

Getting Started with StateWORKS Studio

Table of Contents

GETTING STARTED WITH STATEWORKS STUDIO.....1

1. OVERVIEW.....3

1.1. WHAT IS STATEWORKS ?.....3

1.2. STATEWORKS STUDIO PROFESSIONAL.....4

1.3. STATEWORKS STUDIO BASIC.....4

1.4. STATEWORKS STUDIO LE.....4

1.5. RUN-TIME SYSTEMS.....5

1.6. ABOUT THIS DOCUMENT.....5

1.7. INSTALLATION.....5

2. STARTING UP.....6

2.1. STARTING STATEWORKS STUDIO.....6

2.2. THE VFSM EDITOR.....7

3. CREATE A NEW PROJECT.....7

3.1. THE COMBINATION LOCK PROJECT.....7

3.2. NEW PROJECT: A NEW FINITE STATE MACHINE.....8

3.3. SET UP VIRTUAL I/O FOR THE STATE MACHINE.....9

3.4. COMPLETE THE STATE TRANSITION DIAGRAM.....10

3.5. THE PROJECT WINDOW.....13

3.6. IMPROVE SECURITY OF THE LOCK.....15

3.7. LOCKING.....17

3.8. SAVE AND CHECK THE DESIGN.....17

3.9. A NOTE ABOUT THIS EXAMPLE.....17

4. TESTING.....17

4.1. START SW LAB.....17

4.2. START SWMON.....18

4.3. USE OF SWQUICKPRO.....19

5. RESULTS.....21

5.1. FILES GENERATED: DOCUMENTATION.....21

5.2. SPECIAL FILES GENERATED.....21

5.3. XML IMPORT/EXPORT.....21

6. SYSTEMS OF MULTIPLE VFSMS.....21

6.1. THE PRINCIPLES.....21

6.2. USING THE SMS DIAGRAM.....22

7. RUN-TIME SYSTEMS.....22

8. A FINAL REMARK:.....22

1. Overview.

1.1. What is StateWORKS ?

StateWORKS is a set of tools for producing reliable software. The basis is the use of finite state machines to express the behaviour of a system. These are created by using a special editor, which operates on State Transition diagrams, in a graphical form, and State Transition tables, in a text format, holding all the details. These two different expressions of the same design are closely synchronised throughout the process.

StateWORKS includes powerful tools and techniques for handling complex systems using many state machines at the same time. Normally, state machine designs start to get difficult when all sorts of possible error-handling measures have to be added to a design which would be fine when everything worked around it. This can cause an increase in the number of states, if not handled correctly. StateWORKS makes it easy to split a design into a number of state machines, normally arranged in a hierarchy, and to implement the resulting system. In fact this is one of its strongest points. Most programmers can easily handle the programming of a single FSM, but it can be tricky to deal with a large number of them running in a big system.

The finite state model used by StateWORKS is a pure one, from which input-output processing and numerical computations are kept out of the state machines, which run in what we call a “Virtual Environment”. They are called “Virtual Finite State machines” for this reason, and you will encounter the abbreviation “VFSM” in several parts of the documentation. A VFSM is only aware of significant instantaneous aspects, or states, of its inputs (including lower-level VFSMs in a hierarchy).

Of course, real physical input-outputs have to be connected, and for this a set of routines is provided, linked through a Real Time Data Base to the VFSM structure. This also deals with interconnections between VFSMs and with use of system resources such as timers. We believe that it is vital to separate the behavioural aspects of software from the detailed computations it carries out.

The designer constructs formal models of his system, using the StateWORKS tools such as VFSM Editor. These models are so designed as to be able to be directly executed by the StateWORKS Executor: a sort of virtual machine running in the system. Coding of the intricacies of behaviour of software can in this way be avoided. Neither huge amounts of “if...then...else...”, “switch” and other control statements, nor more formally arranged code expressing finite state machines, need be generated. An important aspect is that the models are “Platform Independent Models” which are exactly the same under test in the I.D.E. and as loaded into the run-time system. There will, of course, be a need to write code in the conventional way for user interfaces, data analysis, special drivers etc. and this code is easily linked to the VFSM structure.

With StateWORKS come tools for testing designs, by manipulating the inputs and monitoring the results. Using these tools, test may be carried out to include all the obscure error paths which could be difficult to handle. This testing is easier to perform and to understand than conventional testing of code, on account of the way in which the environment can be manipulated in subtle fashion.

The complete StateWORKS package goes under the name “StateWORKS Studio” and comes with all the run-time system support software. It is available for various operating system environments, including Microsoft® Windows® NT, 2000 and XP, and various styles of UNIX™ and Linux. Versions are also available for smaller real-time operating systems, such as VxWORKS and O.S.E.

y using StateWORKS, it is possible to make huge improvements in productivity. Typically, a project requiring 5 programmers over 18 months could be done with 3 programmers over 15 months, and with much less after-delivery maintenance to be done. It has even been known for an expert to carry out a 6-month project within 3 weeks! One hears a lot about “code re-use” these days, but it is not always easy to do, as code is hard to fully understand. StateWORKS offers the possibility of “component re-use” on a large scale, as the VFSM models are well documented and easy to understand, by comparison with code of similar complexity. The full product is very complete, of course, and takes some time to fully understand: just the RTDB Class Library Manual has over 160 pages but many users will not need to refer to it.

1.2.StateWORKS Studio Professional

This is the standard and very complete system, with the I.D.E. for design and the Standard Executor for the target system. The Standard Executor includes the real-time data base, and a small A.P.I. library plus source code examples are included to make it easy to connect to other software.

In its normal off-the-shelf version it runs under Microsoft Windows NT, 2000, XP, XP embedded or CE but versions for Linux or other Unix variants can be supplied. And of course variants for other real-time systems, even disc-less, are available on request.

1.3.StateWORKS Studio Basic

This is similar to Pro above but has a limited scope – design and run-time systems are limited to 300 “objects”. It is suitable for small systems, and is sold at a much lower price.

1.4.StateWORKS Studio LE.

For evaluation purposes, for educational use, and for the design of very small systems in which a single VFSM is sufficient, a low cost version of StateWORKS Studio is available. This does include the Standard Executor for Microsoft Windows.

The main restriction is that the “Limited Edition” or LE version can only handle a single finite state machine (but it can display systems of VFSM if provided from an other source): it does not deal with the “SMS Diagram”. Of course, the clever designer might find a way to use Studio LE to design a system with several VFSMs, by inventing means of inter-connecting them. If you wish to do that: good luck! But try to follow the general guidelines we give for use of the full products, so that you can eventually upgrade and save a lot of hassle.

StateWORKS Studio LE is available for download, with a 30-day usage limit. The time limit is only removeable by purchasing our book: *Modelling Software with Finite State Machines*. Each copy of the book contains a unique registration code.

For practical reasons, StateWORKS Studio LE do not have specific documentation and Help files: the differences will be quite obvious when you use this product.

1.5.Run-time Systems.

We recommend use of the complete StateWORKS Executor products for general use in run-time systems, and it is possible in many cases to use the Standard Executor, supplied with StateWORKS Studio for this purpose. You may remember that we mentioned above that the models produced by StateWORKS Studio can be executed directly: this possibility is used for testing a design. In fact the design under test is working exactly in the same way as the run-time system in a target computer, using a special version of the StateWORKS RTDB.

It is also possible to work with our VFSM Run-Time Library: also provided free of charge. This may be suitable for smaller systems where the RTDB is considered too big to be installed easily, or for systems where the user wishes to be in closer control of execution.

The various tools communicate by means of a TCP/IP interface. This could be used, for example, to permit a Windows 2000® computer to supervise a system running in a Linux computer. It could also be applied to situations in which an “Industrial PC” connects to its input-output units over a network, using TCP/IP. Ask your distributor for more information on this topic.

1.6.About this Document

We need to point out that the example you are requested to work though in this “Getting Started Guide” is only meant to help you to familiarize yourself with the development tools. It is not a very good illustration of the benefits of StateWORKS, as the input can be a stream of numbers, the transition conditions are very simple, and it only involves a single finite state machine. So in fact the traditional state machine design and programming methods described in various text books could be used for this project, which creates a sort of “deterministic” state machine as encountered in parsing text.

The way in which the “virtual environment” needs to be specified when designing a state machine, and then the real environment defined in the project editor might seem clumsy to the newcomer, but is a vital feature of StateWORKS as it permits the use of many instances of a state machine to be used in one or many projects, with suitable configuration in each case. It is one of the features which gives StateWORKS its immense power to simplify large and apparently complex projects, and we encourage you to learn how to use it effectively.

1.7.Installation.

