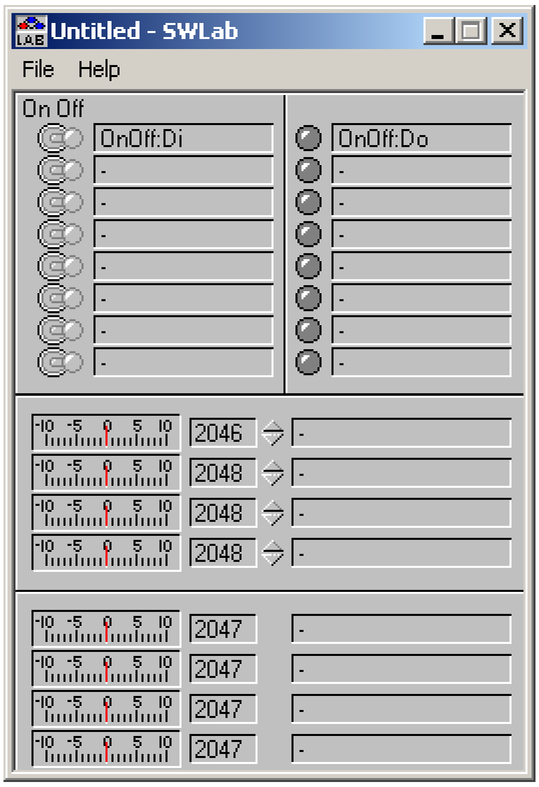


Figure 22 StateWORKS Lab

You start the SWLab by:

Tools / SWLab.

In a running SWLab Lab you can start the RTDB system by opening its SWD file:

File / Open

or restart a running system by:

File / Restart.

The StateWORKS Lab simulates 8 digital inputs, 8 digital outputs, 4 analog (numerical) inputs and 4 analog (numerical) outputs. The digital inputs are represented by switches: you can change their positions with the mouse. The digital outputs activate LEDs. The values of analog input analog output and outputs are shown by analog gauges. You can change the value of the analog inputs by entering a numerical value (clicking on the number windows opens the entry windows) or by activating the arrow symbols that increments/decrements the value by 1 or by 8 (with SHIFT key) or by 64 (with CTRL key).

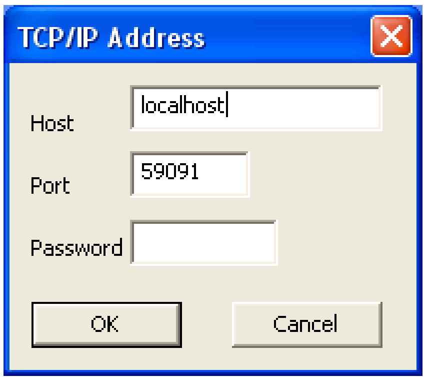
Monitoring RTDB

The StateWORKS Monitor (SWMon) is used to debug the VFSM system of state machines, effectively to monitor the RTDB.

You start the SWMon by:

Tools / SWMon.

When started, the Monitor displays a dialog window to enter the TCP/IP address and port.



Host: localhost
Port: 59091
Password:

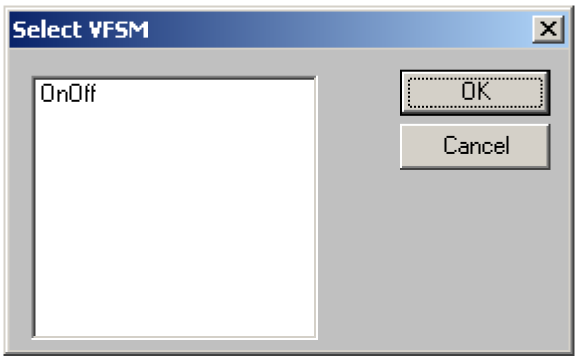
Figure 23: Connecting to SWLab

Accept the default values: *localhost* for the address and *59091* for the port number. Working with SWLab ignore the Password entry as the SWLab does not use a password.

If connected you can display RTDB objects selecting it by:

Select / VFSM

that opens a window with a list of state machines run by Lab.



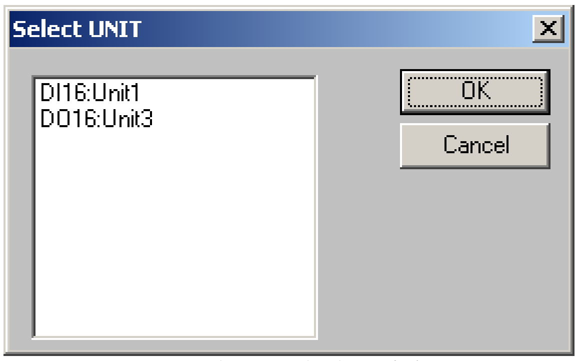
OnOff

Figure 24 Selecting the list of state machines

Alternatively, you may display RTDB objects listed in a Unit:

Select / UNIT

that displays a list of I/O-Units present in Lab:



DI16:Unit1
DO16:Unit3

Figure 25 Selecting the list of I/O-Units

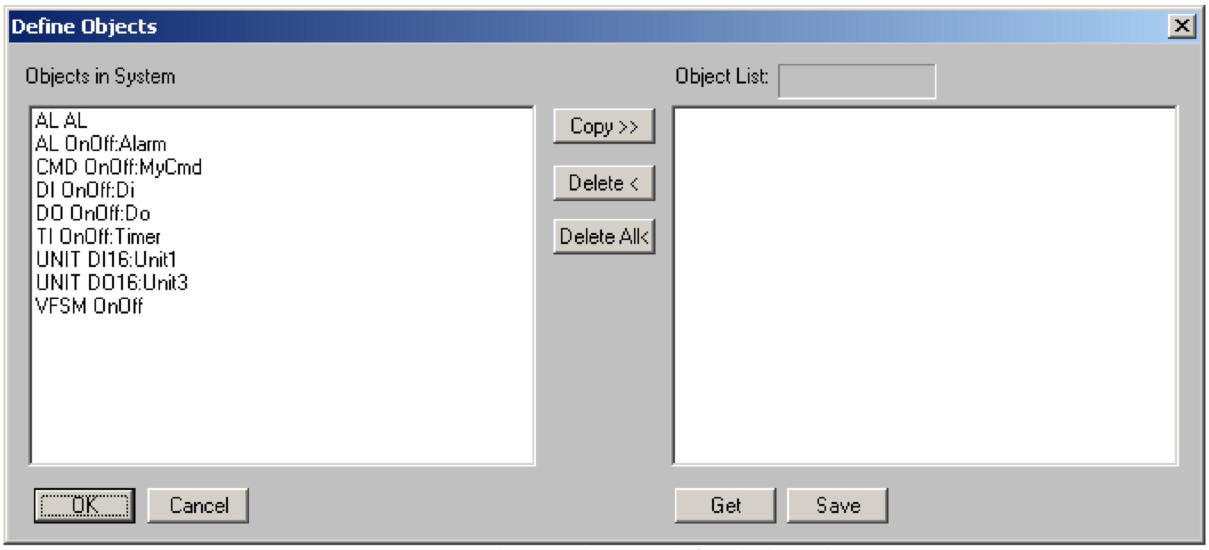
The most convenient way of displaying RTDB objects offers:

Select / User

that displays a list of all objects in the RTDB. You can now define a list of objects to be watched in the Monitor (see Figure 26).

The list can be stored on the disk by **Save**. An already prepared object list can be loaded from the disk by **Get**.

The Monitor display (Figure 32) is divided by a horizontal line into two parts. The upper part contains two lines of edit / display controls. The meaning of the first line of controls depends on the selected object type: VFSM, UNIT or User. The second line of controls is used to handle commands.



AL AL
AL OnOff:Alarm
CMD OnOff:MyCmd
DI OnOff:Di
DO OnOff:Do
TI OnOff:Timer
UNIT DI16:Unit1
UNIT DO16:Unit3
VFSM OnOff

Figure 26 Selecting the user defined object list

If you have selected VFSM the first line of the upper part displays the **VFSM** name and type, and **VIN** (Virtual Input), for instance:

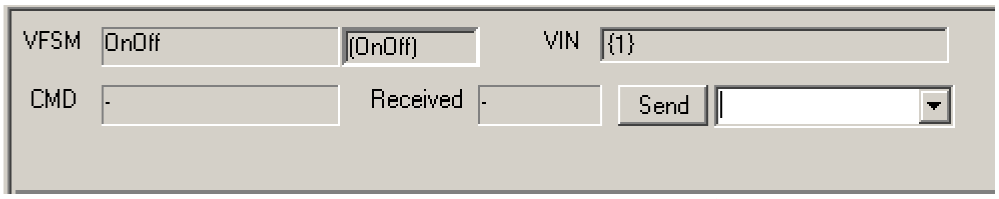


Figure 27 Displaying VFSM virtual input and state

The Virtual Input (**VIN**) shows the input conditions as numbers (you can see the numbers in the StateWORKS Studio by activating the **[123]** icon).

If you have selected UNIT this part displays the **UNIT** name and type, for instance:

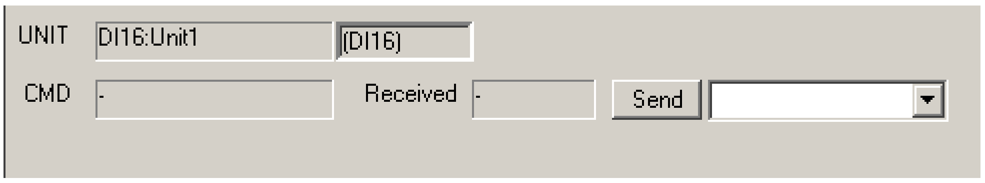


Figure 28 Displaying UNIT name

If you have selected User this part displays the **USER** file name if the list has been stored on the disk (or nothing otherwise):

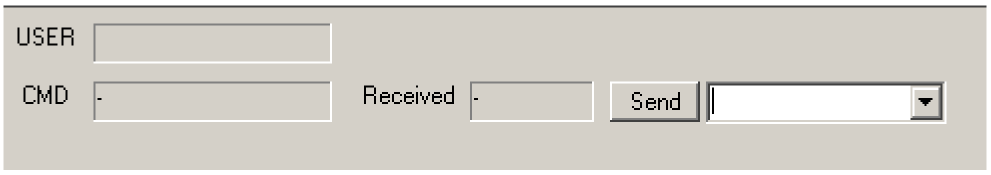


Figure 29 Displaying the name of user defined object list

The **Select** pull-down menu allows you also to select a command object.

Select / CMD

displays a list of state machine commands:

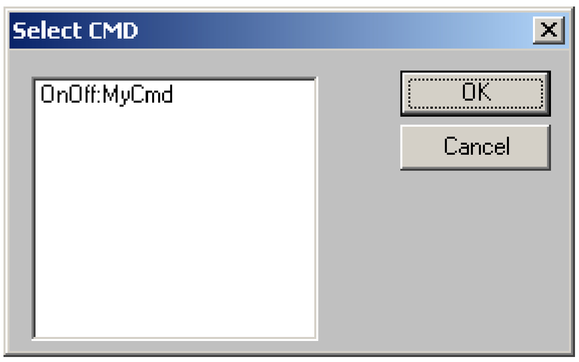


Figure 30 Selecting the list of state machine commands

The command selected from the list appears in the second line of controls in the upper part:

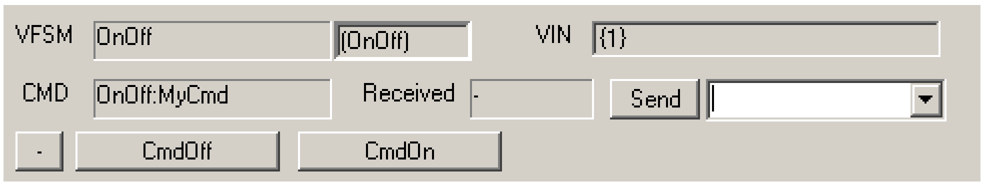


Figure 31 Displaying the command name and values

This line displays information about the selected command object, first of all the **CMD** name.

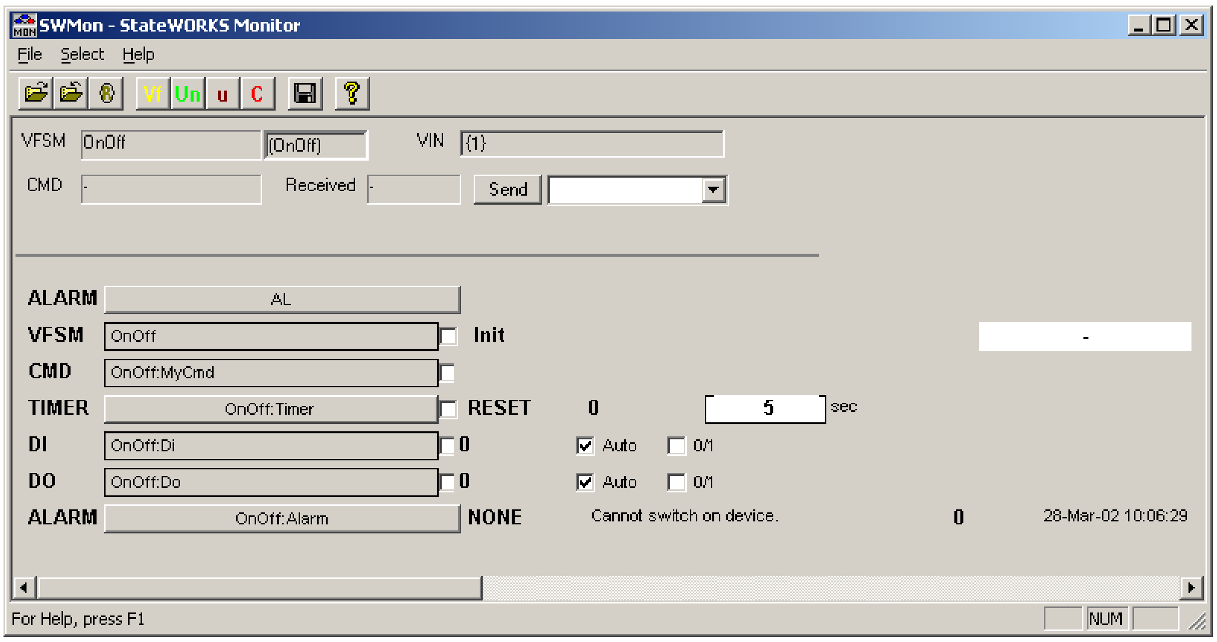
The command that the Command object received from a Master of the state machine is displayed in the second line titled **Received.** Be aware that the (real) command received from the Master state machine and the (service) value, which you can send from the Monitor, are two different things.

