



Gedae 4.6 Release Notes

January 2005

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

Upgrading to Exceed Version 10.0

Version 10.0 is now the default supported version of Exceed and Exceed XDK for Windows installations. Users of Exceed 9.0 and higher should not have to alter the Gedae make system to compile Gedae. Users of Exceed 7.1 through 8.x should perform the following alterations before `initGEDAE` during the installation process:

```
copy %GEDAE%\nt\makeGEDAE_Exceed7_1 %GEDAE%\nt\makeGEDAE
copy %GEDAE%\nt\ent\bin\std_make_info_Exceed7_1
%GEDAE%\nt\ent\bin\std_make_info
copy %GEDAE%\nt\ent\embedded\runtime_make_info_Exceed7_1
%GEDAE%\nt\ent\embedded\runtime_make_info
```

Users of Exceed versions before 7.1 should perform the following alterations before `initGEDAE` during the installation process:

```
copy %GEDAE%\nt\makeGEDAE_Exceed6_1 %GEDAE%\nt\makeGEDAE
copy %GEDAE%\nt\ent\bin\std_make_info_Exceed6_1
%GEDAE%\nt\ent\bin\std_make_info
copy %GEDAE%\nt\ent\embedded\runtime_make_info_Exceed6_1
%GEDAE%\nt\ent\embedded\runtime_make_info
```

It is recommended that users upgrade to the newest version of Exceed as older versions of Exceed are not supported by Hummingbird.

Running Command Programs from the Development Environment

A new feature was added to Gedae. From within the Gedae development environment, users are now able to develop and test launch packages controlled by intermediate command programs.

An application developed and tested from within Gedae can be delivered independently from Gedae as a launch package. The launch package contains all the information and executable code needed to run the application on a target platform. It implements the algorithmic behavior captured by a user's graph. The implementation is based on the specification entered in the Group Control Dialog.. The launch package can be started from a command program that is either user written or automatically generated by Gedae.

Until now, a user could generate a launch package from within Gedae but could not test the launch package and its controlling command program from within the development

environment. A new feature has been added so that the Gedae development environment can compile and test a user written command program running on a Gedae generated launch package. The development environment can send commands to the attached command program to start and stop the application, set parameter values in the running application and collect trace information.

Multihop Embedded Configuration File

You can skip this section if you already know how to set up a multihop configuration using the embedded configuration files. In order to run a command program on an intermediate processor, the user embedded configuration files must specify a multihop hardware configuration. The follow explains what a multihop configuration means and how it can be achieved.

The embedded configuration files are found in the `FGlibraries/emb_config_files` directory and describe the hardware configuration to which a Gedae application can be mapped. The first part of the embedded configuration file, and the only part we'll be concerned with here, is the `Processor_Desc` section. This section describes the processors in the configuration. The following is an example of an embedded configuration file:

```
Processor_Desc: {
  0    local      solaris
  1    local      solaris
  2    local      solaris
  3    local      solaris
  4    chazz      solaris
  5    zathras    solaris
  100  aramis     vxworks
}
```

Each line contains the following three entries.

1. The logical processor number by which the Gedae user can refer to the processor. Group partitions are mapped to logical processor numbers.
2. The name of the physical processor that corresponds to the logical processor number.
3. The processor type.

In the above example, logical processor numbers 0 through 5 are type `solaris`. Logical processor numbers 0 through 3 are mapped to the local host processor. Logical processor number 4 is mapped to the processor named `chazz`, and number 5 is mapped to `zathras`. Processor 100 is mapped to processor `aramis`, which is a `vxworks` type processor.

