



Application Note: Integrating KEEL Cognitive Engines in GE PAC Systems

(General Electric Programmable Automation Controller Systems)

Objective:

Programmable Logic Controllers (PLCs) have recently evolved to integrated control platforms commonly called Programmable Application Controllers (PACs). This has primarily been lead by the integration of motion control into the backplane which allows more complex, integrated applications to be performed. Applications that had once been the domain of DCS (Distributed Control Systems) are now within the domain of some PAC systems.

GE/Fanuc is a leader in industrial automation systems. With their PACSystems™ RX7i and RX3i they have a faster backplane that enables data to be passed between I/O and the processor more efficiently, provide for RIUP (removal and insertion of modules under power), increased memory, and added advanced diagnostics. GE/Fanuc supports their PACSystems with Proficy Machine Edition development environment that supports IEC 61131-3 (1131-3) Programming Languages.

One of the 1131-3 programming languages is known as Function Block Diagram programming. This language allows logic to be encapsulated in a function block. Proficy Machine Edition allows function blocks to encapsulate logic in the 1131-3 languages (Relay Ladder Logic – RLL, Structured Text – ST, Sequential Function Charts – SFC, Function Blocks – FB) and compiled C blocks.

C blocks are commonly used for custom algorithms, or to perform functions that can be handled more efficiently in C than in other languages (moving blocks of memory for example).

This application note focuses on exploiting the capabilities of embedded C Function Blocks to provide solutions to much more complex applications while still avoiding the requirement of designing, debugging, and testing manually developed C code.

Application Example:

This application note will not focus on the structure of a particular KEEL application, so we will not spend time discussing the details of the KEEL design. However, a “sample problem” is provided to show how a KEEL engine is integrated into the solution.

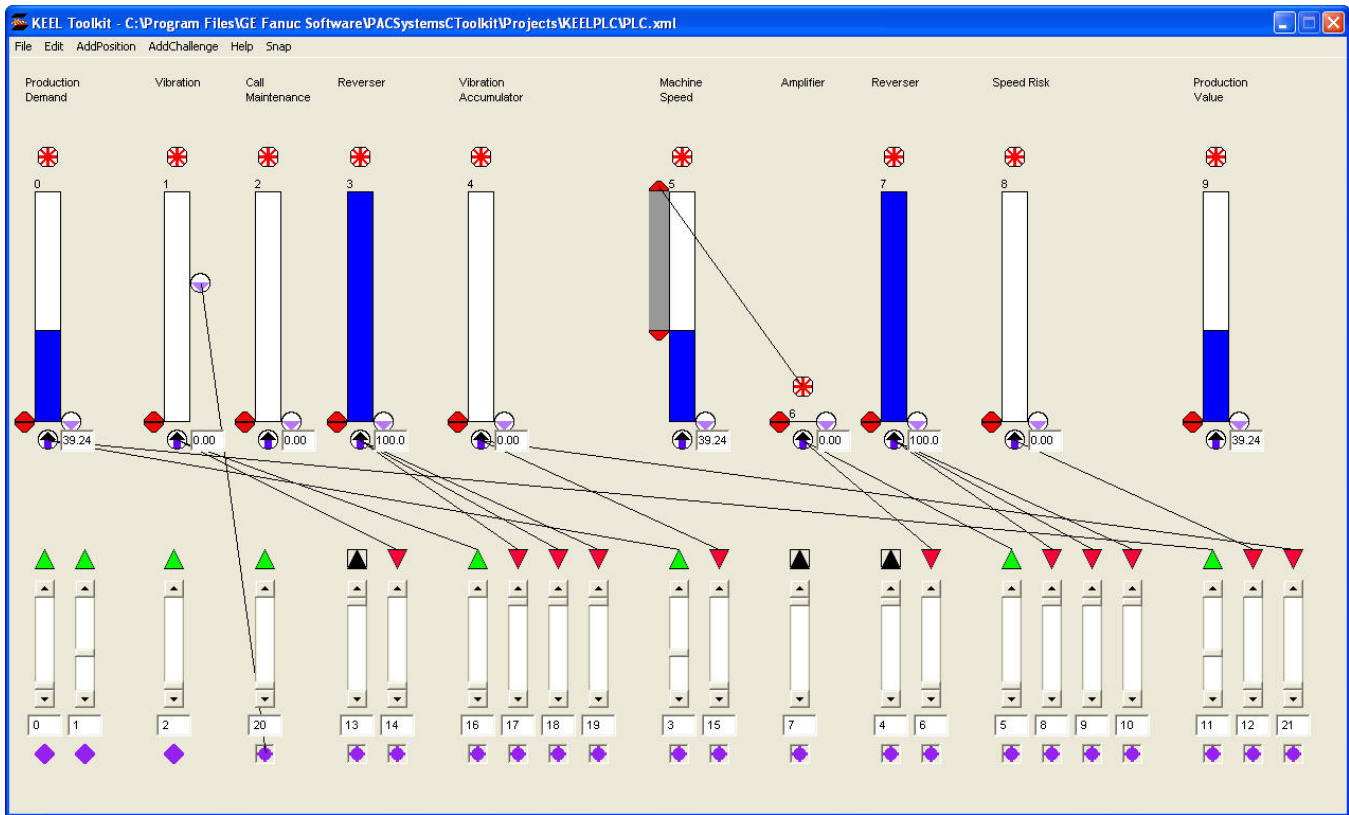


Figure 1

Sample Problem:

In this example we have a production problem where the operator is asked to increase production at his/her work station.

The operator is responsible for the operation of the equipment. The operator has been trained on safe operation of the equipment. The operator is given work orders to define what is expected for that day's production. The operator controls the speed of the machine. By increasing the speed, more parts are produced, but at a certain point the quality degrades, and wear and tear on the equipment increases.

To pose one scenario: During the day, the operator's supervisor comes by and informs the operator of the need to increase production because of a new order. The operator increases the speed a little bit and production increases. The operator detects a new vibration in the machine being used to produce the parts. After ignoring the vibration for a while a maintenance technician is called. Production is stopped while the technician looks over the system.



In this example, the humans made a number of judgmental decisions: 1. The supervisor made a judgmental decision to try and escalate production by telling the operator of the new order and “asking” him to produce more parts. 2. The operator made a judgmental decision on how much to adjust the machine speed by balancing risk against the need to increase production. 3. The operator made another judgmental decision relative to the amount of vibration which justified calling the service technician. 4. The service technician made a judgmental decision about whether to stop production to analyze the system.

None of these decisions are auditable in terms of what one would expect of an automated system. The reasoning behind the decisions is not transferable to other facilities, because they cannot be explained in mathematical terms. There is little opportunity to upgrade or refine the decision-making process, since the existing process cannot be explained or audited.