You can send a command to the selected Command object by typing the command name or selecting the name from the pull-down menu and clicking the **Send** button. The command can be sent as a real command or as a service one: you trigger the service mode on/off using the trigger button "-" to the left of the command buttons. The command object name becomes bold if the command object is in the service mode. Sending the command as a real value (service mode off) you overwrite the command real value (from the Master) with the value sent from the Monitor. Switching to the service mode restores the original real command value.

If the CMD object has commands with values 1..4 they are shown as buttons with corresponding command names. Clicking on a command button you send a command.

Monitoring RTDB objects

The lower part of the Monitor display has a variable size. It shows all objects that belong to the selected VFSM, UNIT or User defined object list. It shows object values and properties. Both values and properties may be temporarily changed for debugging purpose.



```
VFSM  OnOff          (OnOff)     VIN  {1}
CMD   -        Received  -         Send

ALARM  AL
VFSM   OnOff              Init                          -
CMD    OnOff:MyCmd
TIMER  OnOff:Timer        RESET    0          5  sec
DI     OnOff:Di           0        [✓] Auto  [ ] 0/1
DO     OnOff:Do           0        [✓] Auto  [ ] 0/1
ALARM  OnOff:Alarm        NONE     Cannot switch on device.     0     28-Mar-02 10:06:29

For Help, press F1                                                NUM
```

Figure 32 The StateWORKS SWMon monitor

Some objects may be tested using a service mode. The service mode means that the object value is overwritten by a value entered in the Monitor.

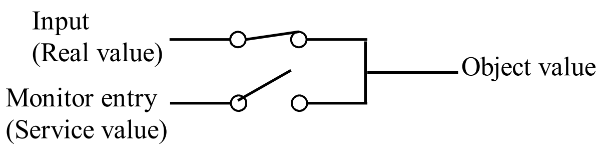


Figure 33 The object service mode

If the service mode is disabled (as in the Figure above) the Object value is determined by the Input (Real value). Enabling the service mode means that the Real value is disconnected and the Object value is determined by the Monitor entry (Service value).

The Monitor shows objects that belong to the selected VFSM, UNIT or User defined list. In case of VFSM an object is displayed in the Monitor if it is defined in the I/O Object Dictionary. You must include an object in the I/O Object Dictionary if you need a condition (input name) or an Action (output name) for specifying the VFSM behavior. You may use objects in the system that do not appear in the I/O Object Dictionary. For instance, a Switchpoint (SWIP object) Limits or Timer (TI object) Const can be parameters. You will rather rarely define a condition on parameter Control Values. So, there is no need to introduce the PAR object into the I/O Object Dictionary. Anyway, you can display objects that are not included in the I/O Object Dictionary, defining your own list of objects (**Select / User**).

VFSM objects are grouped together, each group beginning with the object type name. The following object types can me monitored: ALARM, VFSM, CMD, TIMER, CNT, ECNT, DI, DO, SWIP, STR, PAR, DAT, UDC, NI, NO, XDA, TAB, OFUN. Each object is represented by one line. The line contains basically the elements shown below though the details depend on the object type.

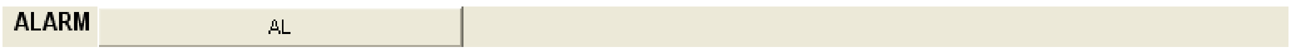
Type	Name	Trace (check-box)	Control Value	Value (Numerical)	Property / Service mode & value (may be many)	Others

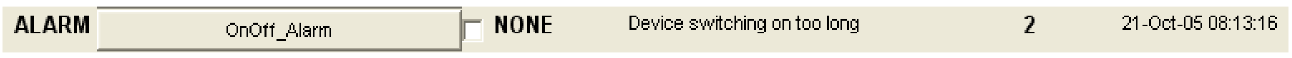
The **Type** is written only once for a group of objects.

Most of the objects have the **Name** on the button. Activating the button you may influence the object: the details depend on the Object type.

All objects have a **Trace** check-box: marking it enables tracing.

Monitoring AL Object


ALARM AL


ALARM OnOff_Alarm NONE Device switching on too long 2 21-Oct-05 08:13:16

The **ALARM** (AL) object line contains a button and the following fields:

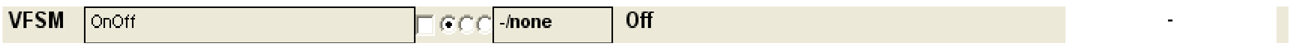
Type	Name	Trace	Control Value	Alarm Text	Alarm Category	Alarm Time
ALARM	Text on a button	Check-box	COMING GOING STAYING ACK COM_GO	String including IDS_ and %references	Integer (preferred values: 1,2,4)	dd-mmm-yy hh:mm:ss

The button carries the ALARM **Name** and is used to acknowledge the alarm.

There are two alarms sections in the Monitor. For a selected VFSM the first line always contains the alarm line which displays all alarms generated by the VFSM. Therefore it has a general name AL. At any time this line shows the last alarm which occurred and has not yet been acknowledged. If there are more alarms present they wait in a queue (first-in-last out). Acknowledging an alarm removes it from the display and the next alarm in the queue is shown. By repeating acknowledgement you can remove all alarms from the queue and the display will then be empty.

At the end of the VFSM objects all VFSM alarms are shown. Each alarm has there its line which shows always its Control Value (status): COMING, GOING, STAYING, COM_GO or ACK.

Monitoring VFSM Object


VFSM OnOff -/none Off -

The **VFSM** object line contains 3 radio buttons and the following fields:

Type	Name	Trace	Run mode	Hold mode	Step	Input Action / Next state	Control Value	Service Value (State)
VFSM	Text	Check-box	Selected if radio-button marked	Selected if radio-button marked	Selected if radio-button marked		State Name	State Value

The **Control Value** field displays a few names of VFSM states. The sequence of the last several states is shown, the left one being the present state. You may scroll the state sequence thus displaying the history of state changes. The scroll buffer is limited to 1000 characters.

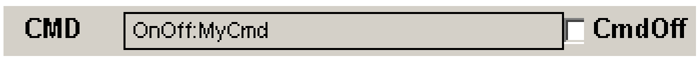
You can overwrite the VFSM State Name by typing the *State Value* in the **Service Value** field. The *State Value* corresponding to a given name can be found in the *.IOD file of the state machine. As long as the **Service value** field contains a number the Master that you have on the screen "sees" the state corresponding to the number. The true state that you still see in the **Control Value** field is overwritten by the *Service Value*.

You cancel the Service Mode by typing a dash (-) character in the **Service Value** field. If you select another state machine to be displayed in the Monitor the Service Mode will be cancelled automatically.

The radio buttons are used to control the Run/Hold mode of the VFSM Executor. The first radio button (**Run mode**) sets the state machine into the Run mode. The second radio button (**Hold mode**) sets the state machine into the Hold mode. The third radio button (**Step**) is used to perform a Step, i.e. one transition of the state machine if the state transition condition is fulfilled or Input Actions which conditions are fulfilled or both.

The **Input Action/Next State** field displays the character *A* or – separated with the slash from the name of the next state if the state machine is in the Hold mode. If there is Input Action due the character *A* is displayed otherwise the dash (-) sign. If no next state is due this field displays the word *none*.

Monitoring CMD Object



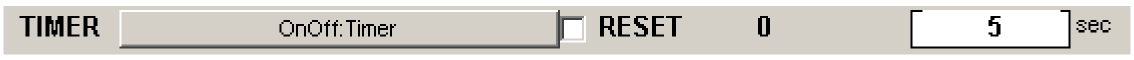
The **CMD** object contains the following fields:

Type	Name	Trace	Control Value
CMD	Text	Check-box	CMD Name or Value

The CMD object line displays the received command Name (or Value if the commands are not named).

The CMD object **Name** becomes bold if the CMD object is in the Service mode.

Monitoring TI Object



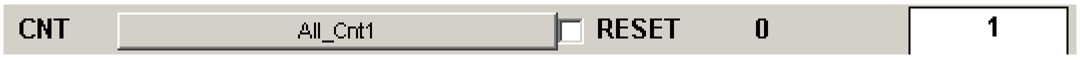
The **TIMER** (TI) object line contains a button and the following fields:

Type	Name	Trace	Control Value	Value	Const (timeout)	Clock
TIMER	Text on a button	Check-box	RESET STOP RUN OVER OVERSTOP	Elapsed time	Integer	min sec 100ms

The button carries the Timer **Name**.

You can overwrite the **Const** *v*alue. The **Name** button is used to restore the original ***Const*** *v*alue.

Monitoring CNT Object



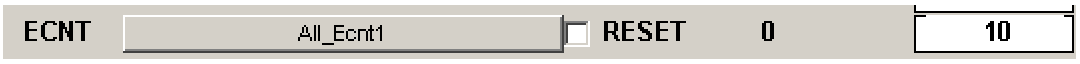
The **CNT** (Counter) object line contains a button and the following fields:

Type	Name	Trace	Control Value	Value	Const
CNT	Text on a button	Check-box	RESET STOP RUN OVER OVERSTOP	Actual count	Integer

The button carries the Counter **Name**.

You can overwrite the **Const** *v*alue. The **Name** button is used to restore the original **Const** *v*alue.

Monitoring ECNT Object



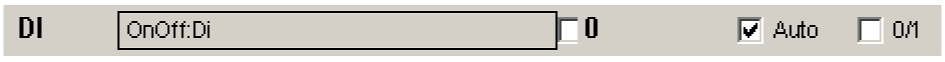
The **ECNT** (Event Counter) object line contains a button and the following fields:

Type	Name	Trace	Control Value	Value	Const
ECNT	Text on a button	Check-box	RESET STOP RUN OVER OVERSTOP	Actual count	Integer

The button carries the Counter N**ame**.

You can overwrite the **Const** *v*alue. The **Name** button is used to restore the original **Const** *v*alue.

Monitoring DI Object



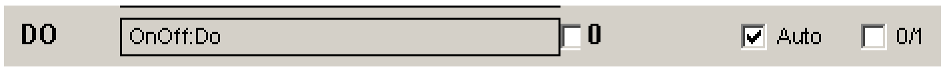
The Digital input object line has the following fields:

Type	Name	Trace	Control Value	Service mode	Service value
DI	Text	Check-box	Boolean: 0 1	Auto: - selected: disabled - not selected: enabled	1/0: - selected: 1 - not selected: 0

The Value is always the **Control Value**.

If you do not select the *Auto* check-box the displayed input value remains the Input value though the system input is equal to the service value. In other words, the **Control Value** always displays the real value (see).

Monitoring DO Object



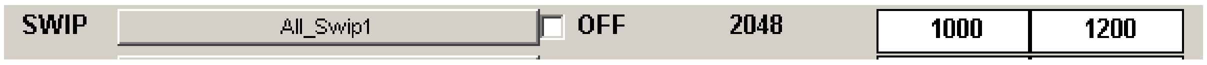
The Digital input object line contains the following fields:

Type	Name	Trace	Action	Service mode	Service value
DO	Text	Check-box	Boolean: 0 1	Auto: - selected: disabled - not selected: enabled	1/0: - selected: 1 - not selected: 0

The **Action** is always the real value.

If you do not select the *Auto* check-box the output value equals the service value. In other words, the **Action** always displays the value applied to the (real) output.

Monitoring SWIP Object

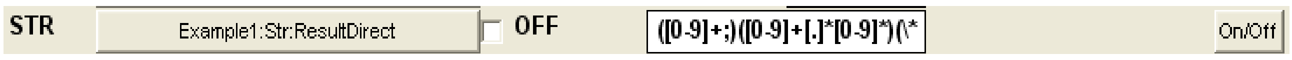

SWIP All_Swip1 OFF 2048 1000 1200

The **SWIP** (Switch-point) object line contains a button and the following fields:

Type	Name	Trace	Control Value	Value	Low limit	High limit
STR	Text on a button	Check-box	OFF LOW IN HIGH UNDEF	Input value: integer or real	Number: integer or real	Number: integer or real

You can overwrite the **limit** values.
The **Value** displays the value of the input controlled by the switch-point.
The **Name** button is used to restore the original limit values.

Monitoring STR Object

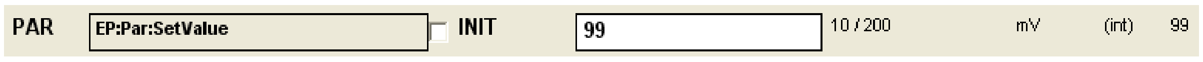

STR Example1:Str:ResultDirect OFF ([0-9]+;)([0-9]+[.]*[0-9]*)(* On/Off

The **STR** (String) object line contains a button and the following fields:

Type	Name	Trace	Control Value	Value	REG	On/Off
SWIP	Text on a button	Check-box	OFF ON MATCH NOMATCH DEF ERROR	Input string	String (Regular expression)	Button

You can overwrite the **REG** string.
The **Value** displays the string of the input controlled by the string object.
The **Name** button is used to restore the original regular expression string.
The **On/Off** button enable/disables the supervision: clicking on it you may force the **Control Value** to *OFF*.

Monitoring PAR Object


PAR EP:Par:SetValue INIT 99 10 / 200 mV (int) 99

The **PAR** (Parameter) object line contains the following fields:

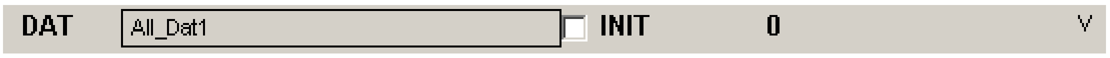
Type	Name	Trace	Control Value	Value	Low limit/ High limit	Unit	Format	Initial value
PAR	Text	Check-box	UNDEF DEF CHANGED INIT	Value	Value	String	bool char short float etc.	Value

The value type of **Value**, **Low limit/High limit** and **Initial value** is defined by **Format**. You can overwrite the parameter value. There is no way to automatically restore the original values if you forget them. There are two parameter types: PP and EP.