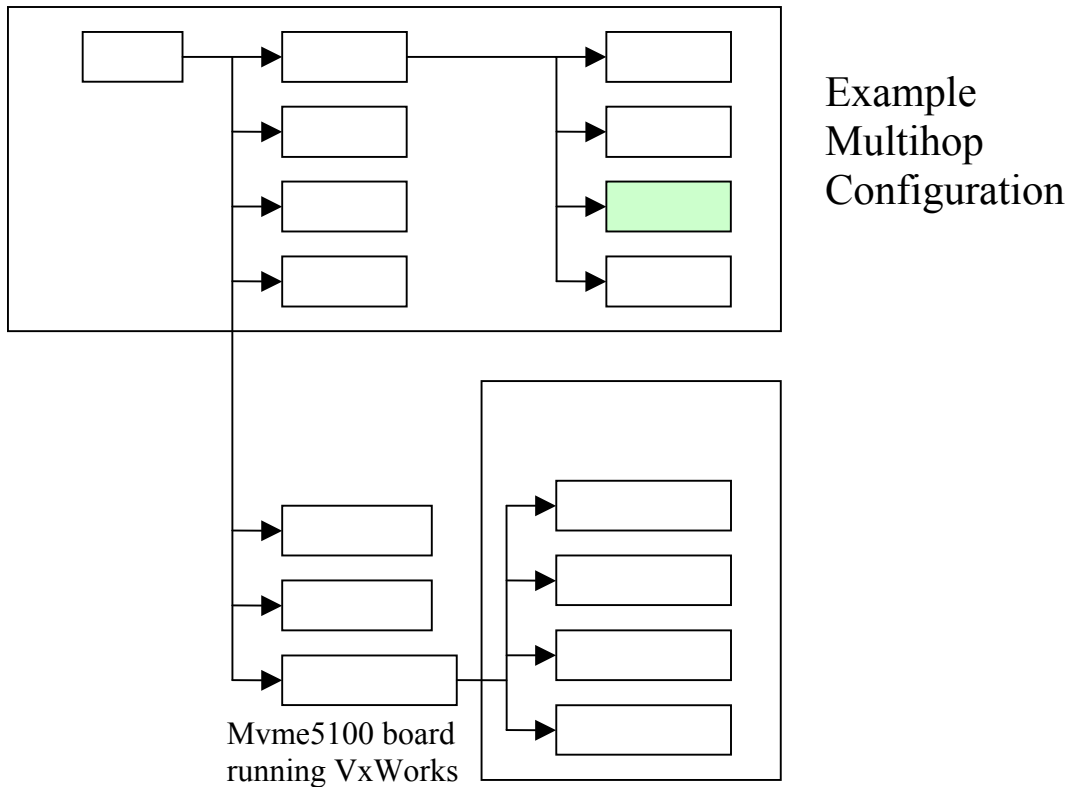
Embedded configuration file names have the format `embedded_config_<procname>`. For example, the embedded configuration file for a processor named “aramis” has the name `embedded_config_aramis`. When an embedded config file is read in, if any of the physical processor names in the configuration file also have an associated configuration file, then that file is read in creating a multihop configuration. Continuing our example, suppose that the above configuration file is the configuration file for the processor on which Gedae is running. In addition, suppose that there exists a configuration file `embedded_config_aramis` as follows:

```
Processor_Desc: {  
    100  2  mcos_altivec  
    101  3  mcos_altivec  
    102  4  mcos_altivec  
    103  5  mcos_altivec  
}
```

Additionally, the file `embedded_config_local` is:

```
Processor_Desc: {  
    0    "local"  "nt"  
    1    "local"  "nt"  
    2    "local"  "nt"  
    3    "local"  "nt"  
}
```

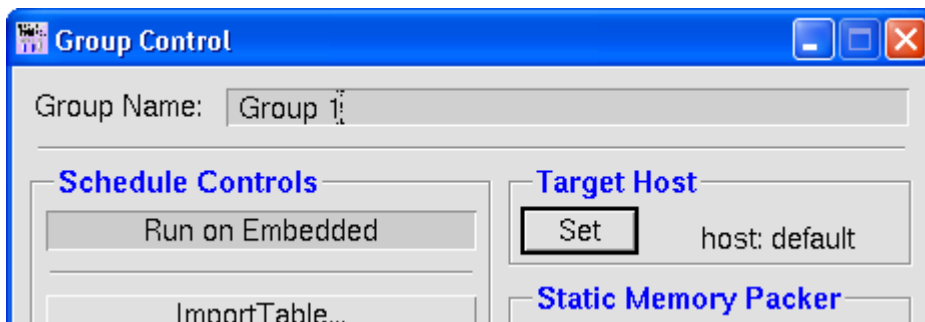
These three files describe a multihop hardware configuration as:



Multihop configurations have long been part of Gedae. To map a partition to Mercury CE5, the user would use a logical processor number 100.103 indicating that the host processor has to go through aramis (logical processor 100) to get to CE5 (logical processor 103 in aramis's configuration file). To map a process to the pale green box marked 2:local, the user would map the processor to processor 0.2 indicating that the host has to go through logical processor 0 in the top-level configuration to get to logical processor 2 in the "local" processors configuration. Note that embedded configuration file embedded_config_local is only read for logical processor 0.

Group Control Dialog Enhancements

The Group Control Dialog was enhanced to allow a user to specify that a particular group in a Gedae graph is run under the control of a separately generated command program. A new section, Target Host, was added to the Group Control Dialog.

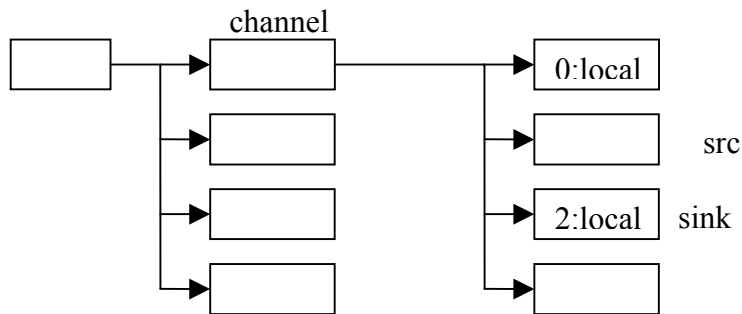


By default, the Target Host is set to the value “host” indicating that the group is running under direct control from the development environment. In this configuration, there is no intermediate command program because the development environment itself is directly controlling the group’s application through the command program API. If the Target Host is set to the logical processor number of a processor that has its own embedded config file, then Gedae will generate a command program as specified by the Launch Package Dialog and will attach to that program at runtime.

Mapping an Intermediate Command Program’s Partitions

After a user specifies the Target Host to be different than the development host, all mapping done in the mapping table will be relative to the target host’s embedded configuration file. Any partition mapped to the host will run in the intermediate command program.

For example, suppose the embedded configuration is designed as below.

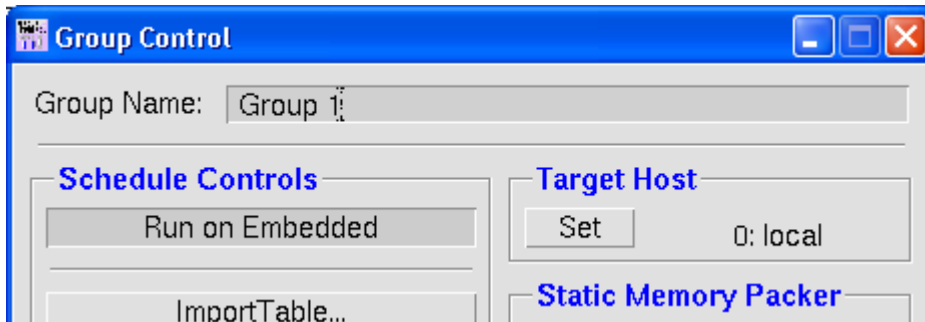


The MapEmbedInfo Table can be used to map the partition channel to processor 0, the partition sink to processor 0.2 and the partition src to processor 0.1 as seen below:

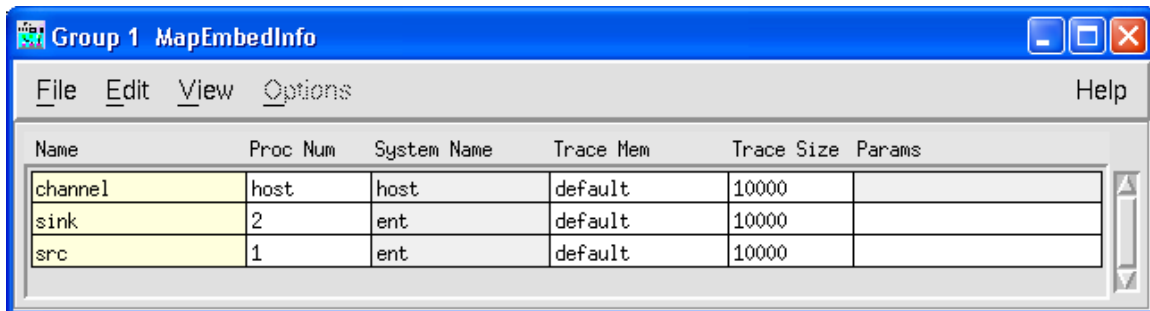
Name	Proc Num	System Name	Trace Mem	Trace Size	Params
channel	0	ent	default	10000	
sink	0.2	ent	default	10000	
src	0.1	ent	default	10000	

The table illustrates the ordinary way of achieving the above mapping of partitions to processors.

To achieve the above mapping using the new Target Host feature, set the Target Host to logical processor 0.



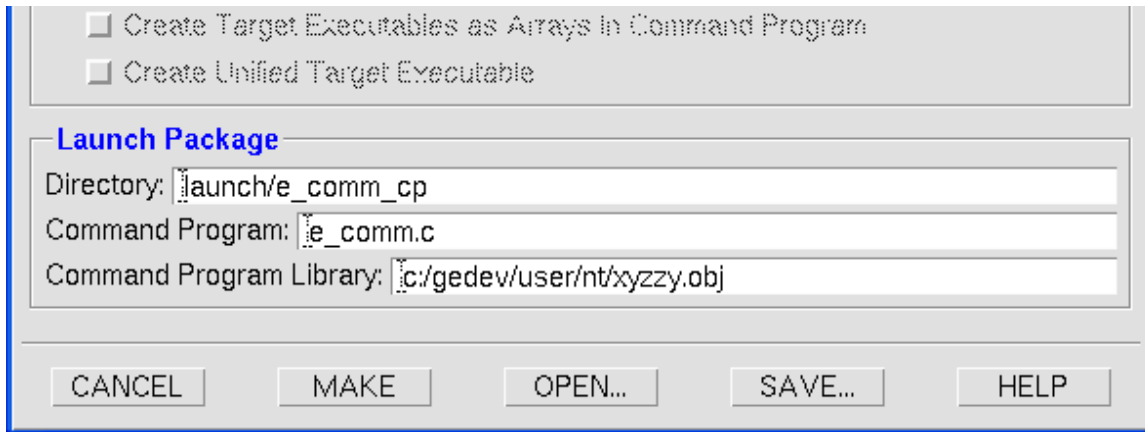
Then the processors are mapped relative to processor 0 as:



Launch Package Dialog Enhancements

Two new text fields that support the ability to have Gedae compile and link user generated command programs have been added to the Launch Package Dialog. These fields are labeled Command Program and Command Program Library. By default, when a launch package is created, Gedae creates a default command program, exec-host.c. Then Gedae, compiles and links this file into an executable program, exec-host. If the field labeled Command Program is set to a command program C language file, then Gedae uses that command program in place of the default exec-host.c. In addition, any libraries specified in the text field Command Program Library are linked with the resulting command program.

If the Target Host field in the group control dialog is set to a value other than “host”, then Gedae will create a command program according to the launch package specification and will run it as an attached command program.



A user can still build a command program using the Make button. If the new fields are set, then the command program will be created with the users .c file and the specified command program libraries. However, if the command program is to be runnable from the development environment, then it must be written as described in the section “User Generated Attachable Command Program.”

User Generated Attachable Command Program

A user generated command program must be written correctly if it is to be runnable from the development environment. The following functions have been added to the command program API to support writing user attachable command programs. These functions are also used by the automatically generated exec-host.c when a user doesn’t specify a command program.