The installer will have allowed various options for the directories. Normally, all the tools and associated help files are placed in the C:\Program_Files\SW

Software\StateWORKS Studio LE folder. Various project files are located in a Projects folder, and each project has its own directory. Note that VFSM designs are easily re-used, so these are always stored in the projects\VFSM folder, rather than with the project(s) using them. On opening a project, you will be asked for the paths to the VFSM and UNIT folders holding files associated with it, if the project can not locate them.

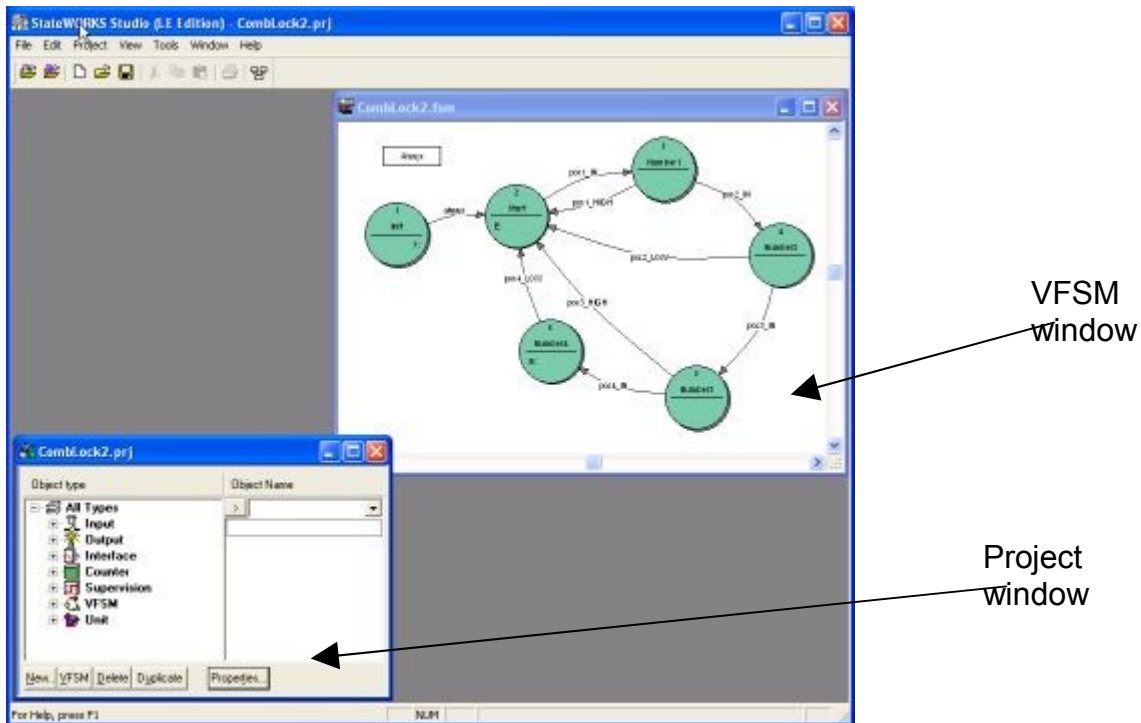
The installer should have set up the configuration data correctly for the “Tools” such as SWLab, SWMon and SWQuickPro. Each tool needs to know the path for finding it, optionally a home directory, and for SWLab and SWQuickPro the current project is normally selected, by setting the Argument as \$Project.

Although the software can be used with a screen resolution of 800 x 600, we strongly advise a large screen with a much higher resolution: 1280 x 1024 for example.

2. Starting up.

2.1. Starting StateWORKS Studio.

Open the StateWORKS Studio editor, by selecting through the *Start -> Programs* menus, or by using a short cut if installed, or double-click on “SWStudio.exe” in the appropriate directory.



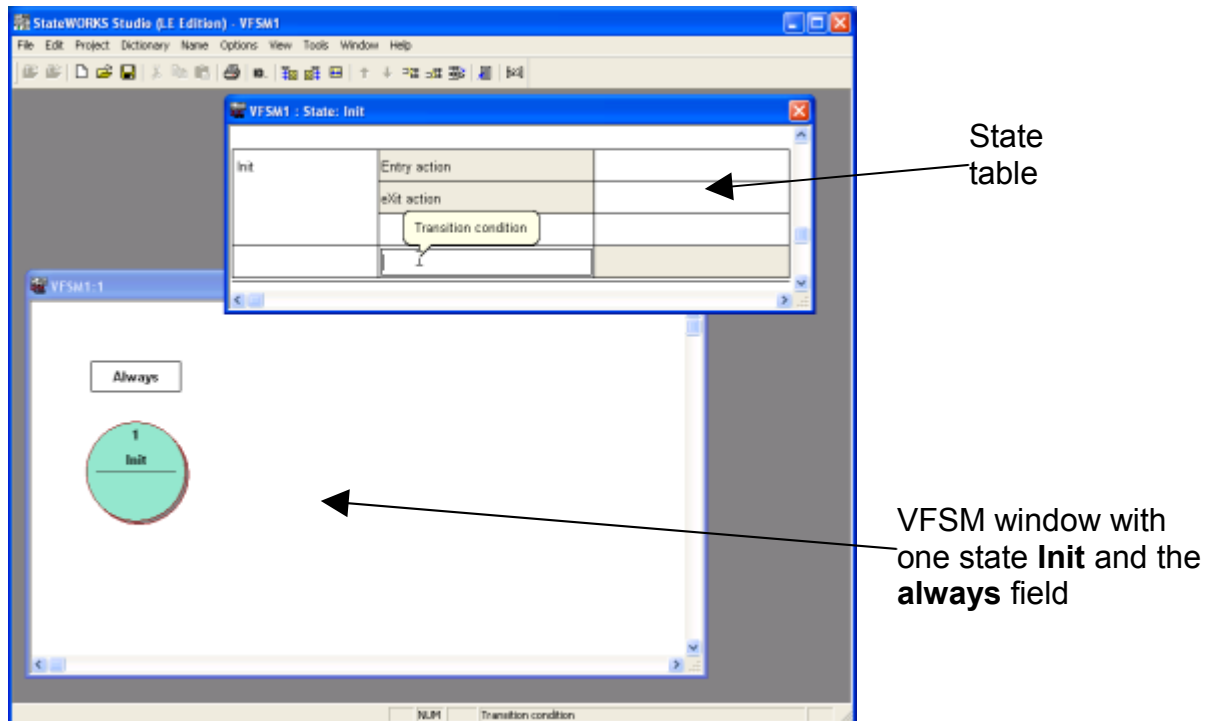
The window will open, with a project called “CombLock” (unless you have already been working on a different project.). We suggest that you should close the project with *Project -> Close*, and start from the beginning in order to learn. Close also the VFSM window if you had opened it.

2.2. The VFSM Editor.

To start a new VFSM design, use *File -> New -> VFSM File -> Generic*. This will open a new window marked VFSM1. The title can be changed if, later, you use *Save*. Notice that the menu items at the top of the Studio Editor window change, according to whether you have selected the Project window or the State Diagram, which we shall refer to as the ST diagram from now on.

This window has not too much at first. The box labelled “Always” is there for technical reasons, but you will find it convenient to use it so as to add your comments about this design. A double-click on this box opens a test window, with a Comment field at the top. Write something such as “My first test.” and then close the window. (The Always box in fact allows for input events to sponsor “Input Actions” not related to any state, which is occasionally useful. For example a VFSM could be a degenerate case with just a logical expression inside it. Don’t worry about this for the moment!).

The diagram has one State shown, as a disc. This “Init” state is the obligatory state where any VFSM must start. Double-click on it, and there will be a table opened in a new window (state table). If you explore this, any time you click on a section of this table, an explanation of its purpose will be displayed. The “Init” state cannot be removed but it can be renamed.