The EP parameter name is bold. The EP parameters are saved in a file .RTDB.par. Hence, after the restart the last parameters shown (set) in Monitor will be loaded. Therefore, the EP parameter line gets at the end an additional property column **Initial value** to show the value defined in the configuration.

The PP parameters are not saved. Hence, restarting the system will load the parameters set from the configuration file (PAR properties values in the Project Manager).

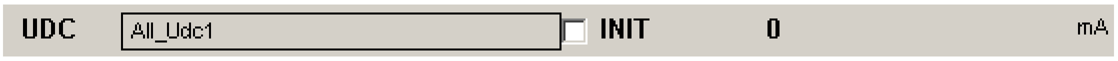
Monitoring DAT Object



The **DAT** (Data) object line contains the following fields:

Type	Name	Trace	Control Value	Value	Unit
DATA	Text	Check-box	DEF CHANGED UNDEF	Actual value	String

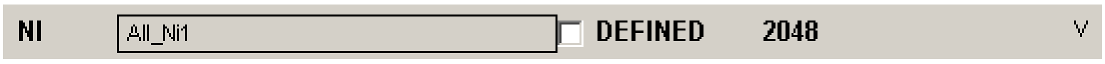
Monitoring UDC Object



The **UDC** (Up-Down Counter) object line contains the following fields:

Type	Name	Trace	Control Value	Value	Unit
UDC	Text	Check-box	UNDEF DEF CHANGED INIT	Actual count	String

Monitoring NI Object

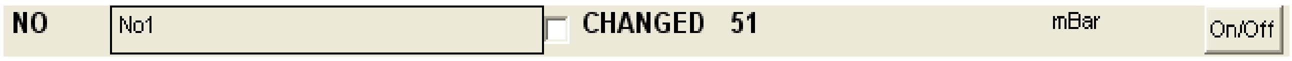

NI All_Ni1 DEFINED 2048 V

The **NI** (numerical input) object line contains the following fields:

Type	Name	Trace	Control Value	Value	Unit
NI	Text	Check-box	DEF CHANGED UNDEF	Actual numerical value	String

The type of the (Numerical) **Value** is defined in the configuration.

Monitoring NO Object


NO No1 CHANGED 51 mBar On/Off

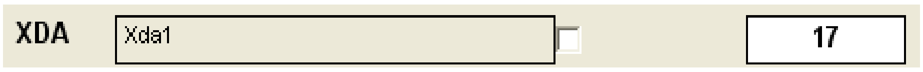
The **NO** (numerical output) object line contains the following fields:

Type	Name	Trace	Control Value	Value	Unit	OnOff
NO	Text	Check-box	OFF CHANGED INIT SET	Actual numerical value	String	Button

The type of the (Numerical) **Value** is defined in the configuration.

The button **OnOff** is a kind of Service Mode for numerical output. If **OnOff** is Off the NO output is determined by the VFSM. A click on the button **OnOff** triggers the object state to ON: in this state the NO output is at any time linked to the parameter that determines the NO value (corresponds to command On).

Monitoring XDA Object

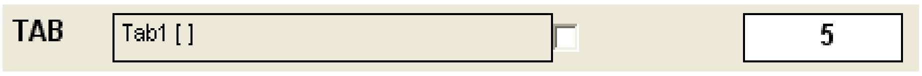


The **XDA** object line contains the following fields:

Type	Name	Trace	Control Value / Action
XDA	Text	Check-box	Integer

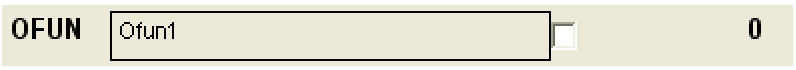
You can change temporarily the object **Control Value / Action**. After restart the original value defined in the configuration will be restored.

Monitoring TAB Object



The TAB object has similar fields as the XDA object, the last filed being an Action field only.
The **TAB** (demultiplexer output) displays the **Index** value.You can change the **Index** value. After resetting the system the **Index** has always the value 0.

Monitoring OFUN Object



The OFUN object has similar fields as the XDA object.
The **OFUN** (Output function) object line displays the Action (function parameter) or the Control Value (function return value).

Tracing

If the Trace check-box of an object is marked all changes of object Control Value or Action are written into a file Trace.txt. The Trace file is open when the Monitor is started. The Trace file can be closed at any time using the icon:

The Trace file can be opened at any time using the icon:

Tracing can be reset using the icon:

Monitoring RTDB using SWTerm

The StateWORKS Terminal (SWTerm) is used to debug the VFSM system of state machines, effectively to monitor the RTDB. Due to not having a GUI the SWTerm program is of course less convenient than SWMon but it has some additional features which make its use interesting.

The basic functionality of SWTerm is equivalent to that of SWMon, i.e. you can get property values of all RTDB objects and you can set the object values. You may also notify SWterm as a client of the RTDB which awaits events (Advise function) but the usage is not very useful, especially if we have many often-changing events – in such a case we prefer of course the usage of SWMon.

The interesting features of SWTerm are: logging and executing of command files. SWTerm logs all typed commands and answers. The command file can be either written by hand or it can be just the log file. The conversion of a log file into a command file requires only the change of the file extension from .log to .com.

The SWTerm program can be started with parameters which determine the usage of log and command files:

- -l -> Without this parameter the log file has a name SWTerm.log.
- -r -> Commands and responses will be logged, otherwise only commands.
- -c -> Take commands from the file CmdFileName, otherwise from the console.

You start SWTerm by:

Tools / SWTerm

It opens the command line window where you type in the commands. The reactions (answers) to the command are also displayed in the command line window.

You find a full description of the SWTerm in [4].

Monitoring RTDB using SWQuickPro

You start SWQuickPro by:

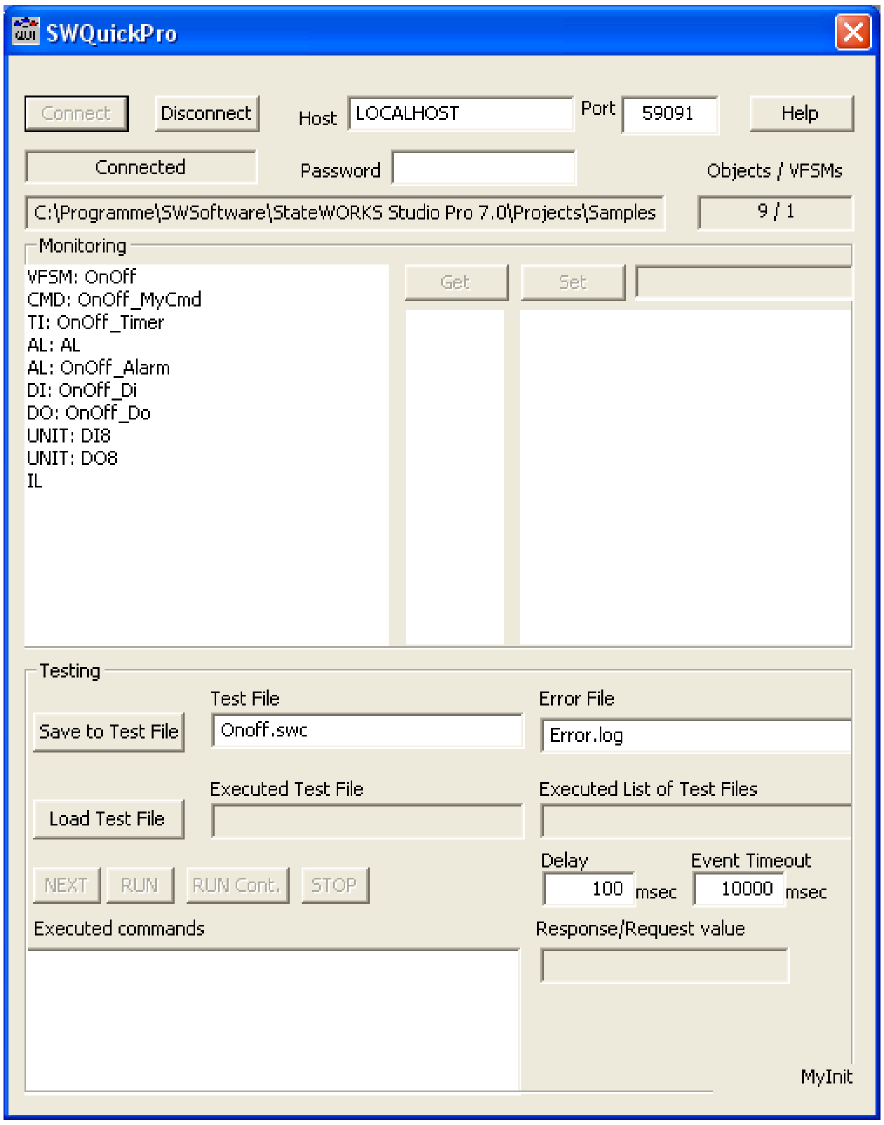
Tools / SWQuickPro

It opens the dialog based application. While connecting to SWLab ignore the password entry.

The StateWORKS SWQuick monitor is used to display all attributes of a single object at a time. If connected all RTDB objects are displayed in the window on the left.

On selecting an object (in the example the VFSM *OnOff*) all its attributes are displayed. A click on the button **Get** reads a selected attribute or all attributes if the *.all* pseudo attribute is selected. A click on the button **Set** writes the value entered in the edit box right to the button into the RTDB. Note that only some attributes may be changed; most of them are just read-only.

In addition to the “display” section SWQuickPro contains a “test” section that is used for automatic testing, i.e. for creating command files that can be used for repeated tests in a step-by-step or continuous mode. Details of the testing facilities are described in [9].



Host LOCALHOST Port 59091

Connected

Password

Objects / VFSMs 9 / 1

C:\Programme\SWSoftware\StateWORKS Studio Pro 7.0\Projects\Samples

Monitoring

VFSM: OnOff
CMD: OnOff_MyCmd
TI: OnOff_Timer
AL: AL
AL: OnOff_Alarm
DI: OnOff_Di
DO: OnOff_Do
UNIT: DI8
UNIT: DO8
IL

Testing

Test File: Onoff.swc

Error File: Error.log

Executed Test File

Executed List of Test Files

Delay 100 msec

Event Timeout 10000 msec

Executed commands

Response/Request value

MyInit

Figure 34: The StateWORKS SWQuick monitor

SPECIFYING VFSM

What will you learn now?

After the introduction we will specify a more complex VFSM. We want to discuss the most often used objects.

Example: Flow

The state machine controls a gas flow regulator. The Flow state machine accepts commands (numbers) and sets an analog signal Flow_Ao and a digital signal Flow_Do. As a feedback from the attenuator the state machine receives the actual gas flow values as an analog signal Flow_Ai and a digital signal Flow_Di indicating the regulator position (open/closed). Three commands determine regulator operations:

1: *Open* the gas flow by setting the *High* value of the digital output Flow_Do,
2: *Close* the gas flow by setting the *Low* value of the digital output Flow_Do,
3: *Regulate* the gas flow to the value determined by the numerical output Flow_Ao.

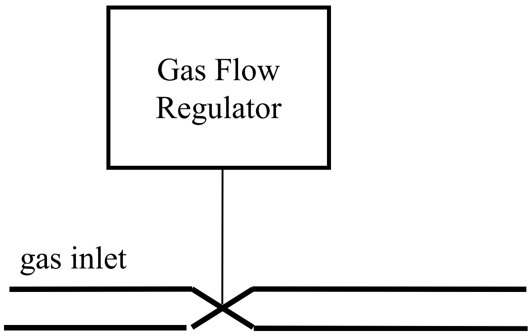


Figure 35 Gas Flow regulator

If the command *Open* has been carried out the state machine should check whether the gas valve has been opened. If the valve does not open after a certain time an alarm should be issued.

If the command *Regulate* has been carried out the state machine should check whether the gas flow has reached the required value. If the flow cannot reach the value in a certain time an alarm should be issued.

The actual gas flow is measured and its value delivered as an analog input signal Flow_Ai to the control system. Normally, it is required that the flow value stays within a certain limits. If the flow value exceeds the required range the state machine should after some delay issue an alarm and count this event. If it happens more than a certain number of times the state machine should issue immediately the alarm if the flow value exceeds the range.

Defining States

We start our design by drawing a state transition diagram. It does not contain all the details but it gives a better understanding of the control problem and it forms the basis of the full specification as expressed in the tables of the StateWORKS Studio. Figure 36 presents the state diagram of the Flow state machine. We do not say that this is our first ST diagram for the Flow state machine! Usually, it takes several steps before we generate the ST diagram that fulfils all of our requirements. In studying this example, we went through several stages; here we present you just the ultimate solution without the intermediate exercises.