Prototypes:

```
void appAttachFromArgs(int argc, char *argv[]);  
int appAttached(void);  
void appHandleAttachedCommands(App app);  
void appSetOnExit(void (*onExit)(void));
```

Description

The function appAttachFromArgs must be passed the argc, argv list of the main routine. It passes these arguments to the appParseArgs function defined in the Gedae Command Program Interface Reference Manual. In addition, if arg “-noattach” is not passed to the function, then the function attaches to the development host.

Function appAttached returns 1 if the command program has been attached by appAttachFromArgs. Otherwise it returns 0.

Function appHandleAttachedCommands provides the protocol engine that listens and responds to commands from the development host. If the user wants the command program to be attachable, then the appHandleAttachedCommands must be called periodically in the main polling loop of the command program. If the command program is written using appLaunch or appLaunchFromGen, then appHandleAttachedCommands

is automatically called. If the program is not attached, then `appHandleAttachCommands` returns immediately.

The function `appHandleAttachedCommands` may cause the program to exit if the user terminates the application from the development environment. The function `appSetOnExit` allows the command program to register a cleanup procedure that will be called by the `appHandleAttachedCommands` routine before the exit happens.

An example program using `appAttachFromArgs`, `appIsAttached` and `appSetOnExit` is given below:

```
#include <commapi.h>
#include <exec-host.h>
extern void *initLaunch_e_comm_cp_launch;

/** LAUNCH PACKAGE INITIALIZATION ***/
void initLaunch_e_comm_cp(App app);

/** BOARD SUPPORT PACKAGE INITIALIZATION ***/

static void *HOST_PORT = 0;
static void *POLL = 0;

static void handleExit(void) {
    printf("ON EXIT COMMAND RUNNING BEFORE EXIT!\n");
}

void main(int argc, char *argv[]) {
    char *dir = ".";
    printf("BEGINNING EXECUTION\n");
    appAttachFromArgs(argc, argv);
    if (appIsAttached()) {
        appSetOnExit(handleExit);
        appLaunch(dir, initLaunch_e_comm_cp,
                 POLL, 0.2, 0.0, 0.2, 0.0001);
    } else {
        .. users normal command program polling loop ..
    }
}
```

The above program assumes that when it is attached, the startup and polling loop will be handled by the `appLaunch` command, and when the program is not attached, the users normal command program will run. If the user wishes to run their polling loop when the program is attached, then a call to `appHandleAttachCommands` must be made from within the polling loop. Such a polling loop is shown in the program below:

```
while (1) {
```

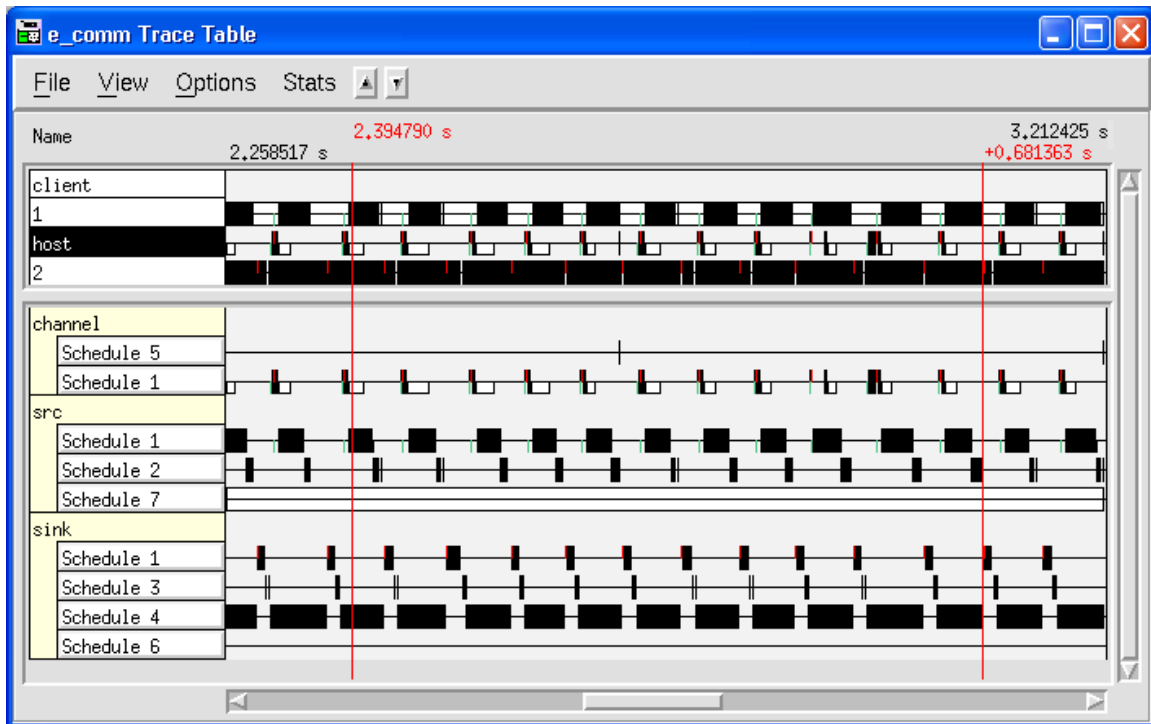
```
/* run primitives mapped to the command program */
appExecute(app,exec_timeout, exec_persistence);
/* handle requests from the target processors */
appHandleCommands(app,0.01,0);
/* handle commands from the development environment */
appHandleAttachedCommands(app);
}
```

