RTP 2000

Software Demo Guide

For assistance in running this demonstration software,

call RTP at 954.974.5500 and ask for Customer Service

or send email to rtpsales@cpipad.cpi.sprint.com

Our office hours are 8 AM to 5 PM Eastern Time

CONTENTS

Setup and Installation

System Requirements

To run the RTP 2000 Software Demo, you need a PC running Windows and the RTP 2000 Demo Download File. Your PC system should include a minimum of:

- IBM AT 286, 386, 486, or compatible PC
- 4 Megabytes of RAM
- EGA or VGA video monitor and adapter
- DOS 3.0 (or later version)
- MS Windows 3.1 (or later version)
- Mouse pointer, 2-button or 3-button

Download Demo Installation

1. Move the **RTP2000.EXE** file into a temporary directory, e.g. C:\RTP-TEMP
2. Open File Manager and expand the file by double clicking, then press F5 to refresh the screen.
3. Double click on the file **INSTALL.exe** in C:\RTP-TEMP
4. Follow the instructions and click “OK” to run the demo program.
5. Select MS-DOS as the Object Engine type.
6. When the GELLO Main Menu is displayed, select **File - Open - sample1.thr** to begin the demonstration.
7. Follow the instructions in the comments on the screen.

Software Key

The RTP 2000 configuration software is a program called GELLO®. Normally GELLO comes with a software key, a small electronic piece enclosed in plastic which connects to the parallel port on your PC. This key must be installed on the LPT1 port to run the RTP 2000 software. The key does not interfere with the use of a printer on the same port.

NOTE: This demonstration program ***does not require*** the software key in order to run.

Mouse Settings

The demo program assumes that the mouse is set to standard (default) operation. If you have trouble accessing Objects with the right mouse button, be sure to check that the mouse is not specially configured. Proper settings should be unassigned or single click as follows:

	Left Button	**Middle Button**	**Right Button**
For 2-Button Mouse	single click		single click
For 3-Button Mouse	single click	single click	single click

On-Line Help

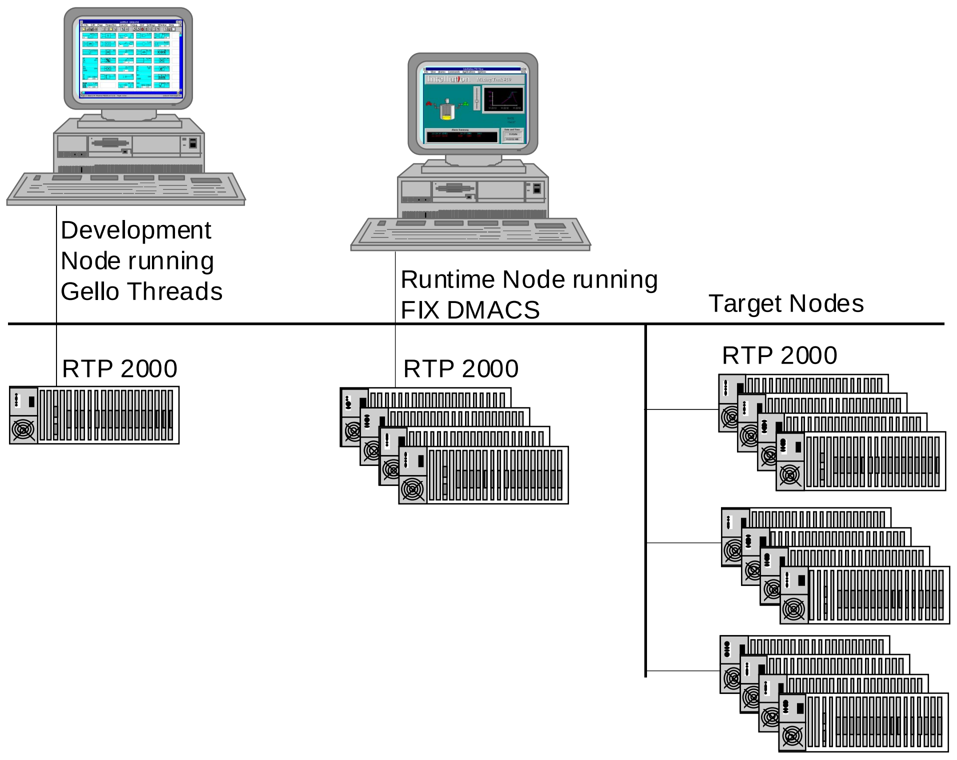
By clicking on the Help menu option, you can access the entire GELLIX user’s manual at all times. It works just as any Windows-based Help facility. Use it to get more information on the specific menu options, object definitions, or for more information on How to ... .

RTP 2000 - The Analog PLC

Traditionally PLC's (Programmable Logic Controllers) have provided a high speed, reliable implementation of digital control with actuators such as relays, solenoids, switches, and valves. A methodology for the design of these systems has been standardized using ladder logic. A ladder of power rails and rungs of ON/OFF actions are processed in sequence. The PLC was designed to efficiently process ON/OFF type activities, not wide range signals found in analog devices such as strain gauges, thermocouples, and variable speed motors. The method for controlling analog processes has been with large scale DCS (Distributed Control Systems) requiring significant processing power, or with highly specialized DSP (digital signal processing) processors. The RTP 2000 extends common PLC functionality by combining the familiar tools of ladder logic with the capability of high speed analog processes, thus the name: Analog PLC. Some of the RTP 2000 advantages, described below, include:

- ➢ Factory hardened and nuclear qualified equipment
- ➢ DCS functionality
- ➢ PLC ease of use and performance
- ➢ Relay level cost of ownership
- ➢ MMI friendly
- ➢ Complete line of high performance analog and digital functions
- ➢ Interconnectivity with multiple network types
- ➢ Graphical control configurator based on ladder logic objects
- ➢ Documentation and Control Logic always in sync

RTP 2000 systems can be purchased in three variations: Development Environment, Runtime MMI, and Target Node, illustrated in the figure below.



Development Environment

The RTP 2000 Development Environment combines GELLO THREADS®' point-and-click ladder logic object-oriented control program development software with popular graphically oriented MMI software (such as FIX DMACS for Windows®) and the necessary hardware to easily develop process control programs and friendly operator interfaces.

Runtime MMI

Whether you are running a custom program, or a standard package, the RTP 2000 Runtime MMI system provides a high performance data acquisition and control system for your process. It includes the FIX MMI running on a user-supplied PC. FIX MMI is a full function operator interface program that supports Windows-based icons and operation as well as graphic representations of processes.

Target Node

Programs developed on the RTP 2000 Development Environment are downloaded to the Target Nodes for high performance I/O.

RTP I/O Option Cards

Each RTP 2000 chassis can support up to 7 additional (optional) RTP I/O chassis for almost unlimited I/O possibilities. RTP I/O Option cards provide a full range of I/O choices: from 5 V to 240 V AC or DC, low level millivolt to 10.24 V, 4-20 mA or 10-50 mA, thermocouples, RTDs, relays, and strain gauges, among others.

DCS functionality

In addition to highly accurate and fast analog input and output capabilities, the RTP 2000 can also have multiple target nodes communicating over a TCP/IP Ethernet network.

PLC ease of use and performance

The PLC has traditionally been programmed using ladder logic. The RTP 2000 builds on the ladder logic form adding functionality for analog processes.

Relay level cost of ownership

The cost of owning a highly reliable RTP 2000 system compares with the cost of implementing and maintaining an equivalent relay system. Plus, the RTP 2000 system gives you flexible configuration with automatic documentation.

MMI friendly

RTP 2000 accepts many MMI programs. Once an application is downloaded to the target processor board, it may communicate to a MMI (man-machine interface) running on the PC host. MMI’s supported are FIX DMACS using a dedicated server, and any software supporting the DDE protocol from Microsoft; for example, Microsoft Excel or Wonderware.

Complete line of high performance analog and digital functions

The RTP 2000 supports the full line of RTP I/O. Typical functionalities are: digital input, digital output, analog input, pulse counter, frequency counter, programmable waveform generator, synchro resolver.