By introducing KEEL technology into the system design as shown in Figure 1, the importance of increasing production would be supplied to the system as a weighted value. This value would be integrated into the performance calculation according to company policy. This company policy would take into account the known risk of increasing machine speed (Figure 2). A separate segment of the KEEL engine would be monitoring diagnostics (vibration in this example). The impact of vibration on Production Value is shown in Figure 3. Again, according to company policy, the service technician would automatically be called when a certain level of vibration was detected. This is shown in Figure 1 when the vibration reaches a normalized value of 60. At the same time the machine speed or other control parameters might be immediately (or continuously) adjusted based on vibration feedback from the machine (again, according to company policy). This is also accomplished in the Figure 1 design. Finally, the decision by the service technician to take the machine out of production for examination might be controlled by the machine itself, rather than waiting for the decision by the service technician. In this design the increase in vibration is fed back into the speed control loop which automatically reduces the machine speed. Figure 4 shows a 3-D graph of the production value as speed is increased, and as detected vibration increases. In all cases, because the policy processing is integrated into the PLC, it can be copied and distributed to other locations. If it is found that the policy needs to change, it can be updated to match the needs of the enterprise. In all cases, the reasoning can be explained and audited.

Expanded Production INPUT

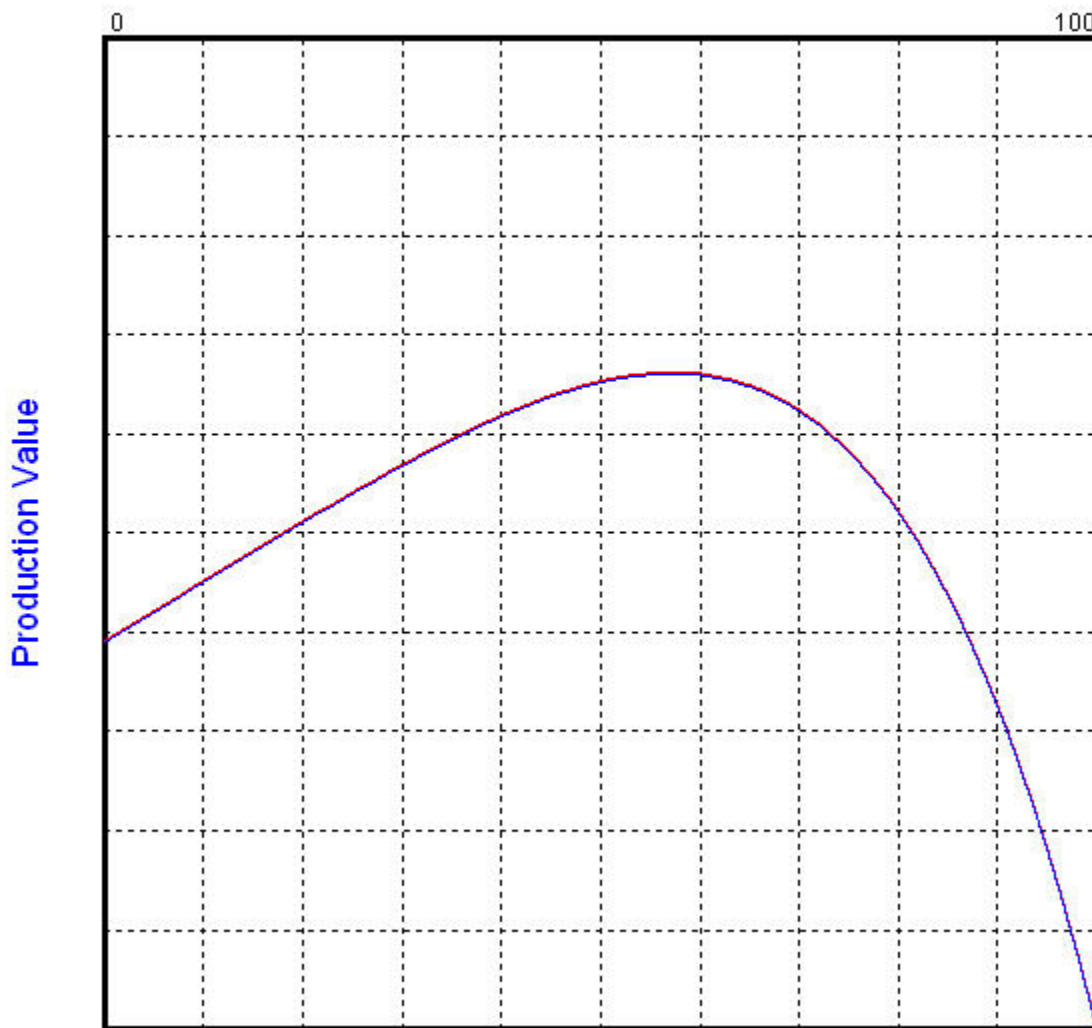


Figure 2
Impact of Increasing Speed

Figure 2 shows the normal operating rate at 40% (left side of the graph). It also shows the company policy that allows the operator to increase production and accept some degradation in overall quality up to a certain point (upward slope of the curve), at which any increase in production would create unacceptable rejects and shorten the life of the machine (downward slope of the curve).

