StateProto – Watch Sample Hsm Implementation

statedriven@users.sourceforge.net

02 July 2006

draft

Table of Contents

Illustration Index

Using StateProto

Starting StateProto

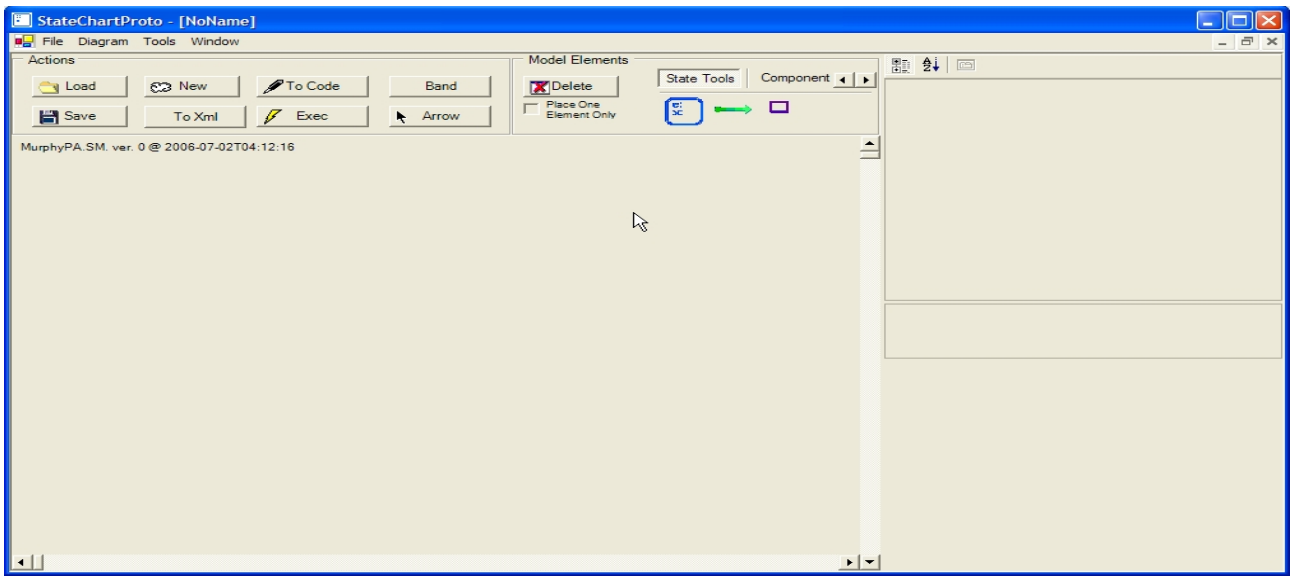
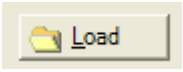


Illustration 1: Fresh start of StateProto

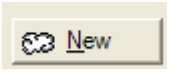
For now the menu items in StateChartProto must be ignored. StateChartProto can only be used with one state machine at a time for now. Run multiple process instances to be able to work with more than one state machine at a time.

Actions

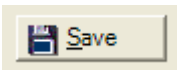
1. Load – Load an existing StateMachine definition. These definitions are saved in a flat file

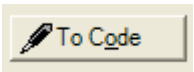
   format with Begin/End block containers. The file extension is sm1.
2. New – Clear any existing state definitions and reset to a clear document.

3. Save – Save the state machine definition.

4. To Xml – Save the state definition in an xml format that excludes all gui parameters.
5. To Code – Generate C# code for the current state machine.

6. Arrow – States and Transitions can be selected via the “State Tools” toolbar. To deselect there is generally an arrow selector somewhere – this arrow button is that selector.

7. Band – This is meant to select a number of glyphs so that they can be moved as a group. Does not work yet.

8. Exec – Allow a state machine to execute the current definition – purely as a simple flow test. Does not work yet.

Model Elements

1. State Glyph – Allows you to draw state model elements.

2. Transition Glyph – Allows you to draw transition model elements.

3. Port Glyph – Ports will be described later.

Property Viewer

The property viewer displays the currently selected model element. If you click on the arrow selector and then click inside the blank design area (beneath the buttons) then you will see that the property viewer shows the state machine model information.