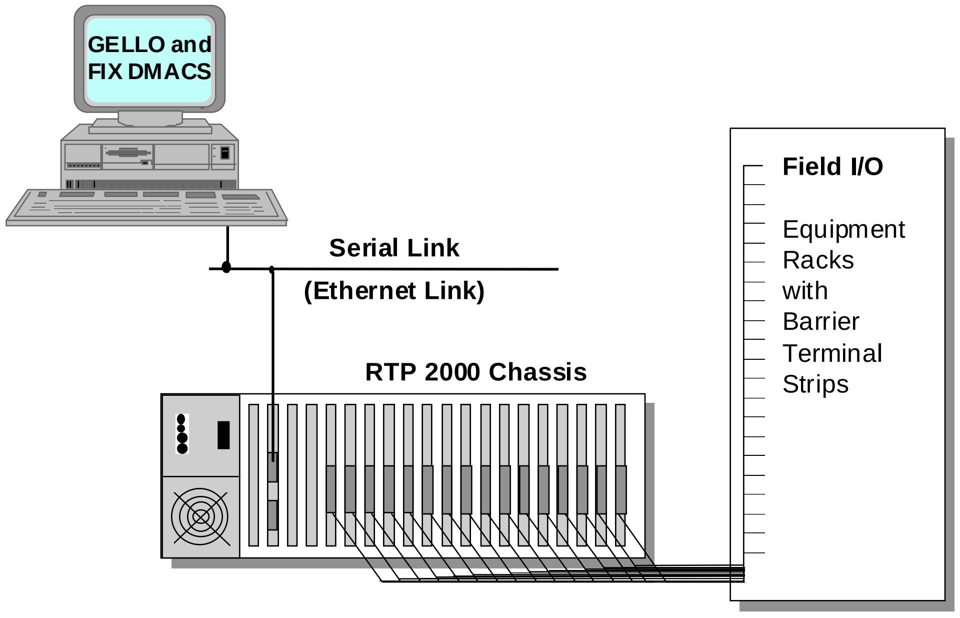
Interconnectivity with multiple network types

The RTP 2000 was designed using open standards. As a result, it can transfer data across a multiple of network types and protocols. All communications drivers process and respond to incoming messages from any source, an editor, another engine, or a user interface. Network communications modules are available for RS-232 serial and TCP/IP.

Documentation and Control Logic always in sync

The graphical design interface of the application development tool provides a visually obvious documentation history. There are no cryptic programming language modifications. A graphical display of the logic clearly shows object and connection changes.

RTP 2000 Architecture



RTP 2000 Configuration Software

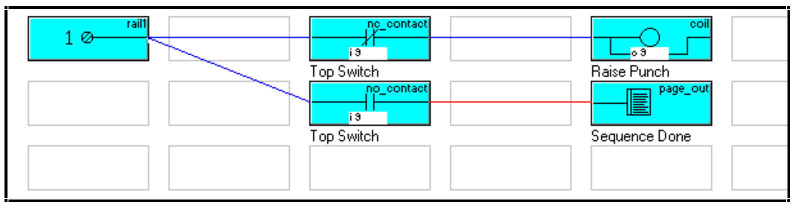
GELLO is an object-oriented control language that provides programming of control by individuals with an understanding of control and automation, in their terminology and symbols. GELLIX is the visual editor, or CASE (Computer Aided Software Engineering) tool, for GELLO which allows the control programs to be generated using symbolic objects.

The GELLO programming environment is more like that of programming a data flow rather than a series of low level steps. By zooming in on each major activity, the macros, or low level steps, are shown in detail.

At the highest level you deal with pages, which are used as your own custom objects, and are equivalent to subroutines, or functions. Each page is a series of program steps, called objects, for one or more items of equipment. You can pass arguments and return values from a page. There are no limitations on the placement or nesting of pages other than memory.

The concept of pages allows proven programming routines for specific machines or process equipment to be incorporated into a common source file library. You can add functions to the library that solve special problems, such as those related to a particular process, which can be used throughout a company by passing along source code.

Programming in GELLO is similar to ladder logic in that objects such as relays and contacts are shown visually on the screen. Connections are shown as lines between objects on the screen.



An important consideration is that programs can be written and substantially tested on any development computer running the GELLIX Editor by using the real-time simulator built into GELLIX. This speeds the debugging of the logic, and means that it is not necessary to have special programming equipment - or even a real-time computer - available for testing.

Similarly, real-time debugging is significantly improved using GELLO. You can attach the GELLIX Editor to the real-time processor running GELLO and watch the current state of objects, as well as make changes to object variables, on-line (similar to making register changes in ladder logic). In addition, GELLO permits single step debugging where the code in the target processor can be stopped at any object and incrementally run from object to object, or a pass at a time, for in-depth understanding and verification of the program.

There is no need to carry the source code on the development computer. You can attach directly to the target processor with a portable computer, for example, and upload the entire program, including comments. A change in software made by one shift technician can be retrieved by another shift technician by simply uploading the program. This keeps the documentation and control logic always in sync.

The use of proven low level modules (objects) permits building very large programs in an extremely short period of time. Because conventional syntactical compiling is not required, code development is simplified and quick. The high level visual editor, GELLIX, provides a fast, self-explanatory method of building code.

GELLO

GELLO is the name of the object-oriented programming language used to write control applications. It is an acronym for **G**raphically **E**nhanced **L**adder **LO**gic.

GELLIX

GELLIX is the GELLO Computer Aided Software Engineering (CASE) tool. It allows you to use preprogrammed objects to compose a GELLO program. GELLIX runs under the Microsoft Windows environment. The GELLIX Editor resides in the PC. After you develop the control application, it can be downloaded across the network to the OBJECT GT ENGINE in the RTP 2000 where it is executed. GELLO and GELLIX are extensible, and objects can be added or deleted to suit custom applications. A GELLO Developer’s Kit is available to allow you to write your own custom objects.

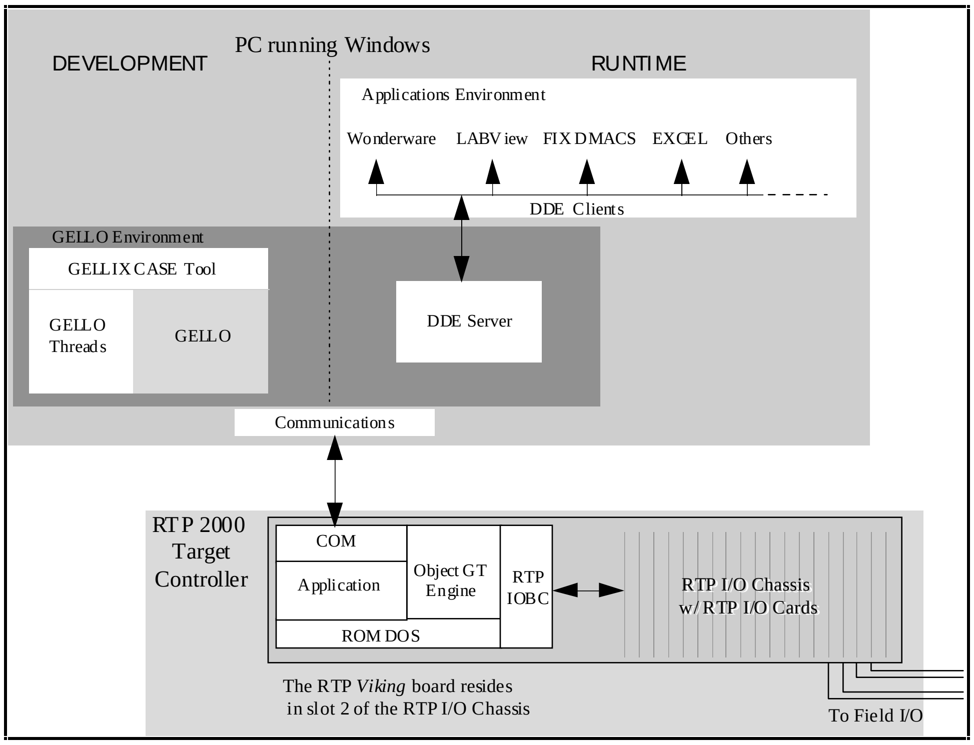
GELLO THREADS

This is the flow control portion of the program. Function objects, decision objects, forks and sync object are used to direct the execution of underlying GELLO THREADS and GELLO pages.

TARGET: RTP 2000 System

The terms target and target controller are used to describe the controller for which you are writing a program. The target hardware is a full-function RTP 2000 data acquisition system with an associated OBJECT GT ENGINE device driver created for it.

Application programs are developed in the GELLO environment and then downloaded to the target RTP 2000 Controller where the program resides. During runtime, the data is transferred to and from the MMI software via the DDE server.



Data flow in the PC and RTP 2000 Target Controller using GELLO.

