



Gedae 4.5.2 Release Notes

August 2004

Address: Gedae, Inc.
18000 Horizon Way, Suite 200
Mt Laurel, NJ 08054
Telephone: (856) 231-4458
FAX: (856) 231-1403
Internet: www.gedae.com

1 New Features

Case 1170: Debugging Support

This case was previously reported in the Gedae 4.5.1 Release Notes. At that time, several new functions that are callable from a symbolic debugger were described including:

```
int lastClosureIndex(void) :  
    returns the index of the closure that just fired or is firing.
```

As of this release, a translation table is output as a file to allow the user to look up the primitive box element (instance) in the flow graph that has the given index. The file containing the table is named: `indexToBox-<part>`. The table is being created for the partition `<part>`. The file is located in the directory where the `exec-<part>.c` file is created. If the application is a launch package, then the table will be in the launch package directory. If the application is run from the development environment, then the table will be in file `embedded/<graph_name>_<serial_no>` directory where `<graph_name>` is the toplevel name of the graph and `<serial_no>` is an integer value.

An example translation table is:

```
Index: Name  
-----  
0: modulator.x_osc.monitor_1  
1: modulator.x_osc  
2: modulator.fft_filter.dyndeq_1  
3: modulator.fft_filter  
4: modulator.x_mult  
5: modulator.x_split  
6: modulator.x_split.send_2  
7: char_encode.unpack.recv_1  
8: char_encode.unpack  
9: char_encode.addstart_stopbits  
10: char_encode.diff_encode  
11: modulator.oqpsk_mod
```

Using this example, if the `lastClosureIndex` function returns 7, then the user knows that the last primitive box element to fire was the `char_encode.unpack.recv_1` box.

Case 1179: Launch Package Compression Enhancements

In preparation for supporting compression of target executables that have been stored as binary arrays in the command program, the BSP support function `embLookupBinaryExec` has been modified. This change only affects developers of BSPs that support code generated executable arrays. The modification to the function is the addition of a third parameter. Previously the function had the prototype:

```
void *embLookupBinaryExec(char *programe, int *nbytes);
```

Now the prototype for the function is:

```
void *embLookupBinaryExec(char *programe, int *nbytes, int *uncompressed_bytes);
```

where

`programe` – is the name of the executable and has the form `exec-<part>` where `<part>` is the name of the partition.

`nbytes` – is the number of bytes the uncompressed executable requires.

`uncompressed_bytes` is the number of bytes in the array returned by `embLookupBinary`, if the image was compressed. If uncompressed, then the value is zero.

If the image was not compressed, then the function returns a pointer to the executable binary image of size `nbytes`. If the image was compressed, then the function returns a pointer to an array of size `uncompressed_bytes`; the `startProc` function must allocate an array of size `nbytes` and uncompress the return value to form the actual binary image.

The method for specifying the compression algorithm used by a given BSP is still under development.

Case 1202: Modify `embSetPeriod` to Set Policy to Dataflow If 0 timeout Entered

The function `embSetPeriod` previously could set the scheduling policy to `timeout` but could not set it back to `periodic`. The primitive can now set the policy back to `dataflow` by calling `embSetPeriod` with a parameter of 0.

Case 1203: Segmented Uncontrolled Merge

Prior to Gedae 4.5.1 any schedule in a segmented subgraph needed to have at least one static input. This requirement has been changed so that schedules consisting of single primitives with only nondet inputs can now be part of a segmented subgraph.

For example, dynamic distribution based on output queue capacity requires the ability to merge results using an uncontrolled merge. If this is to work in a segmented subgraph, then uncontrolled merges need to work there also. Uncontrolled merges by their nature have only nondet inputs without any static controlling input.