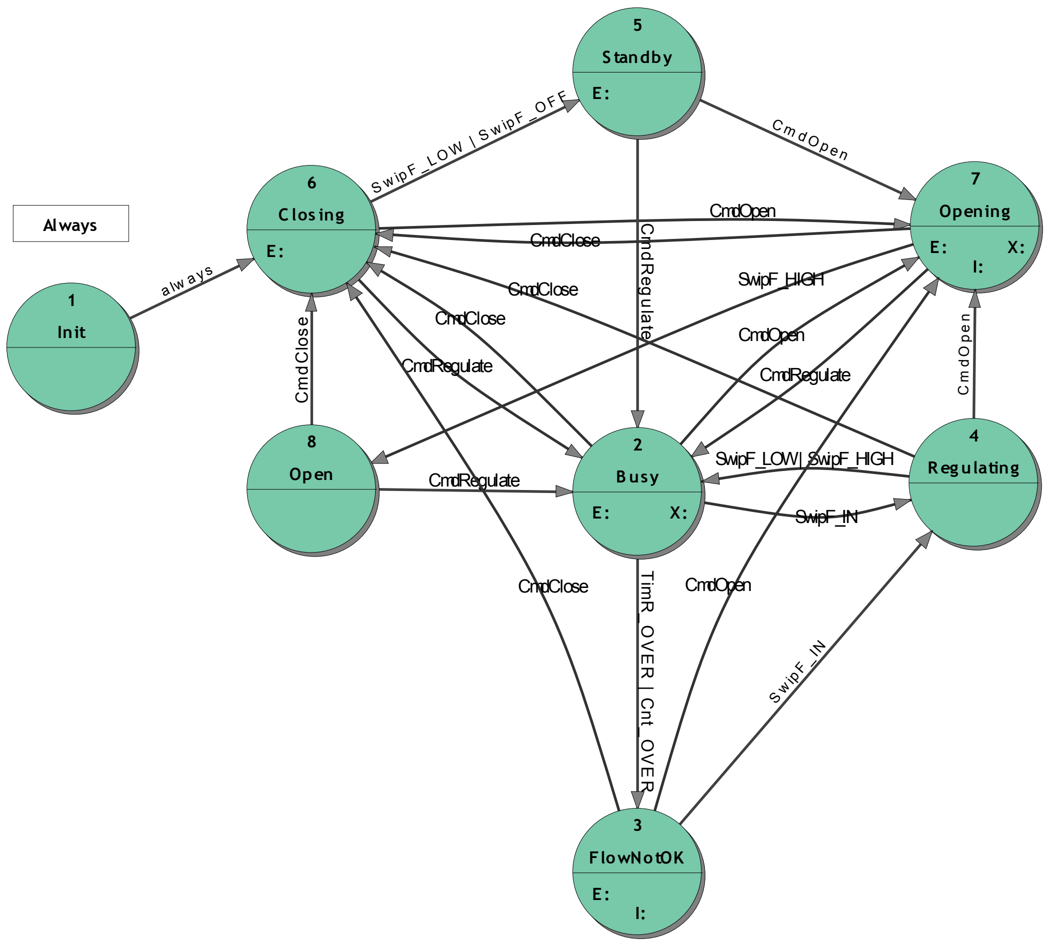


Figure 36 Flow: the ST diagram

We prefer to use the Moore model of the state machine. Therefore, the diagram shows that the states contain Entry Actions but no Input Actions. In addition, we use also a few Exit Actions.

We have decided to solve the problem using the following states: *Closing, Standby, Opening, Open, Busy, Regularing and FlowNotOK*. A detailed description of the states could be found in the documentation of the Flow state machine.

Note the difference in handling of the alarm situations in states: *Opening* and *Busy*. In the state *Opening* the *Flow_Do* output is set to *High* and the state machine checks whether the valve is open by testing the value of the *Flow_Ai*. If the flow does not reach a certain value within a given time an alarm is issued and the state machine stays in the state *Opening*.

In the state *Busy* the analog input *Flow_Ao* is set to a required value and the state machine waits until the value of the analog input *Flow_Ai* is within the required range. If the flow does not reach the range within a certain time the state machine changes to the state *FlowNotOK* where an alarm is issued.

These two different approaches are justified by the different significance of each failure. In the case of the state *Opening* the failure is less important and it is enough to alert the operator who must take an Action and solve the problem. In the state *Busy* the failure is serious and the system must take some Action by itself in addition to alerting the operator. The Flow state machine by itself cannot do more - the appropriate Action depends on the application and will be, normally, effected by another state machine that "controls" the Flow state machine. The state *FlowNotOK* in the state machine Flow is used to "send" the information about the flow error to another state machine. We will discuss this further when dealing with a system of VFSMs.

Defining I/O Objects

We specify I/O objects to define Virtual Input and Output Names for object Control Values and Actions. The MyCmd object is created automatically. Based on the description of the control problem we define the following I/O objects:

- timers: *TimB* used in the state *Busy* and *TimO* for the state *Opening*
- alarms: *AlaF* to signal the flow regulating error and *AlaO* to signal the flow opening error
- a digital output: *Do* to open the flow valve
- a numerical (analog) output: *Ao* to set the flow value
- a numerical (analog) input: *Ai* as an actual flow value
- a switchpoint: *SwipF* to supervise whether the flow value is within the required range
- a counter: *Cnt* to count how many times the flow exceeds the required range

Defining Input Names

For each I/O object we define names using its Control Value. So, we have defined three command names: *CmdClose, CmdOpen* and *CmdRegulate* on 2, 1 and 3 numbers correspondingly.

Similarly, we define names *TimB_OVER* and *TimO_OVER* for timers using only one Control Value *OVER* as we truly need only this condition for the specification.

We define input names for Control Values of the *SwipF* object: *Flow_HIGH, Flow_IN, Flow_LOW, NOT_Flow_IN* (note the use of a complement value) and *FlowControl_OFF*.

Eventually, we define a name for a Control Value OVER of the *Cnt* object: *Cnt_OVER*.

Defining Output Names

Definitions of Output Names are done similarly as in the previous example: for a given Action we invent a name that describes it. We have defined only names for Actions that are really needed by the table specification.

Setting an analog output

The specification of an analog (numerical) Action for the *Ao* object is the only new point. The available Output Values for the *Ao* object are: *Off, Set* and *On*. Choosing the Action *Off* we determine that the numerical output value will be 0. Choosing the Action *On* we determine that a certain value will be set to the numerical output (object *Ao*). The value is determined by a parameter that is linked with the *Ao* object. The parameter and its value will be determined at system configuration. Choosing the Action *Set* we determine that the output value will be set only once, just at this moment when the Action is being performed.

This technique is the simplest one for specifying an analog output. Later, we will learn two other techniques: using tables and writing a user-specific output function.

Filling state transition table

We begin the state machine specification by drawing the first ST diagram. In the next step, we analyze each state and complete the specification with all transitions and Actions that are necessary for the correct functioning of the state machine.

Additional information about the states can be found in the comment fields of the ST table.

Warning: The counter functions like a a timer. Hence, you have to Start or ResetStart it before you apply the Inc (increment) or Dec (Decrement) Action.

Configuring the system

The configuration of the Flow state machine contains several new objects that we have not used in the first example OnOff.

Also in this example we use the top-down approach. So, we begin with the creation of an instance of the state machine Flow that we name as Flow. Defining the properties of the state machine (i.e. determining the objects that belong to it) we choose names for all I/O objects in the system from *Flow_MyCmd* through *Flow_Swip_Range*.

Specifying object properties

After that, we start specifying the properties of all I/O objects. The properties of objects: Commands, Timers, Alarms, Digital input and output are specified as discussed in the first example. Now, we describe the properties of the object types that we have not met so far.

NO (Numerical Output) Properties

The properties of a NO object type include several values:

Format defines a value type: you can choose it from a list that contains typical formats, like integer, real, etc..

Unit is auxiliary information used for display: you can choose from a list of prepared strings like V, mA, etc. or you type any string you like.

Scale Mode decides how the output value will be transformed. There are two modes available: Lin(ear), Exp(onential).

Scale Factor is a coefficient used to multiply the nominal output value as taken from the Out Data.

Warning: The Scale Factor should not be 0.

Offset is a value added to the output value.

Out Data is a name of the Parameter object that supplies the output value. In the example, the output value is determined by the parameter *Flow_Par_FlowSetValue*.

NI (Numerical Input) Properties

Five properties of the NI object type are the same as the properties of the NO object type. They differ only in the last property. Instead of the **Out Data** the NI object type has a property:

Threshold value is a number: up to this value the input signal is ignored and treated as 0.

Warning: The Scale Factor should not be 0.

SWIP (Switch-point) Properties

Switch-point requires the definition of the **Input** and two limits. Switch-points can be set on **Input** objects that have numerical values (NI, DATA and PAR).

The **Limit Low** and **Limit High** values define the range of the input values supervised by the switch-point (see Figure 37). These limits can be defined directly by value or by a parameter (deactivating the switch **By Value**).

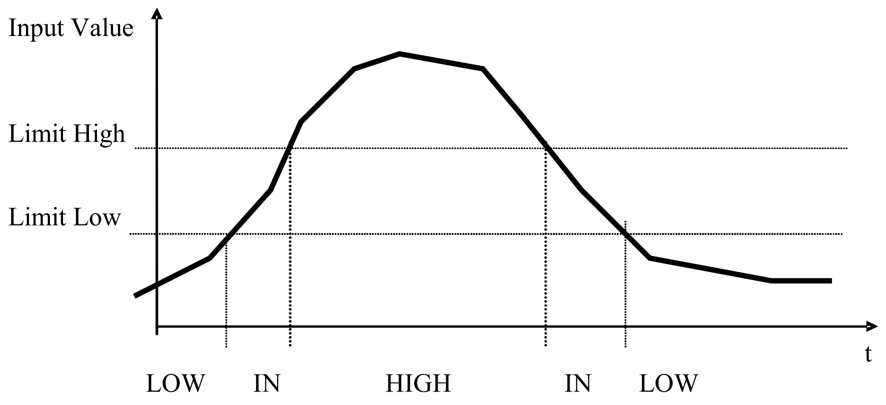


Figure 37 Switchpoint definition

For the Switchpoint *SwipF* we have defined the object *Flow_Ni_ActualFlowValue* as the input and two limit values: 200 and 240 as limits.

CNT (Counter) Properties

An object of a CNT type has only one property: the timeout **Const** value. Reaching the value of **Const** the Counter signals it with the **OVER** Control Value. The counter **Const** could be a parameter if you remove the mark **By value.**

For the *Flow_Cnt_FlowRangeExceeds* we have defined Const = 4.

Specifying I/O Units

The state machine Flow contains three object types that are true external objects, namely: DO, NI and NO. So, we have added the I/O Units to the project and defined which inputs and outputs correspond to the objects *Do8_Unit3, Ni4_Unit5* and *No4_Unit7*.

We repeat at this point that the units must have a predefined Physical Address property if you want to test the system with the StateWORKS Lab. The properties of these units are summarized in the Table below.

The Comm_Port is not used by the StateWORKS Lab, hence we just set it to 0.

Object type	**DI**	**DO**	**NI**	**NO**
Unit name	DI8_Unit1	DO8_Unit3	NI4_Unit5	NO4_Unit7
Physical address	1	3	5	7

MANAGING A SYSTEM OF VFSMS

What is a system of VFSMs?

It is a rather rare case to design a significant control system using only one state machine. If you need many cooperating state machines that will control a complex system you need an organizational framework that makes the job easier. The StateWORKS Project Manager offers you the framework to organize a complex control system.

We prefer and indeed recommend a hierarchical system of state machines [7] as demonstrated in the Gas example below. Of course, you are not limited to a hierarchical structure. If, for whatever reason, you need a system of freely connected VFSM you can do it using the StateWORKS Project Manager.

The crucial feature of a system of state machines is the way in which they communicate with each other. In the RTDB system this communication is done by and exchange of states and commands:

- States of a (Slave) state machine can be used as inputs to another (Master) state machine,
- The (Master) state machine can send commands to another (Slave) state machine.

There is a further possible exchange mechanism using other objects, especially XDA. This type of communication will be presented later as an advanced topic.

Example Gas

The example presents a control system to control a gas inlet of a vacuum chamber used by semiconductor manufacturers (Figure 38). The system contains a flow control and a pressure control. The flow regulator supplies a gas to the chamber and it is controlled by the Flow state machine discussed previously. The chamber is used for a manufacturing process that requires a certain vacuum in the chamber. The low pressure in the chamber is produced by vacuum pumps. Effectively, the vacuum in the chamber is determined by the pumps and the gas flow. The pressure is continuously monitored and if it exceeds the required range the process is interrupted and the gas flow must be discontinued.

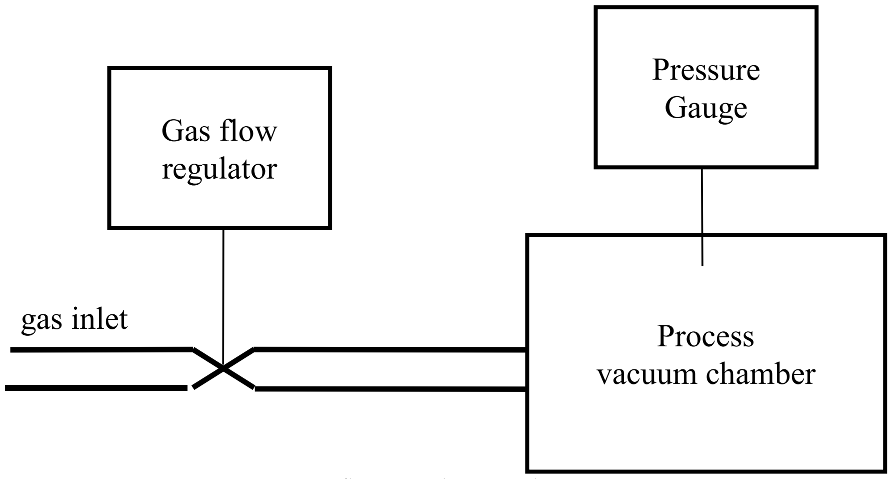


Figure 38 A gas flow regulator with a pressure gauge

The system as designed contains 3 state machines: the Flow state machine for gas flow control, the Press state machine to monitor the vacuum in the chamber and a Gas state machine that is a Master which coordinates the activities of the Flow and Press state machines.

The Flow state machine has been discussed in the previous chapter. The Press state machine can be found in the examples. We will present now the design details of the Master state machine "Gas" The functioning of the Master state machine can be explained in terms of the communication rules among state machines in the RTDB system.

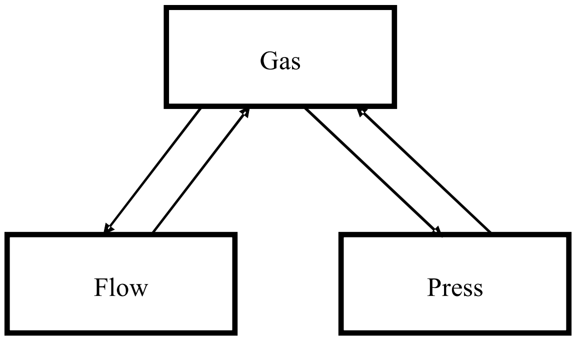


Figure 39 A Gas control system

Opening the project

A single VFSM that has no links to other state machines may be specified outside a project. You can create a project later and include the state machine into it. We handle the Flow state machine in such a way.

A Master VFSM must “know” the states of other state machines. The linkages among state machines are managed in a project. Therefore, before you start to specify a system of several state machines you have to create a project.

Create a project and add to it all state machines that already exist and that you intend to use in the system. Only the state machines that are in the project are “seen” when defining the specification of a Master state machine. At any time, during the development you can add additional state machines to the project.