OBJECT GT ENGINE

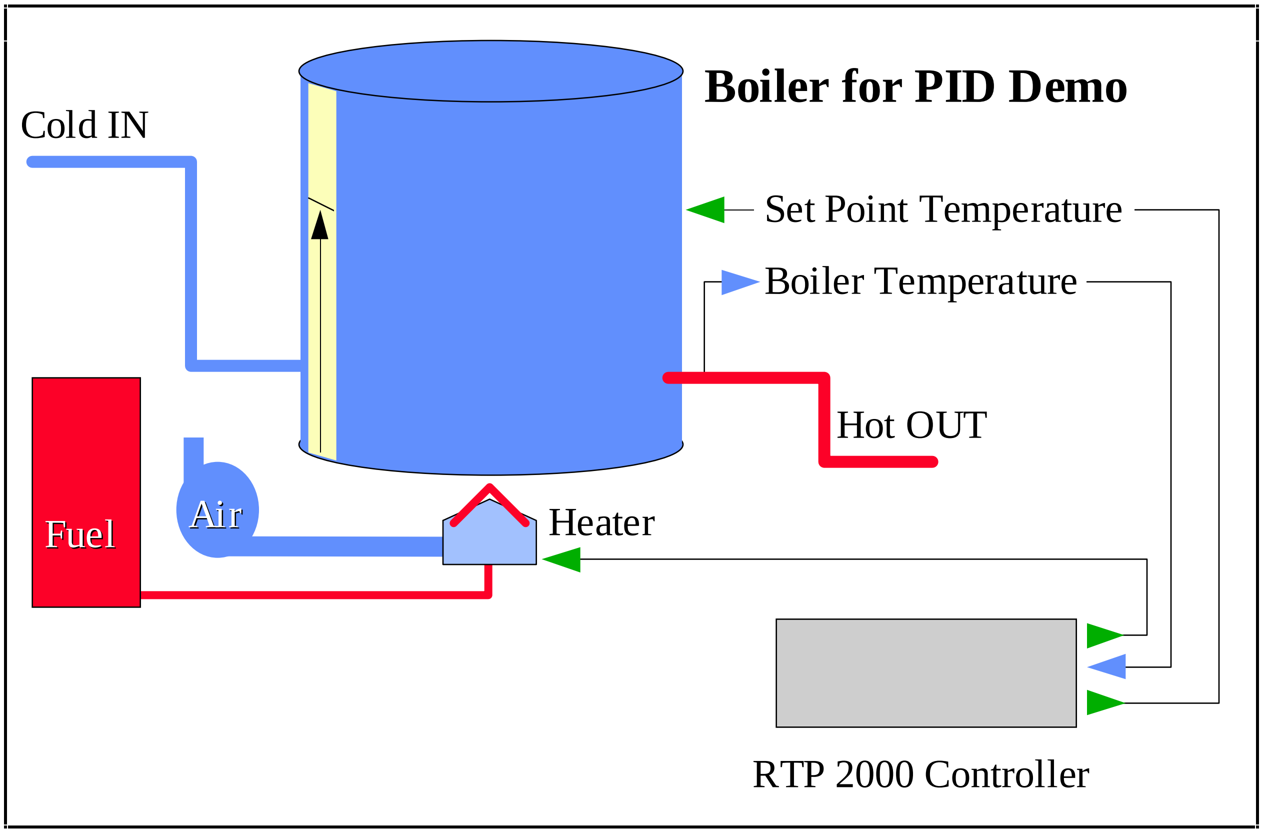
In the RTP 2000, the processor board or “Viking” board contains a 486DX CPU that executes the OBJECT GT ENGINE program code. This is the runtime code resident in the target controller that runs GELLO objects. A complete program, in hexadecimal form, can be downloaded to the target controller using the GELLIX Editor.

The program image is stored in non-volatile ROM on the Viking card for backup and power reset restart. The size of the program image will vary depending upon the complexity and number of symbols used in the GELLO program. An image is created and saved to the file system by the OBJECT GT ENGINE when receiving a new program from the editor. The OBJECT GT ENGINE communicates with the editor using a serial/Ethernet TCP/IP communications driver that is used on a PC-compatible communications port.

Sample Application

Problem - PID Controller for Boiler

With knowledge of ladder logic and control theory, you can create a useful control program in a matter of minutes. The sample program you are about to build represents a typical PID controller for a simple control loop. In this case we've chosen a boiler application where the temperature of the boiler contents is controlled by varying the fuel input to the boiler's heating device.



Solution

The solution for this application requires a simple PID controller. We've attached a waveform generator to the PID Set Point to provide high and low set points for the boiler. By adding several variables to the program, we can monitor and control minute aspects of this control loop. Finally, we added a Strip Chart to record the input and the output to get a visual representation of the performance of our PID control loop as well as to be able to see the effects of adjustments to different variables in the program. This lets us fine tune the PID control loop to our specifications.

Step by Step Instructions

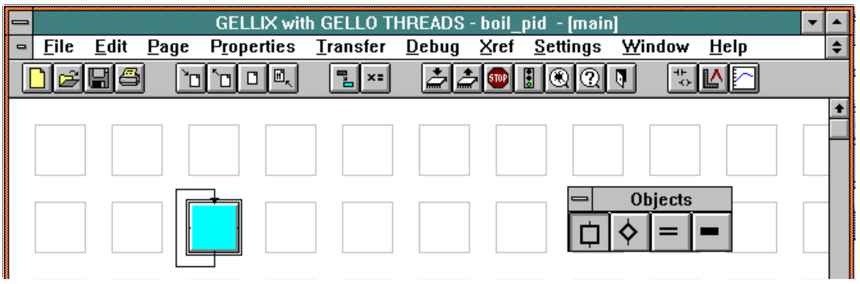
Before you begin creating your PID control program, if you have not already installed the GELLO program, do so now. Once the GELLO Main Menu is displayed, you can proceed.

1. Begin a new THREADS file

- Begin by selecting **File - New**.
- You can now name your program. Select **File - Save As** and give it a unique name such as **BOIL-PID**. Notice that it has a ***.thr** file extension to indicate it is a THREADS file. The program name appears in the Title Bar.

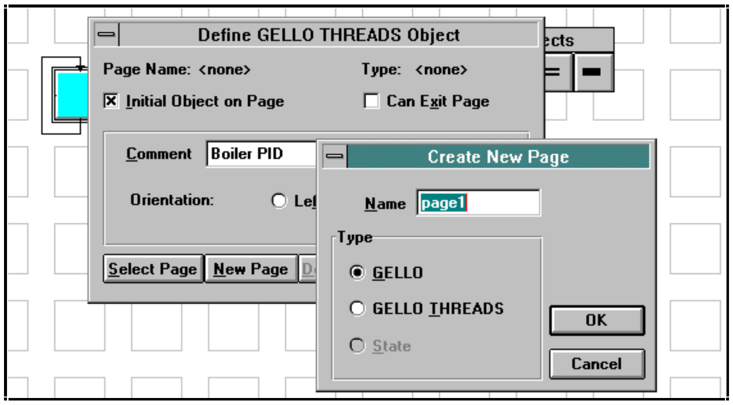
2. Create a THREADS Page and Add Sequencing Information

- To create a THREADS Page, click the right mouse button to display the Objects box.
- Select the **Process object** (square symbol on the left) and place it on the screen. You can place the Process object anywhere on this screen, but it's more convenient to place it somewhere near the upper left portion of the screen. This screen is now the **Main Page** of your program.
- To add the sequencing portion of the program, place the cursor over the lower middle portion of the Process object. The cursor will change appearance to become a **crosshair** if you have positioned it correctly. Now drag the crosshair cursor in a clockwise arc back to the top of the Page in this manner: down and to the left, then up, and then back to the right and down to connect to the top of the Process object. You've just created a continuous process. The screen should resemble the one pictured here.


GELLIX with GELLO THREADS - boil_pid - [main]
File Edit Page Properties Transfer Debug Xref Settings Window Help
Objects

3. Create GELLO Page containing the Control Logic

- GELLO architecture is hierarchical in design. The sequencing information is stored at the highest level called GELLO THREADS. Beneath that level are either THREADS pages containing more sequencing information or GELLO pages containing control logic.
- First, define the THREADS Object created in Step 2. Place the cursor at the lower right corner of the page object so that the cursor changes into a **small gray box** and click the left mouse button to display the **Define THREADS Object**. Here you can label the page object by typing the comment Boiler PID and selecting the position for the label to be displayed in respect to the object (Top, Right, Bottom, or Left).
- Next, create the underlying GELLO page by selecting **New Page** in the **Define GELLO THREADS Object**. When the Create New Page box is shown, check to see that the **Type** of page is a **GELLO** page, and enter a name for the page or accept the default **page1**. Select **OK** or press **Enter** to save these changes.