In order to allow segmented primitives with only nondet inputs, the runtime kernel was modified. The enhanced kernel will put a schedule in the end-of-segment state if either the amount function fails on a draining queue (one that has an end-of-segment marker) or if `embSuspendQueueWait` is called when all input segment queues are draining. Shown below is a correctly written uncontrolled merge function (from `embeddable/stream/logic/umergef`) that can be placed in a segmented subgraph.

```
Name: umergef
Type: static
Comment: "Uncontrolled merge"
Input: {
    nondet stream float [N]in;
}
Output: {
    dynamic stream float out(N);
}

Apply: {
    int T = 0; /* total amount of data available on all input
queues */
    int i;
    /* get input queues ready and calculate value of T */
    for (i=0; i<N; i++) {
        int ain = avail(in[i]);
        T+=ain; /* update T */
        if (ain) amount(in[i],ain); /* prepare in[i] */
    }
    if (T) {
        /* then the box is ready to fire */
        int j;
        for (i=0; i<N; i++) { /* for each input */
            int ain = avail(in[i]);
            e_vmov(in[i],1,out,1,ain);
            out+=ain;
            consume(in[i],ain);
        }
    }
}
```

```

    }
    produce(out, T);
  } else {
    embSuspendQueueWait("waiting for input");
  }
}

```

If the `umergef` determines that no data is available on the input queues, then it calls `embSuspendQueueWait` to say that it won't fire until data becomes available on some queue. In addition, if all the input queues have end-of-segment markers on them, then the call to `embSuspendQueueWait` will put the `umergef` schedule in the end-of-segment state.

A primitive can now prematurely abort processing as soon as it observes that an eos marker has been placed on a queue or queues. A new built in function, `draining`, can be called on a queue to see if an end-of-segment marker was placed on the queue. Using this information, in addition to any other available information (such as internal state), the primitive can choose to force the end-of-segment processing by calling a second built in function – `drain` – on the queue. While several queues may be draining, the `drain` function needs to be called on only one of them to force the end-of-segment.

Below is an example of a segment of code from an `Apply` method that drains a queue.

```

Input: {
  nondet stream in;
}

Apply: {
  ...
  if (draining(in) {
    drain(in);
  }
}

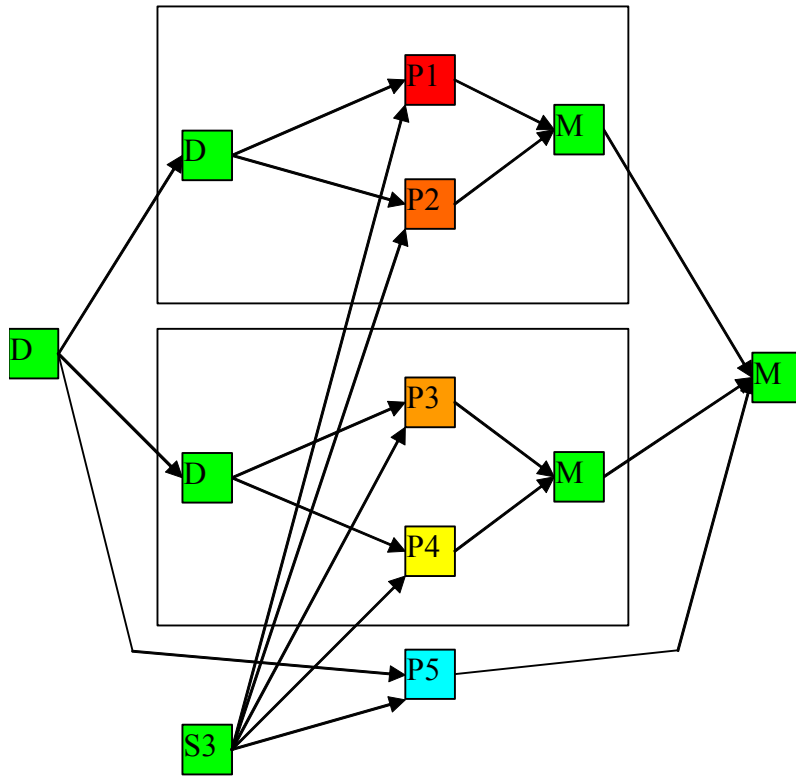
```