You can remove a state machine from the project at any time.

Specifying the Master VFSM

The Gas state machine is a Master of two Slaves: Flow and Press. It "sees" the Slaves through their states and sends them commands. The basic behaviour of the state machine is described by the ST diagram in Figure 40.

The state machine reacts to two commands: 1(Off) and 2 (On). After initializing the state machine is in the state *Off*. Receiving the command *On* it changes to the state *OnBusy* sending to the Flow state machine the command *CmdRegulate*.

If the Flow state machine enters the state *Regularing* the Gas state machine changes to the state *On* sending the command *CmdEnable* to the state machine Press.

If the state machine Press goes to the state *PressOK* the Gas state machine goes to the state *OK*. Disturbances into the flow cause the Gas state machine to return to the *OnBusy* state sending the *CmdDisable* to the Press state machine.

Bad pressure signalled by the Press state machine must interrupt the process: the Gas state machine changes to the state *OffBusy* switching off the pressure monitoring (*CmdDisable* to the Press state machine) and cutting off the gas flow (*CmdClose* to the Flow state machine).

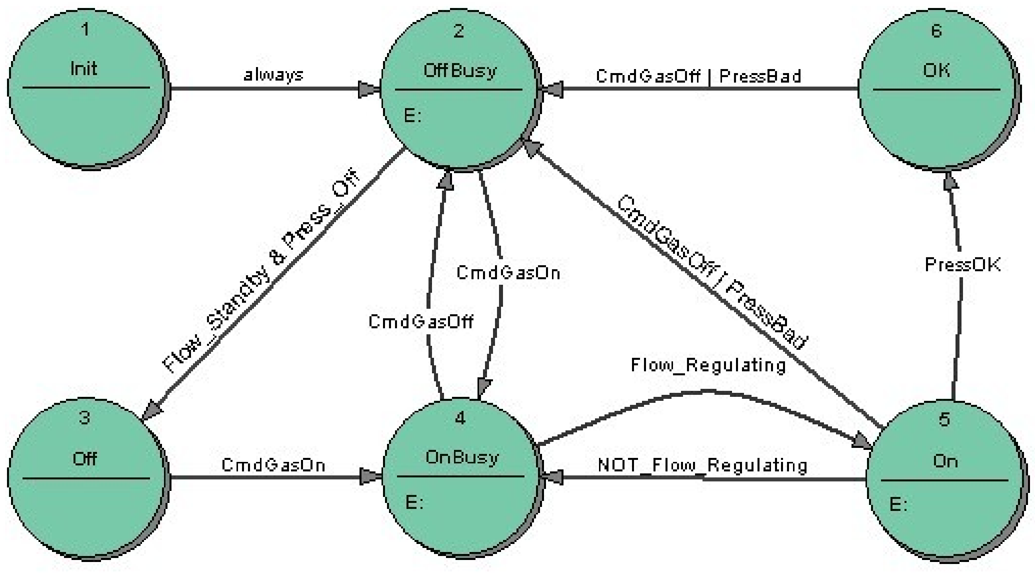


Figure 40 Gas: the ST diagram

Defining States

The Gas state machine has six states defined in the ST diagram: *Init, Off, OffBusy, On, OnBusy* and *OK*.

Defining I/O Objects

The Gas state machine has five objects: an input command that it receives (therefore CMD-IN), two state machines (VFSM) objects: Flow and Press and two output commands (therefore CMD_OUT) sent to the Slave state machines.

Defining Input Names

The two Gas command names (*CmdOff* and *CmdOn*) are obvious ones: they describe the input commands of the Gas state machine. The other input names describe conditions that cover certain state situations of Slaves.

The following text applies only for versions less than 7.0.
You may create complex state conditions in the form of AND-OR logical functions observing the following rules:

- Any number of VFSM states can be linked together with the **OR**-operator creating a complex state condition,
- Any number of state conditions for different state machines can be linked together by an **AND**-operator.

Thus, the name *Flow_Busy* covering two states of the Flow state machine means that the condition is TRUE if the Flow state machine is either in the state *Busy* **or** in the state *FlowNotOK*. You may be more specific and call this condition *Flow_BusyORFlow_NotOK*.
The *Flow_StandbyANDPress_Off* means that this condition is TRUE if the Flow state machine is in the state *Standby* **AND** the state machine Press is in the state *Off*.

Specifying state conditions

You specify the following names that described the situations of the Slave Flow: *Flow_Regulating*, *Flow_Standby* and *NOT_Flow_Regulating*. Note that the name *NOT_Flow_Regulating* is a complement of the state *Regulating* i.e. it covers all states different that the state *Regulating*. Similarly, you define the following names based on states of the state machine Pressure: *PressBad*, *PressOK* and *Press_Off*.
These names will be used for specifying the conditions for state transitions and Input Actions.

The following text apples only for versions les than 7.0.
To specify the name *Flow_Busy* you begin with the selection of Flow as **I/O Object ID**. In the **Input Value** window you get the list of Flow states. You choose the state *Busy*. Clicking on the **Add** button you see in the window **Name** the proposal for the input name: *Flow_Busy*. If you like it you click again on the **Add** button and the entry is copied into the **Input Name** list. Keeping the **Name** and **I/O Object ID** you choose another state in the **Input Value** window - *FlowNotOK* and copy the value into the **Input Name** list by clicking on the **Add** button. Now, under the name *Flow_Busy* you have two values - states *Busy* and *FlowNotOK*. These two values (states) are linked with the OR operator and will be interpreted as: if the state machine Flow is in one of the states - *Busy* or *FlowNotOK* the condition *Flow_Busy* will be TRUE.
To specify the name *Flow_StandbyANDPress_Off* you select first the **Name**, **I/O Object ID** and **Input Value** for Flow as in the previous case. Then, keeping the **Name** you choose the *Press* in the **I/O Object ID** window. In the **Input Value** window you get the list of Press states. You choose the state *Off* and copy the information into the **Input Name** list by clicking on the **Add** button. Now, under the name *Flow_StandbyANDPress_Off* you have one value - state *Standby* of the state machine Flow and one value -

state *Off* of the state machine Press. These two names are linked with the AND operator and will be interpreted as follows: if the state machine Flow is in the state *Standby* **and** the state machine Press is in the state *Off* the condition *Flow_StandbyANDPress_Off* will be TRUE.

You can define any AND-OR combination of states. For instance, the name xxxx defined as:

Name	I/O Object ID	Input Value
xxxx	VFSM1	*state1_1* *state1_2*
	VFSM2	*state2_1*
	VFSM3	*state3_1* *state3_2* *state3_3*

represents the following logical condition:

xxxx = (state1_1 OR state1_2) AND
state2_1 AND
(state3_1 OR state3_2 OR state3_3)

and means: the condition xxxx is TRUE if the state machine VFSM1 is in one of the states: *state1_1* or *state1_2* and the state machine VFSM2 is in the state *state2_1* and the state machine VFSM3 is in one of the states: *state3_1* or *state3_2* or *state3_3*.

Defining Output Names

The list of Output Names contains four items - they are two names that specify commands: *Close* and *Regulate* for the state machine Flow and two names that specify commands: *Disable* and *Enable* for the state machine Press. Note, that the Master state machine does not use all commands of the state machine Flow.

You define also the Output Name *MyCmd_Clear* that is then used in the state *OnBusy* as an Entry Action. That action is required to avoid a restart of the Gas state machine on returning the states: *Off* or *OnBusy*.

Configuring the system

Using procedure described for a single state machine you create all objects required by the state machines: Flow, Press and Gas.

Displaying the system

You may display the system of state machines by
View / Vfsm diagram / Show
or by clicking on the icon .

It opens a state machines system diagram (SMS diagram) which presents all state machines configured in the system. For the Gas example you get the display as in Figure 41.

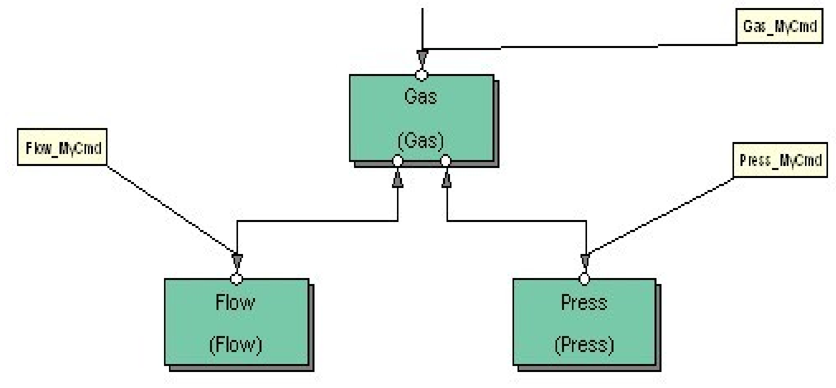


Figure 41 Gas: the SMS diagram

A state machine in this diagram is represented by a rectangle which contains the name of the state machine, for instance Gas and the type in parenthesis, for instance (Gas). The state machines are linked by lines with arrows, which correspond to the basic interface structure among VFSMs:

- An arrow at the top of the rectangle means a command coming from the Master.
- An arrow at the bottom of the rectangle means a Slave state used by the Master.
- A missing arrow means that the Slave does not take a command (at the top) or the Master does not use the state of a Slave (at the bottom).

By placing the cursor on the arrow you can display a tool-tip with the corresponding command or state. Those signals are also shown as labels which may be freely positioned or switched off.

The state machines system diagram (SMS diagram) shows the interface among state machines very clearly: commands sent from Master to Slave and state data sent from Slave to Master. This corresponds to a hierarchical structure of the system, which we recommend using in most situations.

This diagram has also some editing functions, such as:

- A double click on a state machine opens its Properties window.
- A SHIFT+ double click on a state machine opens its ST diagram.
- A single click on a state machine selects a state machine, and at that time also shows (in red) all the links to and from the selected state machine.

You may also select any link by clicking on it; this may help to identify the linked state machines in a complex system where several links overlap on the diagram.

One additional point needs to be made. The SMS diagram only shows the formal links made between the various state machines in the project, but there are some other, "back-door" methods of communication. For example, the use of a shared PAR object to hold some parameter. It is also possible to find that state machines are linked through the process, in that one machine could start a motor and another be influenced by the fact that it is running.

These links are not shown on the SMS diagram, for practical reasons. It is good design practice to restrict all interactions to the formal links shown on the SMS diagram, wherever possible, and the entire structure of state machines will, in the hands of an experienced designer, conform to this practice in all but the most unusual circumstances. Links via shared objects will then be used only to alter some parameters but, in general, not to produce transitions, and links via the process will be reduced by careful partitioning of the task into the various state machines. The designer and his colleagues will then be better able to master the project.

Specifying object properties

For the Flow state machine we just repeat what we have done in the previous example.

The Press state machine represents also no difficulties as it contains only objects already discussed.

The objects for the Gas state machine are the Slave state machines. They are represented in the object list by the VFSM states (Flow and Press) and commands sent to them (FlowCmd and PressCmd).

Having specified the state machines we have created also all other objects we need for the system. Definition of their properties is a repetition of already shown specification steps.

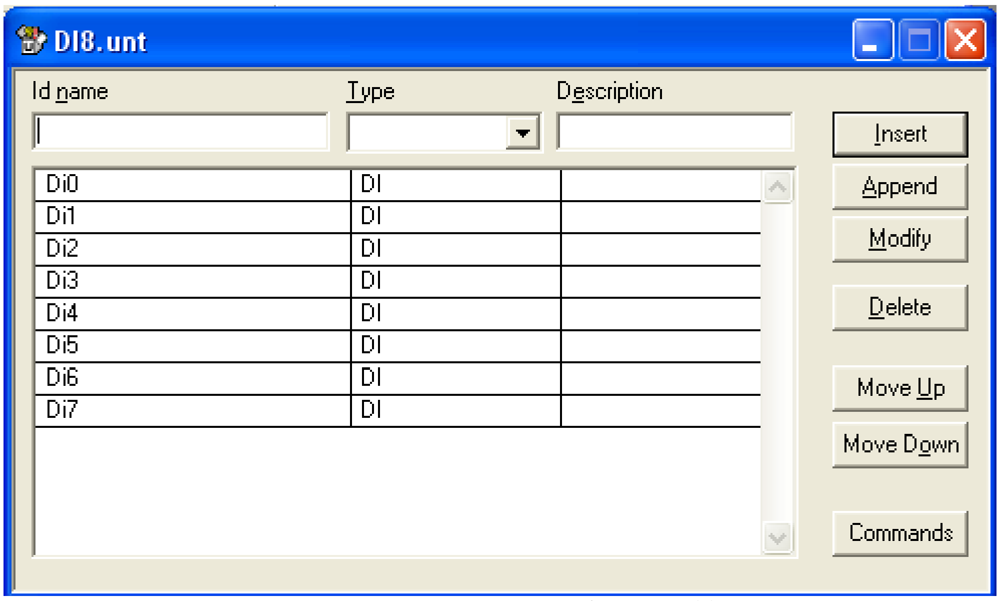
SPECIFYING UNITS

A Unit is a list of objects. Any object type can be used to define the Unit. Units are used by programmers to organize and program I/O drivers.

The most obvious Units are groups of digital inputs (DI) and outputs (DO). Similarly, systems that process numerical (analog) inputs and generate numerical (analog) outputs use groups of numerical inputs (NI) and outputs (NO) that usually represents digital values of numerical (analog) inputs/outputs.

To specify a Unit you have to create a file, in a similar way to specifying a VFSM:

File / New / Unit Document



Id name	Type	Description
Di0	DI	
Di1	DI	
Di2	DI	
Di3	DI	
Di4	DI	
Di5	DI	
Di6	DI	
Di7	DI	

Figure 42 A UNIT DI8 specification table

You are confronted with a table that is used to specify the Unit´s objects.

A specification of a Unit makes only sense if there exists a I/O-driver that can address and use the units. For instance, the StateWORKS Lab has drivers that simulate DI, DO, NI and NO Units as described in the previous sections.

Generally, the Units are application dependent. In the eventual run-time system you may use either Units that correspond to standard serial and parallel I/O´s or you can use your hardware specific Units, supported by I/O drivers written for your hardware.