Generation of the Launch Package Init Function Name

Each launch package contains a file `init.c`, which is linked with the command program and provides an initialization function unique to the launch package. The initialization function name has the form `initLaunch_<name>`. The value of `<name>` was chosen in past releases of Gedae to be the directory name in which the launch package is generated. If the launch package directory is `launch/e_comm_cp`, then the name of the initialization routine becomes `initLaunch_e_comm_cp`. However, if a user specifies a command program file in the new text field `Command Program`, then the command program filename will be used to generate the name of the initialization function. If the launch package command program filename is `e_comm.c`, then the initialization function name becomes `initLaunch_e_comm`. Setting the initialization function name based on the command program name, rather than the directory name, allows the same command program to be used for different launch packages.

Running Attachable Launch Package

When the graph is mapped to the target hardware, as described in the preceding section, the graph can be run as normal. Trace Table information collection remains the same.



Running Attachable Launch Package Outside of Gedae

Command programs created by running a graph in the attached mode can still be run outside of Gedae. To run the command program, add the `-noattach` parameter to the command program's command line. For example:

```
exec-host -noattach -timeout 4
```

will run the default command program without attempting to attach to the host development environment for four seconds.

Features and Limitations

The following features are supported when running an application from the development environment through an intermediate command program.

1. The application can be controlled using the Control->Run, Control->Stop, and Control->Cont menu items.
2. The application can be terminated using the Control->Terminate menu item.
3. Parameters can be set on a running application from the development environment.
4. Application GUIs developed with Trigger and Eval boxes will be popped up by the intermediate command program.
5. Tracing can be enabled and disabled on a running application using the

Utilities->Enable Trace menu item.

6. Trace Table time synchronization is superior when using an intermediate command program vs. using an intermediate target processor running in the multihop mode.
7. The Trace Table can be displayed using the View->Trace Table menu item.

The following limitations exist on running applications from the development environment through an intermediate command program:

1. If a GUI display is developed with Trigger and Eval boxes, then the development environment will pop up its own version of the GUI. The development environments GUI will not be part of the running command program and will be effectively dead.
2. If the group the intermediate command program application is generated from is connected to other groups, then data will not be sent from the intermediate command program to the other groups or from the other groups to the intermediate command program. A common example of a group that is not part of the application is the stream/sink/scope box. Such boxes are added to a graph to monitor signals during development but do not become part of the end application.
3. The following debug features will not work with an attached command program:
 - a. The different View menu options such as View->Execution will not work on either the flattened or normal Gedae graphs.
 - b. None of the Debug Dialog features will work with an attached graph.
 - c. The ability to monitor the values in Gedae internal data structures and primitive box state data structures will not work with an attached graph.