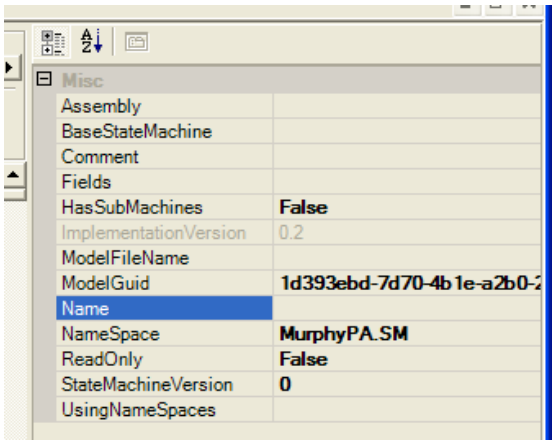


Illustration 2: State Machine Properties

Many of the fields here are reserved for future use. For now the fields that must/can be filled in are:

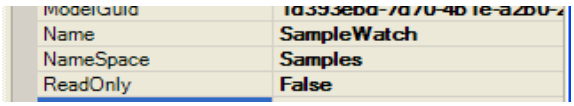
1. Name – provide the state machine name. Do not insert spaces and you must start with a non-digit character.
2. NameSpace – change this to the namespace to be used by your project.
3. UsingNameSpaces – can be used to add some namespaces to place in the generated code. There are other ways to add namespaces as well.
4. ReadOnly – on loading an existing state machine readonly is automatically set to true. Set it to false or press Ctrl-R to toggle.
5. ModelGuid – this is automatically generated and is meant to provide each new state machine with a unique identifier. Do not reuse this identifier in other state machines.
6. Comment – can be used purely as a comment area. This is not used in any other way.
7. StateMachineVersion – this field is auto-incremented every time the state machine is saved.

There is no undo/redo mechanism yet – so save early – save often.

8. Assembly, BaseStateMachine, Fields and HasSubMachines are reserved for future use.

Do it

1. Set the Name to SampleWatch. This is the start of the same state machine that Samek describes in an article samek0008.pdf published in 08/2000 in "Embedded Systems Programming" called "State-Oriented Programming". http://www.quantum-leaps.com/writings/samek0008.pdf
2. Set the NameSpace to Samples



3. Save the file to a folder c:\Hsm\Samples\. The filename should default to

SampleWatch.sm1.

4. Left click on the model elements "State Tools" tab if it is not the current tab. Then click on the State tool to draw states. The cursor changes to a cross-hair indicating that a glyph can be drawn. Now click down on the Design surface and drag. This results in a gray rectangle being drawn with a diagonal through it.
5. When you let go then a state shape will be drawn where the rectangle was initially drawn. The State Glyph will be red and will also a "currently selected" indicator as well as contact points. The state must be named – only then will the colour change from red to its default color for that level.