PRINTING AND PUBLISHING

The ST diagram and the ST tables can be:

- copied into clipboard (Edit/Copy ST table/diagram as Metafile and Directories as RTF file)
- saved as wmf or jpg files (Edit/Save ST table/diagram as graphics and Edit/Save all ST tables as graphics)

In addition, by building the vfsm specification is stored as a XML file (VfsmName.xml). You may display this file in the browser using our tool (vfsmml.dtd and vfsmml.xsl).

LINKING TO OTHER APPLICATIONS

StateWORKS run-time systems

The RTDB (Real Time Data Base) is supplied as a library, and is used to build a StateWORKS run-time system RTDBApp. The application then takes the form of a single EXE file containing the real time date base, VFSM executor and I/O system. The user interface can be any program which is able to communicate with RTDB via TCP/IP. An example of such an application is the SWLab which is used for exercises in the Manual. It is a RTDB run-time system with simulated I/O.

TCP/IP Link

The StateWORKS execution environment supports the TCP/IP mechanism. The real time data base of the VFSM Executor serves as a Server in the TCP/IP Client/Server link.

The TCP/IP connection is realized in two phases. In the first phase, the TCP/IP link between two applications is established - a communication channel is created. After that the applications can exchange data using this communication channel.

Using monitors: SWMon, SWQuick or SWTerm you are shielded from the details of the TCP/IP interface. The monitors may run on any computer in the network in which the RTDB application is running. With a few commands like: *connect*, *disconnect*, *get*, *set*, *advise*, *unadvised* you can communicate with the RTDB application.

OTHER TOPICS

Using tables

Generation of analog output values

A pure logic system such as StateWORKS cannot process or directly create analog values. Analog values are delivered into the system as numerical values (NI objects) and are translated into Control Values using switch-points (SWIP objects). The Control Values of the SWIP object are then used for control purposes.

Similarly, the system produces any analog outputs as numerical values (NO objects). The system can set the numerical value to the output (NO object *On* or *Set*) or clear the output value (NO object *Off*).

There are several ways to determine the value of the numerical output (see Figure 43).

The numerical output is an output of a NO object. The NO object switches on/off the value to the output. The *On* command "closes" the switch between the value and the output. Thus, if for instance a parameter is the source of value any changes of the parameter will be transferred to the output. The command *Set* sets the output only once, transferring the present parameter value to the output (i.e. the switch between the value and the output "closes" only for a while to transfer the values to the output). If you want to repeat the transfer you have to repeat the command *Set*. The source of the value may be a table value, a parameter or a data value. The use of a parameter (PAR object) to define a numerical output has been shown in the Flow example. The use of data (DAT object) is similar and will not be discussed here in detail.

The TAB object as a source of the output value presents the most flexible solution. It allows the numerical value to have several values determined by the table entry. The table entries are parameters or DAT values.

The entire specification of the numerical output consists then of the following steps:
While specifying the VFSM:

- Define a TAB object in the I/O object Dictionary.
- Define output names (in the Output Name Dictionary) that determine the numerical output values using the TAB object entries (indexes).

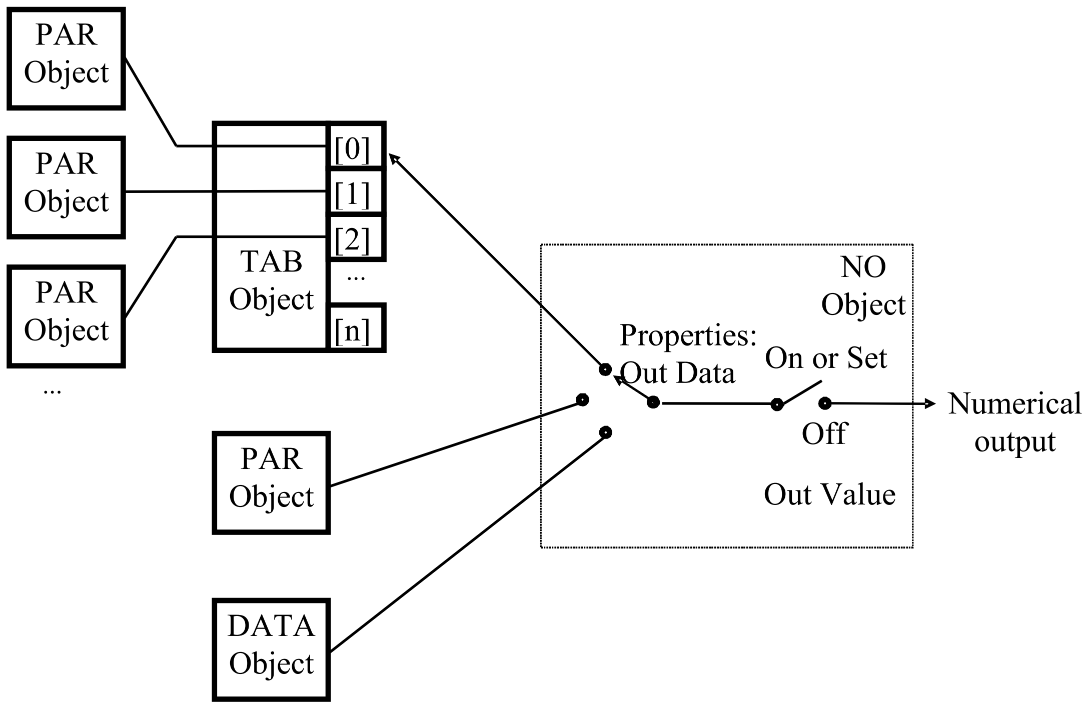


Figure 43 Determination of the numerical output value

While specifying the project:

- Define a set of parameter objects with required values of the numerical output.
- Define a TAB object that belong to the specified state machine.
- Insert PAR or DAT objects into the TAB object.

If you need a more sophisticated output control you can implement it by writing an output function.

Example: SetAo

The procedure as described in the previous section is illustrated by a SetAo example. It is a system with two digital inputs: SetAo_Di0 and SetAo_Di1 and an analog (numerical) output SetAo_No. The system is to realize a step function on the analog output - the value should be set step by step to four values, for instance: 0, 500, 1000 and 2000. The steps are triggered by setting High the input Di0. The input Di1 switches off the analog output, effectively setting it to 0.

The state machine SetAo has six states. Four of them are used to set (in Entry Actions) the value of the analog output according to the requirements. In the state Clear the analog output is switched off. Note the use of Clear Action to remove the input name IncAo by entering a state.

Start also the Monitor to see the state changes. Notice that from the Monitor you can change the output values of the steps by changing the parameter values.

We have put the SetAo state machine into the project *Other*. In this project we also have other state machines for examples that follow. The RTDB that is in fact the application may contain several state machines. The state machines may create an integrated system of state machines as in the Gas project or they are can be quite independent state machines, each realizing some specific functions.

Output functions

It may happen that some requirements cannot be fulfilled using standard objects offered to you by our system. Especially, tasks that require significant execution time should run in their own threads so that the VFSM executor is not blocked. These kinds of tasks should be moved to output functions that are activated by the executor. The output function object (OFUN) represents the programmer's interface.

To use an output function you have to specify an OFUN object in the I/O Object Dictionary. This object appears as an input and output object in Input/Output Name Dictionaries. In both cases the (input/output) value can be an integer.

Specifying an output name for an OFUN object means that this name will call the output function. The value for the output name is a parameter for the output function.

Specifying an input name for an OFUN object means that this name will be triggered by the output function if the return value of the function is equal to the input value.

The XDA object plays an important role in output functions as their private memory.

While specifying the project, in addition to the object name, you have to define the function name and the unit name. The unit name is the unit that is used when programming the output function as it contains the list of objects that the output function may access.

The details of programming interface and how to link output functions with the system can be found in the RTDB Programmer's Guide [3].

We show here only how to use an existing output function in the state machine specification.

Example: Stepper

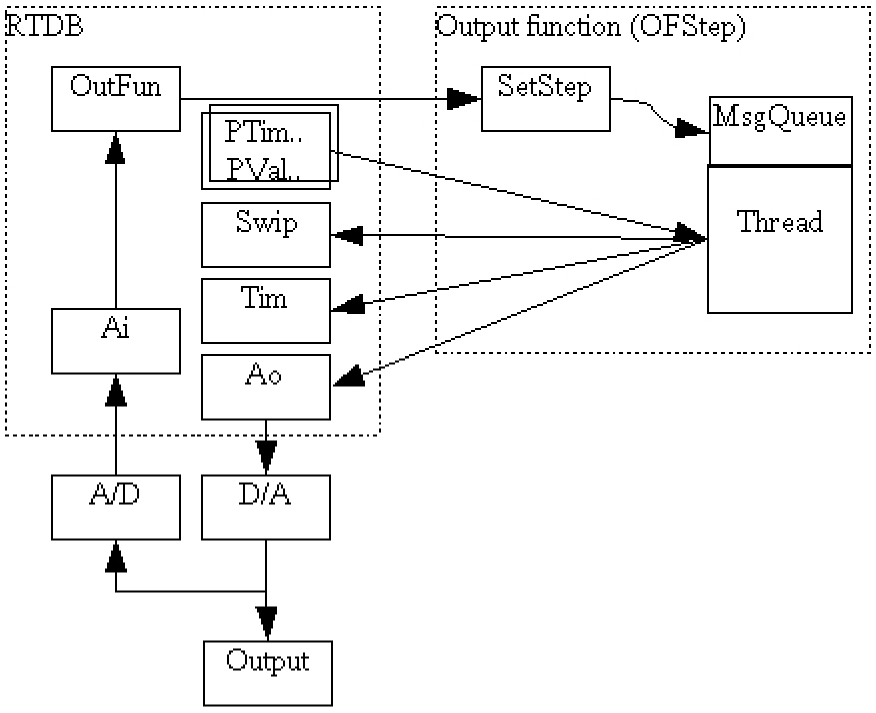


Figure 44 Stepper control

Stepper sets the output voltage, setting numerical output Ao and gets the feedback information about the truly set voltage via a numerical input Ai. Assume that the output voltage sets a digital to analog converter (D/A) which delivers the required voltage and the voltage is in turn sensed by an analog to digital converter (A/D), which supplies the numerical value Ai to the control system. Figure 44 shows objects involved in the Stepper control.

Stepper should generate several voltage steps, each for a defined duration. In the Off state the output voltage should have the value 0. The process starts by a command On. At any moment the stepping process can be interrupted by a command Off. If the output voltage does not reach the set value in the time prescribed for a given step the system waits until the required value is reached. The output voltage should have for instance the following form:

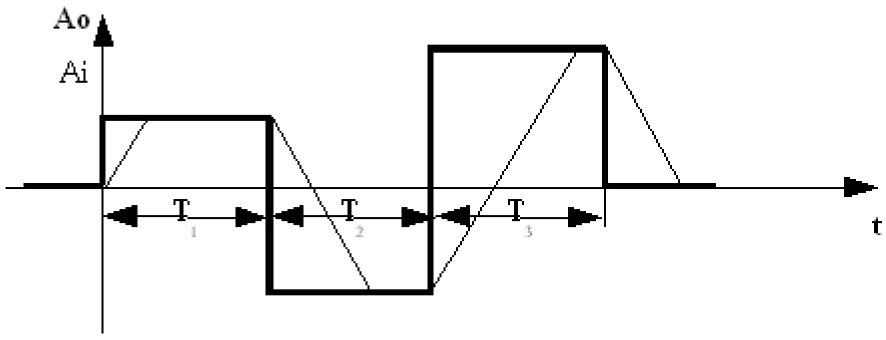


Figure 45 Stpper output voltage (Ao)

If the output value does not reach the -10 V in T_2 time the system prolongs the step 2 and waits until the set voltage is reached.

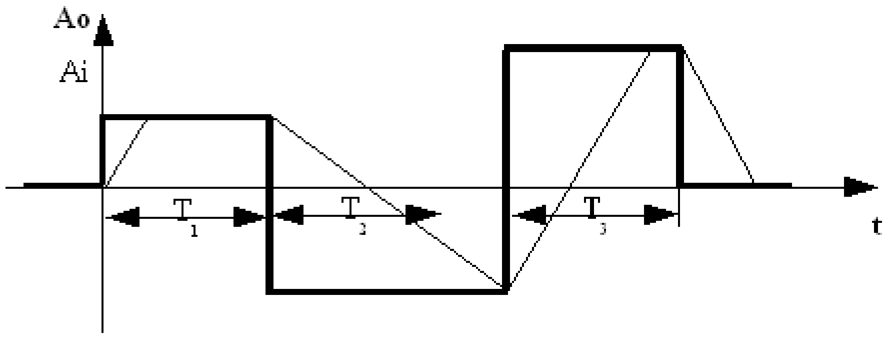


Figure 46 Stepper output voltage with delayed second step

We could realize the Stepper control using only RTDB objects. For the sake of this example we move some Actions to the output function which is going to perform the following tasks:

- set the timer (Ti),
- set the switch-point (Swip) value and enables it (command On)
- set the numerical output (Ao).

All timer timeouts and switchpoint limit values are defined by parameters.

The use of the output function for the Stepper has some (positive) side effects, namely we can use only one Timer and one Swip object independently of the number of required voltage steps. Instead, we shall use many parameters.

