## **Simkit Quick Reference**

Simkit event graphs can be formally described using diagrams depicting state variables, events, event schedules and componentization. Excerpts follow from *Discrete Event Simulation (DES) Modeling*, Arnold Buss, 2022.



* Circles: events A, B correspond to state transitions when fired by an arriving event
* Directed Edge: event A schedules event B to occur after time-delay ***t*** if boolean condition (i) is true



1. State variable **N** is available to all events in the Event graph.
2. The schedule of upcoming events is maintained on the Event List.
3. **Run** **event** is automatically scheduled and commences upon program start.
4. **Run** **event** first sets value of state variable **N** to 0.
5. **Run** **event** schedules **Arrival** **event** to occur after delay sampled from random variate sequence **tA**
6. **Run** **event** is complete.
7. **Arrival** **event** commences when scheduled on event list.
8. **Arrival** **event** sets value of state variable **N** to value computed by N + 1
9. **Arrival** **event** schedules another **Arrival** **event** to (again) occur after time delay sampled from random variate **tA,** with that future event scheduled on the Event List.
10. **Arrival** event has no further steps and is complete.
11. Next event found on Event List is commenced.
12. No overall termination condition explicitly noted.



1. **Event A** cancels next-scheduled **Event B** if boolean condition (i) is true
2. **Event A** has no further steps and is then complete.
3. Next (different) event found on Event List is commenced.



1. **Event A** schedules **Event B** to occurafter delay value of *t* if boolean condition (i) is true, passing internal value **j** as an event parameter from **Event A** as part of this event getting scheduled on Event List.
2. When **Event B** is the next event found on Event List, then it is commenced. Event B accepts passed-parameter j as internal value (k).
3. Event B then executes any internal state-transition code, schedules subsequent events if any (none shown in the example), and is complete.
4. Next event found on Event List is commenced.



1. If **Event A** cancels **Event B** with an accompanying passed parameter **j**, then only the next scheduled occurrence of Event B with a matching input parameter is cancelled.



1. **Event A** schedules **Event B** to occurafter delay value of *t* if boolean condition (i) is true, with priority **p** for breaking ties among events scheduled to occur at a single timestep. This is helpful if execution of one event is dependent on state-variable changes by another event. Allowed values for priority **p** are default simkit.priority.DEFAULT, HIGH, HIGHER, HIGHEST, LOW, LOWER and LOWEST.



1. When surrounded by a box, a smaller event graph becomes a component that can be composed in a larger event graph.
2. An Event Graph Component is simply an Event Graph “in miniature” whose description is a template for instances of the component.
3. Each Event Graph Component instance retains it original initiating parameters and hojds a unique set of internal state variables, all of which are distinct from parameters and state variables in other similar Event Graph Components.
4. All connected components share a common Event List.



1. The forked edge illustrates how a **Listener Component** pattern can hear all events occurring in the **Source Component**.
2. First a Listener Component must register interest in the **Source Component** in order to setup necessary event-passing connections.
3. As each Event in the **Source Component** finishes processing state variables and scheduling other local events, a copy of the completed event is passed to the **Source Component**.
4. One component cannot listen to the initiating **Run Event** in another component.
5. This pattern is sometimes called the Listener Event Graph Objects (LEGO) Framework.



1. The Adapter pattern allows a **Source Component** output **Event A** to be referred to by a **Listener Component** as received **Event B**.
2. Other events in the **Source Component** are not passed to the **Listener Component**. This is a useful filtering mechanism to reduce the number of outputs from a Source to a Listener.
3. **Source Component** output events must have the same parameter signature as the corresponding event defined in the **Listener Component**.



1. A common pattern is to have two components listen to each other via Adapter connections.
2. Avoiding naming collisions and potential confusion between corresponding events is a good practice.



1. A **PropertyChangeListener Component** receives notifications when the value of a state variable changes in a **SimEntity Component**.
2. State variable changes from within the **SimEntity Component** are dispatched as PropertyChangeEvents.
3. If no specific state variable is named, then all **SimEntity Component** state variables are accessible.
4. PropertyChangeEvents do not interact with the event list.
5. PropertyChangeListeners are useful for monitoring, debugging and statistics.
6. PropertyChangeListeners should never be used for actual dynamic behavior modeling being performed by the Event Graphs themselves.
7. TODO perhaps a different shape is better for PCLs rather than a box (which is used by components)?



1. The **PropertyChangeListener** only receives value updates from a single named **aStateVariable**.
2. This is a useful filtering mechanism to reduce the number of outputs from a Source to a Listener.

**Questions**

1. How to best annotate events or PCLs with descriptions?
2. How to best show code blocks?
3. Is there a *shutdown* or *finally* event?