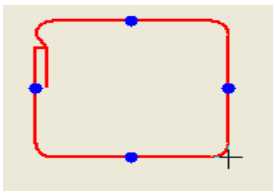


Illustration 3: Selected Unnamed State

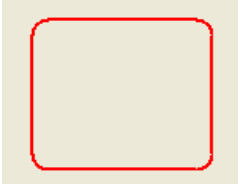


Illustration 4: Unselected Unnamed State

The State Glyph

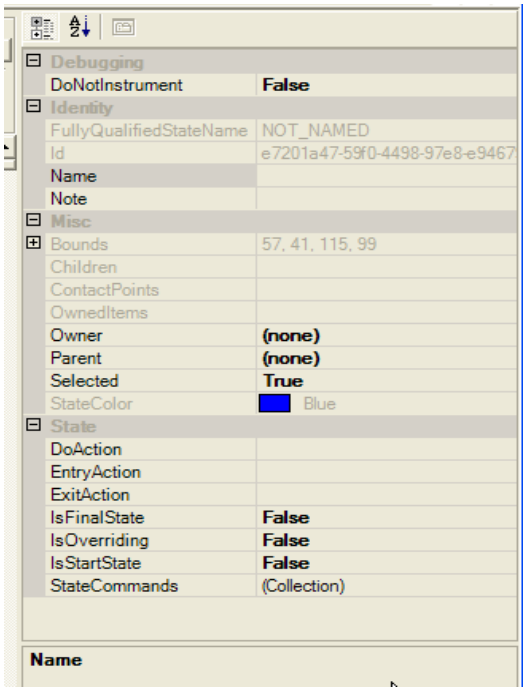


Illustration 5: State Properties

1. Name – the name of the state. This is used during code generation and so must be a valid part of a C# function name. The name can be entered here or press Ctrl-n to get an entry dialog.
2. Note – this is essentially a comment and is not used for anything else.
3. DoNotInstrument - will not code generate log statements within the method for this state if set to true.
4. DoAction – Not yet supported.
5. EntryAction – The C# code to execute when entering this state. To keep things simple make this is a method call. The action can be entered here or press Ctrl-e to get an entry dialog. Within the dialog Ctrl-Enter will insert a newline. Semi-colons are automatically placed into the code generated at the end of newlines.
6. ExitAction - – The C# code to execute when exiting this state. To keep things simple make this is a method call. The action can be entered here or press Ctrl-x to get an entry dialog. Within the dialog Ctrl-Enter will insert a newline. Semi-colons are automatically placed into the code generated at the end of newlines.
7. IsStartState – If this is set to true then code is generated to ensure that the parent state will move into this state asap. Toggle this here or by pressing Ctrl-t.
8. IsFinalState – If this is set to true then the IsFinalState(...) generated method will return true when the state machine is in thi state. No other action is currently taken – but this behaviour might change. Toggle this here or by pressing Ctrl-f.
9. IsOverriding – Indicates that this state overrides a state by the same name and parent tree in a derived state machine. This is reserved for future use.
10. StateCommands – a list of strings that is placed as attributes of the state that can be used to indicate a set of commands that a gui (or any other subsystem) could execute when the state machine (hsm) reaches this state.

Do It

1. Give the state a name. Press Ctrl-n and type TimeKeeping. Note that the state is drawn in blue once it gets a name. This is the colour for states at level 1.
2. Select the "Arrow" to take you out of state making mode. Usually you will draw a number of states and name them immediately during your design phase. As you can size the state on initial draw – you can even draw substates easily while staying in the state drawing mode. After this you can enter "transition" drawing mode to place out a number of transitions.
3. All this said – you can at any point click on "Arrow" to then be able to select a state or transition and do context specific actions against them. This includes naming, moving, resizing (via the contact points) and finally deleting.
4. After clicking on "Arrow" the cursor should change to an arrow. If for some reason the unnamed state is not selected then click on it to select it. Then move the mouse over the bottom contact circle, click down and drag downwards to enlarge the state vertically.
5. Then move the mouse over the right contact circle, click down and drag to the right to enlarge the state horizontally.

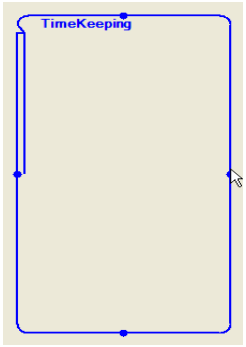


Illustration 6: State with Name and resized.

6. If you click anywhere within the state and drag then you will move the state around the design area.
7. Let's draw a substate within "TimeKeeping" called "Time". Click on "state" model element. Click within the "TimeKeeping" state and drag out a substate within the top half of "TimeKeeping". Press Ctrl-n and give it the name "Time". The colour prior to naming was once again red. After naming it should become darkOrange.

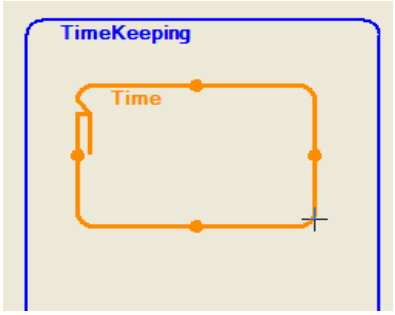


Illustration 7: State with Nested SubState

8. Note that the cursor is still a cross-hair which means that if you click again then it will drop another state onto the design surface.
9. Do this to add all the states as indicated in the next diagram.

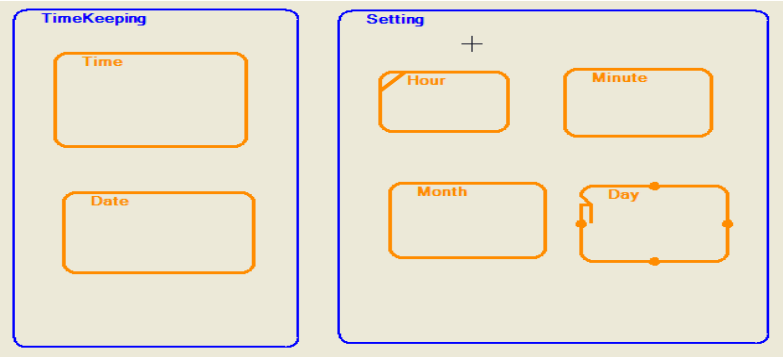
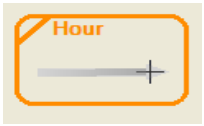


Illustration 8: Watch - states only

10. Save by clicking on Save or pressing Alt-S. A save dialog should be shown.
11. Now click on the Transition tool. The state tool should become unpushed.
12. Left click inside the hour state and drag out a transition line while still remaining inside the hour state.



