Issue No. 4, January 2013

# G E D A I C A

**pl n 1.of or relating to the culture, artifacts, customs of Gedae**

## SCOTCH, JEANS, FRIENDSHIP, GEDAE…

*Some things just get better with age!* 2013 marks the 25th anniversary of the birth of the Gedae technology. That's right! It was 25 years ago this year that Kerry Barnes and Bill Lundgren sketched out their first plans for Gedae. Back then the goal was much less ambitious. Initially Kerry and Bill set out to create an autocoding tool to solve the big data problem for signal processing applications, specifically RADAR and SONAR. Gedae's predominant historic use for signal processing made the use of the block diagram dataflow language appropriate. 25 years later Gedae's become so much more than originally intended.

To that end, we're pleased to announce the release of Gedae 6.5, the first full release of the Idea Programming Language and initial support for programming GPU's. Formulated from the integration of mainstream language features the Idea language is familiar and easy to learn. The Idea language was created to allow all parts of an application access to the parallel optimization capabilities of the Idea Compiler. Built on the time tested, customer proven Gedae Idea Compiler, the Idea Language offers a new and easy way to access the benefits of the compiler.

Be sure to check back often for updates to Gedae's roadmap. And if you have feedback please share! Our customer's feedback, support, and vision have helped guide Idea to where it is today.

## GEDAE IS GROWING:

### CSP, Inc. becomes Gedae VAR!

CSPI MultiComputer Division announced a Value Added Reseller relationship with Gedae, Inc. CSPI MultiComputer Division will market, sell, and distribute the Idea Development Environment with its high performance embedded computing systems. This agreement significantly enhances the set of application development tools available to CSPI customers, expediting time to deployment by reducing the complexity of programming CSPI's advanced multicore systems with complex memory architectures.

*"We are excited to offer our customers this tool set that allows for rapid development of signal processing solutions and the very quick dynamic reallocation of processes in multicore environments for optimum tuning of performance"*
Paul Martino, Director of US Sales, CSPI.

Gedae offers an application development environment consisting of a language, compiler and thread scheduler automatically optimizes the software targeted for multiprocessor, multicore or heterogeneous systems beneficial to CSPI's TeraXP™ OpenVPX™ Embedded Servers and 4000 SERIES AdvancedTCA product line.

*"We are pleased with CSPI's selection of Gedae's Idea Development Environment as the solution to their customer's demands for multicore software tools,"*
said Bill Lundgren, President and CEO of Gedae, Inc.
*"Idea makes it easy to get the best performance from CSPI's top of the line computing systems. This agreement represents a significant event in Gedae, Inc.'s growth."*

## THE WATER COOLER

*Submit questions via email at thewatercooler@gedae.com.*

**Can Idea be used to write your entire applciation?**
Yes, but there are really 3 parts to the question.

Idea is a textual language but it can be edited using either a graphical or textual editor. Historical users of Gedae expect that the application will be a block diagram – and indeed the textual and graphical versions can be freely intermixed. But if a developer prefers she/he can build an application that is completely textual.

The second aspect to consider is "can a complete system be built using Idea?" Again, yes! Idea is a general purpose language that equally supports signal, image and data processing, and control. Idea can and should be used to write the whole of the software for any sensor processing system.

A final consideration is "When the compiler and/or development tools fall short, what recourse do I have?" In short, Idea is far more general purpose and capable of handling entire sensor processing applications TODAY. And if there is a shortfall Gedae's unsurpassed customer support will be there to live up to our mission of ensuring every customer's success.

On top of that the Idea Compiler, the heart of the technology, is the same reliable, time tested compiler for targeting multicore and multiprocessor GPPs and DSPs you know and love. Of course, Idea can't extend system performance beyond the limits of the hardware!

## DOLLARS AND SENSE

*25 hours for 25 years!* Limited to the first three new customers to respond to our 25 for 25 offer. With the purchase of any production license, get 25 hours of world class development support to help you get started harnessing the power of Gedae.

## ***CODE, TOOLS & TIPS*** *Tracking Example*

**Is there an example you'd like to see in CODE, TOOLS & TIPS?** – We're looking for industry parallel processing applications to implement in Idea Language and publish results. Submit yours today via cc@gedae.com.

Gedae 6.5 introduces constructs that support the direct expression of database operations. An Idea header file (.ih extension) is shown in **Figure 1**. It generally follows the C syntax for a data structure but takes its name from the structure name. The example shown has only integers but can also include mixed data types including other data structures.

```
struct Record { // represents a track
  int idx; // the index of this record in database
  int roi; // sensor input - region of interest
  int cnt; // number of times record has been updated
}
```

**Figure 1:** Idea Header File

Code for the example track database processing is shown in the code in **Figure 3**. One of the key characteristics of the Idea language is the limitation that a name can only be used one time on the LHS of an assignment expression. But a database is by its very nature created and then modified many times. In this example we show how that can be done using the Idea language.

The syntax for declaring persistent data is shown in **Figure 2** below.

```
do ( <initialize persistent data> ) {
  pop <variable from outside loop body>; // stream data into loop
  <code to process persistent data>
  push <variable to outside loop body>;  // stream data out of the loop
} while(1);
```

**Figure 2:** Idea Syntax for Declaring Persistent Data

Variables declared in the initialization section of a loop are called iteration variables. If iteration variables are used on the RHS of expressions in the body of the loop, the variable refers to the current value of the token, not the new value computed during the current iteration. In the example below db[n] and cnt are iteration variables. The variable db is a vector of records (nominally tracks). It sets the maximum size of the database. The variable cnt is initialized to zero and indicates the number of records currently stored in the database. A modified record is inserted back into the database using the set() function.

In these examples an infinite do/while loop is used. (Gedae will introduce the equivalent syntax, database () { }, in Gedae 7.0 release in May 2013.) One of the unusual features of the Idea language is that many infinite loops can happily run at the same time. The keywords pop and push are used to get data into and out of an infinite loop. A pop takes one token from an external stream of data and a push puts one token on an external stream. In the example shown in **Figure 3** a region of interest (roi) is streamed into the loop and the new updated record (rec) is streamed out of the loop.

A more complete tracking example is implemented in the application examples/Tracking/test_InstantiateTracks.fg found in Gedae's released library.

```
/* roi        - represents target information
  MaxRecords - maximum number of tracks in database */

rec Simple(int roi, int MaxRecords) {
 range n = MaxRecords;
 Record initialize = {-1,0,0};
 do (Record db[n] = initialize, cnt = 0) {
   pop roi;
   range n1 = constrain(cnt,MaxRecords);
   roi_match[n1] = db[n1].roi == roi; // is roi in db?
   idx[found] = find(roi_match,1); // which record if any?
   if(#found != 0) { // use logical operation to constrain
     idx0 = idx[0];
     Record rec = {idx0, roi, db[idx0].cnt+1};
   } else {
     idx0 = cnt; // new record
     cnt = cnt + 1;
     Record rec = {idx0,roi,1};
   }
   db[n] = set(db,rec,idx0);
   push rec;
 } while(1);
}
```

**Figure 3:** Infinite Do/While Loop Example