The ST diagram of the Stepper is as follows:

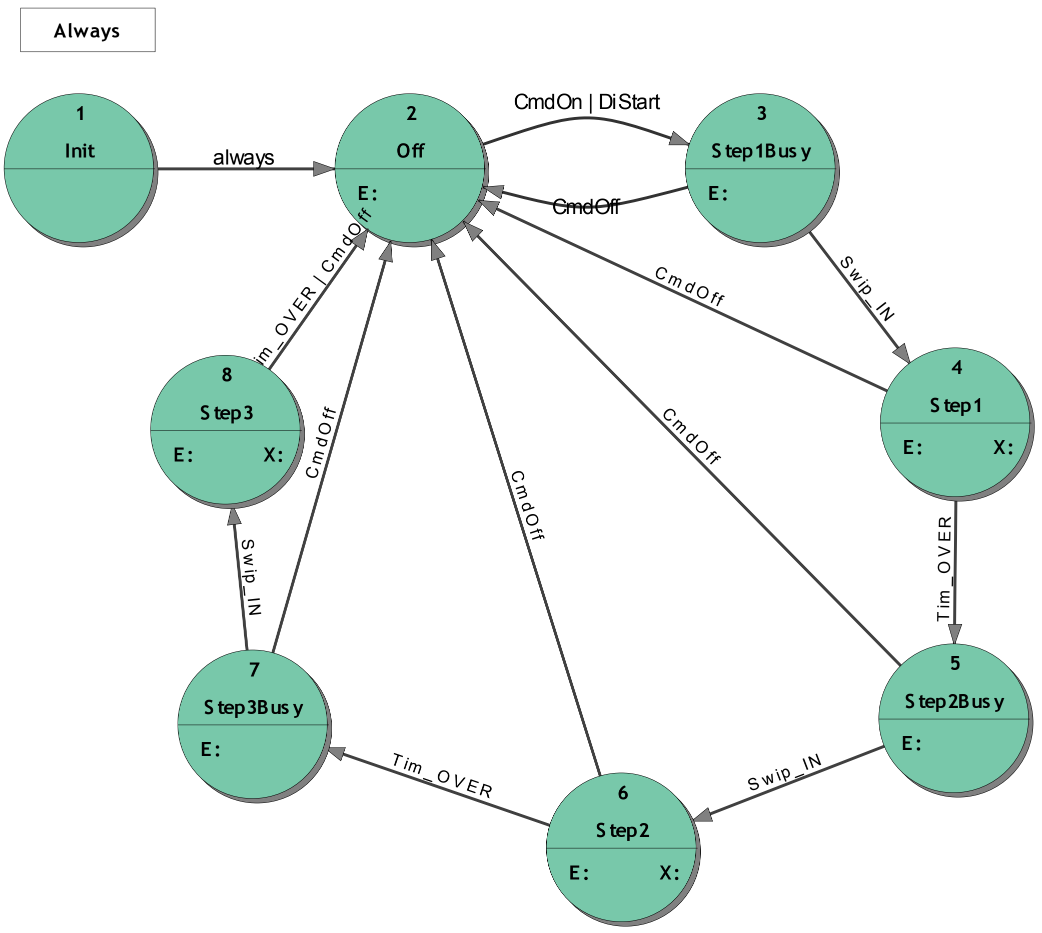


Figure 47 Stepper: the ST diagram

The Stepper waits in the Off state for the start. On receiving the command CmdOn or when the DiStart signal is active, Stepper goes to the state Step1Busy where it calls the output function (SetStep1) and starts the Timer. The output function does the above listed Actions and Stepper waits until the Swip object signals (Swip_IN) that the output voltage sensed by Ai has reached the required value. If this is the case Stepper goes to the state Step1 where it switches on a digital output Do1 and disables the switchpoint. The digital output just indicates that the voltage is ok. Stepper remains in the state Step1 until the Timer is OVER which causes the transition to the state StepBusy2. The transition condition Tim_OVER is AND-ed with the switchpoint state OFF to avoid slipping through state on the control value Swip_IN. When leaving the state Step1 Stepper switches off the digital output Do1, stops the Timer and switches off the Swip. Processing of the next steps is done exactly in the same manner.

If you analyze the list of IO objects defined for the Stepper state machine you notice that it contains several objects (XDA and PAR types) that are not used by the specification of the Stepper behaviour. The reason is that we require also a list of objects used in the output function. Instead of preparing a special Unit for the output function we included all objects in the IO object list for the Stepper state machine. The XDA object is there as a supplier of a “private” memory for the output function thread.

Implementing combinational systems

Sometimes, a state machine reduces to a pure combinational logic system i.e. a system that does not need states to “store” the history of input changes. You can use the StateWORKS Studio to specify such a state-less system.

A combinational system is specified in the Always table. Effectively, such an application contains one default state Init that always exists in the table. Therefore, you can specify the combinational system also in the mandatory Init state as its Input Actions. In such a case the other fields: Entry, Exit, Clear and Transition are meaningless and are not used.

For each output two logical conditions must be specified: the On-function that sets the High (TRUE) value of output and the OFF-function that clears the output (Low). Note that a true don’t care situation can be achieved here. For instance, specifying logical condition for an output Y (A, B, C) as:

$A B' + B'C$	Y
$A'C' + B C'$	Y'

you effectively decide that the condition B=1 and C=1 is ignored. If both B and C are simultaneously TRUE the output Y will not change (remains High or Low).

Example Combi

The Appendix contains a more complex example of a combinational system: COMBI. This system has three outputs: X, Y, Z and six inputs: A, B, C, D, E and F.

The logical functions are realized as fully defined functions (without any don’t care input conditions). For instance, the Karnaugh table for the output X looks like this:

		ABC							
		000	001	011	010	110	111	101	100
DEF	000	0	0	0	0	0	0	0	0
	001	0	0	0	0	0	0	0	0
	011	0	0	0	0	0	0	0	0
	010	0	0	0	0	0	0	0	0
	110	0	0	0	0	0	0	0	0
	111	0	0	0	0	0	0	0	0
	101	0	0	1	1	1	1	1	1
	100	0	0	1	1	1	1	1	1

X

Multi-valued objects

The “natural” variables often used for control are the Boolean ones. The VFSM concept adds to the Boolean variables multi-valued control variables. In fact, in the VFSM concept the Boolean variable does not play any special role: it is just a Control Value as any other multi-valued variable.

Input values

All input and input/output objects (DI, TI, CNT, ECNT, SWIP, STR, UDC, NI, DAT, PAR, OFUN) “generate” a few Control Values. For instance, a timer (TI) has the following Control Values: RESET, STOP, RUN, OVER, OVERSTOP. A state machine (VFSM object) has several Control Values which are its states.

Another problem exists if we have to “translate” a number of values into Control Values. In certain situations we call the values “commands”. In other situations the values may represent a movement direction, working mode or some physical value: temperature, pressure, etc.

We may try the “traditional” way of representing a value as number of base 2 using DI objects which are to store Boolean values. It will work at least for integer values. This solution is not very convenient as the decoding of the values is relatively complex. The easier way is to use object types which can represent multi-valued variables: CMD, XDA, NI.

A CMD [7] object can be used either as an input or as an output object. Normally, a CMD object is an output object in one state machine (Master) and the same object is then an input object in another state machine (Slave). The CMD object used as an input object (CMD-IN) can be used also to generate an output name – for details see [8]. The CMD object is a positive integer: 1, 2, 3, … and its various values are given names that are used as input names in the Slave state machine and as output names in the Master state machine. Thus, these names representing the numbers are the Control Values of CMD objects.

An XDA object can be used as an input or as an output object even in the same machine, the only restriction being: a XDA-input name and a XDA-output name must not be used in the same state. The XDA object has several functions. It is used to:

- supply a memory for output functions (see “Example: Stepper”, p. 75)
- pass information between RTDB and I/O units
- represent a multi-valued variable
- exchange information among state machine (extending CMD objects)
- storing information in a state machine

A typical usage of an XDA object is to convert the contents of a message to Control Values: the message is analyzed in an I/O unit and the result is coded into several XDA values which are then used to define input names for a state machine. The same works for the other direction: the XDA values are used to define output names that define some Actions in the I/O unit, or some messages.

The usage of XDA to store information in a state machine should not be overused as it is nothing other than a kind of flag to store some information to be used in the future. In a properly designed state machine there should be no need to use this kind of flag.

Similarly, XDA objects should not replace commands in the communication among state machines. Commands are the proper interface between two state machines. Any additional exchange of control information makes the system more complex and error prone. The use of XDA for state machine communication will generally mean that the state machines are not well designed.

The NI object itself is of no direct relevance in control. It can only store a value of any software type (int, float, bool, char, etc.) but it has no Control Value. Therefore the NI object does not appear in the IO Object ID list of the Input Name Dictionary and cannot be used directly to define an Input Name. The Control Values of an NI value can be generated using a SWIP object: see "SWIP (Switch-point) Properties", p. 57.

Output objects

Similarly to inputs, the RTDB has not only a digital output object (DO) which stores and outputs a Boolean value but also a few objects to supply multi-valued outputs: NO, TAB, AL. The usage of the TAB object has been discussed in "Generation of analog output values", p 73. AL objects are used for a very specific function - to generate an alarm.

The only object which is actually used to supply a multi-valued output is the NO object but it does it in an indirect way. The NO object itself has only 3 output values: Off, Set, On (see "Generation of analog output values", p. 73) which are used to pass the true output value which can be of type PAR, DAT, NI.

In-Out objects

There is a group of objects, which are both: input and output objects: TI, CNT, ECNT, UDC, OFUN, XDA, SWIP, STR. This feature of these objects allows triggering a change of its internal state using an Action, which in turn leads to a change of a Control Value which is based on its state. For instance, starting a timer with an output *ResetStart* (effectively it is a timer input) results some time later when the timeout elapses in an *OVER* timer state which may be used as an input Control Value.

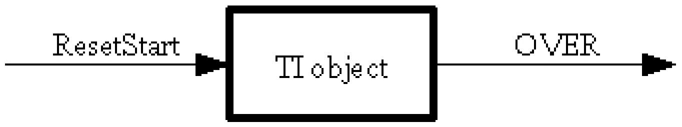


Figure 48 Timer object

The counter objects: CNT and ECNT have the same features as a TI object. The fourth counter UDC is different from the CNT object. The UDC object is an event counter which in addition can be triggered by commands: *Clear*, *Down* and *Up* but does not generate an *OVER* state. Its state can be detected and translated into Control Values using a SWIP object, as for NI objects.

OFUN object can be used only with an output function. The OFUN output is a parameter passed to the output function defining the task to be done by the output function. The return value of the output function is the OFUN Control Value.

The usage of the XDA object has been discussed in the previous section.

The usage of a SWIP object has been discussed by describing the translation of multi-valued inputs into Control Values of an object (NI, PAR, DAT, UDC) supervised by SWIP. The output to a SWIP object (*Off*, *On*) does not influence the Control Value of the supervised object; it simply enables/disables SWIP functioning. The Control Values evaluated by the SWIP object reflect the Control Value of the supervised object.

The STR object is used for string analysis. Similarly to the SWIP object, a STR object "supervises" an input object that can be a DAT or PAR with string format. A regular expression, which defines the string used as a matching pattern, can be hard coded or a content of a DAT or PAR object. The output to a STR object (*Off*, *On, Set*) does not influence the Control Value of the supervised object; it simply enables/disables / acknowledges STR functioning. The Control Values evaluated by the STR object reflect the Control Value of the supervised object. In addition, if matched the result is stored (and automatically converted into appropriate format) into DAT; PAR, NI or NO objects (the result may contain several strings).

Principally, the CMD object is also an object that is used as an input and an output but as we mentioned in the previous section it cannot be used in these two functions in the same state machine. The object CMD serves as an interface between two state machines.

Using VFSM-Templates

What is a Template?

A complex control system contains several cooperating state machines. They cooperate by exchanging state and commands. As a rule, any state machine does not need full information about its cooperating counterparts. A state machine is interested only in a few (stable) states of its partners and can only send a few commands to them.

When developing the specification of a (Master) state machine that coordinates several other state machines we would like to have a freedom to alter the Slave state machines. A development and maintenance process requires steady modification of the state machines. If the state machines are linked together, any changes in a Slave state machine resulting in introduction or removing of states and commands may require changes in the Master state machine.

A Template is a skeleton of a state machine containing definitions of its states and commands. The Template contains only those states and commands that are interesting for and used by other cooperating state machines.

When creating a new state machine we decide whether the state machine is Generic or based on a Template. If the state machine is Template-based the Template can be considered as an interface among state machines. Other state machines that communicate with a Template based state machine "see" only the Template.

As a Template contains only a subset of states and commands it may be used as a common basis for several similar state machines. This makes easy to design a Master of such state machines. Also changes of a Template-based state machine do not have any influence on the Master if the Template does not change.

Example: Template

We will illustrate the concept of a Template by a more complex example of the previously-discussed Gas control system. This time the system contains five state machines: the Master Gas, the pressure control Press and three gas flow control Flow. We use the previous VFSM Press. We replace the Flow VFSM by a Template based version. If you compare the specifications of the Generic and Template based Flow state machines you will not notice any differences. The only difference is that using the Template based Flow VFSM you define the Template TFlow as a Slave of the Gas VFSM.

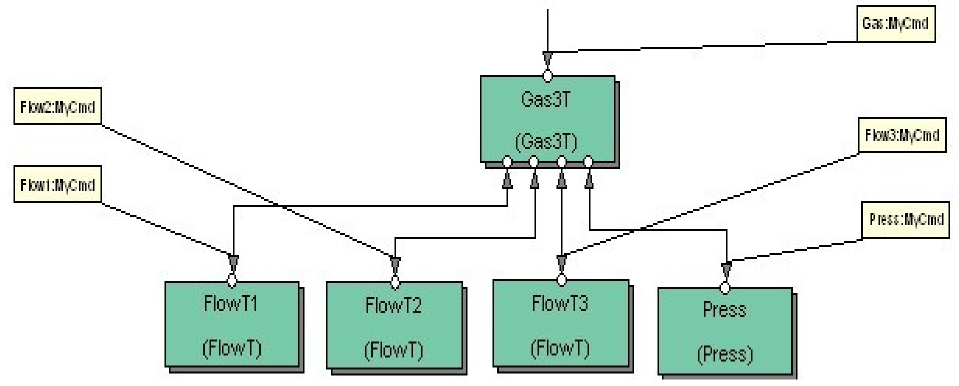


Figure 49 Gas with 3 Flow Slaves: the SMS diagram

Specifying a Template

We specify a Template for the Flow state machine opening the Template:

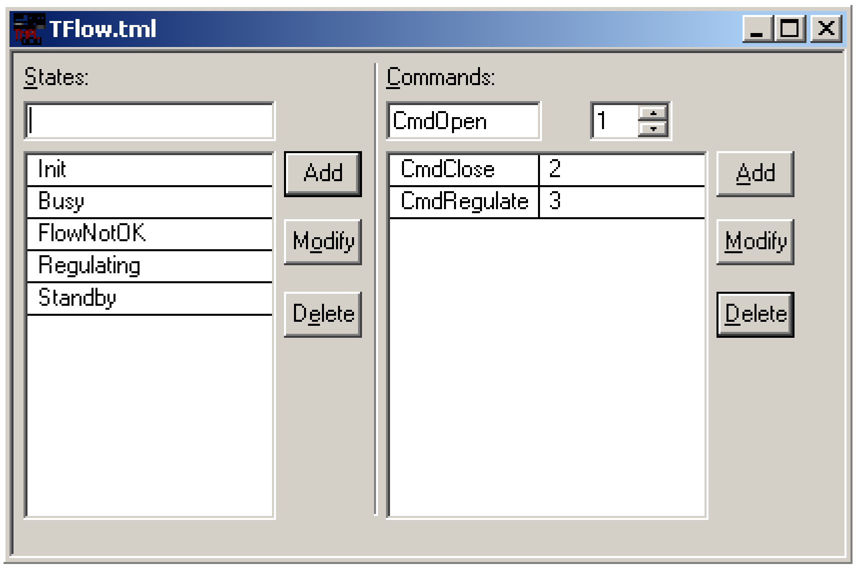
File / New / Template Document

The Template window contains two tables. The state table is initialized with the obligatory *Init* state. For the Gas example we have defined: *Busy, Flow NotOK, Regulating and Standby* states. Defining commands we have to specify names and command values. For the Gas example we have used: *CmdClose* (2) and *CmdRegulate* (3).