13. Release the mouse button to drop a transition glyph inside the hour state.

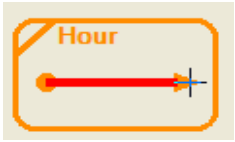


Illustration 9: Transition without Signal definition

14. Transitions do not need a name by default. What they do need is a signal.
15. Press Ctrl-V and set the signal to ModeEvt.

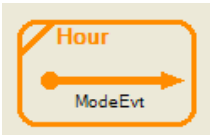


Illustration 10: Transition with Signal definition

16. You can select a transition by clicking on its circle or pointer contact points – not on its line body. Notice that the currently selected transition is drawn bolder than any other transition. In future instances you will also notice that a transition takes on a colour that gradiates from the colour of its source state to its target state.

The Transition Glyph

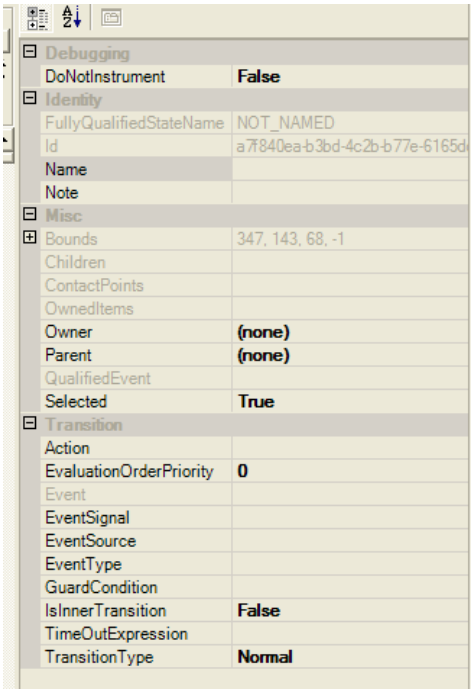


Illustration 11: Transition Properties

1. Name – simple transitions do not need to be named. A unique name must be provided for transitions with guard conditions or timeout expressions.
2. Note – a place to put a comment.
3. DoNotInstrument – reserved for future use.
4. Action – the action to execute when the transition is taken expressed as C# code. It is recommended that this just be a method call.
5. EvaluationOrderPriority – orders the testing of guard conditions on transitions from the same state that share the same signal. Order is from lowest to highest.
6. EventSignal – This is the signal that must be raised by sending it to the state machine via an hsm.AsyncDispatch(ev) for the transition to be taken. Signals must not contain any spaces and must start with a letter or underscore.
7. GuardCondition – The guard condition is best defined as a C# method that returns true or false. Otherwise it can be any valid C# boolean expression.
8. EventSource – Indicates the source of the event. By default the source is not set – and will execute as if there were no specific source. This will be discussed in more detail later with ports[1].
9. EventType – this is only outputted against the TransitionEventAttribute and can be useful for introspection during execution – but is not used as yet.
10. IsInnerTransition – sets the Transition to be dealt with as an inner transition. Inner transitions only make sense on transitions where the start and end states of the transition is the same state. Inner transitions do not run the exit and entry actions of the state.
11. TimeOutExpression – a timeout expression that can support a single value to indicate timeout in seconds, can start with “after” as in “after 10” to indicate a timeout in 10 seconds or “every” as in “every 10” to indicate a timeout every 10 seconds.

1 In the follow on document: Interaction between Multiple StateMachines.

12. TransitionType – there are three transition types (normal, shallow history and deep history) of which only normal and deep history are currently supported.

Do it

1. Draw the further transitions as indicated in the following diagram. Note that each nested state in Setting has a ModeEvt self transition and a SetEvt transition from one substate to the next.