3. Create a new project

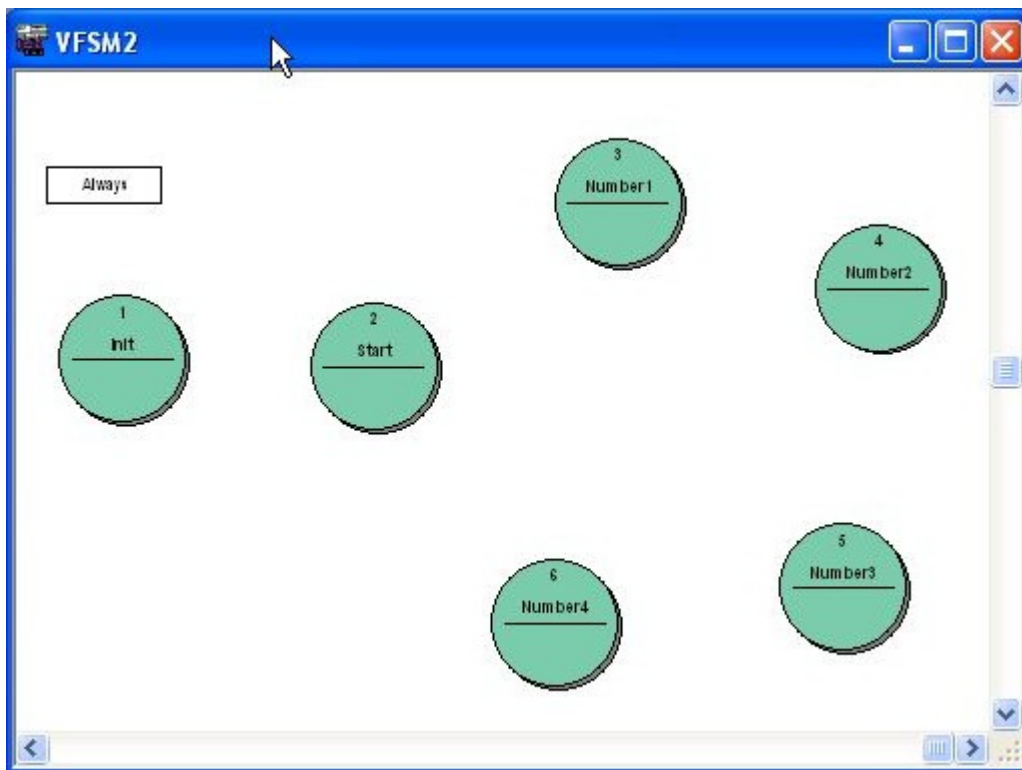
3.1. The Combination Lock Project

Now for a short tutorial: a combination lock has been chosen. This is a very simple design, but it illustrates several important points about StateWORKS.

The combination lock is an electronic version of a popular design of lock for safes and filing cabinets. Its workings were very amusingly described by Richard Feynman, in his auto-biography “Surely You Are Joking, Mr. Feynman?” and he learnt about them while working at Los Alamos on the Manhattan project. A single knob, with a scale 0 – 100, has to be turned to a sequence of four positions, going in alternate directions, and the door can then be opened. For this example, we assume that the knob has first to be turned “upwards” towards the first number. As Feynman discovered, there are some tolerances so that ± 2 steps from the correct position may be acceptable.

3.2. New project: a new *Finite State machine*.

To create a new project use *File -> New -> Project* menu. Enter the project name “Comb_Lock2” and ensure the project folder will be created in the proper directory. We shall not use the project window for a while, as we shall design the lock as an abstract model, before we link it to the real-world environment. This is an important feature of StateWORKS: state machines designed in this way can be taken and used in various quite different projects, as components, so we need to bear in mind that there are always the two views; the abstract VFSM design and its concrete instances in the various projects.



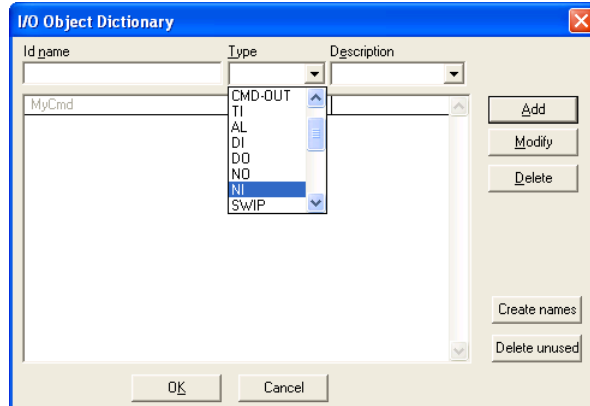
Create a new VFSM file using *File -> New -> VFSM file -> Generic*. Here we will design the ST diagram for CombLock2 VFSM. You need to design a finite state machine, which means inventing states and transitions between them. Start by putting a few state symbols on the diagram: double-click on a blank area to the right of the Init state symbol. A list “State Name Dictionary” opens.

The list only has Init, greyed-out because you can not remove this state. Enter a name “Start” in the text box, and click on *Append* to put this new state onto the end of the list.

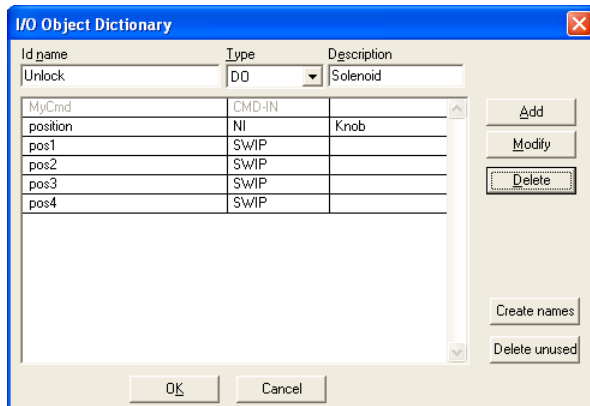
With *O.K.* you will see the new state on the ST diagram. (The reason why a Start state is wanted will be made clear in a moment.) Repeat to add states “Number1, Number2, Number3 and Number 4, one at a time. Move the symbols around, using the mouse, to make a nice arrangement.

3.3. Set up Virtual I/O for the state machine.

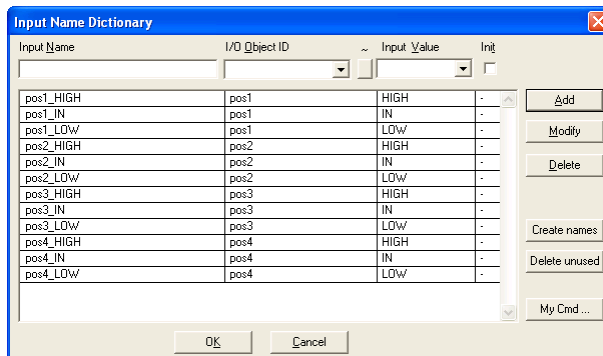
Now we see that we do not know how to set up the transition equations. It is clear that we shall need to know about the position of the big knob, so open the list of *I/O Objects* by selecting “*Dictionary -> I/O Object...*” on the tool-bar and select a NI object (Numerical Input) from the Type drop-down list in the centre at the top of this window. Set the *Id name* to “position”, and *Description* to “Knob”. The Add button adds this object to a list. Keep this *I/O Object Dictionary* window open.



Next, we need to set the knob positions for the four numbers. For this we can use the SWIP object. The SWIP (switch-point) examines a numeric value, and checks whether it is “in range” i.e. between the lower limit and the upper limit. Create 4 instances of this as follows. Select SWIP as a Type, give it a name “pos1” and click on “Add”. Change the name to “pos2” and repeat the Add, and repeat to add the pos3 and pos4 objects.

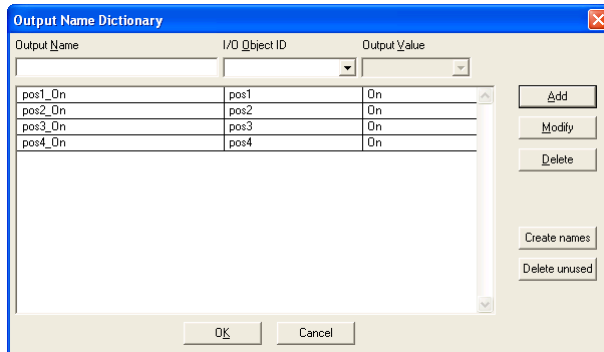


Now we need to set up each SWIP, so close the *I/O Object Dictionary* window and open the *Input Name Dictionary* window, by selecting “*Dictionary -> Input...*” on the tool-bar. From the I/O object ID drop-down list, select the pos1 object, and then look at the drop-down *Input Value* list which becomes available to its right: it shows the possible values “OFF, LOW, IN and HIGH”. We might not need all of these, but let us fix input names for LOW, IN and HIGH values. Select LOW, and then click on the *Name* window at the left: a name is suggested, “pos1_LOW” and it can be accepted for this design. Click on *Add* and the new Input Name is added to the list. (If you don’t click on *Name* then