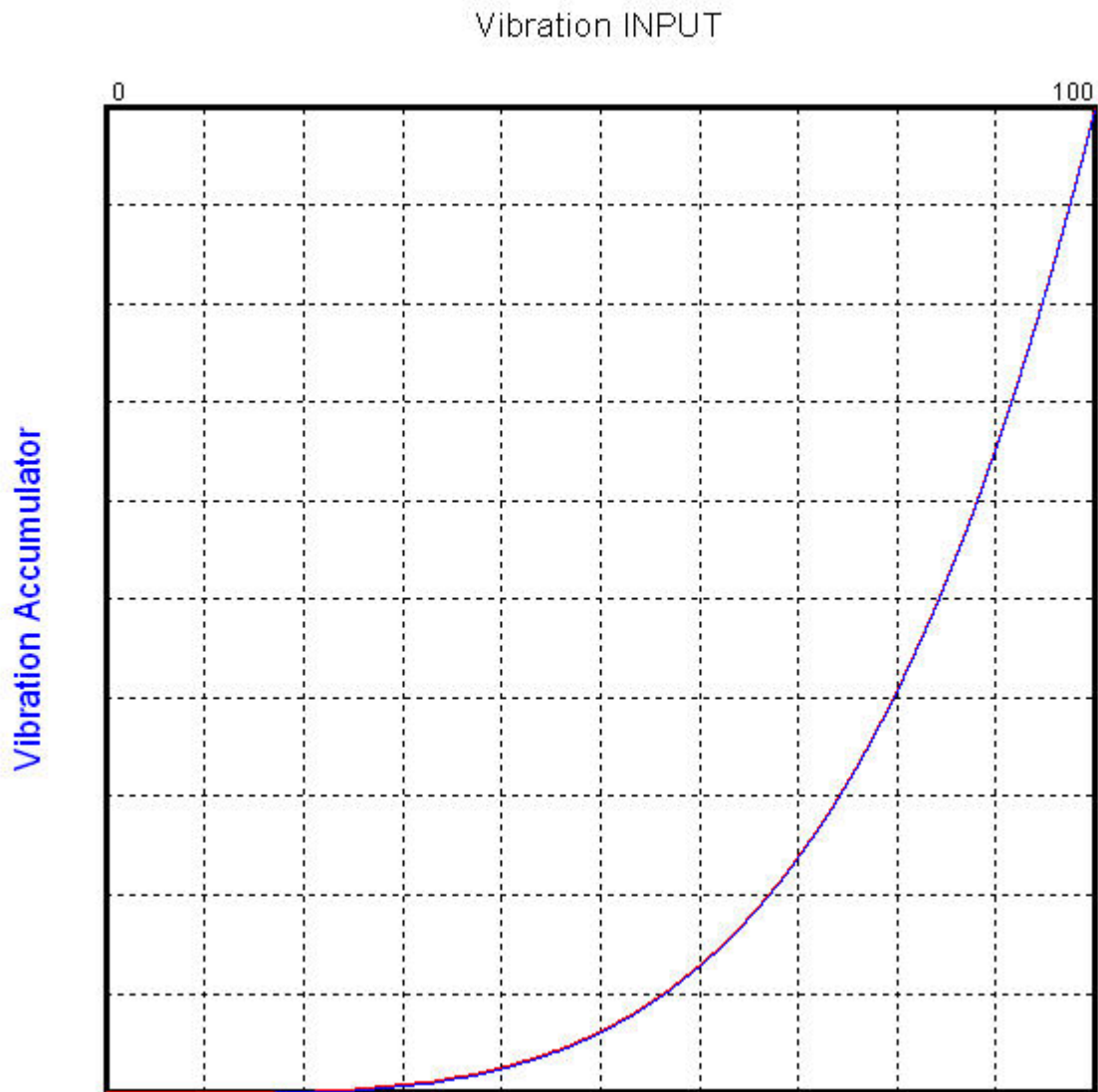


Figure 3 shows the policy for interpreting vibration on the life of the machine. In this case a little vibration is acceptable (and expected).

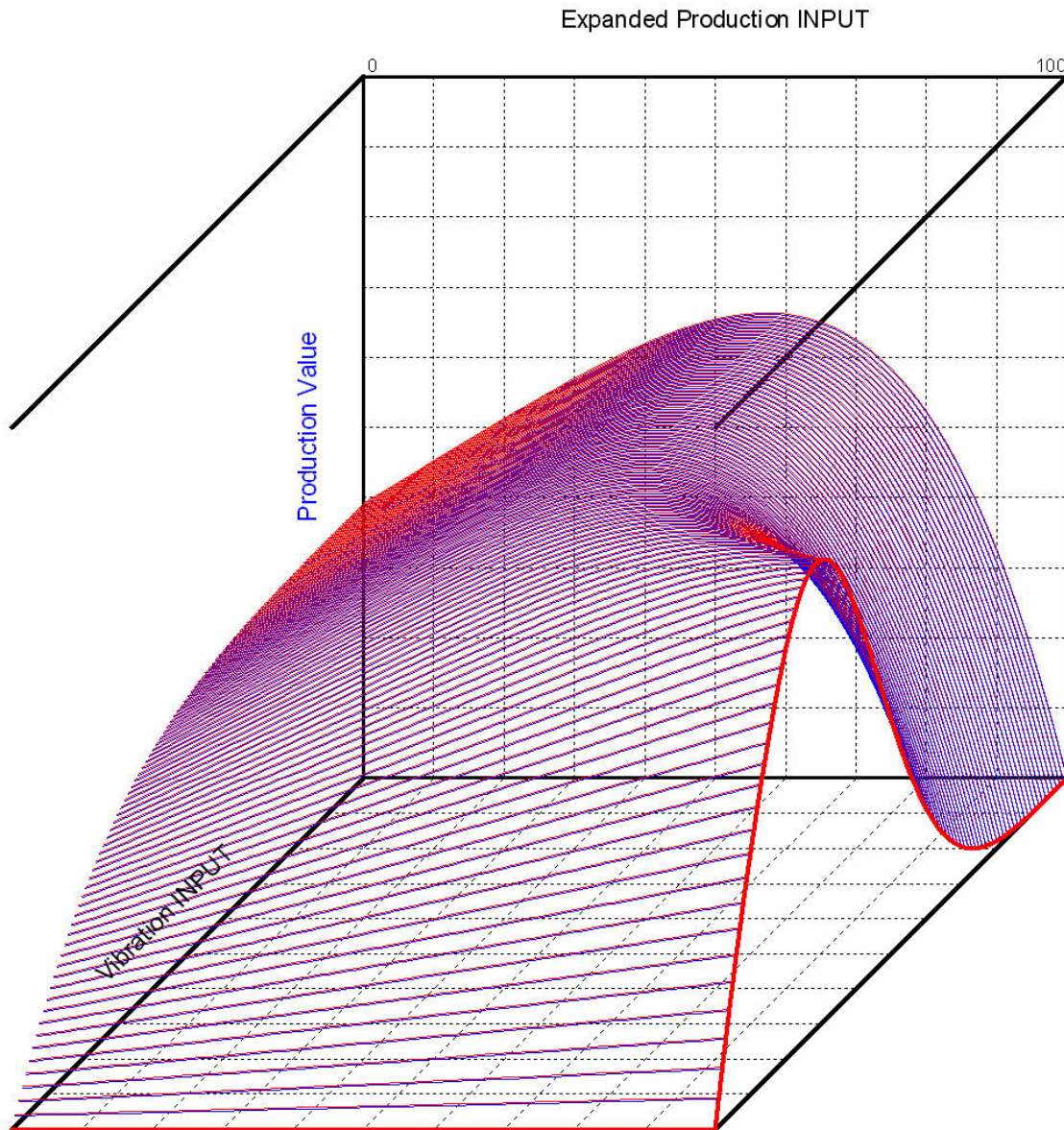


Figure 4
Integration of Machine Speed and Vibration on overall Business Value

Figure 4 shows how vibration can be used together with desired machine speed to identify the appropriate maximum speed. It would be difficult (or impossible) to tell operators how to integrate these values in their head and control the equipment according to policy.