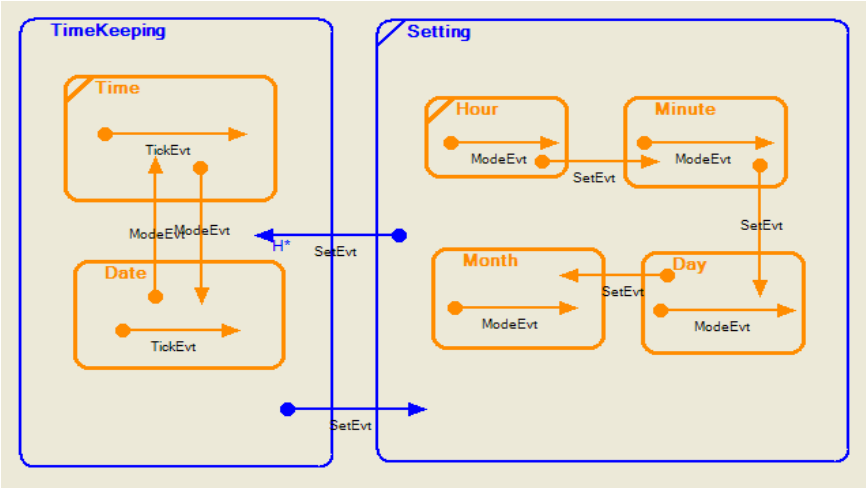


Illustration 12: State diagram with Actionless Transitions

2. Note that these transitions do not have any actions against them yet.
3. Note that the SetEvt transition from Setting to TimeKeeping is a deep history transition.
4. You can select the Hour::ModeEvt transition and press Ctrl-A or click in the Action property. Ctrl-A brings up a dialog in which you can enter the action code. Note that there is not code-completion (intellisense) as yet. Type "IncrHour()" without the quotes.
5. Next we will convert this to code.

Converting to Code

1. Create a C# winForms project in c:\Hsm\Samples called SampleWatch so that the project file is saved to c:\Hsm\Samples\SampleWatch\SampleWatch.csproj.
2. Press the "To Code" button (or Alt-O) to generate code. Save the SampleWatch.cs file to c:\Hsm\Samples\SampleWatch\SampleWatch.cs.
3. Include this file in your project.

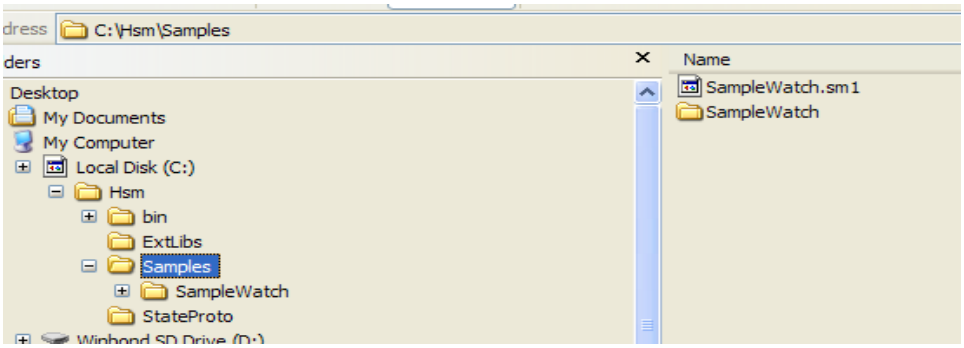


Illustration 13: Samples Project Folder Structure

4. In the Visual Studio Solution Explorer click "Show All Files" and include SampleWatch.cs

into your project.

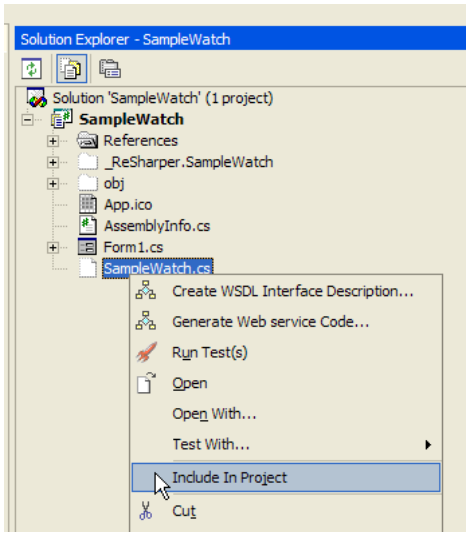
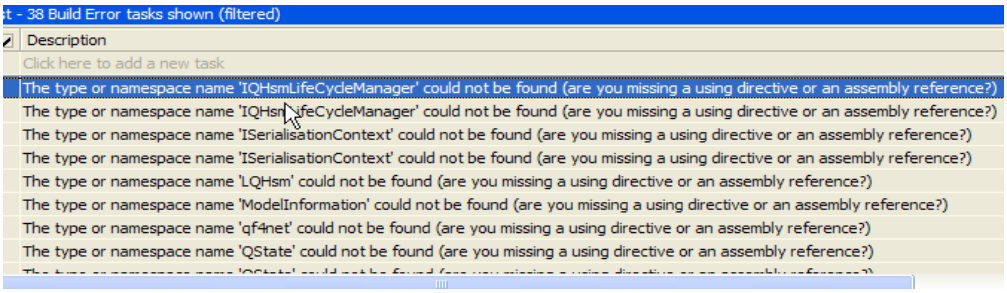
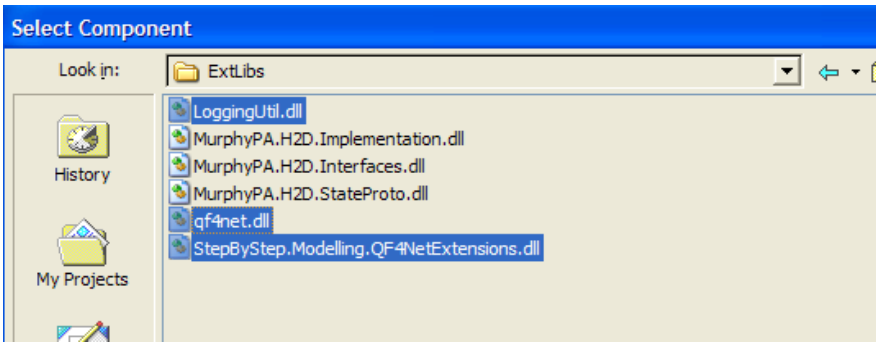


