



# Gedae 5.0.2 Release Notes

September 2006

Address: Gedae, Inc.  
1247 N Church St, STE 5  
Moorestown, NJ 08057

Telephone: (856) 231-4458  
FAX: (856) 231-1403  
Internet: [www.gedae.com](http://www.gedae.com)

This is a minor release in which three experimental features have been included. Three bugs have also been fixed. See the Gedae 5.0.1 release notes for the rest of the new features and bug fixes in Gedae 5.0.2.

# 1 New Features

## Sanitizing Trace Table

A new capability has been added to Gedae to allow a user to sanitize a Trace Table. Gedae provides the ability to save a Trace Table to a file which can be loaded from Gedae without having the original graph. This feature allows users to send Trace Tables to Gedae Inc. to help with analysis of dataflow problems. However, occasionally some of the names of subgraphs or primitives in the Trace Table are classified and cannot be freely distributed. Also the timescale at which things happen may be classified.

To sanitize a trace table the user must create a synonym file. The file consists of lines of the form

```
<classified string> <replacement string>
```

Where the `<classified string>` is the name in the trace table to be replaced and the `<replacement string>` is the string that will replace it. If the classified string appears as substring within a name appearing on the trace table then that substring is replaced with the replacement string.

For example the line

```
CfarAlg Xyzzy
```

Will cause the names

```
CfarAlg, CfarAlg_1, HiResCfarAlg
```

 to be replaced with

```
Xyzzy, Xyzzy_1, and HiResXyzzy
```

 respectively.

Lines beginning with a # sign are commented out of the synonym file. There is one exception. Lines of the form:

```
#Scale <factor>
```

Multiply all times in the trace table by <factor>

For example the line:

```
#Scale 3145
```

will cause all of the times to be multiplied up by a factor of 3145.

## User Events

A new API has been developed that allows user to add trace events from within a primitive Apply method. Each event is numbered. The user can add events that appear on the trace table as tick marks and these events can be added with an additional integer or floating point value. The user can add one level of interval events (events with a begin time and end time) that appear on the trace table as

### Prototypes:

```
/*** USER TRACE EVENT FUNCTIONS ***/  
void embUserEvent(int number);  
void embBeginUserEvent(void);  
void embEndUserEvent(int number);  
void embUserIntEvent(int number, int x);  
void embUserFloatEvent(int number, float x);
```

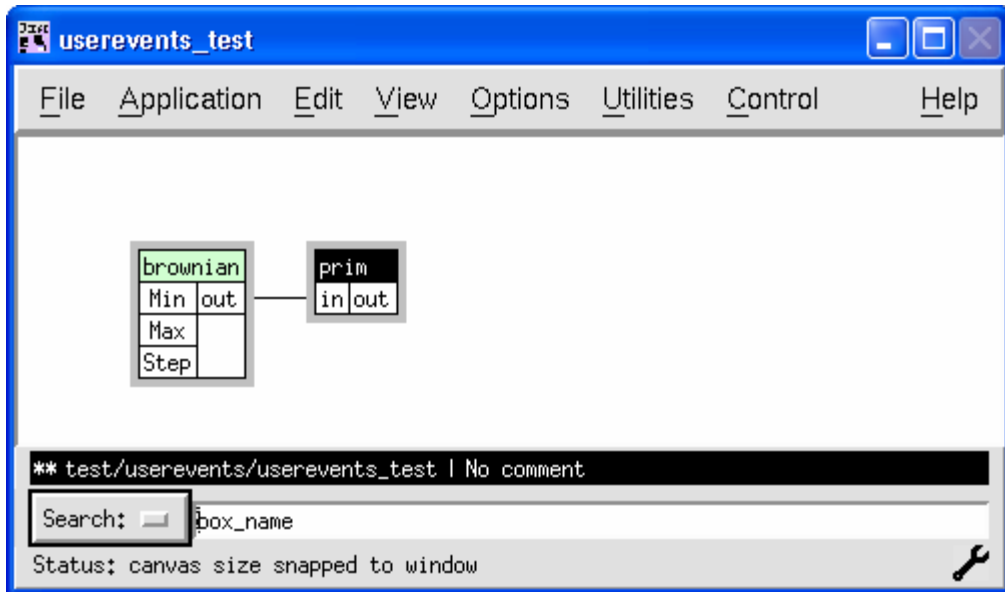
### Description:

The embUserEvent, embUserIntEvent and embUserFloatEvent functions all create a timestamped event. Each event has a user set number that typically is used to identify where in the primitive the event was created. In addition the embUserIntEvent and embUserFloatEvent take an additional value that is recorded with the event.

The embBeginUserEvent and embEndUserEvent provide one level of interval events. The embBeginUserEvent takes no parameter and will be paired with the next call to embEndUserEvent that provides the event number. Nesting of interval events is not currently supported. To record nested intervals the user should call embUserEvent twice to mark the beginning and ending of the intervals.