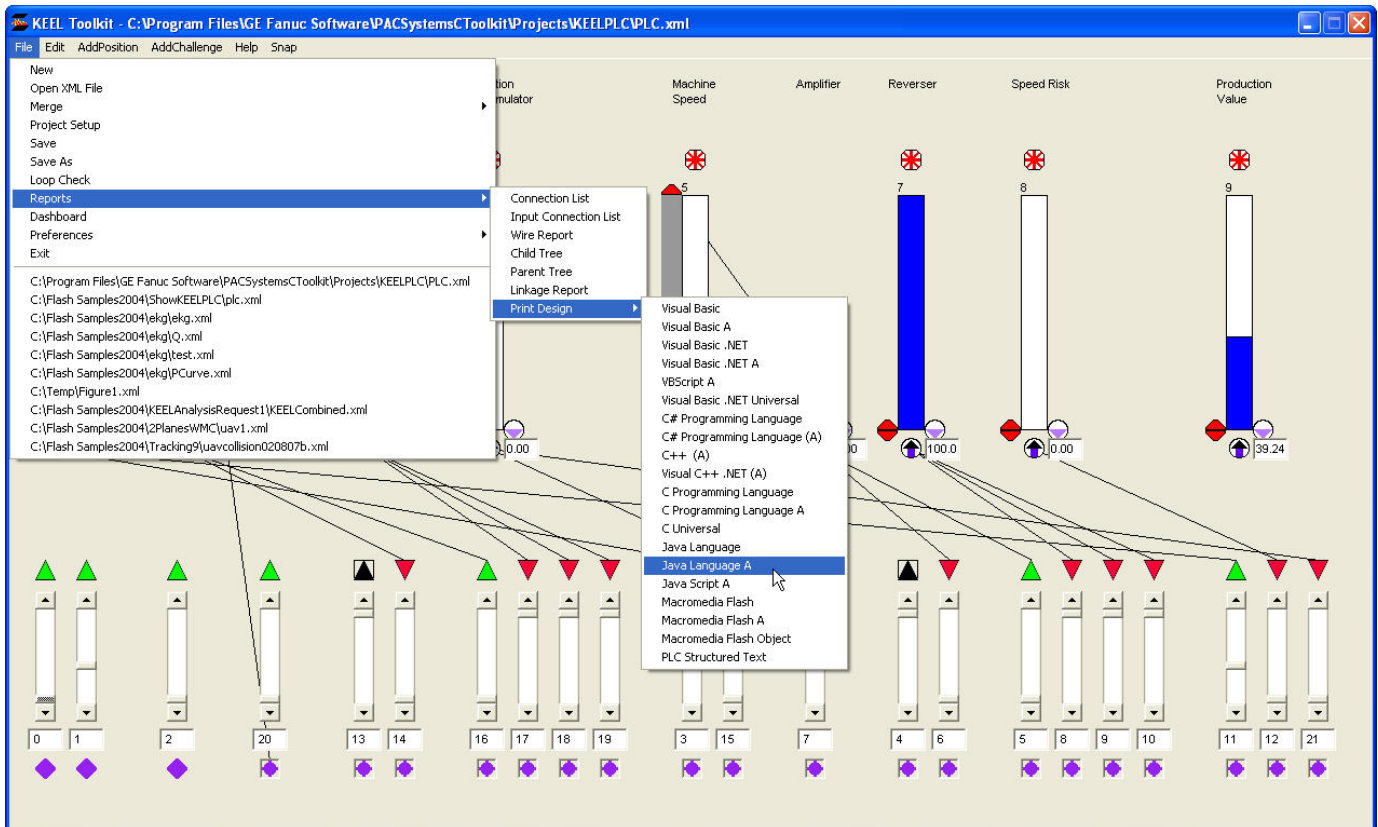


Figure 5
Creating a KEEL Engine in C

Referring to Figure 5, it took approximately 15 minutes to develop this sample model. The output of the model is translated into bug-free C source code as a text file. In this case a file named KEELPLC.txt is created.

GE/Fanuc provides a “C Toolkit for PACSystems” application that runs on Windows-based PC’s. They provide a user’s manual that describes its installation and use: “C Programmer’s Toolkit” for PACSystems (User’s Manual, GFK-2259B, November, 2005).

This Application Note assumes that the GE/Fanuc documentation is sufficient to describe the general use of C Function Blocks.



PACSystem C Code:

GE/Fanuc provides *include* files that provide the interface between the Proficy Machine Edition Function Blocks and the C code:

PACRXPlc.h – is used for any PAC System

PACRX7iPlc.h – is used when the target platform is the RX7i

PACRX3iPlc.h – is used when the target platform is the RX3i

In our example, we will target our KEEL engine for any PAC system.

This Application Note will assume that the “C Toolkit for PACSystems” was installed in the **C:\ProgramFiles\GE Fanuc Software\PACSystemsCToolkit** directory.

It also assumes that a **Projects** subdirectory was also automatically created during the install. The C Toolkit instructions suggest creating a separate subdirectory under **Projects** for each C Toolkit project. In this case, we will create a subdirectory called **KEELPLC** for our C KEEL engine development.

To create the appropriate code for the “C Toolkit for PACSystems”, copy the KEELPLC.txt file to “KEELPLC.c” using Windows copy and paste or rename services.

The new file will exist at:

C:\ProgramFiles\GE Fanuc Software\PACSystemsCToolkit\Projects\KEELPLC\KEELPLC.c

Open the KEELPLC.c file with Notepad, or other text editor.

Insert the following line at the beginning of the KEELPLC.c file:

```
#include <PACRXPlc.h> /*For C blocks that run on any PAC System*/
```

If the target had been for either the RX7i or RX3i, then one of the next two lines would have been inserted:

```
#include <PACRX7iPlc.h> /*For C blocks that run on RX7i PLC*/  
#include <PACRX3iPlc.h> /*For C blocks that run on RX3i PLC*/
```




The KEEL Toolkit includes a section in the header with the suggested calling procedure for the KEEL Engine. Figure 6 shows the commented area from the KEEL header.

```

/*****
/* General process for calling KEEL routines */
/*
/* This code is 'almost' valid C code. You are asked to develop your own code to */
/* read the external inputs and load them into the argvalues table. If you copy the */
/* following code into your source, you will need to personalize a couple of lines. */
/*****

/* {add code to load external inputs into argvalues() table} */
/* (Replace 'x' below with the appropriate values normalized between 0 and 100.) */
/* argvalues[0]=x; /* Expanded Production INPUT */
/* argvalues[1]=x; /* Basic Production INPUT */
/* argvalues[2]=x; /* Vibration INPUT */
/* dodecisions(); */
/* calculate outputs based on buffered inputs */
/* {post the output values or derived values from modposvalues(), posvalues(), */
/* threshvalues() to external control functions.} */
/* You may want to call the loginputs(); routine to log inputs to a file. */
.