Illustration 14: "Show all Files" and "Include In Project"

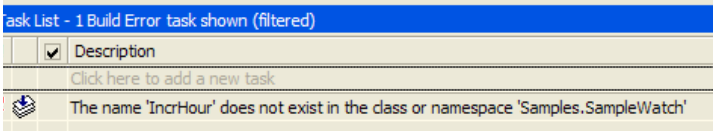
5. If you try to build now you will get a number of build errors – mostly to do with missing assembly references.



6. Go "Add References" and browse up to the c:\Hsm\ExtLibs folder and include the following files: LoggingUtil.dll, qf4net.dll and StepByStep.Modelling.QF4NetExtensions.dll[2].



7. If you try to build now you should get the following error: "The name 'IncrHour' does not exists...'.



8. If you double click on the error VS will take you to the point that this method is called. We did not provide this method yet – which is why we're getting the error. Go to the top of the file and look for the "//Begin[[ClassBodyCode]]" block.

2 I am planning to rename the Namespaces from StepByStep to StateDriven when I get a chance. Even so – I do like StepByStep.

```
public class SampleWatch : LQHsm, ISigSampleWatch {

    //-------------------------------------------------
    //Begin[[ClassBodyCode]]
    //End[[ClassBodyCode]]
    //-------------------------------------------------
    Boiler plate static stuff
```

9. Between the Begin and End tags press newline.

```
public class SampleWatch : LQHsm,

    //----------------------------
    //Begin[[ClassBodyCode]]

    //End[[ClassBodyCode]]
    //----------------------------
    Boiler plate static stuff
```

10. The IncrHour() is defined as follows:

```
//----------------------------
//Begin[[ClassBodyCode]]

int _Hour;
void IncrHour()
{
    _Hour++;
    if(_Hour > 23)
    {
        _Hour = 0;
    }
}

//End[[ClassBodyCode]]
```

11. The project will now build and run. The only problem is that the state machine will not actually execute yet – we need to construct it.
12. Constructing an Hsm is easy. There are six steps:
    - Create a QTimer. This can be a QSystemTimer().
    - Create an EventManager. This can be a QMultiHsmEventManager().
    - Create a Runner. This can be a QGUITimerEventManagerRunner() for Gui's. For server side create a QthreadedEventManagerRunner().
    - Now create the state machine passing the EventManager to the constructor.
    - Call init on the state machine.
    - Hook up events to the state machine.

```
private void Form1_Load(object sender, System.EventArgs e)
{
    _EventManager = new QMultiHsmEventManager(new QSystemTimer());
    _Runner = new QGUITimerEventManagerRunner (_EventManager, 1);
    _Runner.Start ();
}

private void buttonCreateWatch_Click(object sender, System.EventArgs e)
{
    _SampleWatch = new Samples.SampleWatch (_EventManager);
    _Hsm = _SampleWatch; // use it as a straight on state machine
    _HsmSignals = _SampleWatch; // use it as a source of signals -- inst

    SetupHsmEvents (_Hsm);
    _Hsm.Init ();
```

Illustration 15: Creating a State Machine

```
private void RegisterEvents()
{
    _Hsm.StateChange += new EventHandler(_Hsm_StateChange);
    _Hsm.UnhandledTransition += new DispatchUnhandledTransitionHandler(_Hsm_UnhandledTransition);
    _Hsm.DispatchException += new DispatchExceptionHandler(_Hsm_DispatchException);
}
```