Case 1207: External State No Longer Needs to Go to Every Mode

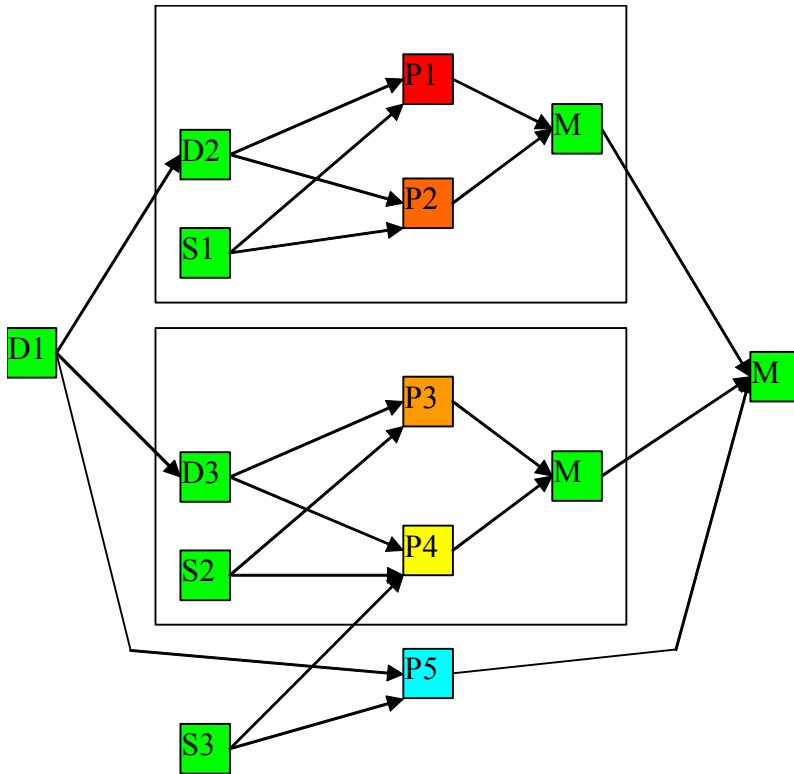
A previous limitation in the use of distributed external state was that the external state had to be used by every segmented subgraph controlled by an exclusive output tree. This limitation has been removed.

A couple of examples will help illustrate the limitation. In the examples, boxes marked D are exclusive branches. Both examples have an exclusive branch tree with three boxes. Boxes marked P are subgraphs controlled by the exclusive branches. Boxes marked S are external state boxes. Boxes that are the same color are mapped to the same processor.

The first example shows a legal use of external state in Gedae 4.5.1. In this example, a tree of exclusive branches – all mapped to one processor – distribute control to processing blocks running on different processors. The example is legal because the state box S3 is connected to a state input in all five subgraphs.



The next example was not legal in Gedae 4.5.1. In this example, the three state boxes are not connected to all five subgraphs, which are controlled by the exclusive branch tree. An undesirable workaround for state S3 is to connect the state to dummy inputs in subgraphs P1, P2 and P3. This workaround is inefficient because it causes state to be transferred unnecessarily. The problems with states S1 and S2, however, cannot be fixed. Because S1 is in a subgraph controlled by D1 it cannot be sent out to P3, P4 or P5. If the state S1 was brought up to the toplevel of the graph (at the same location as S3), then it would be possible to distribute it to dummy inputs but it would change the behavior of the graph. In the graph above, state S1 is reset every time a new segment is output from D1; but if S1 were brought to the toplevel of the graph, then it would only be reset when the graph started up.



The above graph is now legal as of Gedae 4.5.2. When control is passed to a subgraph that does not use the state variable, the state information remains on the processor that last used it. When control is later passed to a subgraph that requires the state, it is sent from the processor that last used it to the next mode that needs it.

Case 1126: Setting Internet Protocol Using the Linux eredhat BSP

It is now possible to set the Internet protocol used for communicating between target eredhat processes. The default Internet protocol is TCP/IP, but the user can now modify this default. To use the feature for setting the Internet protocol, you'll need to know what number corresponds to the protocol you want to use (usually found in a .h file in the /usr/include directory). The easiest way to change the protocol is to edit your embedded_config file and change the Processor_Types section from:

```
Processor_Types: {
redhat: {
Type: "eredhat"
Make_Params: "eredhat/runtime_make_info"
Info: ""
Memory_Desc: {}
}
}
```

To:

```
Processor_Types: {  
redhat: {  
Type: "eredhat"  
Make_Params: "eredhat/runtime_make_info"  
Info: "-family 17"  
Memory_Desc: {}  
}  
}
```

Where 17 should be replaced with whatever number corresponds to the protocol you want to use. If the family field is not set, then the family will default to the normal value.