```

Figure 6

From the GE/Fanuc C Toolkit documentation we know that C Function Blocks use a GefMain routine as the interface.

From our KEEL design (and from the pseudocode) we know that this KEEL Engine (KEEL Function Block) will have three (3) inputs that are loaded into the **argvalues** array at index positions 0, 1, and 2.

argvalues[0] will hold an input signal for “expanded production”
argvalues[1] will hold an input signal for “basic production”
argvalues[2] will hold an input signal for “vibration”

From our KEEL design, we know that we will be using two (2) outputs. These will come from the **modposvalues** array at index positions of 0 and 2.

We will need one additional input to the KEEL Engine that will trigger the first time the engine is called to initialize the variables used internal to the KEEL Engine.

The inputs and outputs will be tied to the KEEL engine by reference pointers.



The following code is added at the very end of the KEELPLC.c file:

```
int GefMain(T_BOOLEAN *init, T_REAL32 *extrarate, T_REAL32 *baserate, T_REAL32 *vibration, T_REAL32
*combinedrate, T_REAL32 *kalarm)
{
  if((init==NULL)|| (extrarate==NULL)|| (baserate==NULL)|| (vibration==NULL)|| (combinedrate==NULL)||
(kalarm==NULL)) return(GEF_EXECUTION_ERROR);
  if(*init==1){
    initializefixedtables();
    *init=0;
  }
  argvalues[0]=*extrarate;
  argvalues[1]=*baserate;
  argvalues[2]=*vibration;
  dodecisions();
  *combinedrate=modposvalues[0];
  *kalarm=modposvalues[2];
  return(GEF_EXECUTION_OK);
}
```

Figure 7

Figure 7 shows the GefMain subroutine that defines the interface to the KEEL engine.

*init references a BOOLEAN value that acts as a flag for initializing the variables within the KEEL Engine.

*extrarate references a REAL value that defines extra production

*baserate references a REAL value that defines the base production rate

*vibration references a REAL value that defines the amount of sensed vibration

*combinedrate references a REAL value for the integration of the extrarate, baserate, and vibration

*kalarm references a REAL value that will trigger an alarm for service if the interpreted risk associated with vibration rises above the trigger point

The simple logic of this routine is:

- If any of the variable pointers are null, indicate an error and exit
- If this is the first time, call initializefixedtables to initialize all internal variables
- Load the inputs
- Call the internal dodecisions routine to process the information
- Return the outputs



Use the PACSystems C Toolkit to create the object file:



When the PACSystems C Toolkit is installed on the PC, an icon is placed on the desktop. Double clicking on this icon will open the C Toolkit in a DOS Window.

Figure 8 below shows the results of first using the `cd` command to change to the KEELPLC directory (line 5).

```
cd KEELPLC
```

Lines 7 and 8 show the command to compile the code:

```
compileCPACRX KEELPLC
```

NOTE: The C Toolkit assumes the `.c` extension to the filename. (KEELPLC.c)

The remainder of the window shows the progress of the compile process.

If there had been any errors during the compile process they would be indicated here.

```
C:\ PACSystems(TM) C Toolkit
Toolkit Release: 3.5; Toolkit Build Number: 34a1
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Program Files\GE Fanuc Software\PACSystemsCToolkit\Projects>cd KEELPLC

C:\Program Files\GE Fanuc Software\PACSystemsCToolkit\Projects\KEELPLC>compileCP
ACRX KEELPLC
Toolkit Release: 3.5; Toolkit Build Number: 34a1
Compiling code with features available at or before
release 3_5 of the PACSystems C Toolkit.
gccElfX86 -mcpu=pentiumpro -march=pentiumpro -nostdlib -fno-builtin -fno-defer-p
op -Wall -I. -DCPU=PENTIUM3 -c -finstrument-functions -oKEELPLC.gef0 KEELPLC.c

ldElfX86 --entry=_start -o test.sm KEELPLC.gef0 ctkGefStartup.gef0 ctkBlkToolki
tInfo.gef0 ctkPlcLibFault.gef0 ctkPlcLibRefMem.gef0 ctkPlcLibFunc.gef0 ctkPlcLib
Bus.gef0 ctkPlcLibStackCheck.gef0 ctkPlcLibVariables.gef0 ctkStackCheckStub.gef0
 ctkCRunTimeStub.gef0 ctkTargetStub.gef0
ldElfX86 -r -d --entry=_start -o plc\KEELPLC.gefElf KEELPLC.gef0 ctkGefStartup.
gef0 ctkBlkToolkitInfo.gef0 ctkPlcLibFault.gef0 ctkPlcLibRefMem.gef0 ctkPlcLibFu
nc.gef0 ctkPlcLibBus.gef0 ctkPlcLibStackCheck.gef0 ctkPlcLibVariables.gef0

C:\Program Files\GE Fanuc Software\PACSystemsCToolkit\Projects\KEELPLC>
```

Figure 8

The compile process creates a new subdirectory inside of the KEELPLC directory called **plc**.

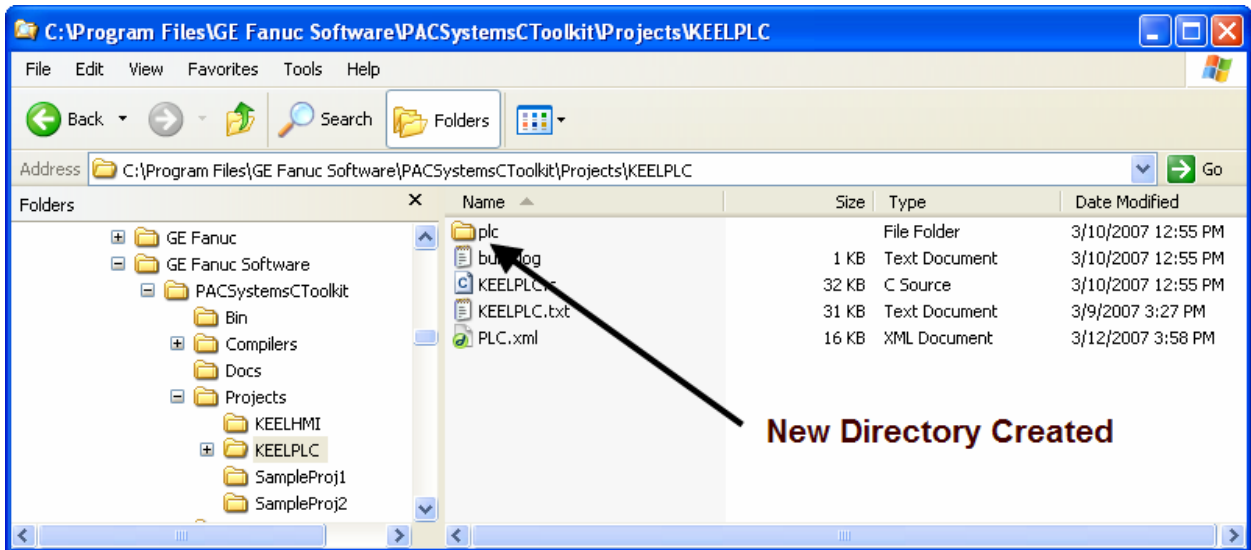


Figure 9

This directory holds the object file that was created by the PACSystems C Toolkit. (KEELPLC.gefElf)

