



GSEOS V5.2 User Manual

Copyright GSE Software, Inc. 1998 - 2005

Gseos

© 1998-2005 GSE Software, Inc.

All rights reserved.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: April 2005 in Marina del Rey, CA.

Table of Contents

Foreword	0
Part I Welcome to GSEOS	7
1 A Quick Tour	7
Add a screen window	8
Histogram Data	11
2 About this document	14
3 Features	14
4 Introduction	15
Part II Architecture Overview	18
1 Data Flow	18
2 Real-time control	19
3 System Structure	20
Part III User Interface	23
1 Desktop Management	24
2 GSEOS Explorer	24
Alarms	25
Blocks	26
Conversions	27
Decoders	27
Expressions	28
Monitors	29
Network	30
Sequencers	31
System	32
Tasks	32
3 Log Windows	34
4 Menus	34
The File Menu	35
The Help Menu	38
The Tools Menu	38
The Data Export Menu	39
The View Menu	39
The Window Menu	40
5 Screen Windows	42
Menus	43
Draw	43
Command Button	44
Data Item	46
Ellipse	48
Expression	48
Image	50

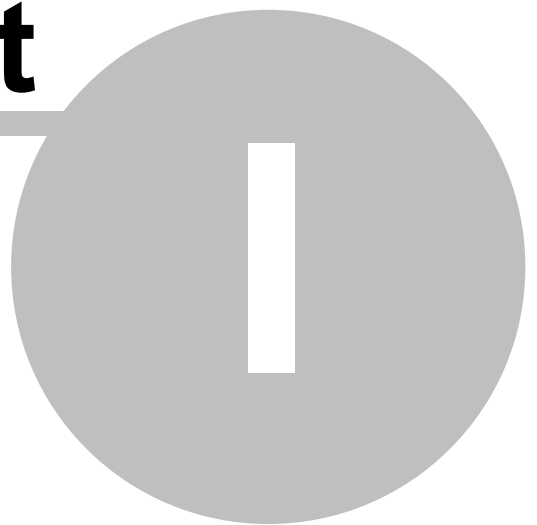
Line	50
Rectangle	51
Rounded Rectangle	51
Scale	51
Text	52
Edit	53
Options	54
Align	54
Grid	54
Zoom	55
Style	55
Color	56
Data Item	57
Alarm Properties	58
Bargraph	59
Binary	60
Bitmap	61
Casting	64
Decimal	65
Float	66
Hexadecimal	67
Octal	68
Signed Decimal	69
Text	70
Y(t)	71
Y(x)	73
Line	74
Orientation	74
Range	76
Text	77
Window	78
Snapshot	79
Placing objects	79
Adjust Display Style	80
Selecting a Drawing Region	81
Selecting a Drawing Tool	81
Selecting Object Properties	82
6 The Command Dialog	84
7 The Console Window	85
8 The Data Export Dialog	87
9 The GSEOS Main Window	90
10 The Message Window	91
11 The Recorder Dialog	92
Part IV GSEOS Reference	96
1 Configuration Files	97
Alarm Limit Files (*.alarm)	98
Block Definition Files (*.blk)	99
Command Batch Files (*.cpd)	103
Command Menu Files (*.cm)	103
Configuration Files (.cfg)	105

Alarm Monitor Configuration.....	105
Alarm Monitor Sample.....	109
Formula Definition Files (*.qlf)	110
gseos.ini	113
Buffer	116
Command	117
Config	118
Console	118
Instance	119
Net	120
Printer	122
Project	124
PyStartup	124
Recorder	125
Timebase	126
System	127
Text Reference Files (*.tr)	128
2 Directory Structure	130
3 Gseos Python Interface	130
Modules	131
__main__	132
GseosBDM.EnableDataSource.....	132
send	133
Decoder	133
bEnable	134
constructor	134
Delete	135
dwCnt	135
Examples	135
Simple Decoder	135
Variable Length Decoder.....	138
strName	140
Gseos	140
Gseos.FileMenu	140
Gseos.FileOpenDialog.....	141
Gseos.GetInstance.....	142
Gseos.GetProjectPath.....	142
Gseos.help	142
Gseos.InputDialog.....	143
Gseos.InputDialogModeless.....	143
Gseos.Log	143
Gseos.LogReload	144
Gseos.LogSave	144
Gseos.MakePathRelative.....	145
Gseos.MessageBox.....	145
Gseos.MessageBoxModeless.....	146
Gseos.PumpWaitingMessages.....	146
Gseos.SetActiveDesktopPage.....	147
Gseos.SetStatusText.....	147
Gseos.ShellExecute.....	148
Gseos.WaitDialog	148
GseosBlocks.....	148
GseosCmd.....	149