Illustration 16: State Machine RegisterEvent

```
private string StateNameFrom(QState state)
{
    if (state == null) return "NULLSTATE";
    return state.Method.Name;
}

private void _Hsm_StateChange(object sender, EventArgs e)
{
    LogStateEventArgs sa = (LogStateEventArgs) e;
    switch(sa.LogType)
    {
    case StateLogType.Init:
    case StateLogType.Entry:
    case StateLogType.Exit:
        {
            Logger.Info("StateChange: {0} {1}", sa.LogType, StateNameFrom(sa.State));
        }
        break;
    case StateLogType.EventTransition:
        {
            Logger.Info("StateChange: {0} {1} {2} {3}", sa.LogType,
                        StateNameFrom(sa.State),
                        StateNameFrom(sa.NextState), sa.EventDescription);
        }
        break;
    default:
        {
            Logger.Info("StateChange(defaultHandler): {0} {1}", sa.LogType, StateNameFrom(sa.State));
        } break;
    }
}
```

Illustration 17: Sample StateChange Handler Logging

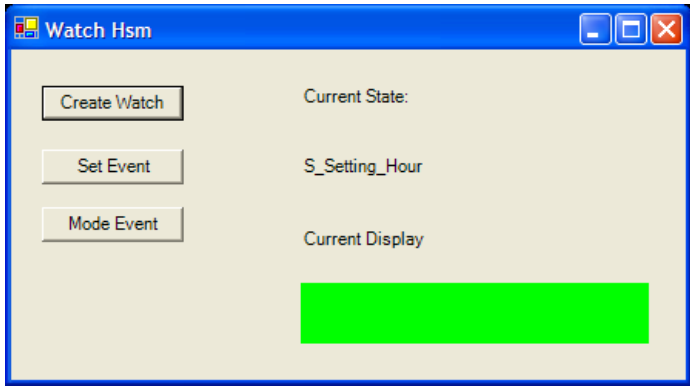


Illustration 18: Sample Watch FrontEnd

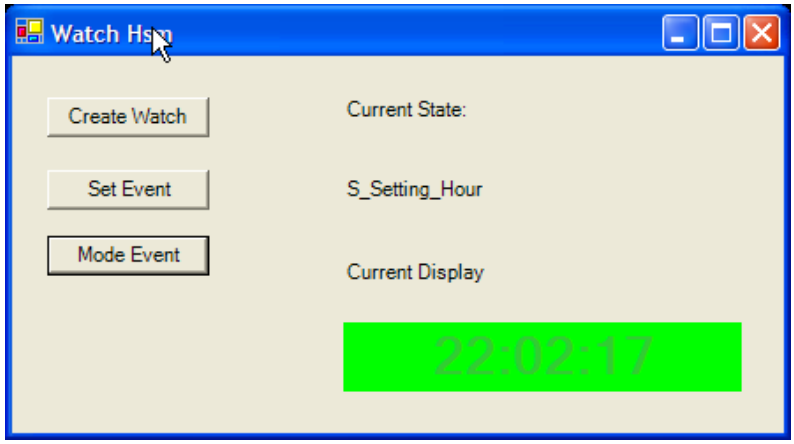


Illustration 19: After ModeEvt is sent

```
20060702 09:02:18.008 INFO SampleWatch.ConsoleStateEventHandler StateChange: Entry S_Setting
20060702 09:02:18.008 INFO SampleWatch.ConsoleStateEventHandler StateChange: Init S_Setting
20060702 09:02:18.008 INFO SampleWatch.ConsoleStateEventHandler StateChange: Entry S_Setting_Hour
20060702 09:03:44.152 INFO SampleWatch.ConsoleStateEventHandler StateChange: EventTransition S_Setting_Hour S_Setting_Hour ModeEvt/IncrHour()
20060702 09:03:44.162 INFO SampleWatch.ConsoleStateEventHandler StateChange: Exit S_Setting_Hour
20060702 09:03:44.162 INFO SampleWatch.ConsoleStateEventHandler StateChange: Entry S_Setting_Hour
```

Illustration 20: Logged after Init and ModeEvt

This state machine is incomplete and does not really behave like a watch. It is in the Setting::Hour state. In a normal watch the hour will be blinking. If I change over to the TimeKeeping::Time state (by pressing SetEvt a couple of times) – I would expect the time to be displaying and ticking over every second. This does not happen.