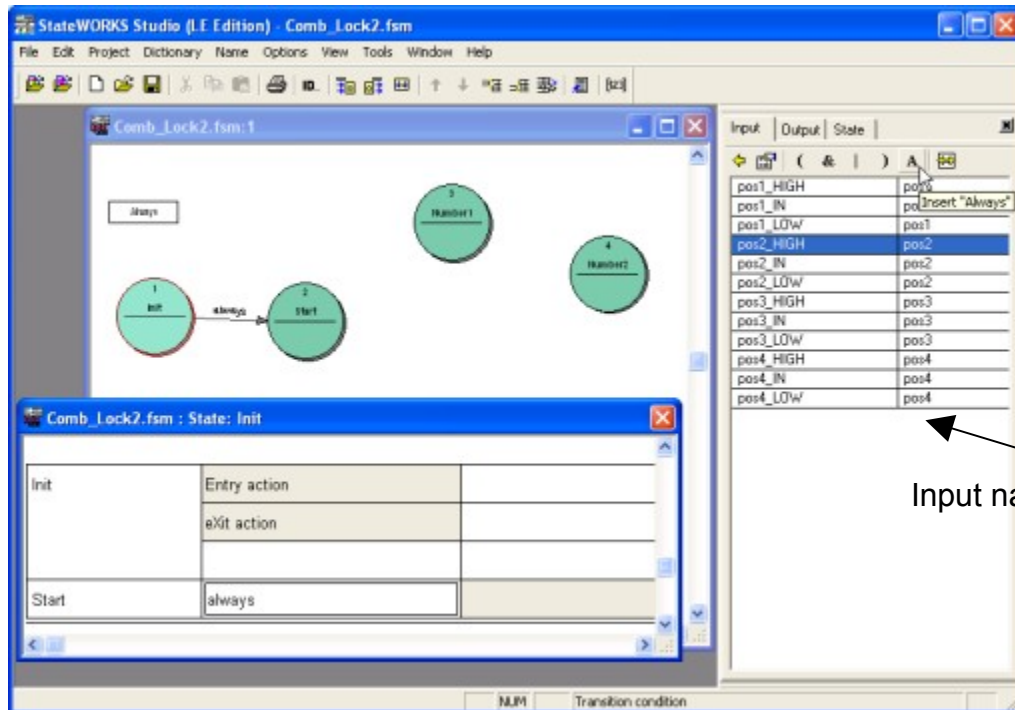
a click on *Add* has the same effect, so you can just click on *Add* twice, in fact!) Repeat this for IN and HIGH, and then do the same for the other three SWIPs. Close the Input Name Dictionary window.

SWIP objects can be enabled or disabled, and the latter is the default state. We therefore need to be able to switch them all on, so we need to create suitable “Output Actions” for the state machine. Open “*Dictionary -> Output...*” and select “pos1” which is the first SWIP in the I/O object list, select “On” for a value, and accept the name “pos1_On” for the action. Repeat for the other three SWIPs, and click “O.K.” to save the new list.



3.4. Complete the state transition diagram.

Now we have enough of a “virtual input” structure to specify the state machine more completely. Just to start with, we shall just put the five obvious transitions in place for correct opening of the lock. Right click on the Init state in the ST Diagram, and drag the



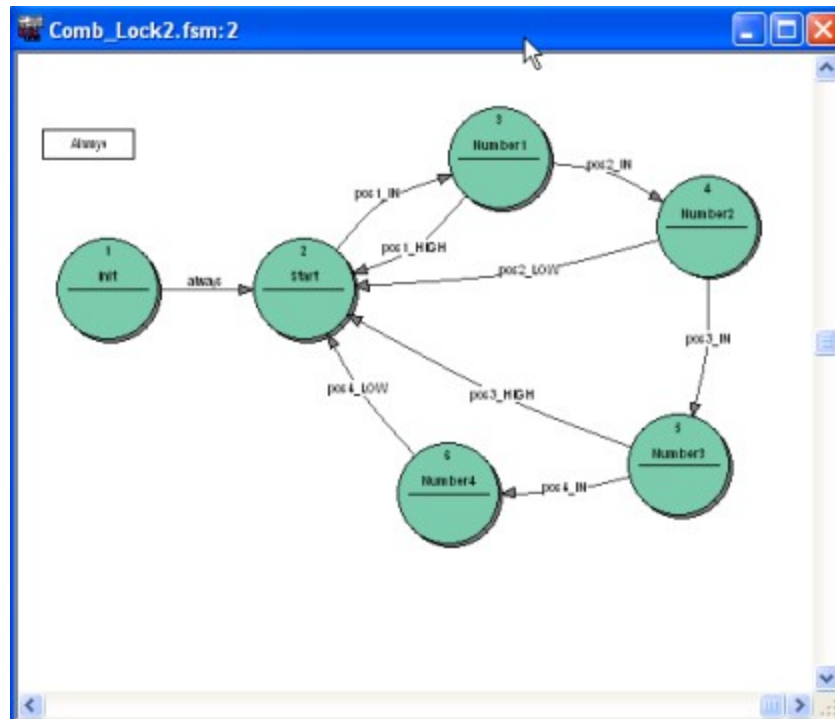
cursor into the Start state. An arrow will be drawn from the Init state to the Start state, to represent a state transition. A window will open, for setting up the conditions for this transition. Select the “transition condition” zone to the right of the “Next state” zone: it is probably already selected, and the Next state is already marked “Start”. (These boxes are not directly labelled, which can be confusing for a beginner, but if any box in this window is selected an explanation appears). Don’t write anything: Doubleclick on “A”

(which stands for “*always*”) on top of the input box to the right. The input name is copied as a transition condition to the state table window.

We shall not use these other possibilities just now, but notice that, at the top of the Input name list, there are also the symbols (, & , | ,) besides A: these can also be selected, so as to compose a Boolean expression. There is no *Not* available, for very good reasons.

Keep the Input name list open, and make a transition from Start to Number1. This time, enter “pos1_IN” as the transition expression. Repeat for the remaining transitions, to Number2, Number 3 and Number 4, using pos2_IN, pos3_IN and pos4_IN respectively.

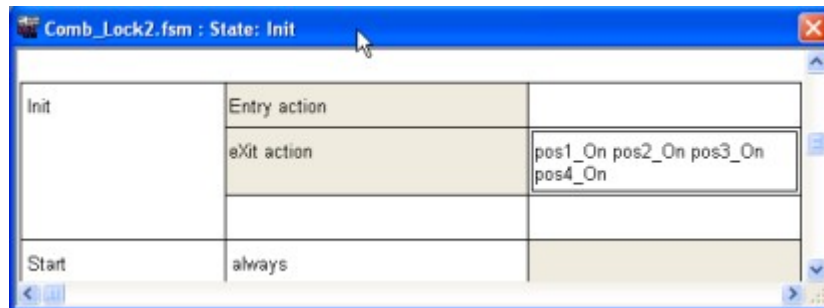
Now we have a machine which should permit the lock to be opened by setting the knob to the four defined positions in turn. Each time the knob reached the next position, the state machine advances to the next state, until it reaches Position4. But with this design the lock would also open if the knob were to be turned at random for a while! So we need to add a few more transitions.



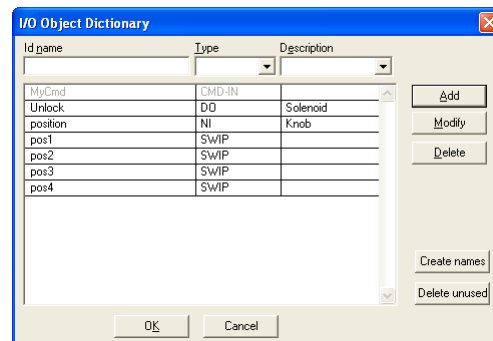
Add a transition back from Number1 to Start using the condition pos1_HIGH. This might have been given a Description “too far up” perhaps. On the ST diagram this second transition will be a curved line: drag it, using a mouse select-and-drag motion, so that it is nicely separated from the straight transition in the “forwards” direction. Repeat this process, adding transitions from Position2, 3 and 4 to Start, using pos2_LOW, pos3_HIGH and pos4_LOW respectively. Now we have effectively simulated the combination lock, in that a turn of the knob too far at any stage will kill the unlocking process.

Your ST diagram may look a trifle untidy at this stage, so play with it a little. Using the mouse, move the state symbols, and alter the curves or lines of the transition arrows to make the diagram look clear and tidy.

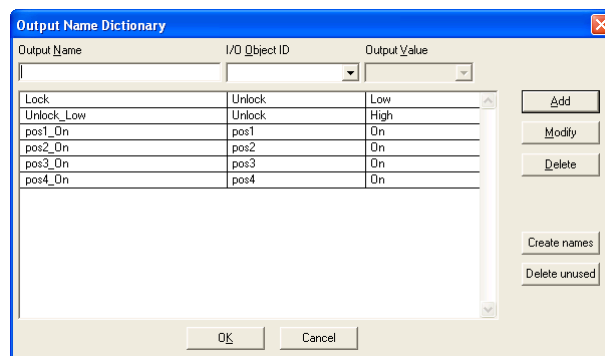
Now the SWIPs need to be enabled. Open the table for the Init state and click into the “Exit Action” field. Doubleclick on the “pos1_On” output name in the output list to the right in order to add it to the list of exit actions of state “Init”. Do the same for all remaining output actions “pos2_On”, “pos3_On” and “pos4_On”. This will ensure that all the SWIPs are activated when the VFSM starts to function. This is why the Start state was required, in fact.