### Example:

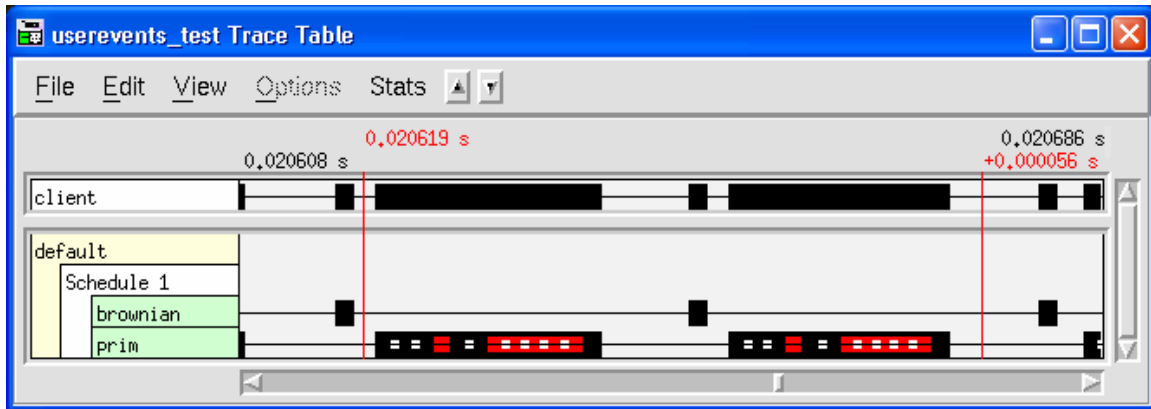
The prim function in the following graph creates each type of user event:



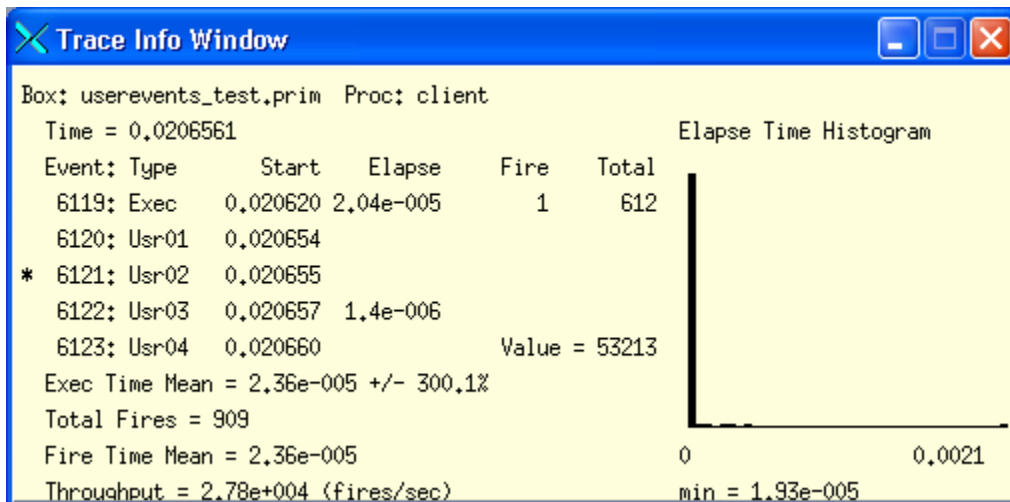
The Apply method for prim is:

```
Apply: {
  embUserEvent(1);
  embUserEvent(2);
  embBeginUserEvent();
  embEndUserEvent(3);
  embUserIntEvent(4, cnt);
  embBeginUserEvent();
  embUserFloatEvent(5, 3.7);
  embUserFloatEvent(6, 3.7);
  if (cnt > 10000) {
    embUserEvent(7);
  }
  if (cnt > 10001) {
    embUserEvent(8);
  }
  cnt++;
  embEndUserEvent(9);
}
```

The firing of Prim produces the following trace table.



Clicking on the second white dot in one of the firings of `prim` shows the following display with more detailed information about the user events. From this display the user can read the even number, the elapse time (if it is an interval event) and the events value (if any).



## Setting Break Points

This is the least filled in of the three features. The goal of this feature is to be able to set breakpoints at any even that would normally be traced. Currently the ability to set breakpoints on user events has been implemented.

To set break points the user must load a break point file. This can be done either by selecting the menu item `Application->Open Breakpoints` or by loading the breakpoint file from the command line interface using the `-bkpts` command line argument. For example:

```
gedae -fi demo/comm./e_comm -pa default -bkpts e_comm_bkpts
```

An example breakpoint file is shown below:

```
type=user
box=prim type=user
type=user number=1
box=prim type=user number=4 value=20000
box=prim type=user number=8 value=(...,10]
box=prim type=user number=4 value=(30000,30002)
box=prim type=user number=4 value=[40000,40002)
box=prim type=user number=4 value=(50000,50002]
box=prim type=user number=4 value=[60000,60002]
box=prim type=user number=4 value=[70000,...)
```

The format of the file is to describe an event by a series of qualifiers. The first entry into the file says to stop on any user event. The second entry says to stop in user events for the box prim. The third entry says to stop on just user events have a user set number of 1. The next 7 events show the format for stopping on user events that have a particular value.

```
value=20000          stop if value = 20000
value=(...,10]      stop if value is <= 10
value=(30000,30002) stop if 30000 < value < 30002
value=[40000,40002) stop if 40000 <= value < 40002
value=(50000,50002] stop if 50000 < value <= 50002
value=[60000,60002] stop if 60000 <= value <= 60002
value=[70000,...)   stop if value >= 70000
```

In future release of Gedae the event file will be extended to handle different types of box, queue and schedule events. It will be possible to set events for an entire box class. Range tests will be available for things like number of tokens in queue, amount consumed, granularity of box firing, time of execution, total number of firings, and many other things. The format for expressing the range will be as shown above for the user event value.

When a break point is hit the debug dialog pops-up showing the box that has caused the break. The stop toggle button is pressed and the user can either Step execution or continue by turning off the Stop toggle. The user can bring up the trace table and examine variables.

## 2 Bugs Fixed

### **SCR2377 Saving the Trace Table causes Gedae to segfault.**

Saving the Trace Table causes Gedae to segfault. This problem was introduced in version 5.0.1 and fixed in version 5.0.2.

### **SCR2372 Setting partition of graph containing a typedef segfaults**

Setting the partition of a graph that contains a typedef box causes Gedae to segfault. This problem has been fixed.