2 Bugs Fixed

The following is a list of the bugs that have been fixed.

Case 1193: Boxes That Can Fail and Have Parameters Driven by Encoded Streams Do Not Work

A box that can fail during execution (calls functions `embSuspendRetry`, `embSuspendQueueWait` or has a nondet queue that isn't satisfied) and also has parameters driven by encoded streams will not work correctly. The problem is that the parameter value will be removed from the stream prior to firing the box. When the box is resumed after the failure, the next parameter value will be taken from the stream.

Case 1201: Gedae Goes to Sleep Unnecessarily

Some hosts such as NT support the ability for the Gedae process to go to sleep, which frees the CPU for use by other processes; however, on such hosts, it was possible that Gedae could go to sleep unnecessarily, slowing execution. This problem has been fixed.

Case 1205: Segments Get Out of Order

Observed graph in which graph moves from segment 3 to 4 and then back to 3. The problem can occur when a box has two static inputs (call them A and B) and one nondet input (call it C). The nondet queue C has segments up to 4 on it. A gets a null segment 3 (begin and end with no tokens). At the end, the C input drains and goes to the next segment - 4. Later on B, a null segment 3 arrives. This arrival of a null segment changes the segment number of the schedule back to segment 3. It does not necessarily cause a segfault but does cause segments to get mixed up. This problem has been fixed

Case 1211: Problems with Subscheduling

The following three problems have been fixed:

1. after scheduling, bringing up the autosubschedule tool crashes Gedae
2. gain sets for gainHier table are not created correctly at dynamic boundaries
3. send primitives are put into the wrong subschedule, if the destination is dynamic

Case 1212: Embedded Box Initialization Method Serial Number Causes Undefined Symbol Problem

To limit a code-generated function name to a maximum number of characters, a serial number replaces the directory path that the box is in. These serial numbers are maintained in the file `object/gedae_library_serial_no.dat`. An example file is:

```
widget
discrete/stream
embeddable/stream/complex
embeddable/stream/host
discrete/integer
demo/comm
embeddable/stream
embeddable/vector/complex
embeddable/comm
embeddable/stream/source
internal
embeddable/vector
```

In the above example file the directory `internal` is the eleventh entry, so the `internal/send` box will be generated with an initialization procedure named `I0011_send`.

When compiling new boxes a problem can occur where different serial numbers are used to create and call the box's initialization function. For example, the problem manifests itself with a message like:

```
undefined symbol I0012_send
```

when the box initialization function is declared in the `object/internal/send.c` file as `I0011_send`. The problem is that the code generator and development environment use different serial numbers for the same directory name. This problem has been fixed.

Case 1219: Save Trace Table Crashes

Saving the Trace Table causes Gedae to crash with a segfault. This problem has been fixed.

Case 1124: Hardcoding path C:\ . . . to be the First Argument to appLaunch or appRead Does Not Work

appLaunch and appRead do not recognize the path on NT beginning with C:\ as an absolute pathname and do not correctly create a launch package directory name. This problem has been fixed.

Case 1125: Segfault When Deleting Dataflow Parameter Shared by Two Groups

Gedae can segfault when deleting a dataflow parameter (a parameter that effects dataflow and requires rescheduling when the value is changed) shared by two groups. This problem has been fixed.

3 Known Bugs

The Known Bugs are the same as those reported for Gedae 4.5. No new bugs have been added to this list.

Case 1024: Accommodate Different Exceed Versions in `initGEDAE`

In order for Gedae to work with Exceed/XDK 9.0, `makeGEDAE` has to be modified.

Currently users must view the FAQ for 7.1 changes

http://www.gedae.com/SUPPORT/FAQ/exceed_7_1.html

and then remove `xlibcon` from `makeGEDAE` and `ent` files mentioned in the FAQ.

Case 1026: Broadcast Transfer Mechanisms

Currently all Gedae data communication is point to point; however, many target processors support efficient broadcast mechanisms. The goal of this task is to extend the Gedae BSP API to allow broadcast data transfers to be part of the BSP. A second goal is to fix the DSA mechanisms that cannot be implemented when there is fanout (see Case 1149). Rather than fixing that problem directly, a broadcast DSA capability will be implemented.