Oops: we forgot something! How can this VFSM actually unlock the door? We shall need another Output action. Open the *I/O object dictionary* from the tool-bar menus, or with F5, and select a DO type – a digital output. Give it a name Unlock and a description Solenoid, add to the list, and close the list using O.K. Now open the Output dictionary, and in the drop-down list for I/O Object ID, select the new Unlock object. Go to Output value and select Low, because of the drive circuit we shall be using, and when you select Name the name Unlock_Low will appear. Not brilliant, but click on Add, then on O.K. While we are here, also select a High Output Value, and add a new output Name. Change this to Lock.



Now we plan to drive the solenoid when the knob reaches Position 4, so double-click on the Number4 state, and then select the “Entry Action” field. Select Unlock_low in the output name list to the right. So now the safe can be opened, in theory. Of course, this particular safe needs to be wired to a specific solenoid driver. We did not setup this item in the project window so far. Before we do that, add an entry action of Lock to the Start state. Otherwise the safe could never be locked again.



At this stage it would be as well to save your files, in the usual way by means of the menu items *File -> Save as...CombLock2*. You will be asked to enter a three letter prefix for your VFSM. Accept the default. You will learn later why we need prefixes. We always advise saving *VFSM* designs in a *VFSM* directory, while the other files should be

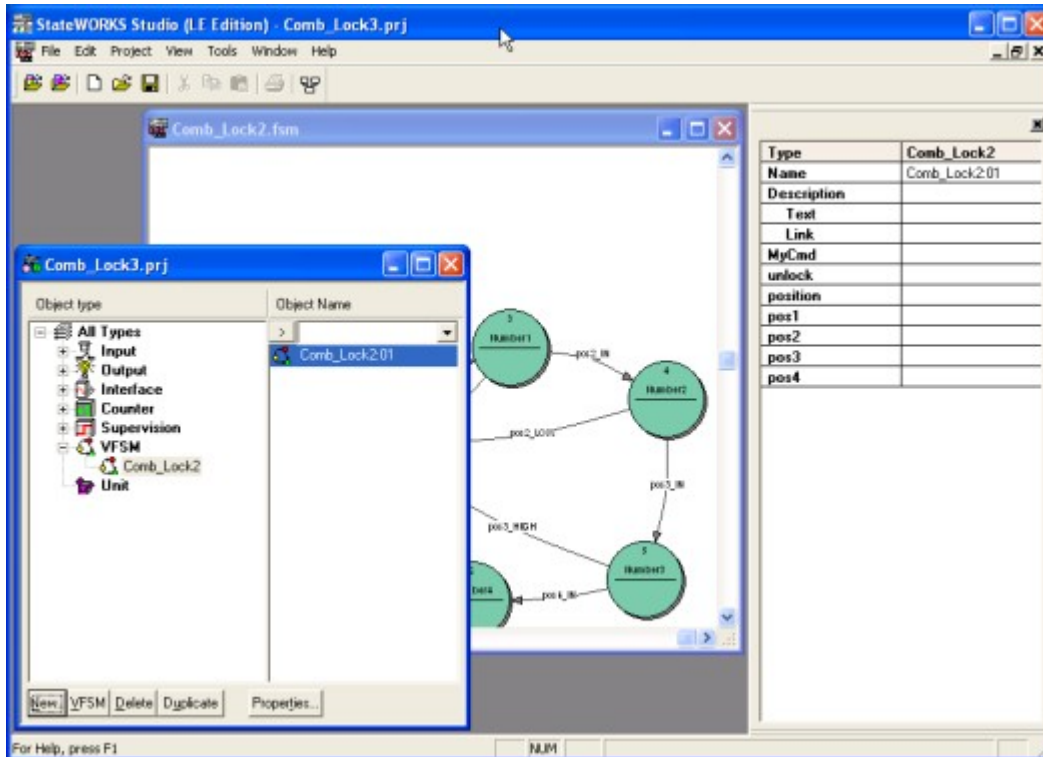
saved in a Project directory. When you use StateWORKS Studio for your own projects you will no doubt prefer to set up your own directories for them.

3.5. The Project Window

The objects which you created in the VFSM editor are “virtual” rather than real: the VFSM uses “object control properties” to define the virtual environment. Contrary to this, objects in the Project Window are specifications of real objects defining their properties. These real objects can be all handled by the Real-Time Data Base (RTDB) which is an important part of the StateWORKS system. Virtual objects defined in the VFSM editor are linked with real RTDB objects in the Project Window. Any object defined in the VFSM editor may be linked with several incarnations of objects in the Project Manager. In a significant project, you might need to use half a dozen identical incarnations of the same state machine, with different properties set for each of them, in the project window.

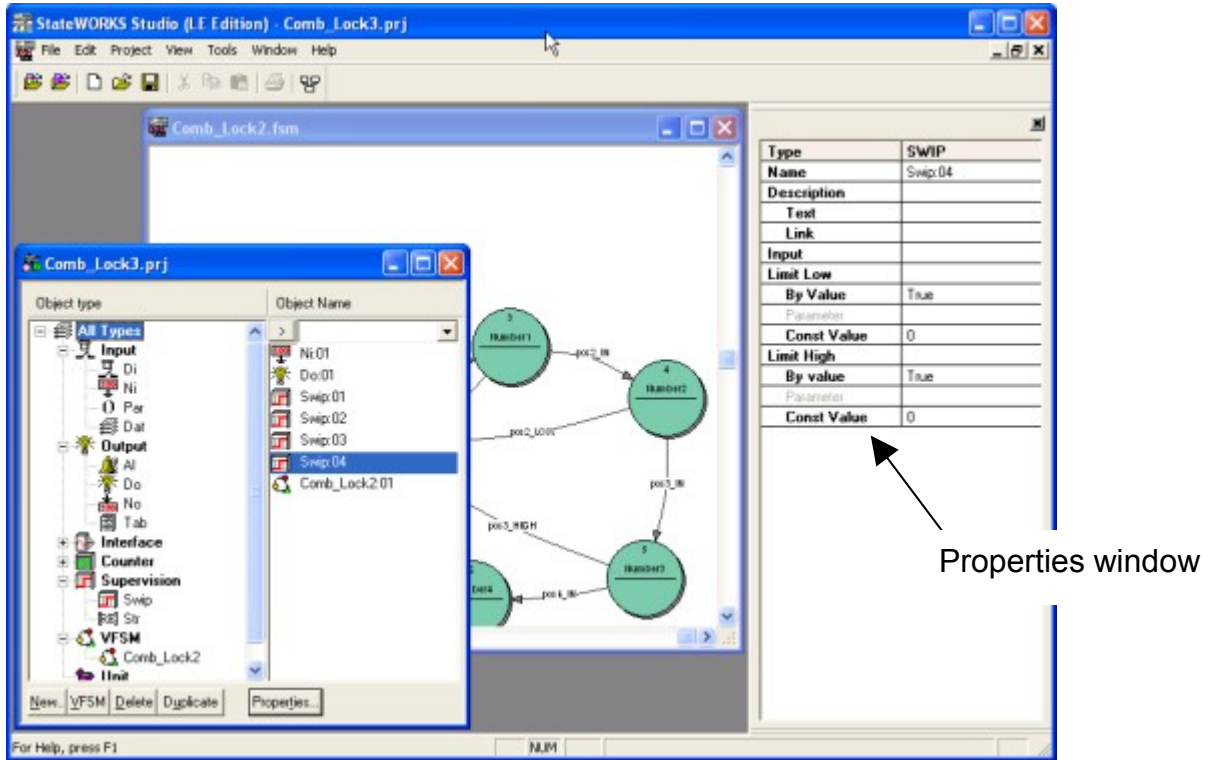
IMPORTANT: From the menu bar, select *Project -> Edit*. Add the VFSM file “comb_lock2.fsm” to the empty list of files in this project. This now specifies that the f.s.m. you have designed is going to be part of the project. Technically, this VFSM is in fact added to the list of objects available to the project, in the same way that many standard I/O objects are in the list already.