Page Name: <none> Type: <none>

[✓ Initial Object on Page] Can Exit Page

Comment: Boiler PID

Orientation: Lef

Name: page1

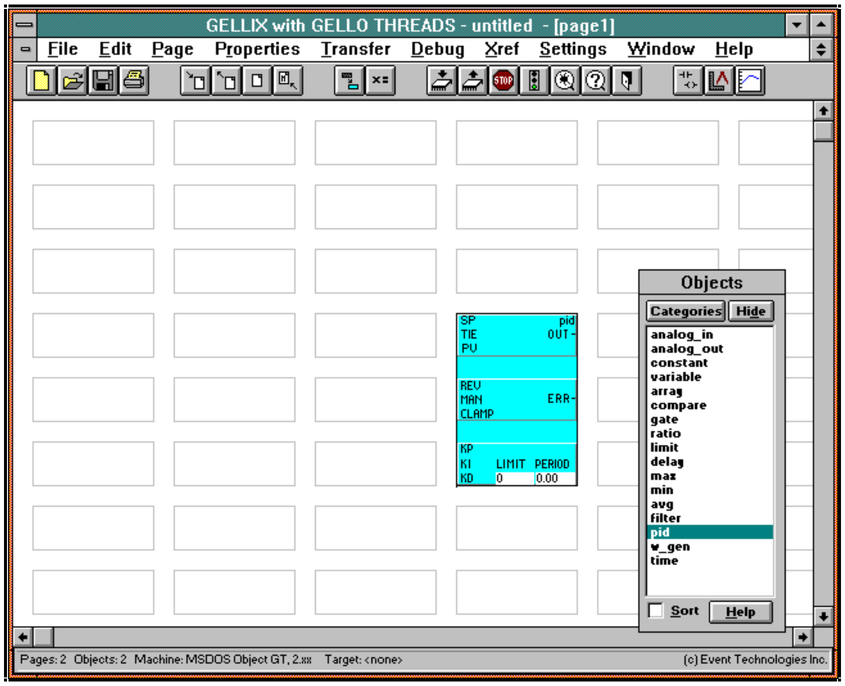
Type: [✓ GELLO] GELLO THREADS State

- To begin adding the GELLO objects, you need to open the next level down, or go to Page1. Do this by placing the cursor just inside the lower right corner of the Process object. Click the right mouse button. There are other methods of getting to Page1 you can explore yourself.
- You may return to the Main Page from any of the GELLO pages by pressing **Ctrl + M** at any time.

- Once Page1 is displayed (you should notice that the program objects have changed size from squares to rectangles to accept GELLO objects), you are now ready to begin programming the PID controller.

4. Add PID Object

- To add a **PID object**, click on the right mouse button. The **Object Categories Box** is displayed.
- Click on the **ANALOG** category and then click on pid. This selects the PID object to add to a Page.
- Now place your cursor in the 4th row in the 4th column and click the left mouse button to place the PID object on the Page.

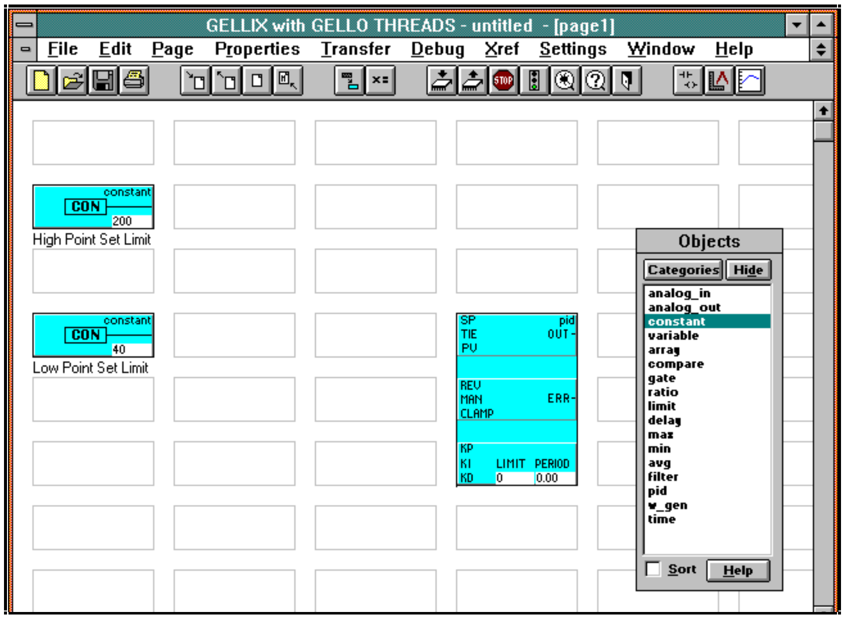

GELLIX with GELLO THREADS - untitled - [page1]
File Edit Page Properties Transfer Debug Xref Settings Window Help
SP pid
TIE OUT
PV
REV
MAN ERR
CLAMP
KP
KI LIMIT PERIOD
KD 0 0.00
Objects
Categories Hide
analog_in
analog_out
constant
variable
array
compare
gate
ratio
limit
delay
max
min
avg
filter
pid
w_gen
time
Sort Help
Pages: 2 Objects: 2 Machine: MSDOS Object GT, 2.xx Target: <none>
(c) Event Technologies Inc.

Easy, wasn't it? What you *didn't* see happen is that all of the programming code necessary to create the PID functions has automatically been added to your program now. This is truly point-and-click programming with objects. Now, let's finish the program.

5. Add Analog Constant Objects

- Add other objects in the same manner as you just added the PID object - by selecting them from the object list and pointing and clicking.
- To add a **High Limit Constant** (an input for the Waveform Generator we'll add later), click the right mouse button to display the object **Categories** and select **ANALOG** and then **constant**. Place the cursor on any position and click again to drop the object onto the Page. We recommend placing the constant at the far left-hand edge of the page for convenience. However, you are able to move objects (including their connecting threads) at any time by simply clicking them off, positioning the cursor at the new location, and clicking them on again.
- Label the constant by placing the cursor just below and outside the object's center and click to display the **Enter Comment** box. Type **High Set Point Limit** and press **Enter** or select **OK**.
- Next, set the **Initialization Value** for the constant by clicking in the lower right corner of the **constant object**. This value represents the upper limit of the Set Point. Enter the number ***200*** and select **OK**.
- Add another constant object for the **Low Limit Constant** and use the number ***40*** for the Initialization Value. Label it **Low Set Point Limit**.

♦ Your program should appear like the figure below when you've finished this step.



6. **Add Waveform Generator Object**

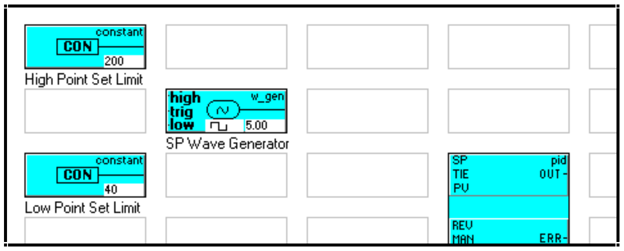
♦ Add a **"w_gen" Waveform Generator** object from the **ANALOG** category to the right of the two constants added in the previous step.

♦ Label it **SP Wave Generator**. (Click just under the center of the object to display the Enter Comment box.)

♦ Click on the bottom **waveform symbol** inside the Waveform Generator object to select the form of the wave to be generated. Choose the **square wave** and select OK.

♦ Now enter the **Period** value. The period is measured in seconds, from a maximum of 327 seconds to a minimum of 0.00 seconds. Enter a value of **5** seconds.

♦ Your program should resemble the figure below when you've finished this step.



♦ Want to know more about the Waveform Generator? Place the cursor on the g in w_gen at the upper right corner of the object and click the left mouse button. A **Help** screen displays details about the selected object. This works similarly for all objects.

7. **Add Variable Objects**

♦ There are five variables to be added to this program. Each one is added the same way as described in the previous steps, but they are set to different values and labeled differently.

♦ Position the cursor in the column to the right of the waveform generator and add **Variable 0**.

