

[TracNav](#)

- [JPFWiki](#) - Welcome Page

[Introduction](#)

- [What is JPF](#)
- [Testing vs model checking](#)
- [Random Example](#)
- [Race Example](#)
- [JPF classification](#)

[Installing JPF](#)

- [System requirements](#)
- [Download snapshots](#)
- [Download repositories](#)
- [Create site.properties](#)
- [Install NetBeans IDE plugin](#)
- [Install Eclipse IDE plugin](#)
- [Building and testing](#)

[User Guide](#)

- [Application Types](#)
- [JPF Components](#)
- [Configuring JPF](#)
- [Running JPF](#)
- [JPF Output](#)
- [The JPF API](#)

[Developer Guide](#)

- [Design](#)
- [Choice Generator](#)
- [Partial Order Reduction](#)
- [Attributes](#)
- [Listener](#)

[MJI](#)

- [Mangling for MJJ](#)
- [Bytecode Factory](#)
- [Logging](#)
- [Report](#)
- [Embedded](#)
- [JPF tests](#)
- [JPF project layout](#)
- [Create a JPF project](#)
- [Coding Conventions](#)
- [Hosting update site](#)

[Projects](#)

- [jpf-core](#)
- [jpf-actor](#)
- [jpf-awt](#)
- [jpf-awt-shell](#)

- [jpf-concurrent](#)
- [jpf-cv](#)
- [jpf-delayed](#)
- [jpf-guided-test](#)
- [jpf-mango](#)
- [jpf-racefinder](#)
- [jpf-rtembed](#)
- [jpf-statechart](#)
- [net-iocache](#)
- [jpf-aprop](#)
- [jpf-numeric](#)
- [jpf-symbc](#)
- [jpf-concolic](#)
- [jpf-symbc-load?](#)
- [jpf-extended-test-gen](#)
- [jpf-parallel-spf?](#)
- [eclipse-jpf](#)
- [netbeans-jpf](#)
- [jpf-inspector](#)
- [jpf-shell](#)
- [jpf-template](#)
- [jpf-trace-server](#)
- [standard NB example](#)
- [Summer Projects](#)
- [External Projects](#)
- [Change\(B\)log](#)

[About](#)

- [About this Wiki](#)
- [About the Mailing Lists](#)
- [About the Development Process?](#)
- [About the Repository?](#)
- [How to Contribute](#)
- [JPF contributor account](#)
- [Events](#)
- [Presentations](#)
- [Papers](#)
- [FAQ](#)
- [History?](#)
- [Support](#)
- [People?](#)
- [Playground](#)
- [Table of Context](#)

jpf-statechart Guidance Scripts

Guidance scripts can be used by JPF to define event sequences to explore. The purpose of guidance scripts is to enable verification and testing of incomplete models. They are used to drive the state machine into a specific target state, either to test correct transitions (in case of simulation), or to start state space exploration from there (in case of model checking).

Grammar

```

script ::= {section | sequence}.
section ::= 'SECTION' ID {' ID' {' {sequence} '}'.
sequence ::= iteration | selection | event.
iteration ::= 'REPEAT' [NUM] {' {sequence} '}'.
selection ::= 'ANY' {' {event} '}'.
event ::= ID ['(' [parameter {' parameter}]]').
parameter ::= STRING

```

All parameters are treated as strings, but string literals preserve the double quotes.

Event ID can contain any of the chars #.:@\$* (e.g. for further parsing of targets etc.)

An asterisk '*' event ID represents an "any event", which means it is expanded at execution time into a set of events that are obtained from inspecting all trigger methods of all active states.

The keyword ANY precedes a set of set of state names or an "*", and means the model checker will try all choices. If used in the simulator, a random element of the set will be selected.

Parameters of events can contain regular expression terms, which are expanded into a set of all possible combinations:

```
e("one"|"two", [12])
```

is expanded into a set of events

```
e("one",1), e("one",2), e("two",1), e("two",2)
```

Empty parameter lists "()" can be omitted.

".." and "*".." comments are ignored.

Sections can be organized hierarchically, i.e. potential sequences for an active state are looked up recursively.

Example 1

```
a b
```

will produce two corresponding events:

```
a(), b()
```

Example 2

```

SECTION s1 {
  a
  b
}

SECTION s2 {
  ANY {e, f(true|false), g}
}

```

will be expanded into the following event sequences

```

if (activeState == s1) => a(), b()
if (activeState == s2) => {e(), f(true), f(false), g()}

```

Example 3

```
SECTION superState {  
  a b  
}  
  
SECTION superState.subState1 {  
  ANY {*}  
}
```

means all substates of 'superState' except of 'subState1' will process the sequence

```
a(), b()
```

and 'subState1' will process all events for which it has trigger methods defined (i.e. systematically checked)