In the Project window you should now start to investigate the list of objects in the left-hand panel. All sorts of possible objects are listed here, and if you expand the VFSM section you will see the Comb_lock2 state machine you have designed. Select it, and click “New”. An instance is created, in the right-hand section, called Comb_lock2:01 which name we can leave for this project as it is.



Now, still working in the project window, select a NI object in the Inputs group, and add an instance of this by clicking New. Select a DO in the Output group, and do the same. Then select SWIP in the Supervisory group, and click on New four times.

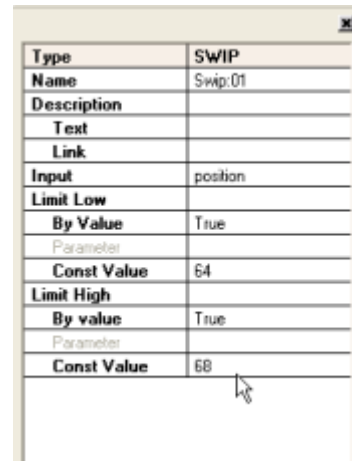
If you select “All types” at the top of the left-hand list, you will see all the real objects in the project. You need to set up the properties of the various objects and you will need to select physical objects for each of the virtual objects. In most cases there is only one name offered, and it should be selected.



Select the NI:01 object in order to see the Properties list in the right window. Change its name to “position” which is easier to remember. Select “ushort” as the format, “<none>” as the units and “Lin” as the scale mode.

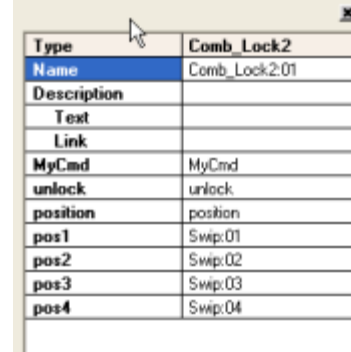
Click then on Do:01 object, rename it to “unlock” and sent Invert option to “true”.

Now set up the SWIP properties. For each SWIP you should choose the input number which it will supervise, and this is always “position” in this project. In a real project we should set the combination as parameters which could be changed easily, but to simplify matters we shall set each limit value directly. Open the properties for the SWIP called SWIP:01 and select “position” from the list for the Input. Edit the low limit to, say 64 and the high limit to 68. Repeat for the other three SWIP objects, entering the limits 31-35, 82-86 and 46-50 respectively.



There is a small problem with the default MyCmd object. If this is left, an ERROR will be signalled, and although in fact this error can be ignored, we suggest that you should create something. In the project editor, select a Cmd in the Interfaces, make a new one, call it MyCmd, and in the properties set a type of CMD-IN.

As the NI and DO objects correspond to physical input/output, a further stage of configuration is needed, with "Units". Use *Project -> Edit -> Add* to add files for DO8.unt and NI4.unt which you will find in the "UNIT/" directory. Then open Unit in the Project window, and make new NI4 and DO8 units, and set their properties. This process is only needed when you wish to link I/O to the functions in SWLab, for testing. The DO8:01 physical address should be set to 3, and for Do0 select "unlock". Then the NI 4:01 physical address should be set to 5, and "position" selected for Ni0. These actions will link the SWLab tool to your design.

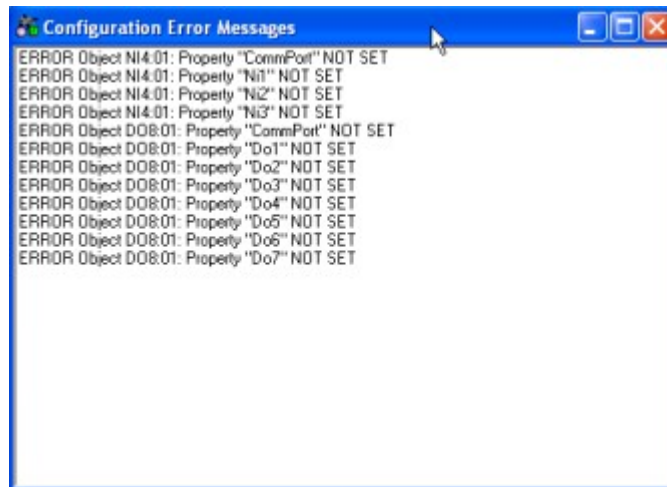


Type	Comb_Lock2
Name	Comb_Lock2:01
Description	
Text	
Link	
MyCmd	MyCmd
unlock	unlock
position	position
pos1	Swp:01
pos2	Swp:02
pos3	Swp:03
pos4	Swp:04

Now the final stage: we still need to link the virtual objects whose properties are used in the ST diagram window to the real objects created in the Project window.

Select your VFSM

Comb_Lock2:01 in the Project window. You will see a list of used objects here, and for each one you need to make the link, by selecting one name in each case. On most cases there is only one object name offered, but you will notice that the SWIP objects could be mixed up because we have four of them. (This might seem a little ponderous in this rather trivial project, but in a real, large project, using many objects of each type, we shall appreciate all the flexibility which it provides.).



Then, at the top, select *Project -> Build All*. You will see a list of errors relating to properties not set of the un-used inputs and outputs; just close it, and ignore the errors.

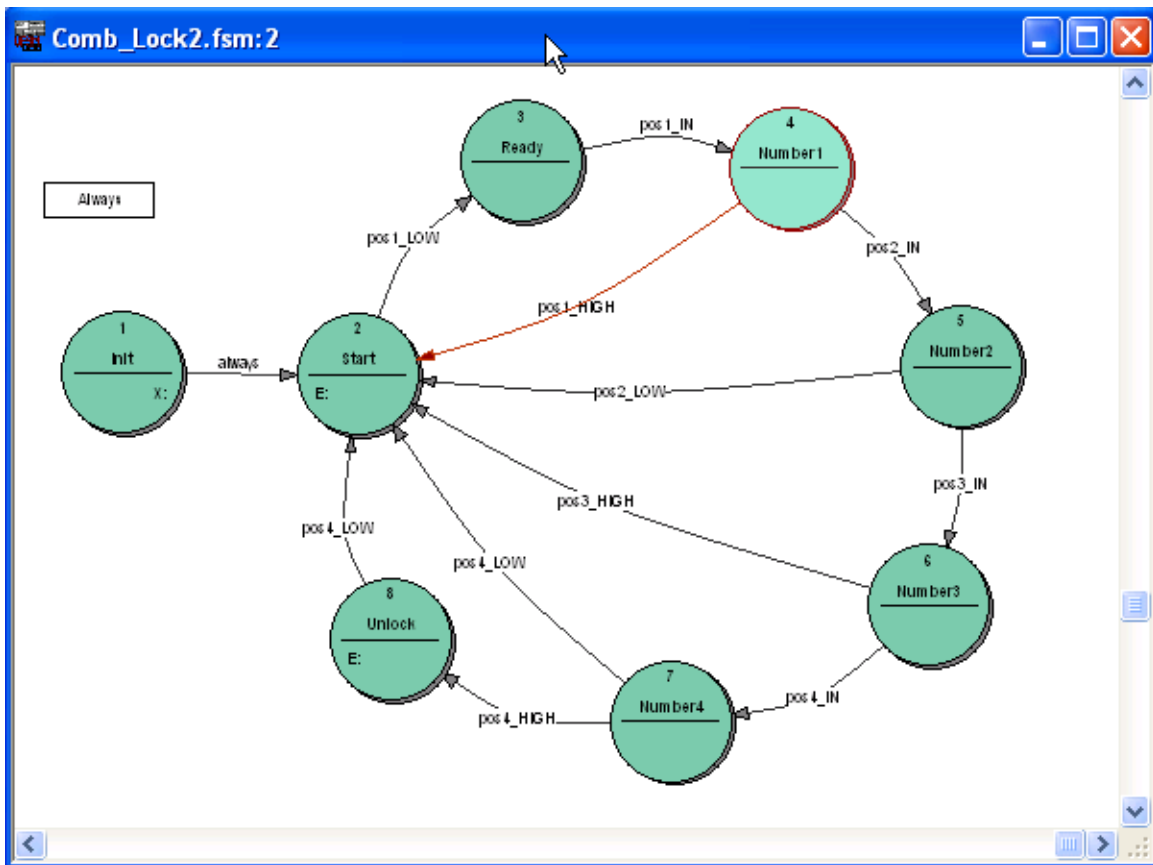
You might like to save or book-mark this section, as the process it describes is very important and not too obvious if you are not used to it.

3.6.Improve security of the lock.