Attempting to use the above debug functions may crash Gedae.

2 Bugs Fixed

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 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 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.

Case 1153/SCR2007: 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/SCR2011: 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 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. This problem has been fixed.

Case 1222, SCR2010: 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](#)"). This problem has been fixed.

Case 1247: Unsigned Integer with Value Over 2^{31} Does Not Get Parsed Correctly in Apply Method

Unsigned integer values over 2^{31} that appear in primitives do not get parsed correctly and are printed out with wrong values during primitive code generation. For example, the following code:

```
HIGHFREQ = 2799880000U;
```

Gets printed as:

```
HIGHFREQ = 279988000U;
```

dropping the last 0. This problem has been fixed.

Case 1250: Command Line Parameter -setp Does Not Work If No Parameter File or Group Setting Is Loaded

If Gedae is started

```
gedae -file path/graph -setp "var=setting"
```

the setp is not applied. The setp is only applied if the param or group flag is used. The setp flag should be applied every time it is used. This problem has been fixed.

Case 1270/SCR2009: The `embSetPeriod` Function Does Not Work with Fully Static Graph

If a graph is fully static (no boxes have nondet or dynamic inputs and outputs), then the periodicity of the schedule set by the `embSetPeriod` function is ignored. This problem has been fixed.

SCR2006: Avail on Variable Vectors is Unreliable

On variable vectors the `avail` function returns the number of tokens available on the vector and not on the variable part. As a result, `avail` may return a number too large, so the variable part may not actually be ready. Operating on a vector that is not ready may cause a segfault. This problem has been fixed.

SCR2008: Cannot Create Launch Packages "as Arrays in Command Program" That Have No Parameters

Launch packages with no parameters subsequently have no synonyms. As a result, the `launch.c` file fails to compile due to the fact that there is an empty synonyms array. This problem has been fixed.

SCR2013: Cannot Run Launch Package when Target Schedules Are Created in the Target Executable

If an application graph has its launch package group settings set to create the target schedules in the target executables and if identical boxes are mapped to two partitions, then Gedae only creates one target executable for the two partitions. Creating one executable for both partitions is an error as each partition requires the generation of a unique schedule. The problem has been fixed.

SCR2021: Trace Table Segfaults on Button3 Push and Release

Clicking on the trace table with button 3 brings up an information dialog. Sometimes this action would cause Gedae to crash. This problem has been fixed.

SCR2037: Function `e_vshrink` Crashes When the Stride is Greater Than 1

The function `e_vshrink` crashes when the stride is set to be greater than 1. This problem has been fixed.

SCR2040: Autoresize of Dynamic Pointer Stream

If the runtime-resize queues feature detects a capacity increase is needed on a dynamic pointer stream, then there is a problem with the graph. This problem was not detected and caused a segfault. The problem is now detected, and the user is informed of the problem via a message dialog.

SCR2043: Distributed Dynamic Graphs Can Stall in the Development Environment

Graphs in the development environment can stall when all of the following conditions are met:

1. There exists a static schedule that is distributed.
2. A portion of the schedule is mapped to the host.
3. There is an input queue on a target processor that can cause the schedule to fire at different granularities.

The problem could occur when the schedule is preempted in the middle of execution. This preemption happens in the development environment to provide a timeslot for handling user interactions with the development environment interface (like popping up the Trace Table). The problem has been fixed.