♦ In the same column as Variable 0, add **Variable 1** below it, and continue down the column adding **Variables 2, 3, and 4.**

♦ Next at the top of the column with the PID object, add **Variable 5.**

- Set and label the variables as follows:

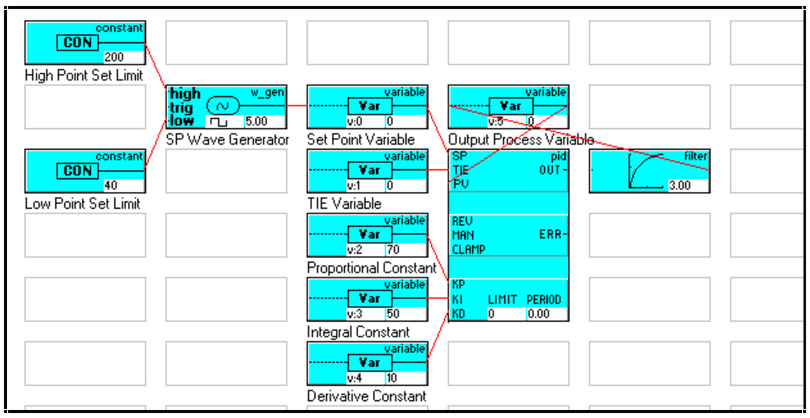
Object	Initial Value	Label
Variable 0	-	Set Point Variable
Variable 1	-	TIE Variable
Variable 2	70	Proportional Constant
Variable 3	50	Integral Constant
Variable 4	10	Derivative Constant
Variable 5	-	Output Process Variable

- Add a **filter** in the column to the right of the PID object. Set the period value of the filter to 3 and then hide the Objects box.

8. Tie Inputs and Outputs Together

- Another key advantage to this point-and-click programming: you can tie the inputs and outputs together quickly and easily. **Input connections** to an object are always on its left side while the **output connections** are on the right. The object indicates an input or output by visible lines in the object icon. Connections between objects are made by drawing a line from an **input** of one object to the **output** of the preceding object.
- To connect the **Waveform Generator** inputs, position the cursor on the **high** input line of the Waveform Generator object.

  NOTE: If the object disappears when you clicked, the cursor wasn't positioned correctly - simply click the left mouse button again to re-apply the object and start this step over.
- Press and hold the left mouse button and drag the cursor to the output line of the **High Limit Constant** object. With the cursor on the output line (make sure the cursor is inside the object), release the mouse button to draw a red (analog) line between the objects.
- Repeat this process for the **Low Limit Constant**.
- Connect the input of the Set Point Variable to the output of the **Waveform Generator**.
- Connect the **PID SP** input to the **Set Point Variable** output.
- Connect the **PID TIE** input to the **TIE Variable** output.
- Connect the **PID KP** input to the **Proportional Constant** output.
- Connect the **PID KI** input to the **Integral Constant** output.
- Connect the **PID KD** input to the **Derivative Constant** output.
- Connect the **PID PV** input to the **Output Process Variable** output.
- Connect the **Output Process Variable** input to the **Filter** output.
- Connect the **Filter** input to the **PID Output**.
- Your program should appear like the figure below when you've finished this step.



9. Add Strip Chart

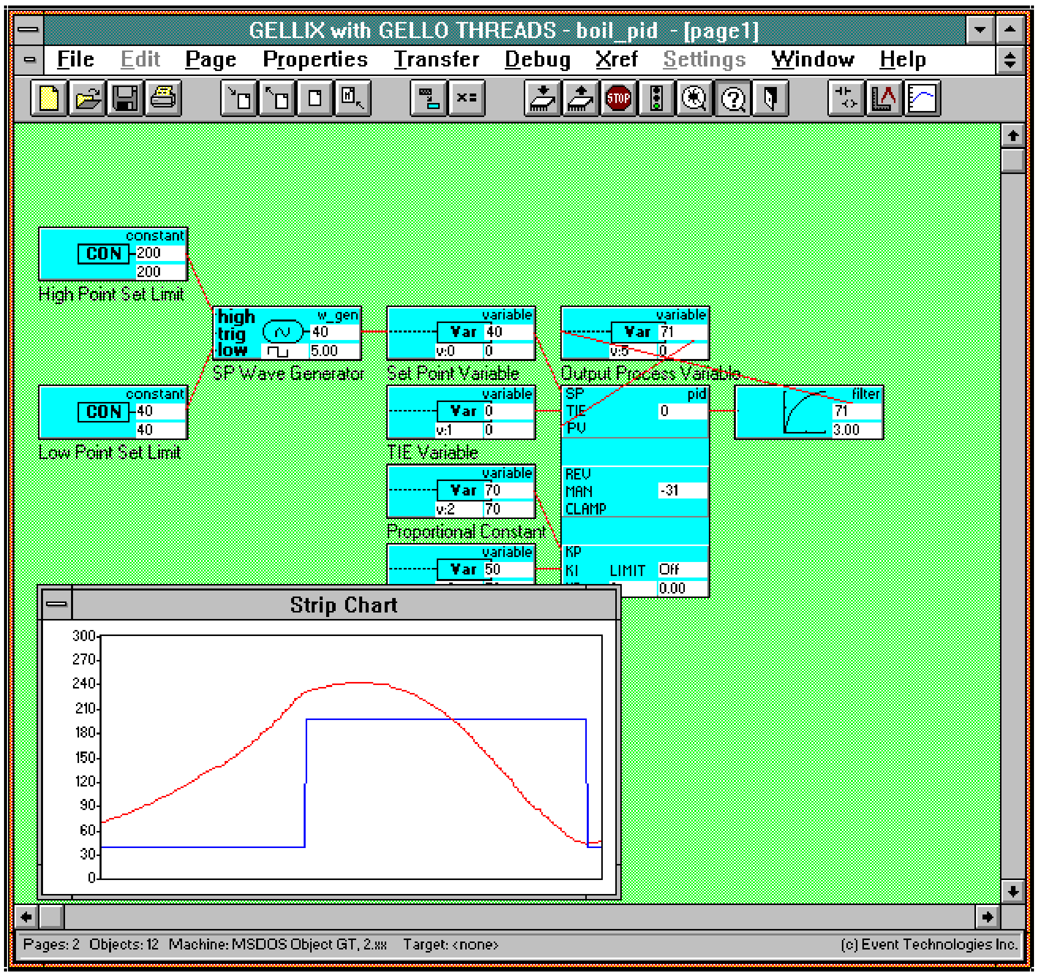
- The program is nearly complete, but we need to add some means of viewing our results. To add a Strip Chart, select **Debug - Strip Chart**. The **Strip Chart Assignments** box is displayed. At the lower portion of this box, set the **Overall Chart** values to the following: **Interval** to **0**, **Low Limit** to **0**, **High Limit** to **300**.
- Set the remaining **Point Definitions** values as follows:

Line Color	**Data Type**	**Point#**	**Interval**
Blue	Variable (Square Wave Generator)	0	0
Red	Variable (PID Output)	5	0

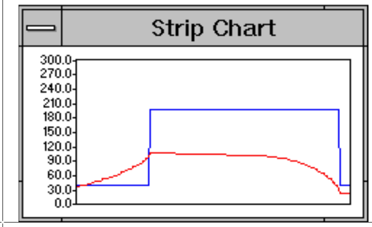
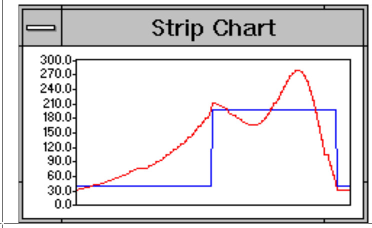
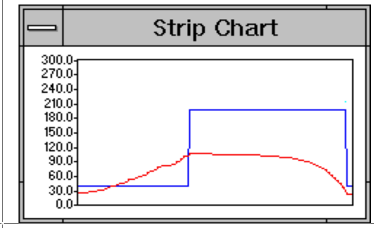
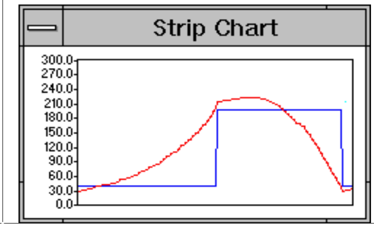
- Use the mouse to position and shape the Strip Chart on the screen.

10. Save Your File & Run a Simulation

- Select **File - Save** (or press **Ctrl + S**) to save your work.
- Another advantage to advanced software of this kind, you can debug the program and run a simulation before you ever connect it to hardware. This saves a lot of time and avoids costly mistakes. It also gives you the opportunity to develop programs before you need to implement them.
- To run a simulation, open the program file, select **Debug - Simulate**. The program will run its course, or, in the case of continuous programs like this one, run until you select **Debug - Stop simulate**.
- The running simulation should resemble the figure below.