A little thought will lead to suggestions for improving security. For example, the solenoid will be activated immediately when Position 4 is reached, even if the knob is turned further. So one does not need to know the final setting at all! One could introduce a short delay, before going a new state where the solenoid is energised, and with a transition to Start activated by turning the knob too far. You can experiment with the

Timer object if you wish. In this case, we introduce a new rule: after reaching Position 4, one should turn the knob back a little. So we add a new state “Unlock” which is reached from position4, on the condition pos4_HIGH, and we move the solenoid activation to this new state.

Another difficulty is that when you examine the transitions, it is clear that the transition from Start to Number1 will occur as soon as its SWIP senses the IN condition, from either direction. This means that, by starting at 100, one can open the lock without knowledge of the first setting, but only of the second. So another state, called Ready, should be inserted before Number1, and the transition made immediately from Start to Ready on the condition pos1_LOW.



Perhaps you are starting now to appreciate how, by discussing a state transition diagram with a colleague, you might come to see where your design may have weaknesses, and find ways to correct these, much more easily than if you had started with coding. In fact, although testing is important, many problems can be identified and corrected through close study of the ST diagram.

Although, at first glance, the lock seemed to have $100^4 = 100$ million combinations, but as Feynman discovered, there are only $20^4 = 160$ thousand, on account of the tolerances. And until the flaws detailed above are fixed, there are only 400 combinations to try, starting at 100 for the first number, testing the second and third, and moving the knob slowly until the solenoid click shows you the fourth! Three hours of work for the safe cracker, perhaps. Once these flaws are fixed, you might enhance security by adding

time-outs, and ensuring that any mistake will cause the state machine to stay in Start for 45 seconds or more before it will accept new inputs from the knob.

3.7.Locking

It would be important in a real safe to de-energise the solenoid: in fact it can be so done by turning the knob, but a better way is to have a micro-switch, indicating Door Open. When this is active, the solenoid should be shut off and the Start state reached once more. We assume the door locks by a mechanical latch, but perhaps when it is closed, a second solenoid should be activated, for a couple of seconds, to lock the door securely. Check this with your engineers in the design department.

3.8.Save and Check the Design.

Save both the VFSM and the Project, and now in the project menu activate *Build All*. An error window opens, and with luck there will not be any errors listed. This action makes sure that files used by other tools are created, and the documentation aspects are discussed below. The error window can also show some Warnings, if you change the project options, and these might be useful.

3.9.A Note About this Example

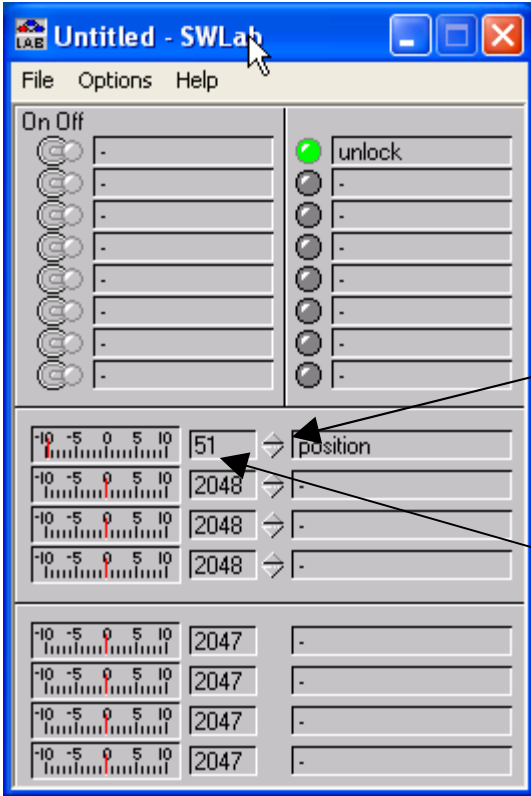
Please remember that the example is given here merely to illustrate how to use the StateWORKS I.D.E. and it is not representative of the typical StateWORKS project, in the following respects:

- It only uses one F.S.M. and you can find very many suggestions for programming such a system in published articles and books. The real power of StateWORKS is displayed in projects requiring 10 or more F.S.M. In fact, splitting the behaviour-control software into a number of F.S.M. is very much more powerful, when implementing the system, than the StateCharts™ technique of regarding groups of states as states.
- As the input is a single object, the knob position, the ability of StateWORKS to handle very complex expressions for transitions or actions is not demonstrated.
- This F.S.M. could be “event-driven” which is a dangerous concept.

4. Testing

4.1.Start SW Lab.

If your design has no serious errors, the *Build All* command will have set up all the project files needed for execution by the run-time system, or for carrying out some tests. On the Tools menu, click on *SWLab*. Your design is running! You will see an *Unlock* output and a *position* input.

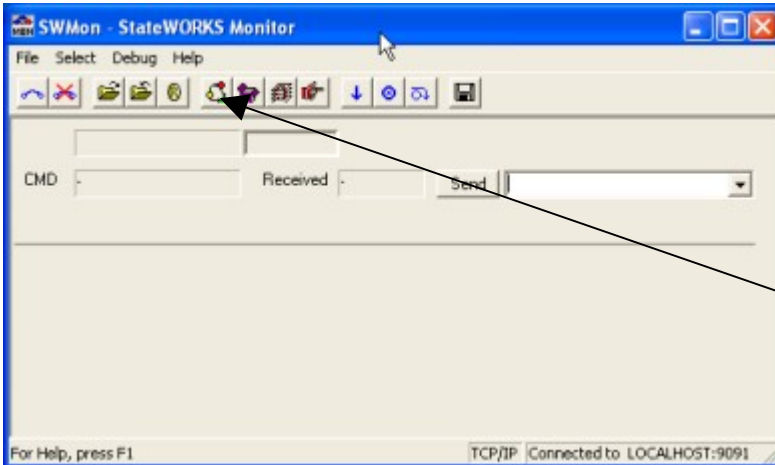


Up/down arrows

Doubleclick the field to enter a value

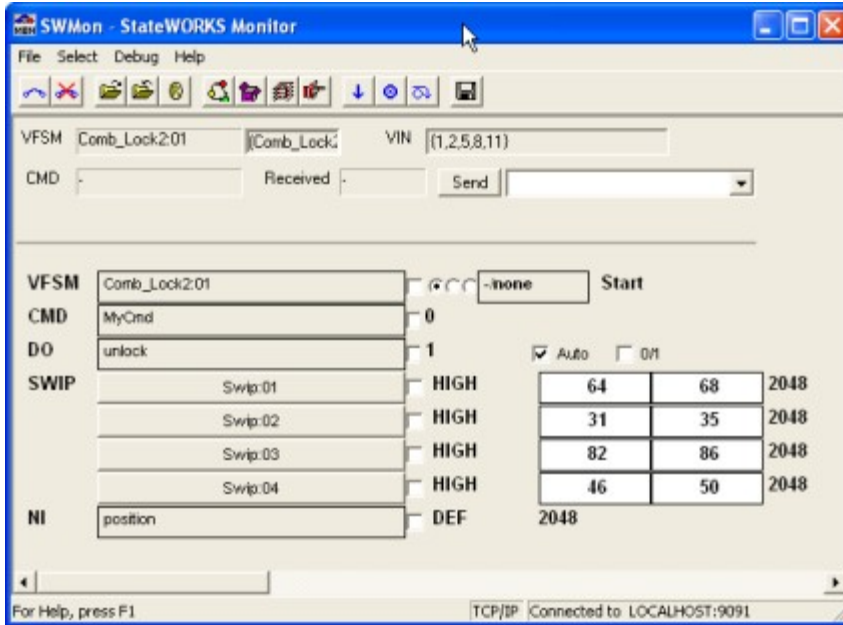
4.2. Start SWMon.

Now open SWMon in a similar way (Tools -> SWMon). You will see a small tcp/ip set-up window: accept the default values for Host (= LocalHost) and Port (=9091). Click O.K. and the Monitor window opens. It does not show much, but a VFSM icon can be used to show the list of state machines in the system, and you can select Comb_lock:02 to see what it is doing.



VFSM Icon

The upper section can be ignored for the present. In the lower section you see a list of objects, starting with the state machine itself. It is in the Start state and nothing much is happening, as the NI position input is set to 2048 by SWLab.



Now in SWLab, using the cursor, or preferably the up/down arrows, alter the setting to 50 or so, and then to 68. This is tricky, but you may note that the *Shift* key accelerates the up/down increments by a factor of 8, and the *Control* key by 64. The state machine performs a transition to Number1.

Continue, by changing the position value to 33, 84 and 48 in turn, so as to arrive at the state Position4, and up a little (55) to reach Unlock where the Unlock output line will pass from 1 to 0 to drive the solenoid. Remember, we had specified an inversion in the output driver for this function..