Case 1056: FGU of Hierarchical Typedef Boxes

FGU does not transfer hierarchical typedef boxes correctly. The typedef used to define the input of the box is set to the old directory rather than the new.

Case 1057: Graph Stalls

A rare condition can cause a graph to stall (or segfault) when the controlled static schedule is partitioned to two processors in the following form:

A->B->A

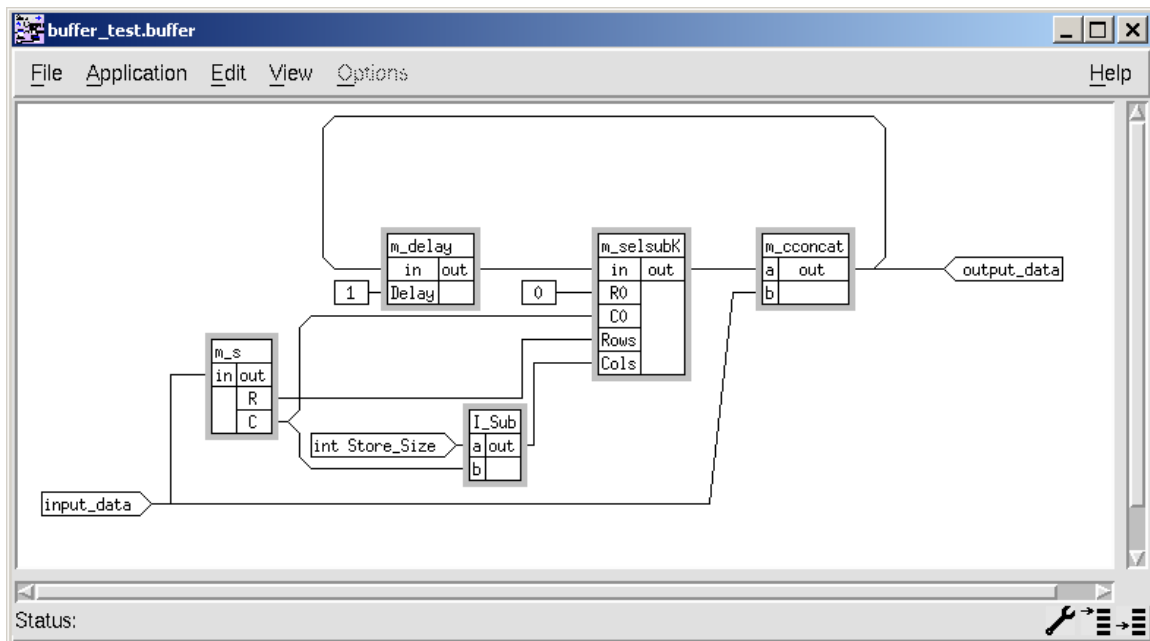
The problem scenario is that the schedule is partitioned into three parts, with the first and last parts mapped to the same processor. Usually Gedae puts the parts mapped to processor A in the same static schedule; however, to allow efficient pipelining, Gedae splits the two parts mapped to processor A into two different static schedules. They are numbered `n.1` and `n.2` (for example `2.1` and `2.2`). To see if any schedules have been

broken into two parts, the user can pop up the Schedule Info Dialog and see if any of the schedule names contain a decimal point.

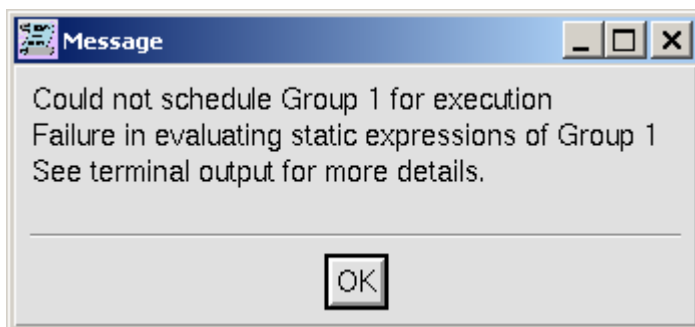
The decimal point in the schedule name does not necessarily indicate a problem. The problem only occurs when the data source driving the processing is faster than the graph, causing the control message queue to back up and overflow. The condition is rare because the problem only happens when the graph is not keeping up with the input data rates.