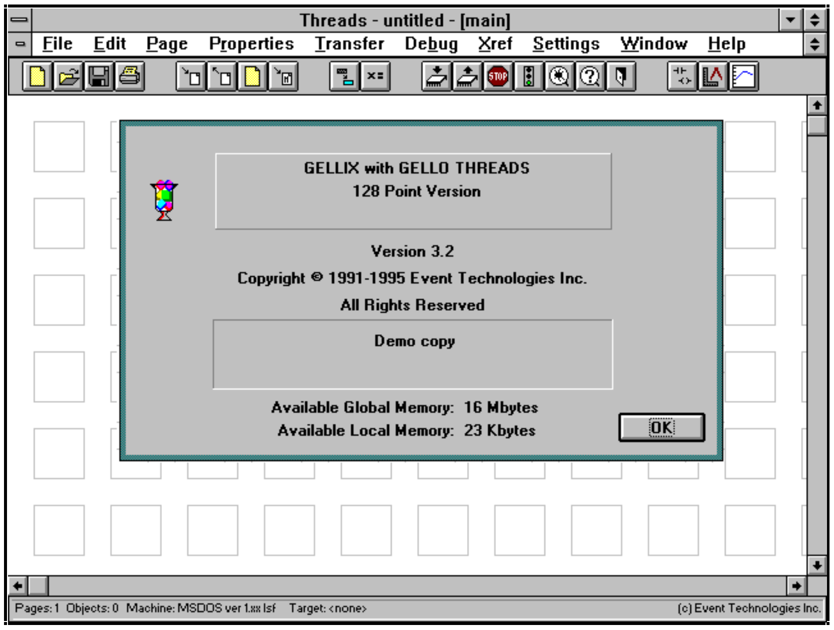
- By adjusting the variables (V2, V3, V4), you can fine tune the controller to compensate for equipment characteristics. The chart below shows different effects on the Strip Chart output as a result of PID settings.
- Click on the value just under the word "variable" while simulating to force the variables to the desired values.

Type	Gains			Wave
	V2	V3	V4	
P	120	0	0	
PI	120	480	0	
PD	120	0	480	
PID	120	100	480	

GELLO Basics

This section contains basic GELLO operation and details for operating the program in general.

Opening Screen


Threads - untitled - [main]
File Edit Page Properties Transfer Debug Xref Settings Window Help
OK

GELLIX with GELLO THREADS
128 Point Version

Version 3.2
Copyright © 1991-1995 Event Technologies Inc.
All Rights Reserved

Demo copy

Available Global Memory: 16 Mbytes
Available Local Memory: 23 Kbytes

Pages: 1 Objects: 0 Machine: MSDOS ver 1.xx lsf Target: <none> (c) Event Technologies Inc.

This is the first screen you'll see after a successful installation of the RTP 2000 Software Demo.

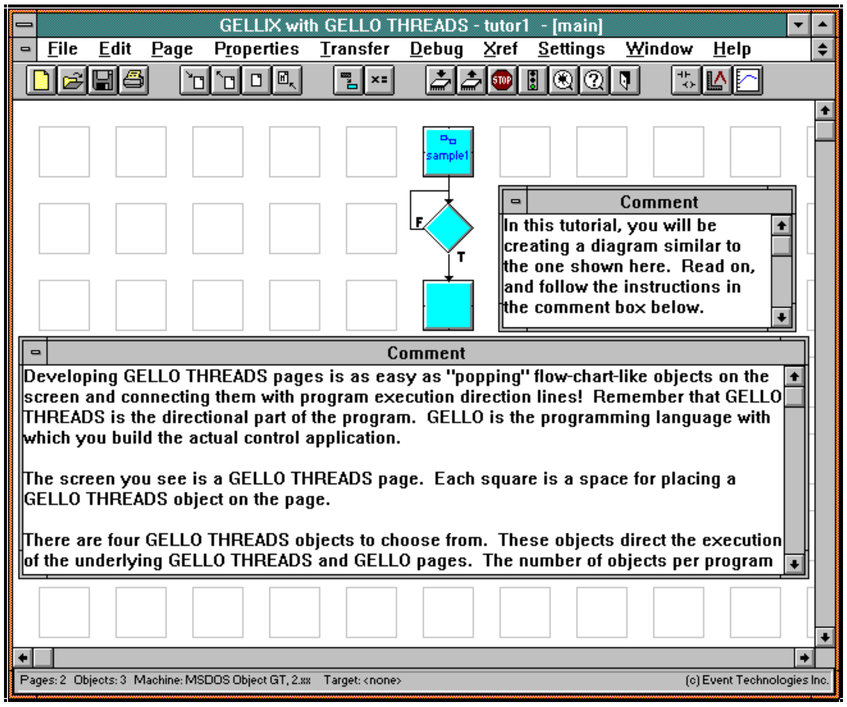
Five sample THREADS files are included to demonstrate the basic GELLO elements:

- SAMPLE1.thr
- SAMPLE2.thr
- SAMPLE3.thr
- SAMPLE4.thr
- SAMPLE5.thr

To learn more about the details of how GELLO is used to design programs, start by opening file SAMPLE1.thr and follow the instructions in the Comments Box. By going through the five sample THREADS files, you can become a productive GELLO designer in less than an hour.

Main Menu

The RTP 2000 Main Menu uses the GELLO THREADS editor running under MS Windows, version 3.1. The basic GELLO THREADS screen has several labeling elements for quick reference.


GELLIX with GELLO THREADS - tutor1 - [main]
File Edit Page Properties Transfer Debug Xref Settings Window Help
sample1
F
T
Comment
Comment

In this tutorial, you will be creating a diagram similar to the one shown here. Read on, and follow the instructions in the comment box below.

Developing GELLO THREADS pages is as easy as "popping" flow-chart-like objects on the screen and connecting them with program execution direction lines! Remember that GELLO THREADS is the directional part of the program. GELLO is the programming language with which you build the actual control application.

The screen you see is a GELLO THREADS page. Each square is a space for placing a GELLO THREADS object on the page.

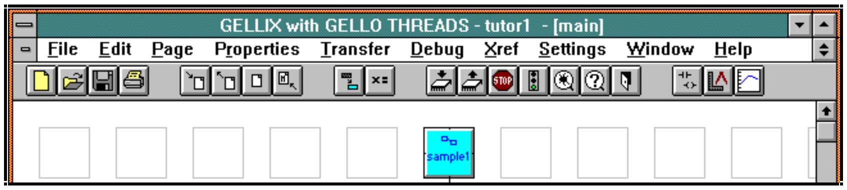
There are four GELLO THREADS objects to choose from. These objects direct the execution of the underlying GELLO THREADS and GELLO pages. The number of objects per program

Pages: 2 Objects: 3 Machine: MSDOS Object GT, 2.xx Target: <none> (c) Event Technologies Inc.

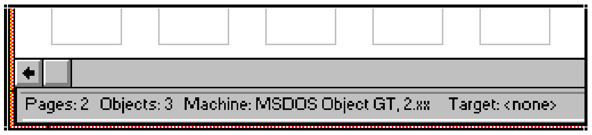
First screen of the SAMPLE1.thr demonstration file.

Menu Tool Bar

The top of the screen lists the different pull-down menus grouped according to tasks. Below the menu titles a row of icons is displayed to provide quick access to program functions. Simply click on an icon to activate a function.


GELLIX with GELLO THREADS - tutor1 - [main]
File Edit Page Properties Transfer Debug Xref Settings Window Help
sample1

The bottom portion of the screen is used to display information specific to the file that is open, such as: number of pages, number of objects, type of engine, and type of target controller. The number of objects per program as well as the number of pages per program is dependent on the available data memory in the target machine. In this demo package, the program is restricted to support only 50 objects per thread file.



Program Functions

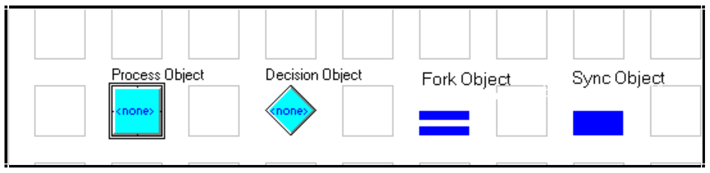
All the program functions operate as they would for any Windows style program. Mouse buttons, scroll bars, function keys, all operate according to standard Windows conventions.

GELLO THREADS Objects

There are four objects in the GELLO THREADS library:

- ♦ Process object
- ♦ Decision object
- ♦ Fork object
- ♦ Sync object

These objects direct the underlying GELLO pages. They possess connection and click zone properties.