4.3. Use of SWQuickPro.

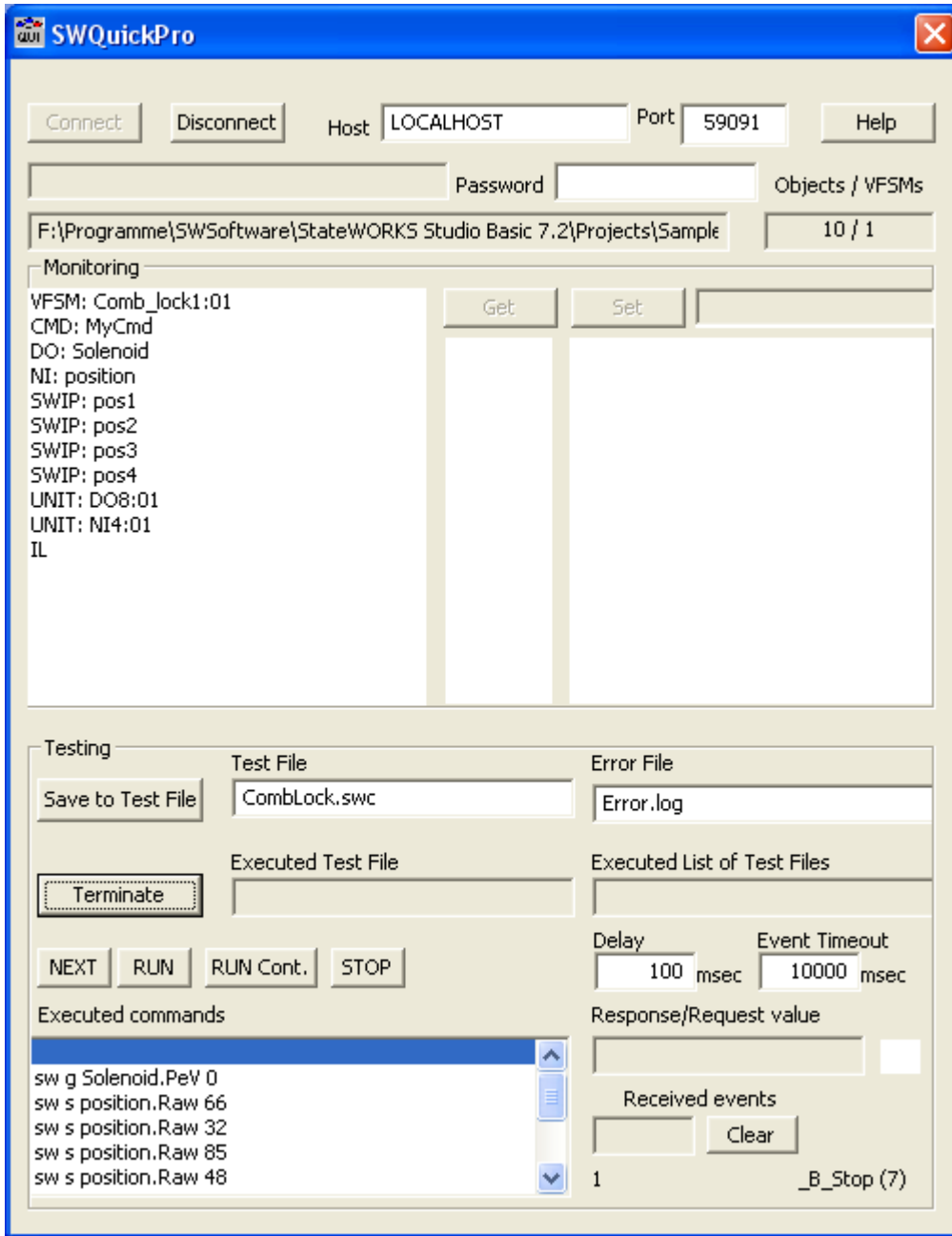
We think that this utility is a little too complicated to explain in this guide, and is best left for explanation elsewhere. But be aware that it permits command file scripts to be generated and used for repeating test sequences. Anyway, you may easily try that testing facility:

- Start SWQuickPro (*Tools/SWQuickPro*)
- Connect to SWLab (button *Connect*)
- Load the command file CombLock.swc (button *Load*)
- Execute the command file using buttons NEXT, RUN, RUN Cont. and STOP.

If you have studied U.M.L. or Extreme Programming you will know that these techniques emphasize regular testing of models or of software, to confirm that it is functioning correctly. StateWORKS Studio assists you to automate such tests, and so save a lot of time. You can create prepared command scripts, to express what is supposed to happen in the external environment, and see how the software will react. Using these, or working by hand with individual commands, you may complete a series of tests, and a log file can be saved as a record. Furthermore, this log file can be used to form a new

command script file, and re-used to check that design changes have not had adverse side effects.

Although much emphasis is given in published literature to the formalisation of “Use Cases” we must emphasise that such tests need to cover not just the “sunny-day scenarios” when everything works as it should, but must also explore as many error possibilities as possible, to see how these will be handled. This can get quite complicated, and you will also wish to spend time discussing your VFSM designs with colleagues, so as to detect possible flaws in your reasoning: much easier with reference to an ST diagram than by going through source code of conventional software.



5. Results

5.1. Files Generated: Documentation.

Two printing possibilities are provided, for the text file and the ST diagram respectively.

You may print either of these to a file, in the usual way, and if you have Acrobat Distiller installed on your computer you can generate .pdf files. The text file may include the contents of your various dictionaries, as well as the state transition tables: in fact various options are offered. Don't forget that you can add comments to the ST tables, and these will be very helpful in understanding the project later.

5.2. Special Files Generated.

Most of the files which are generated by "Build" are special files, for use by the testing software and/or the run-time systems of various sorts. The .cpp file, for example, is used to create a "C tables" which are required to form the RTBD in situations where there is no disc drive in the target system. You should not need to worry about these files.

5.3. XML import/export.

Designs are exported in XML format by the Build command. This will permit StateWORKS designs to be processed, and used in conjunction with other tools, or U.M.L. finite state machine specifications to be used as a basis for a detailed VFSM implementation, for example. You can look at these files with an Explorer, as the DTD and XSL are also provided, to produce a pleasing format. On account of the special features of StateWORKS, it needs to use its specific XML DTD and this is fully explained in papers which you can access on the Web site.

Future versions of StateWORKS Studio will also provide for XML import of designs, and other features. So make sure we can contact you, to advise you about upgrades.

6. Systems of Multiple VFSMs

6.1. The Principles.

Although an ST diagram with 10 to 20 states can be manageable, it is often found that when all the possible system problems are taken into account, the number of states increases to a point where the design is hard to understand. This problem is best resolved by splitting the design into one which employs several state machines, working together. Although the StateWORKS tools permit any arrangement you like for linking these, we advise the use of a hierarchy. A very large system might use a hundred or more VFSMs, arranged in five or six levels: StateWORKS can deal with this.

At the upper level, there are masters, and below these there are slaves. The masters should issue commands to the slaves, while the slaves, being visible to the masters,

provide state. Technical papers on the StateWORKS web site discuss this issue in some detail.

6.2. Using the SMS Diagram.

When you have a design using several VFSMs, the special “System of State machines” diagram (SMS diagram) can be opened, from the project window. This shows the various VFSMs as rectangles, and shows the command links between them. We suggest that you open the “gas” project in the Examples folder, to see how this is done. The SMS section of the help file provides more information.

7. Run-Time Systems

The files produced by StateWORKS Studio completely define the behaviour of each VFSM in the project, together with the input-output systems, so that their specifications can be directly executed by the “VFSM Standard Executor” software module. SWLab is a ready-built system of Real-Time Data Base plus the Executor, and is not different in principle from the run-time system. It is possible to use the Standard Executor in many run-time systems, for example if you are using an “Industrial PC” for your project. The I/O system is a trifle restricted, but it would be possible to use external I/O units linked to the computer by a TCP/IP network or through serial lines, for example.

We provide all the tools and documentation needed to build specific systems with your own input/output drivers, integrated into the real-time data base. The design of the latter is quite open so special versions are readily produced.

Consult us for further details.

8. A Final Remark:

From here on, after following the guide for an hour or two, you should find that working with StateWORKS Studio is fairly obvious, and you should get up to speed quite soon.

Don’t hesitate to use the Help system. Further information and technical papers can be seen at the Web site <http://www.stateworks.com>.

If you are happy with the product, tell your colleagues and friends. If you have a problem, tell us! Perhaps there will be a small misunderstanding, or an error in the Help files or other documents – give us a chance to fix it! Call your distributor, or e-mail sw-info@stateworks.com with your complaint, quoting your user code number.