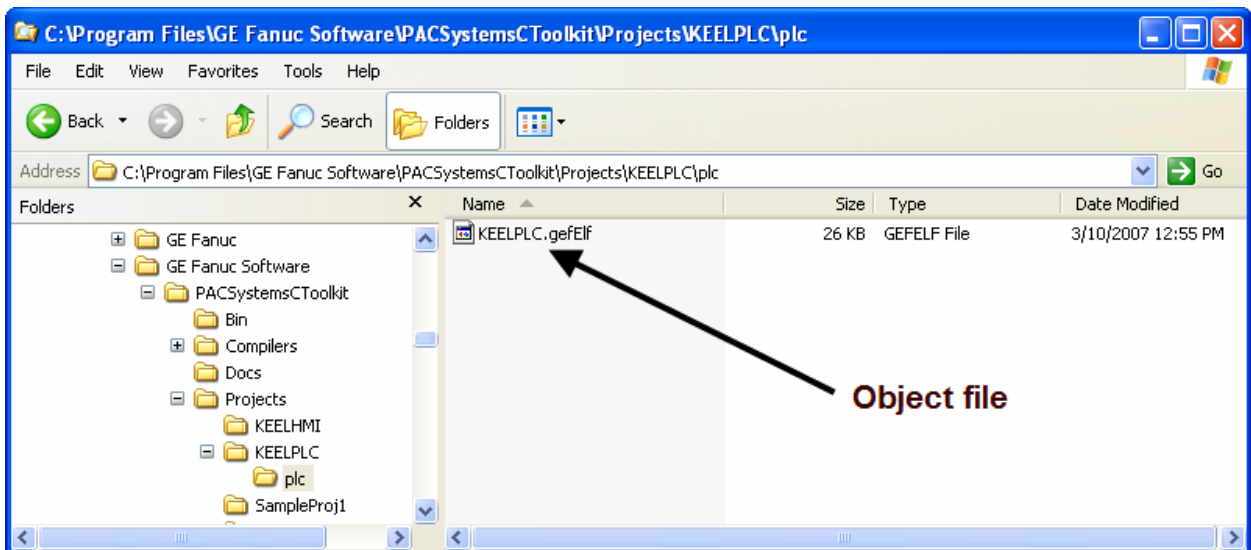


Figure 10
Object File: KEELPLC.gefElf

At this point a new C Function Block is available for use within the Proficy Machine Edition environment.

Proficy Machine Edition:

{This Application Note is not intended to be a tutorial for the use of Proficy Machine Edition}

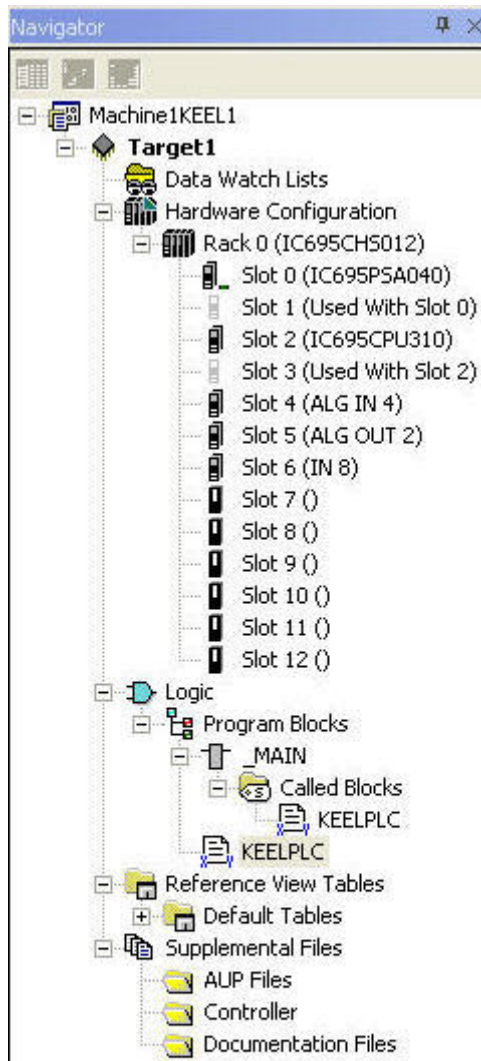


Figure 11 shows the Navigator window from within Proficy Machine Edition. This shows the hardware configuration selected for this example. It includes an analog input card in slot 4 and an analog output card in slot 5 and a discrete input card in slot 6.

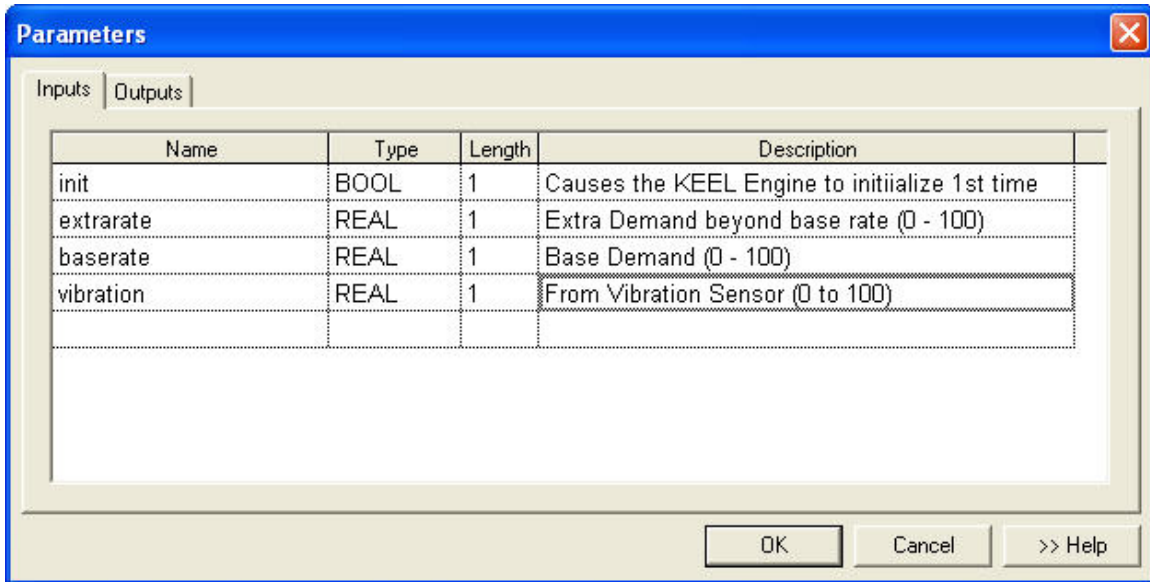
This figure also shows the logic components used in this design. It includes a main RLL program and a Function Block (KEELPLC) that includes the KEEL engine previously designed.