Case 1071: Dimensions Not Set Correctly

The dimensions are not set correctly in the graph below due to an interaction between the output parameter from the `m_s` box and the propagation of the dimensions around the feedback loop.



The error manifests itself as:



The workaround is to directly set the `m_selSubK` input parameters from graph parameters rather than from the `m_s` output.

Case 1078: Gedae Overwrites Protected Files

Gedae overwrites files that are protected as a result of being checked out of a CMS. This overwriting causes a problem because it defeats the CMS system.

Case 1108: Setting Default Subschedule Queue Capacities Correctly

The queue policy and capacity of a queue feeding a subschedule may not be set correctly.

Case 1122: Embedded Build Can Require a `makeGEDAE CLEAN`

If an application is repartitioned, then the target executables do not get relinked. The problem is that all the `.o` files are older than the targets, and the fact that there is a new link line does not force the target library and target executables to rebuild.

Case 1127: Static Schedule Gain Setting Algorithm Propagates Through Dynamic Queues

If there is a dynamic queue internal to a static schedule, then the gain setting algorithm propagates the gain through the dynamic boundary. Since the decimate/interpolate rates set for such boundaries are not necessarily a reflection of the actual data flow gain, these rates can lead to a report of a gain error in the graph.

Case 1139: Unterminated Comments

Unterminated comments cause the Gedae parser to segfault.

Case 1140: Parser Problem

The Gedae parser does not handle an odd number of quotes (") well.

Case 1141: Problem with Variable Vectors and Delay

If a dynamic variable vector input is preceded by a `vv_delay`, then the graph segfaults during scheduling.

Case 1142: Arrays of Strings Not Allowed

Gedae currently allows string array graph parameters to be declared as:

```
const string X[] = {"hello", "world"}
```

or

```
string X[i] = [i]Y
```

where `Y` is a family of strings.

In either case, the values so declared are not correctly set, and therefore, should be considered illegal.

Case 1143: Inconsistent Data Type Declarations

Gedae aborts when the same stream type is multiply defined. For example, suppose two primitives define data types with the same name but different definitions. If the primitives use these types in their `Input`, `Local` or `Output` sections, then Gedae aborts.

Case 1144: Function `appFree` Memory Leak

A command program running on VxWorks does not free all the resources allocated (memory, sockets, etc). The `appFree` function must release everything allocated. Gedae should automatically generate a call to `appFree` for the standard `exec-host` command program.

Case 1145: External Code Does Not Recompile

`Make` is not called after a successful run, so code changes to code listed in the `Personal_Emb_Obj_List` do not get recompiled. To force the recompile, it is currently necessary to change something from the Gedae GUI - like saving a primitive or toggling the Group "Run On Embedded" toggle off and on.

Case 1146: Large Graphs Fail to Display on Flattened Graph

If a graph is too large, then it cannot be displayed on the flattened graph. That is, if the flattened width or height exceeds the allowable pixmap width or height.

Case 1147: Primitive Cannot Recompile

If a primitive `Input`, `Output` or `Local` section is modified at runtime, then Gedae segfaults when the primitive is recompiled, and the graph is rerun. The user must currently exit Gedae after a primitive `Input`, `Output` or `Local` section has been modified.

Case 1148: VxWorks `embWallclock` Function Misses Wrap

When collecting trace information, the `embWallclock` function timer can wrap without being detected. This failure to detect the timer wrap causes VxWorks processor timelines to appear compressed.

Case 1149: DSA with FanOut Does Not Work for Some BSPs

If one box output fans out to several boxes mapped to several different processors, then the DSA communication mechanism does not work correctly for Mercury and Sky BSPs.

Case 1150: Trace Table Saved on NT Not Readable on Solaris

Files saved from NT are byte reversed from what is expected on Solaris. Files saved on a big-endian platform cannot be read by Gedae from a little-endian platform.

Case 1151: FFT Primitives Only Work with Power of 2 Sized Vectors

The FFT boxes do not support non-power-of-2 lengths; however, the comments make no mention of this fact. If these boxes only support a power-of-2, then it would be useful to have a separate set of boxes that support a non-power-of-2.