3 Known Bugs

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

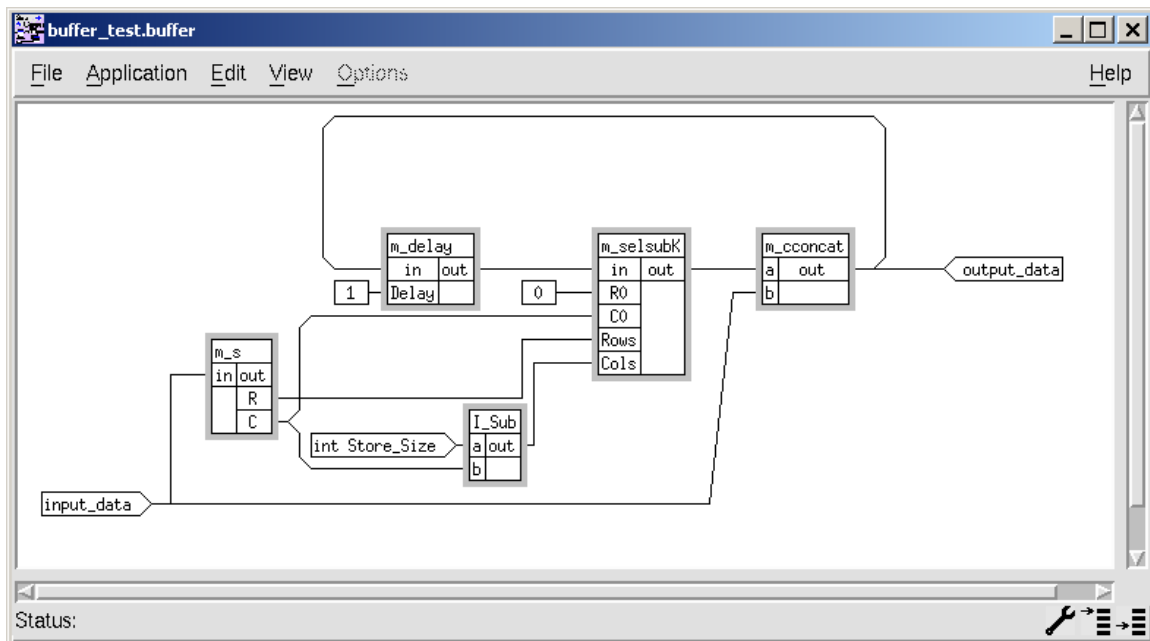
The problem scenario is that the schedule is partitioned into three parts, with the first and last parts mapped to the same processor. Usually Gedae puts the parts mapped to processor A in the same static schedule; however, to allow efficient pipelining, Gedae splits the two parts mapped to processor 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,

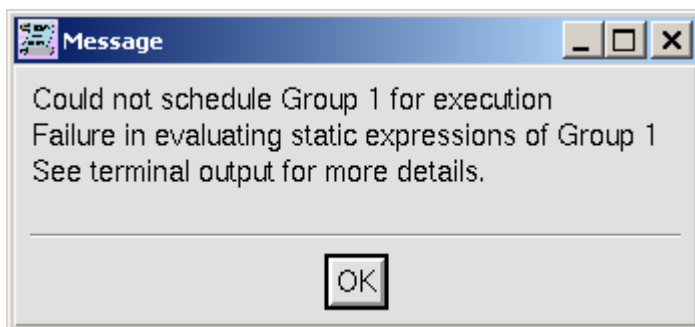
causing the control message queue to back up and overflow. The condition is rare because the problem only happens when the graph is not keeping up with the input data rates.

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 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 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 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 – such as, 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. Currently, the user must 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 a 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 1155: Constants Propagated Through typedef Boxes

Constants propagated through typedef boxes cannot be used for instantiation.

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 1210/SCR2015: 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 1239: Static Schedule in Segmented Subgraph Not In Scope Of Segment Controller Asserts

An assert error is caused by a static schedule in a segmented subgraph that has no input in scope of the segment controller. For example, a constant box driving a merge input in the subgraph can cause the error. The error manifests itself as an assert as shown below:

```
Error: assert(dq->eos[inptr].begin_level > 0) failed.  
File: ../../source/segment.c, Line: 935
```

Such static schedules should be declared to be illegal at scheduling time.

Case 1249: Transferring Doubles Between Host and Target with Different Endian (Dy4av2)

Currently, the Gedae BSP does not support providing byte swapping of doubles when transferred between the host and target processors.

Case 1271/SCR2012: Running Two Vxworks Processes on the Same Processor

This problem occurs when trying to run two separate Gedae generated VxWorks executables on the same processor; however, the entry point for each executable has the same name, `VxWorks_main`, making this impossible.

4 Solutions

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 the 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 when the source was 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.