The following procedure was used to incorporate the C Block in the application: (from Chapter 3 Writing a C Application in the GE Fanuc “C Programmer’s Toolkit” User’s Manual, GEFK-2259B)

1. In the Project tab, expand the Logic node.
2. Right click the Program Block node under the Logic node.
3. Select Add C Block. This brings up a file navigation dialog box.
4. Navigate to the *.gefElf file and click the Open button to add the C Block to the folder.

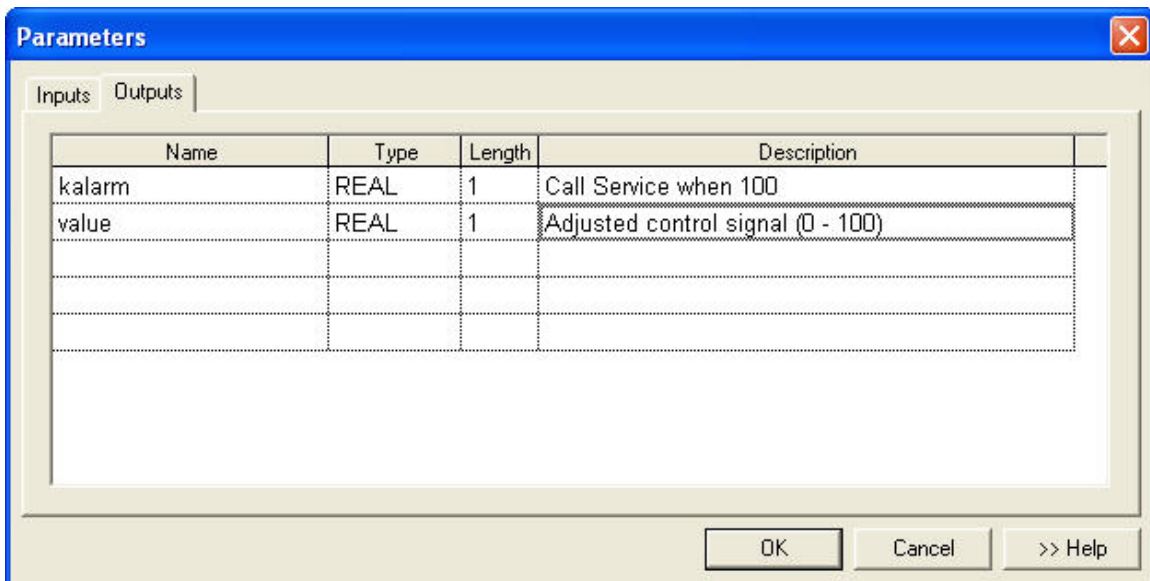
Figure 11

To specify the parameters for our C Block, click on the C Block in the Navigator. In the properties page for the C Block, click on the Parameters item and then click on the button provided. This opens the Parameters dialog box with two tabs, one for inputs and one for outputs. For this example they are filled in as follows:



Name	Type	Length	Description
init	BOOL	1	Causes the KEEL Engine to initialize 1st time
extrarate	REAL	1	Extra Demand beyond base rate (0 - 100)
baserate	REAL	1	Base Demand (0 - 100)
vibration	REAL	1	From Vibration Sensor (0 to 100)

Figure 12
Inputs



Name	Type	Length	Description
kalarm	REAL	1	Call Service when 100
value	REAL	1	Adjusted control signal (0 - 100)

Figure 13
Outputs



The C Block can be invoked in a number of ways. See the C Programmer's Toolkit User's Manual for details.

In Figure 14 we show it as a sub-block of the main block.