To achieve the more complete watch feel – you will need to expand the state machine to the version below:

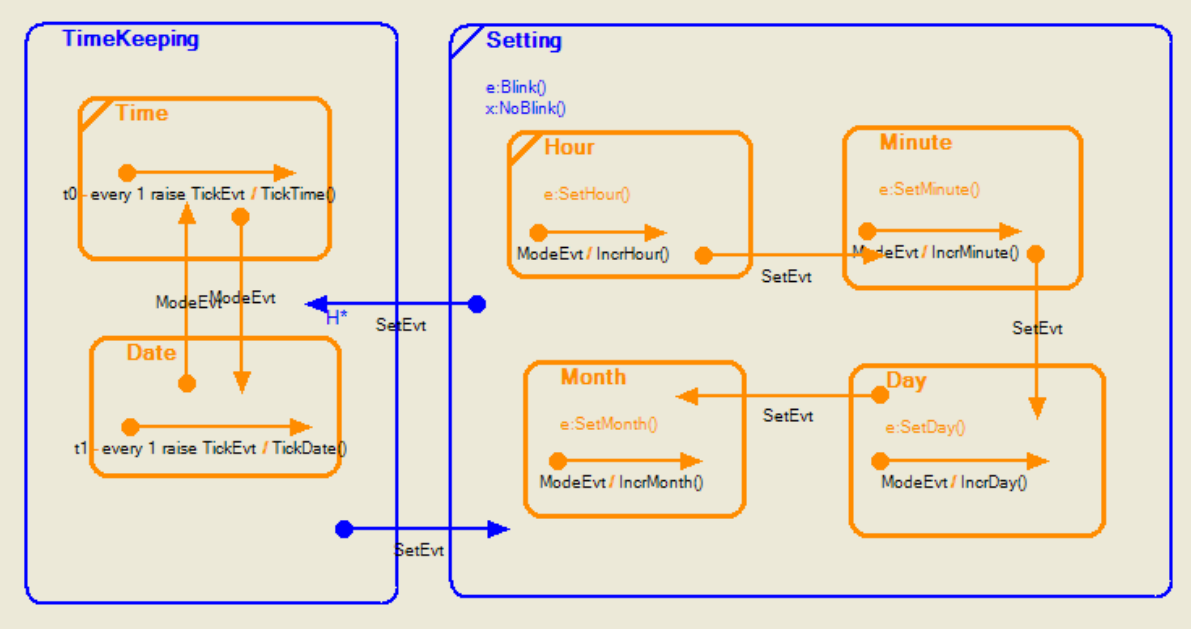


Illustration 21: Sample Watch - Fill out more of the actions.

1. Note the timeout “every” expression.
2. Note the entry action.

Here is a sample of the application running in Setting::Hour mode under this new state machine. Note that extra work is done in the gui to achieve the actual blinking effect.

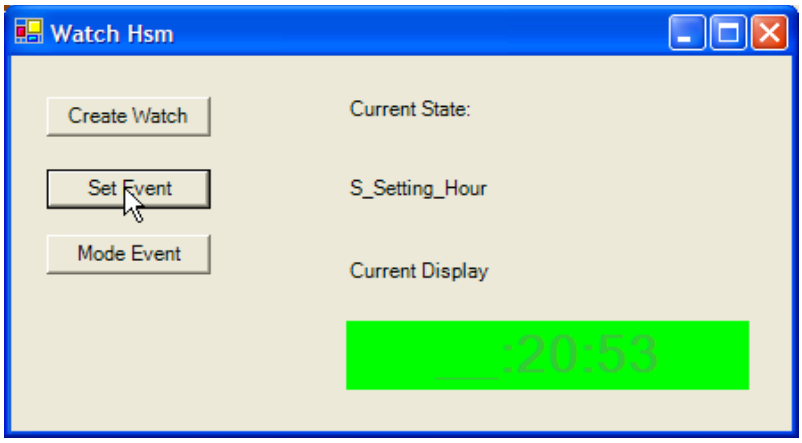


Illustration 22: Blinking Effect while in Setting::Hour state

Conclusion, for now...

Look at the sample code provided with this document.

Shortcuts for StateProto

Alt-S: Save As
Alt-O: Convert To Code (remember - this will use the saved cs source file if it exists - so save it in the Visual Studio IDE if you've been working on it). Also, unless this is a quick code translation check - remember to save the model first.
Alt-L: Load an existing model.
Alt-N: Replace the current model with a new model and designer view.
Alt-A: Change the current selector to the Arrow pointer ("select single item and move" selector)

If state is selected:

Ctrl-N: Name the state
Ctrl-T: Toggle default/start state
Ctrl-F: Toggle final state
Ctrl-E: Set an entry action
Ctrl-X: Set an exit action

If a transition is selected:

Ctrl-V: Set the event signal
Ctrl-G: Set the guard
Ctrl-A: Set the transition action
Ctrl-K: Set the event source port name
Ctrl-N: Set the transition's name