Decision Object

The decision object allows If-Then-Else direction. Therefore, you have the standard “input”. But you have two “outputs”: one if the page being called proves true; one if the page being called proves false. The first connection you make will be the true, and the second will be the false, always in that order. A small T or F differentiates the two on connecting lines.

Process Object

The process object requires two connections. The first is the incoming connection from the previously executed page; the second is the outgoing connection to the next page to be executed.

Fork Object

The fork and sync objects function in a different manner. For each of these types, you will have two or more like objects linked together by being next to each other. They act as one object. The fork will only have an input coming in at the farthest left section in the row. However, it will have an output from each fork section.

Sync Object

Similarly, there will be an input connection to each sync object in a row, but only one output connection. If an output from the sync object is made at any other section of the row, an error will occur at link time.

Object Click Zones

The GELLO THREADS objects have click zones in which a click of the mouse will activate different functions. The fork and sync objects have connection point and add/remove zones. The decision and function objects have connection point, add/remove and page assignment zones.

Connections

The decision and process objects have four connection points, each in the center of the sides of the object. The fork and sync objects have connection points on the top and bottom of the object. All of the objects have varying connection requirements.

An exception for connection rules comes in the case of an initialization object (the first object to be executed on a page) or an exit object (the last object to be executed on a page when you do not want a virtual connection), both of which will execute properly with one connection.

GELLO Objects

An object appears in the program as an icon, and performs a particular function during engine execution. Objects can be data types, inputs, connections, reference field inputs, parameter field inputs, or outputs.

Data Types

There are three basic types of data objects:

♦ **Binary Values**	Stored as single bits and manipulated as 0 (off) or 1 (on). Binary values represent discrete (on/off) states typically passed on a conventional ladder logic rung.
♦ **Analog Values**	Stored and manipulated as 16-bit values. All analog math and test objects assume that analog input and output values are linear signed integers. Conversion objects are provided to convert bcd inputs and outputs to a linear 16-bit value.
♦ **Float Values**	Stored and manipulated as IEEE floating-point designation. These objects are not automatically included with all packages.

Inputs

Each object may accept inputs as connections (INx) or field entries (ARGx). Connections are lines joining the output of one object to the input of another. Depending on the object, connections may be required or optional. Field entries indicate you may enter a value directly to the object, and are required when the field is visible on the icon/click zones.

Connections

The click zones illustrates the type of connection allowed or required for a particular input.

Reference Field Input

An object may be associated with a device such as a coil, timer, counter, etc. Typically, argument field 2 (ARG2) is used to indicate which device the object is referencing. If the object references one type of device, this field only requires the reference number of that device. If the object may be associated with multiple devices, then an alpha character prefix is necessary before the device reference number indicating the device type.

Parameter Field Input

An object may require a value for computation or references. This value is usually entered in ARG field 3 and/or 1.

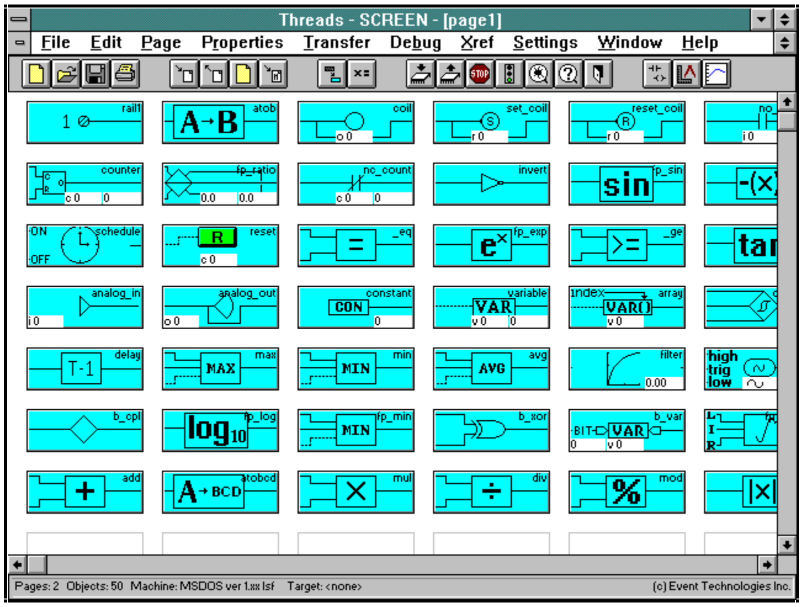
Outputs

Outputs are limited to connections only and are optional. The click zone identifies the output point with the word analog or binary, which identifies its type.

PID Object Features

Features of the extended pid object:

- User choice of ISA dependent or conventional independent gain equations
- Selectable I/O range (for different analog I/O resolution)
- Engineer scaling for PV, SP, Error and Zero crossing (true floating point)
- Zero-crossing dead band
- Selectable derivative term (PV or Error)
- Direct or Reverse error calculation
- Clamp input for integral term
- Output Limits with alarm
- Manual mode with bumpless transfer to automatic
- Integral Feedback mode (Track) for integral tracking of auxiliary signals
- Feed forward (bias) input
- Input parameter out-of-bounds alarm



Some of the GELLO Objects available for control synthesis.

Analog

analog input
analog output
array (fp available)
average (fp available)
compare (fp available)
constant (fp available)
delay (fp available)
filter (fp available)
gate (fp available)
limit (fp available)
maximum (fp available)
minimum (fp available)
pid
ratio (fp available)
time
variable (fp available)
wave generator
* (fp available) indicates that a corresponding object also exists in floating-point format.

Analog Math

absolute value (fp available)
add (fp available)
divide (fp available)
equal (fp available)
greater than or equal (fp available)
greater than (fp available)
less than or equal (fp available)
less than (fp available)
not equal to (fp available)
modulo
multiply (fp available)
negate (fp available)
subtract (fp available)

Binary

and
coil
counter
flip
hold open / hold closed
invert
normally closed contact
normally closed count
normally open contact
normally open count
or
rail1
rail0
reset
reset coil
rst
schedule
set coil
timer
xor

Bit Operation

bitwise analog and operator
binary complement
bitwise analog or operator
bit accessor of a variable
bitwise analog xor operation

Conversion

analog to binary
analog to bcd
bcd to analog
binary to analog
analog to floating point
floating point to analog

Page

escape
page
page_in
page_out

GELLO THREADS Objects

decision object
function object
fork object
sync object

Floating Math

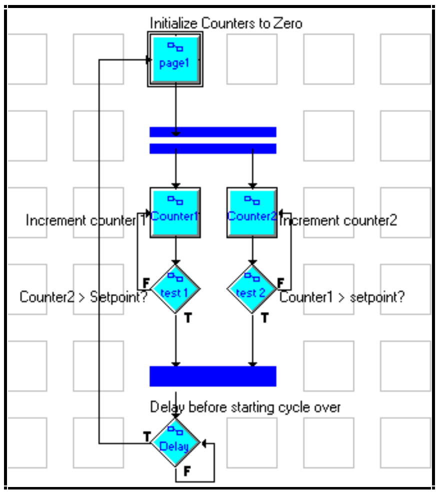
cosine
exponential function
integral
natural log
base 10 log
pid
power
sin
tangent

Symbols

If a page refers absolutely to specific I/O addresses (e.g. ai0) any program using that page must dedicate those points to the same addresses. This restriction is removed in GELLO by adding the feature of symbols. A text string, or symbol, may be placed in the reference field of the I/O objects, instead of an absolute I/O address. When you load the page with symbols to a program, those symbols may be assigned to any I/O reference by using Properties/Symbols.

Pages

At the highest level of GELLO are pages, which are equivalent to subroutines, or functions. Each page is a series of program steps - objects - for one or more items of equipment. You can pass arguments and return values from a page. There are no limitations on the placement or nesting of pages other than memory.



The concept of pages allows proven programming routines for specific machines or process equipment to be incorporated into a common source file library. You can add functions to the library that solve special problems, such as those related to a particular machine, which can be used throughout the plant by passing along the source code.

The Page Object

Functions, or pages, are referenced by page objects in your program. A page object can actually be used like a custom written object because of the function of the page it represents, and it can be used throughout the program.

Generic pages of code that perform a particular function can be created by using symbols. These can be reused in any program by simply using the File/Import Page menu option, and defining the symbols according to that program's requirements.

Importing or Linking Pages

Pages from your source library can be used in other programs in two ways. They can be imported and made a part of the new program, or they can be linked into the new program. When pages are linked, they are actually still part of the parent program. It is not possible to edit a linked page except from its original program. When edits are made to a page that is linked to another program, the changes affect the program when the page is linked, and not until then.