Case 1152: Queues Do Not Auto-resize When Opening Group Settings

If a queue needs to be autoresized and the autoresize option is turned on in the group-setting file, then the autoresize does not occur. A workaround for this problem is to toggle the Group Control Dialog Autoresize toggle off, and then on, and then resave the group settings.

Case 1153: Cannot Put Parentheses Inside Single Quotes

A line like `c = '{'` inside a primitive causes an error during primitive parsing. The parser will count the `'}'` as a bracket and fail to find a matching closing bracket. Because brackets inside double quotes are ignored, the expression `"{"` will not cause a problem.

Case 1154: Set Partition by Equation Error

When setting a partition by equation on a subgraph family, and if any of the contained boxes are also families, then the `$1` variable applies to the deepest family member giving the wrong results.

Case 1155: Constants Propagated Through typedef Boxes

Constants propagated through typedef boxes cannot be used for instantiation.

Case 1157: Segmented Schedules with `nondet` Inputs

If a segmented static schedule has only `nondet` input queues, then it will not respond to segment ends correctly. Such schedules should be disallowed at development time. It is possible that in some situations such schedules should be legal, and this possibility is also being investigated.

Case 1158: Primitives with `EndOfSegment` and No `Apply`

Primitives that have an `EndOfSegment` method but do not have an `Apply` method do not get included; therefore, the `EndOfSegment` method does not run. A workaround for this problem is to include an empty `Apply` method in the primitive.

Case 1159: Stream Box with `push` in Hostless Launch Package

If a stream box contains a call to `push` and it is made part of a hostless launch package, then the launch package will fail to compile, as the code for the `push` is not included in the standalone library.

Case 1161: `avail` on Variable Vectors is Unreliable

On variable vectors, the `avail` function only returns the number of tokens available on the vector and not on the variable part. Usually these are the same, but when the variable vector is coming from another processor over a dynamic queue they may be updated at different times and may be different. As a result, `avail` may return a number too large. When the data is processed, the variable part may not actually be ready, causing a segfault. A workaround is to check the `queues_ready` flag after the call to `amount` and only continue with execution if the `queues_ready` flag is non-zero.

Case 1209: `mz_rnconcat` & `mz_cnconcat` Do Not Work

Both `mz_rnconcat` & `mz_cnconcat` flowgraphs need a modification to get them to work. The input has N family members. The `splitx` typedef has only one family member, and therefore, the internal `mz_rnconcat` does not know how many family members there are and can not instantiate the arc on its inputs.

Case 1210: Multiple Exclusive Sources with Some Sources Are Not Used by Every Mode

If there are multiple exclusive sources to a family of modes and some of the sources are not used by every mode, then Gedae crashes during development time scheduling. For example, if two exclusive branches drive three downstream modes and one of the branches has one of its outputs unused by the third mode, then this causes a segfault during scheduling. The workaround is to add dummy inputs to the modes to allow all the sources to be used by every mode.

Case 1222: Launch Package Stored in a Directory with Spaces in the Name Will Not Run

An issue with launch packages on windows is that they will not run if they are in a subdirectory that has spaces in its name. For example, if a launch package resides in "[D:\data\sapu](#)", then it will run; however, if it is in "[D:\Data Directory\sapu](#)", it will not

run. This problem becomes obvious if you put the launch package on the desktop and run it (since it is in "[C:\Documents](#) and Settings\<>username>\Desktop").

4 Solutions

“Solutions” is a new section of the Release Notes. We will post answers to questions that are of general interest here.

Case 1195: Perl Delivered with Gedae May Not Work

On Solaris and Redhat, the Perl delivered with Gedae may not run with the users installation. The solution is to copy the Perl found in /usr/bin/perl to \$GEDAE/

Case 1206: Outfail Event Caused by Internal Queue

A queue internal to a schedule (source and dest of queue is same static schedule) may cause an outfail event if the source produces data and the dest does not consume the data.

This outfail event happened within a segmented subgraph where the source was just producing one token at the beginning of the segment and the dest did not intend to consume it. The dest does not drain the queue on the end-of-segment because it's an internal queue.

Be aware that whatever is produced on an internal queue must be consumed by the destination of the queue when it fires.