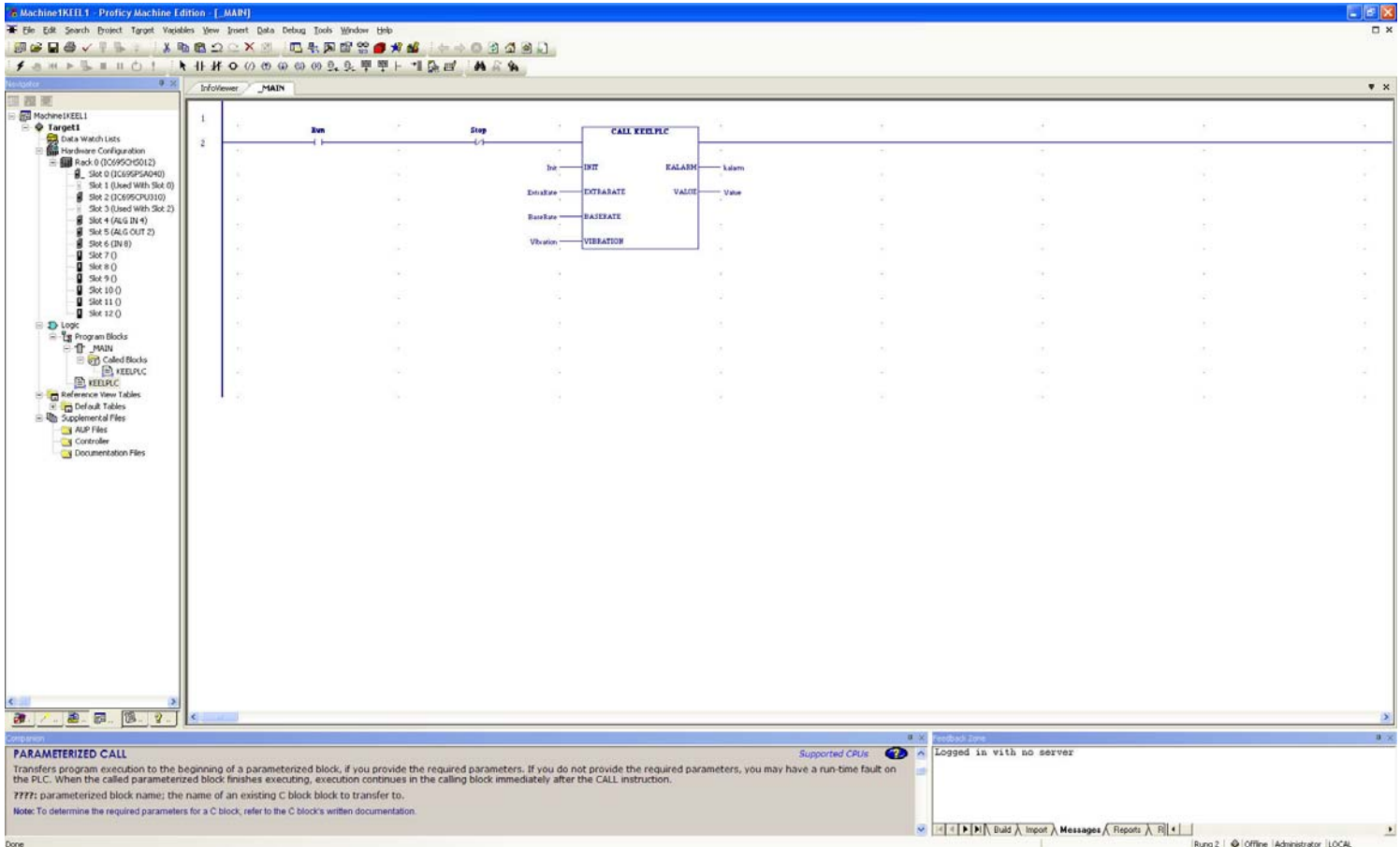


Figure 14

A better view of the function block is shown in Figure 15.

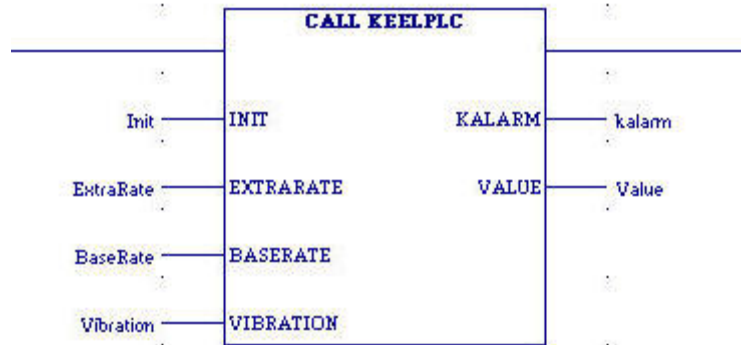


Figure 15

Right clicking on the spaces outside of the function block allows the assignment of specific hardware registers.

Figure 16 shows the assignment of a BOOLEAN variable for Init that defaults to True (on) when the application starts. This variable is reset the first time the KEELPLC function block is called.

Figures 17 – 20 shows the assignment of symbolic references to specific hardware registers.

Inspector	
Variable [Target1]	
Name	ExtraRate
Description	
Publish	False
Array Dimension 1	0
Data Source	GE FANUC PLC
Ref Address	%AI0001
Data Type	REAL
Current Value	0.0
Initial Value	0.0
Default Display Format	Scientific notation
Precision	7
Retentive	True
General	

Figure 16

Inspector	
Variable [Target1]	
Name	Init
Description	
Publish	False
Array Dimension 1	0
Data Source	GE FANUC
Ref Address	
Data Type	BOOL
Current Value	Off
Initial Value	On
Default Display Format	On / Off
Retentive	True
Initial Force State	Not Forcec
Current Force State	Not Forcec
General	

Figure 17

Inspector	
Variable [Target1]	
Name	BaseRate
Description	
Publish	False
Array Dimension 1	0
Data Source	GE FANUC PLC
Ref Address	%AI0002
Data Type	REAL
Current Value	0.0
Initial Value	0.0
Default Display Format	Scientific notation
Precision	7
Retentive	True
General	

Figure 18

Inspector	
Variable [Target1]	
Name	Vibration
Description	
Publish	Internal
Array Dimension 1	0
Data Source	GE FANUC PLC
Ref Address	%AI0003
Data Type	REAL
Current Value	0.0
Initial Value	0.0
Default Display Format	Scientific notation
Precision	7
Retentive	True
General	

Figure 19



The image shows a software window titled "Inspector" with a close button (X) in the top right corner. The window contains a table with two columns: "Variable [Target1]" and "Value". The table lists various properties of a variable, including Name, Description, Publish status, Array Dimension 1, Data Source, Ref Address, Data Type, Current Value, Initial Value, Default Display Format, Precision, and Retentive status. The "General" tab is selected at the bottom left of the window.

Variable [Target1]	Value
Name	
Description	
Publish	Internal
Array Dimension 1	0
Data Source	GE FANUC PLC
Ref Address	%AQ0001
Data Type	REAL
Current Value	0.0
Initial Value	0.0
Default Display Format	Scientific notation
Precision	7
Retentive	True

Figure 20

Dependencies:

This Application Note utilizes Compsim's KEEL Toolkit to develop the KEEL Engine C source code. A text editor was used to add the few additional lines of C code to the KEEL C code. The GE/Fanuc C Toolkit for PACSystems was used to compile the KEELPLC.c source code and create the KEELPLC.gefElf object file. The GE/Fanuc Proficy Machine Edition development environment was used to create the application files for the GE-PAC system.



Summary:

While this example shows an almost trivial KEEL application, it would have been significantly more difficult to develop this application manually in C. The code would have to be developed, debugged and tested using other means.

Should the manually developed C code ever need to be audited and extended, the life cycle costs would escalate even higher.

KEEL Technology enables GE Programmable Automation Controllers to satisfy the needs of much more complex applications that include dynamic, non-linear, inter-related, multi-dimensional problem sets. This satisfies three market demands: 1) the need to utilize COTS industrially hardened equipment to address more complex problems, 2) the need to support life cycle cost issues at the same time, and 3) the opportunity to incorporate business practices in the form of executable policies directly into the controller.

The GE Fanuc PAC family of controllers provides an excellent host for KEEL Engines encapsulated in IEC 61131-3 Function Blocks.

Contact:



Compsim LLC is a provider of next generation cognitive technology for application in automotive, industrial automation, medical, military, governmental, enterprise software and electronic gaming markets. The company is headquartered in Brookfield, Wisconsin.

Compsim LLC
PO Box 532
Brookfield, Wisconsin 53008
(262) 797-0418
<http://www.compsim.com>