Both of these options allow you to use code that has already been written and proved accurate. They can save significant amounts of development and testing time, allowing the programmer to spend time on new ideas.

Simulations and Debug

Simulation speeds the debugging of your logic without special programming equipment, so you can find programming errors before sending the program to the target controller (or downloading). After a control program is written in GELLIX, it can be run through a built-in real-time simulator.

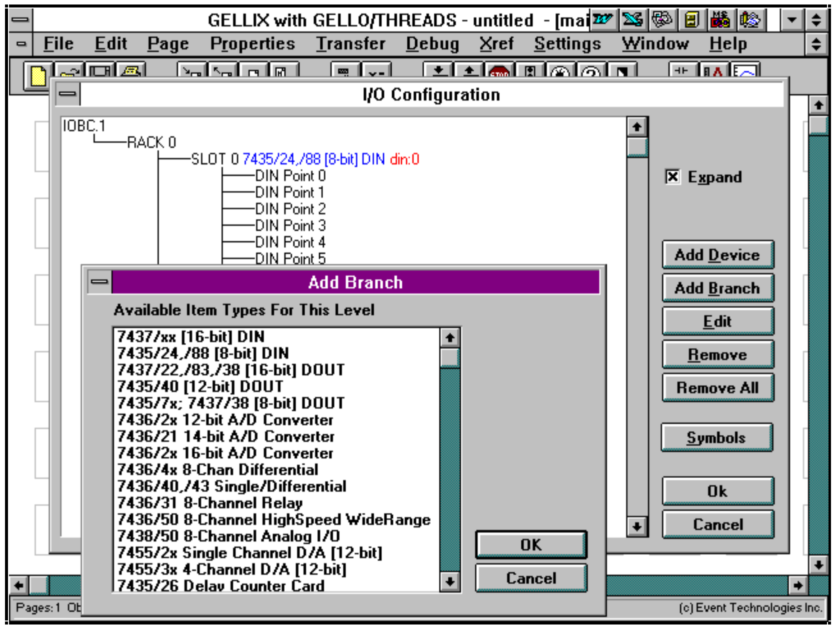
Similarly, you can attach the GELLIX Editor to the real-time processor running GELLO and watch the current state of objects in debug mode. In this state, you can make changes to object variables, or incrementally run from object to object using single-step debug to understand and verify the program in depth. NOTE: The debug feature is not operable in the demo program.

I/O Configuration

The OBJECT GT ENGINE runs in a continuous loop. When the OBJECT GT ENGINE is running a program, the objects in that program are scanned and executed in sequence. Then all of the real-world I/O points are scanned and updated by I/O drivers. These I/O drivers are external software routines which are linked to the OBJECT GT ENGINE and are called by the engine at the beginning of an I/O scan to communicate with the physical I/O hardware. Thereafter, the loop repeats itself, starting again by scanning and executing the objects.

The GELLO objects do not write values directly to those real-world I/O points, however. Instead, they are written to an intermediate image of the real I/O points. When executing input objects (contacts, analog_in), I/O values are read from the input image array into memory. And when executing the output objects (coils, analog_out), I/O values are written to the output image array in memory.

Setting up I/O Configuration in GELLIX is very simple. The screen below is accessed through the menu options. It is as simple as clicking on the appropriate I/O types and drivers and entering the correct points being used.



```
IOBC.1
  └──RACK 0
       └──SLOT 0 7435/24,/88 [8-bit] DIN din:0
            ├──DIN Point 0
            ├──DIN Point 1
            ├──DIN Point 2
            ├──DIN Point 3
            ├──DIN Point 4
            └──DIN Point 5
```

```
7437/xx [16-bit] DIN
7435/24,/88 [8-bit] DIN
7437/22,/83,/38 [16-bit] DOUT
7435/40 [12-bit] DOUT
7435/7x; 7437/38 [8-bit] DOUT
7436/2x 12-bit A/D Converter
7436/21 14-bit A/D Converter
7436/2x 16-bit A/D Converter
7436/4x 8-Chan Differential
7436/40,/43 Single/Differential
7436/31 8-Channel Relay
7436/50 8-Channel HighSpeed WideRange
7438/50 8-Channel Analog I/O
7455/2x Single Channel D/A [12-bit]
7455/3x 4-Channel D/A [12-bit]
7435/26 Delay Counter Card
```

The figure above shows the IOBC devices available for the RTP 2000. The I/O Configure window shows the device / rack / slot / and digital input channels for each input and output. The Add Branch window displays the available I/O cards for a slot. Each one is selectable by a mouse click.

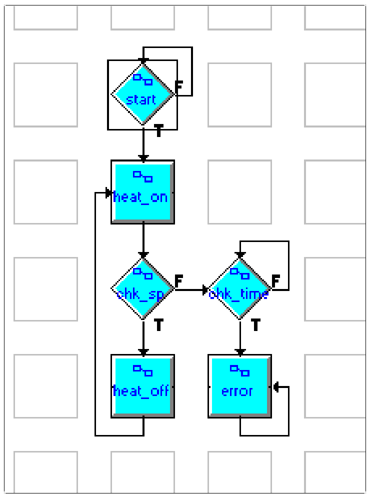
When the I/O update scan begins, outputs are written by the I/O drivers from the corresponding values in the output image array, the inputs are read by the I/O drivers, and the values are placed in the input image array.

Because physical I/O hardware devices differ in their software interface requirements, a unique set of software routines is typically required to control and retrieve values from these devices. In the GELLO concept, these routines are separate from the OBJECT GT ENGINE and reside in external software modules called device drivers. Therefore, for every different "type" of physical device installed on the target machine, there will be an associated device driver module. Since these device drivers are separate from the OBJECT GT ENGINE, they may be changed or added (with an associated physical device change) at any time without disturbing the OBJECT GT ENGINE and other device drivers.

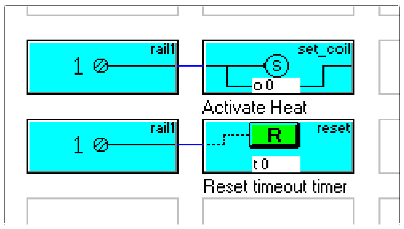
Program Example

The THREADS page shows a typical temperature controller application called RTP_DEMO. The heat_on page starts the heater and resets the timeout timer. The temperature is compared to the setpoint continuously in the check_sp page. If equal, the heater is turned off. If not equal, the timer is checked and rechecked until it reaches the timer count (in the check_time page) and calculates the error (in the error page). The diagrams below show the GELLO THREADS page named process and the GELLO page named heat-on that is nested below the process page.

Program Name: RTP_DEMO Pagename: process (THREADS)



Program Name: RTP_DEMO Pagename: heat-on (GELLO)



Configuration Setup, Cross Reference List, and Symbols

```
Program Name: RTP_DEMO              I/O CONFIGURATION
AI:000-007 IOBC.1          7436/50  8-CHAN HIGH SPEED        0    7    3    8
AO:000-011 IOBC.1          7455/30  [12-BIT]                 0    0    3    12
AO:018-018 IOBC.1          7435/26  DELAY TIME               0    7    0    1
DI:000-007 IOBC.1          7437/20  [16-BIT]                 0    5    0    8
DO:000-007 IOBC.1          7437/22  [16-BIT]                 0    6    0    8
DO:008-015 IOBC.1          7437/22  [16-BIT]                 0    6    0    8
```

```
Program Name: RTP_DEMO            CROSS REFERENCE
Ains
          IO                      Page: chk_sp             Refs: 1
Aouts
          IO                      Page: simulate           Refs: 1
Counters
          <none>
Contacts
          i0                      Page: start              Refs: 1
          o0                      Page: simulate           Refs: 1
Coils
          o0                      Page: heat_on            Refs: 1
          o0                      Page: heat_off           Refs: 1
          o0                      Page: error              Refs: 1
          o1                      Page: error              Refs: 1
Timers
          t0                      Page: chk_time           Refs: 1
Integer Variables
          <none>
Pages
          start                   Page: process            Refs: 1
          heat_on                 Page: process            Refs: 1
          chk_sp                  Page: process            Refs: 1
          chk_time                Page: process            Refs: 1
          heat_off                Page: process            Refs: 1
          error                   Page: process            Refs: 1
          rail                    Page: main               Refs: 1
          process                 Page: main               Refs: 1
          simulate                Page: main               Refs: 1
```

```
Program Name: RTP_DEMO            Symbols
                        chk_set                            = I0
                       chk_time                            = t0
                       heat_tmp                            = o0
                          start                            = i0
                           test                            = O0
```