To be used, the Template must be **Build** like a normal VFSM and added to the project:

Project / Edit

In the **List Files of Type** window you can select the **TMPL** type files and **Add** the displayed Templates to the Project.



TFlow.tml

States:

Init
Busy
FlowNotOK
Regulating
Standby

Commands:

CmdOpen 1

CmdClose	2
CmdRegulate	3

Figure 50 Flow: the template window

Generating a Template from a VFSM

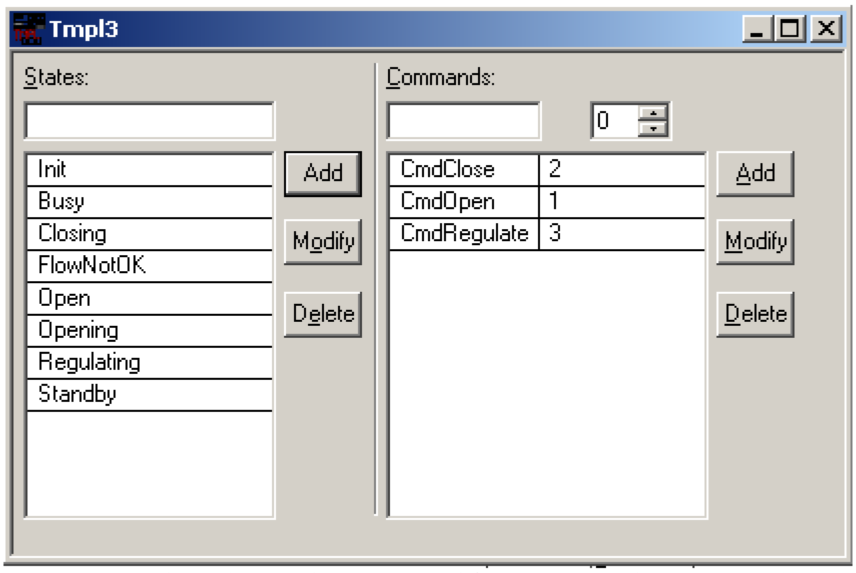
Alternatively, we may take the already existing Flow state machine and produce a Template:

File / Generate Template

A Template is then generated with all Flow states and commands.

By deleting the states (Closing, Open, Opening) and commands (CmdOpen), which are not used by the Master (Gas) state machine we get the required Template.

This command not only allows you to generate Templates at a later stage in the project from the state machines designed already, but also to change the basis (Generic or Template) of the VFSM (see the next section).



States:
Init
Busy
Closing
FlowNotOK
Open
Opening
Regulating
Standby

Commands:	
CmdClose	2
CmdOpen	1
CmdRegulate	3

Figure 51 Flow: the template generated from existing Flow state machine

Changing a VFSM Template

A state machine (VFSM) must be connected with a corresponding Template. It is done by definition of the VFSM base and connecting it to the Template. The basis (Generic or Template) of a VFSM can be changed at any moment. Using:

File / Change Template

you can either change a Generic VFSM to a Template based machine or vice-versa: a Template based VFSM to a Generic one. When changing a Generic VFSM to a Template based one you can use Templates that are part of the project (Project/Edit/Add).

Connecting VFSM to Template

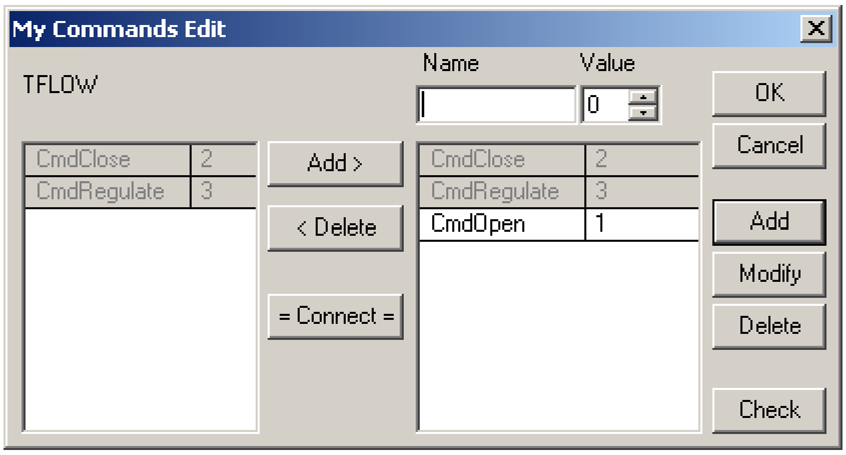
A Template based state machine is connected to its Template in the following window:

Dictionary / Input / My Cmd ...

You have to connect each command separately, selecting them on both sides: Template and VFSM.

The changes to the VFSM basis sometimes require a manual test and adaptation of VFSM commands: First of all, you have to **Add** Template commands to VFSM commands. Then you are free to introduce other commands that you need for the state machine.

Remember that the Master will use only the Template commands.



TFLOW

Name	Value
	0

CmdClose	2
CmdRegulate	3

CmdClose	2
CmdRegulate	3
CmdOpen	1

Figure 52 My command window used to connect VFSM to Template

A Template is a way of defining the interface between Master and Slave state machines. When specifying the system objects in a Project we have to create true state machines and not Templates. Therefore, in a Project we use state machines that are connected to corresponding Templates.

Application Notes

The number of topics related to StateWORKS Studio is very large. A User's Guide cannot exhaust all of them. Searching for other themes you may consult the Technical Notes on our web site (www.stateworks.com).

VFSMS DOCUMENTATION

The files available in the Project/Samples directory in the installation package contain full documentation of the examples discussed in the manual:

- OnOff - Standard on/off control
- Press - Pressure control
- Flow - Flow control
- Gas - Master flow control (with generic state machines)
- Gast - Master flow control (with template based state machines)
- SetAo - Setting analog output
- Combi - Combinational system
- Stepper – Using output function

For your convenience, the state transitions printout of the first example, *OnOff* has been appended to the printed manual.

OnOff example

Object ID dictionary

Name	Object Type	Description
1 MyCmd	CMD-IN	
2 Timer	TI	
3 Alarm	AL	
4 Di	DI	
5 Do	DO	

Input name dictionary

Name	I/O Object ID	Input Value	Init
1 always			+
2 CmdOff	MyCmd	1	-
3 CmdOn	MyCmd	2	-
4 Timer_OVER	Timer	OVER	-
5 DeviceOff	Di	LOW	-
6 DeviceOn	Di	HIGH	-

Output name dictionary

Name	I/O Object ID	Output Value
1 Timer_ResetStart	Timer	ResetStart
2 Timer_Stop	Timer	Stop
3 Alarm_Coming	Alarm	Coming
4 Alarm_Going	Alarm	Going
5 SwitchOff	Do	Low
6 SwitchOn	Do	High

State name dictionary

Name
1 Init
2 Off
3 OffBusy
4 On
5 OnBusy

OnOff

OnOff is a simple state machine (vfsm), which demonstrates basic control principle.
It controls switching of a DO output which is triggered by commands:On and Off.
The output changes have to be acknowledged by a feedback signal in a form of a DI input.
When switching On the feedback is supervised by a timer and in case of failure an alarm is generated.

Always		

This state is a done state.
Nothing happens when entering the state.
The vfsm waits for a command and reacts only to a command Off.

On	E:	
	C:	
	X:	
▨		
OffBusy	CmdOff	▨

On entering this state the Do is set Low and the vfsm waits for an acknowledgement from Di.
If the acknowledgement comes the vfsm goes to state Off.
If the command On comes before the acknowledgement the vfsm changes to state OnBusy.

OffBusy	E:	SwitchOff
	C:	
	X:	
▨		
Off	DeviceOff	▨
OnBusy	CmdOn	▨

This state is a done state.
Nothing happens when entering the state.
The vfsm waits for a command and reacts only to a command On.

Off	E:	
	C:	
	X:	

▨		
OnBusy	CmdOn	▨

All Vfsms starts from this state.
By choosing a suitable next state you decide what action will be carried out when starting the system.

Init	E:	
	C:	
	X:	

▨		
OnBusy	CmdOn | DeviceOn	▨
OffBusy	CmdOff | DeviceOff	▨

On entering this state the state machine sets Do high and starts the Timer.
If Di acknowledges that the controlled device has been switched on the vfsm goes to state On.
If the command Off comes before the acknowledgment the vfsm switches to state OffBusy.
If Timer expires an Alarm is generated.

OnBusy	E:	SwitchOn Timer_ResetStart
	C:	
	X:	Timer_Stop

▨	Timer_OVER	Alarm_Coming
▨	DeviceOn | CmdOff	Alarm_Going
On	DeviceOn	▨
OffBusy	CmdOff	▨

REFERENCES

[1] Wagner F., al.: Modeling Software with Finite State Machines: A Practical Approach. Auerbach Publications. New York, May 2006.
[2] Wagner F.: The Virtual Finite State Machine: Executable Control Flow Specification. Rosa Fischer-Löw Verlag, 1994.
[3] SW Software. StateWORKS RTDB Programmer's Guide. Release 5.0.6. 2004.
[4] SW Software. SWTerm.pdf. 2003.
[5] TN1-Virtual Environment.pdf
[6] TN2-What Is StateWORKS.pdf
[7] TN3-Hierarchical system.pdf
[8] TN4-Commands.pdf
[9] TN20-Testing with StateWORKS.pdf
[10] TN22-Complement control values in VFSM concept.pdf
[11] StateWORKS. Specifying state machine – Tutorial. 2005.
[12] StateWORKS. Specifying system of state machines – Tutorial. 2005.
[13] StateWORKS. Specifying RTDB – Tutorial. 2005.
See also other Technical Note on www.stateworks.com.

Index

SW Studio Quick Reference

ST table and diagram

Abbreviations used:
State transition table = ST table
State Transition diagram = ST diagram
State Machine System diagram = SMS diagram

Operation	**In the ST table**		**In the ST diagram**		**Remarks**
	Short cut	**What to do**	**Short cut**	**What to do**	
Open ST diagram	CTRL/O	Command File/Open	📂		
Select state		Move cursor to a state field		Click on the state in the ST diagram.	
Select transition		Move cursor to a state field or a transition condition field		Click on the transition arrow in the ST diagram.	Double click on the transition arrow opens the ST table, with the cursor in the transition condition field
Add new state		Commands: Edit/Insert expression Edit/Append expression	▤	Click on any point in the STdiagram, It opens the state name dictionary. After defining the state name and OK the state appears on the ST diagram and the ST table opens.	
Delete state		Command Edit/Delete expression	☒	Select state. Command State/Delete. Delete or Back key	A deleted state stays in the State Name Dictionary and may be still used. Ultimate deletion is done in the State Name Dictionary.
Add new transition		Commands: Edit/Insert expression Edit/Append expression		Click the right mouse over the state in the ST diagram and drag the appearing arrow to another state. The ST table opens to specify the state transition condition. The transition is appended after the last existing transition.	Click the right mouse over the state to create a transition to the same state. Details can be filled in later if you prefer. Note order of transitions defines priority.
Delete transition		Command Edit/Delete expression		Select transition. Delete or Back key	

Open ST table		Already open in this mode.		Double click on state in the diagram	CTRL/double click on state opens a new ST table window.
Open next ST transition table				CTRL + double click on state in the diagram	Up to 4 ST tables can be open at the same time
Display state information		Entire information is contained in the displayed ST table.		Position the cursor over the state name, transition arc, action symbol (E:, X:, I:)	
Move state				Click on the state and drag it to the new position	
Scroll ST diagram				Use arrow keys or a mouse scrolling wheel	
Scroll ST diagram on the print page				Use SHIFT+ arrow keys or SHIFT + a mouse scrolling wheel	Use Print setup to display the page boundaries.
Zoom ST diagram			🔍 to fit the window size	Use "+" / "-" keys of the numerical pad or CTRL+ mouse scrolling wheel	
Default size of the ST diagram				SPACE	

SMS diagram

Operation	In the SMS diagram		Remarks
	Short cut	What to do	
Open SMS diagram	⊞		The icon is visible only for a selected Project window
Open Properties window of a state machine or unit		Double click on the state machine or unit in the SMS diagram.	
Open ST diagram		SHIFT+ double click on the state machine in the diagram.	
Move state machine		Select the state machine by a click on it and drag it to the new position.	If ▦ (View / SMS diagram / Grid settings) is selected the new position is on a grid.
Display state machine information		Position the cursor over the state machine.	
Display command		Position the cursor over the arrow entering the state machine on top.	This is a command sent from a Master to a Slave state machine.
Display state		Position the cursor over the arrow entering the state machine on bottom.	This is a state of a Slave state machine.
Select a link		Click on the link	The color of the link changes to red.
Display all state machine links		Select the state machine by a click on it	The color of all links reaching the state machine changes to red.
Scroll SMS diagram		Use arrow keys or a mouse scrolling wheel.	
Scroll SMS diagram on the print page		Use SHIFT+ arrow keys.	Use Print setup to display the page boundaries.
Zoom SMS diagram	SPACE to return to default size and position	Use “+” / “-“ keys of the numerical pad or CTRL+ mouse scrolling wheel	