Quick Introduction to IAR visualSTATE®

Integrated tools for:
• State machine design
• Code generation
• Testing
• Additional features

Getting Started:
• Tutorials
• Example projects
Designing state machine models

- State machine models are commonly used for describing discrete systems, where the current behavior is a result of previously occurring events.
- A state machine consists of a hierarchy of states and transitions between the states, which you create by drawing state machine diagrams. The diagrams provide you with a high-level view that helps you manage the complexity of your application.
- State machines allow you to develop your embedded application in a natural, iterative way where states in the machine correspond to states in your application. A transition represents a change from one state to another. The system can change states depending on input from the environment, typically when an event occurs. Transitions can perform actions on the environment. For example, when the LoadKey event occurs, the action Open is performed, which causes the state machine to change state from Closed to Open.

```
1. evLoadKey() / Open()
2. Close()
```

- visualSTATE can be used for modeling hierarchical state machines as described in the UML standard. Thus, a state machine can contain other state machines.
- A state machine can only be in one state at a given time—the states are mutually exclusive. However, if your system must be in more than one state at a time to manage concurrency, you can organize states in separate regions. This way, your system can be in more than one state at a given time.
- visualSTATE provides mechanisms for reuse of design:
  - State machine templates and submachine states
  - System instances
visualSTATE Designer

- Design tool—specialized on state machine models
- Simple to use

Project Browser window where you get an overview of your system organization

Diagram toolbar where you pick the design elements for your state machine diagram

State machine diagram window where you design your state machine model
Code generation from
state machine models

• Automatic code generation from your state machine models.
• State machine concurrency is taken care of automatically.
• Your design and code are always synchronized.
• The generated code can be executed on any platform for which a standard C (or C++) compiler is available.
• The generated code does not require any compiler-specific features.
• Support for using compiler-specific keywords to place state machine code and data in the memory areas of your choice.
• The generated code can also be used with a real-time operating system.
• Size-of-data entities can be forced to 16 or 32 bits to match your target architecture for speed purposes.
• Simplified maintenance because you can focus on the design instead of implementation details.

• Target-independent code
• All generated code constructs adhere to ISO/ANSI C
• Support for C++
• API for readable or table-based code
The workflow for generating code and integrating it with your own source code using an API is straightforward:

- **Design and simulate:**
  - **Select API**
  - **Generate code for the API**
  - **Set up the compiler project with appropriate file structure**
  - **Use the API functions to initialize and handle your state machine**
  - **Write the rest of your code**
  - **Compile your project**

- The API provides an interface between the Coder-generated code and your own code. visualSTATE comes with two APIs, with different support for table-based code or readable code, C/C++ support, and hardware debugging support. Choose the API that suits your needs.
- **Set up for code generation in visualSTATE:**
  1. **Select the project**
  2. **Choose which API to generate code for**
  3. **For the Adaptive API, optionally choose Readable or C++ code generation**

- You can configure the Coder in many different ways to balance the needs of the target MCU, the compiler, and coding standards.
- A final application will consist of:
  - The actual application based on the state machine model
  - The API files for the execution engine
  - A set of Coder-generated files
  - Action functions which you implement and which are called by the state machine model.
Tools for testing

visualSTATE Validator

• Simulate your state machine model for functional testing

• Graphical animation—animate your debug session graphically while simulating using the Validator, or while debugging on hardware using C-SPYLink or RealLink. When you send an event in the Validator that fires a transition, the affected states and transitions can be viewed in the Designer.

• Automatic simulation—test your model automatically by applying a test sequence of events and assignments that you have recorded earlier to a sequence file.

• Tracing—to obtain a sequence of events that will get the system into a desired configuration. Tracing can be used for answering the question *How do I get from the initial state to a specific state configuration?*

• Analyze your state machine model with regard to element use and test coverage—static analysis and dynamic analysis, respectively.

• Record one or more test sequences to a sequence file, which can be used as a source of reference in future simulation sessions, for example after changes in the model design.

• In addition to using the Validator for simulation, you can also use it for testing your state machine model in a target application by means of RealLink.

When an event has been sent, various elements will be affected. In the Validator you can view the reactions, and control the execution by using breakpoints.
• visualSTATE Validator for simulation and functional testing
• visualSTATE Verificator for formal verification
• C-SPYLink and RealLink for testing on hardware

**visualSTATE Verificator**

• Formal verification for checking the logical consistency of your state machine model:

  - Many different checks for complex properties to choose from, such as local dead ends.
  - Complete animation of models with large state spaces.
  - Computes traces that show how a model might reach a state in which a warning or error condition holds true.

**C-SPYLink**

• Integrates with C-SPY in IAR Embedded Workbench.
• C-SPY provides support for debugging on hardware.

**RealLink**

• Requires a UART to communicate state machine data between the Validator and the target.

Both C-SPYLink and RealLink let you debug on hardware with feedback directly in the state machine diagram—to see exactly which state configuration is active and which transition that was taken to enter that state configuration.
Additional tools and features

- To document your visualSTATE projects, create a project report using the visualSTATE Documenter.
- Use the built-in support for Altia Design to test how your state machine model interacts with a graphical user interface.
- View all your state machine models without having access to the visualSTATE product; useful for showing design ideas to those without a license.
- Use visualSTATE remotely via the Control Center.
- Transfer state machine models between visualSTATE and tools from other vendors by using the XMI® file format.
- Use the visualSTATE State Machine API for programmatic manipulation of models for various programming languages that support calling C functions.
- Comprehensive documentation in a User Guide (PDF) and an online help system; press F1 to get context-sensitive help.

Getting Started

- Tutorials give you a smooth start with visualSTATE.
- Example projects for different example designs, sample applications, and reusable code for mechanisms such as event queues and using the visualSTATE APIs.

Finally

- For system requirements, see the Release notes on your product installation media.
- For licensing information, see the Installation and Licensing Quick Reference booklet in the product box.
- For technical support, see www.iar.com/support.