GseosCmd.batchstart.....	150
GseosCmd.batchstop.....	150
GseosCmd.msg	151
GseosCmd.send	151
GseosCmd.sound	153
GseosCmd.winexec	153
GseosConsole.....	154
GseosConsole.off.....	154
GseosConsole.on.....	154
GseosConsole.write	154
GseosConvert	155
GseosConvert.ftol	155
GseosConvert.ltof	156
GseosConvert.signed.....	156
GseosMsgWindow.....	157
BringToTop	157
Clear	157
Close	157
New	157
Print	157
GseosNet	157
GseosNet.ClientConnect.....	157
GseosNet.ClientDisconnect.....	158
GseosNet.ClientStatus.....	158
GseosNet.Disable	158
GseosNet.Enable.....	159
GseosNet.IsEnabled	159
GseosNet.ServerReset.....	160
GseosNet.ServerStatus	160
GseosRecorder	160
GseosRecorder.AddPlaybackBlock.....	161
GseosRecorder.AddRecordBlock.....	161
GseosRecorder.GetDataPath.....	162
GseosRecorder.GetPlaybackBlocks.....	162
GseosRecorder.GetPrefix.....	162
GseosRecorder.GetRecordBlocks.....	163
GseosRecorder.IsPlayingBack.....	163
GseosRecorder.IsRecording.....	163
GseosRecorder.RemovePlaybackBlock	163
GseosRecorder.RemoveRecordBlock	164
GseosRecorder.SetDataPath.....	164
GseosRecorder.SetPrefix.....	164
GseosRecorder.StartRecording.....	165
GseosRecorder.StopRecording.....	165
GseosSys	165
FileAppend	165
FileOpen	165
StartApplication	165
WindowClose	165
WindowMaximize.....	165
WindowMinimize.....	165
WindowPrint	166
WindowRestore	166
Histogram	166

Histogram.bAutoscale	168
Histogram.Clear	168
Histogram.dwRange	168
Histogram.Histogram1D	168
Histogram.Histogram2D	170
Monitor	172
bEnable	173
constructor	173
Delete	174
dwCnt	174
Examples	174
CounterCheck	175
LimitCheck	176
strName	177
Sequencer	178
Sequencer.Delete	182
Sequencer.InputDialog	183
Sequencer.MessageBox	183
Sequencer.Sequencer	184
Sequencer.Sleep	184
Sequencer.Start	185
Sequencer.Stop	185
Sequencer.Wait	186
Sequencer.wStatus	186
4 Recorder File Format	187
Part V How do I...	200
1 How do I display alarm information?	200
2 How do I configure an alarm monitor?	200
3 How do I configure the networking module?	200
4 How do I configure startup settings?	203
5 How do I open a screen file programmatically?	205
6 How do I use Expressions and Conversion Functions?	206
7 How do I write a GSEOS extension DLL?	207
Index	210

Part



1 Welcome to GSEOS

Ground Support Equipment Operating System



1.1 A Quick Tour

This section will take you on a quick tour through GSEOS and explain the most important concepts in GSEOS. It will outline the general structure and configuration of the system and give an overview of the various capabilities of GSEOS. For a detailed description please refer to the GSEOS Reference chapter.

GSEOS is a 'shrink-wrap' application that you can run stand-alone. However, you have to configure the system to your needs. This paragraph will outline the various files you will have to create/modify in order to set up a working system.

gseos.ini

This file configures the system parameters like network settings, printer settings, recorder path, project name, etc.

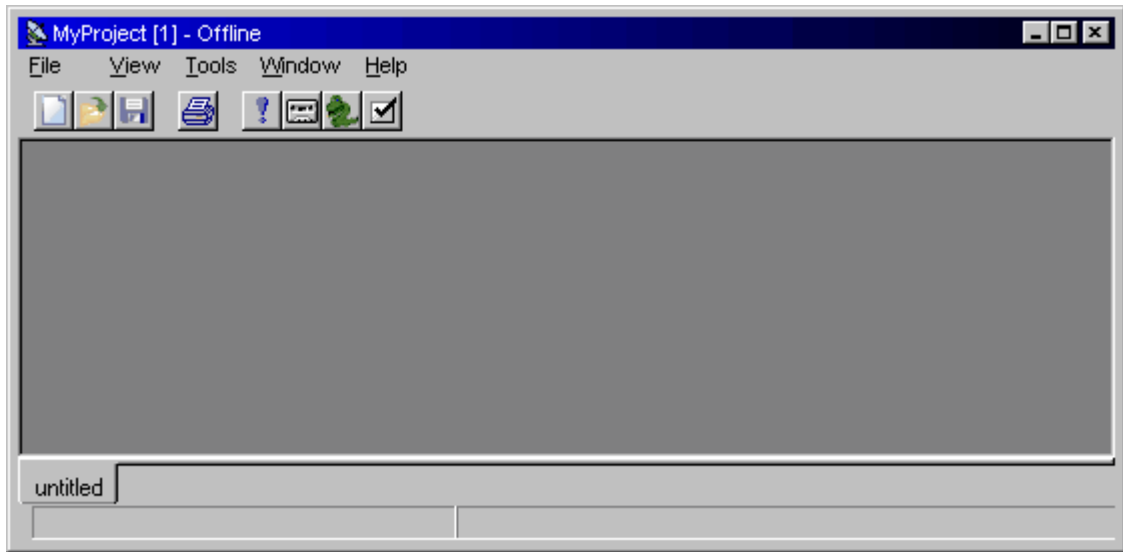
MyProject.blk

The block definition file(s) will hold your telemetry and other block definitions. You can split your block definitions over multiple files if you like. The block definition files to be loaded are specified in the gseos.ini file. These settings have to be initialized when the system starts up. If you have to change your block definitions you will need to shut down and restart GSEOS for these settings to take effect. The following paragraph shows a sample block definition. For a detailed description of the block definition format refer to the section: Block Definition Files.

```
TLM      {
          (ApId          ,,,,16;)
          ,,,, 5;
          InstId         ,,,, 4;
          PacketId       ,,,, 7;
          Seq            ,,,,32;
          Len            ,,,,32;
          Data [40000]   0 ,,,, 8;
        }
```

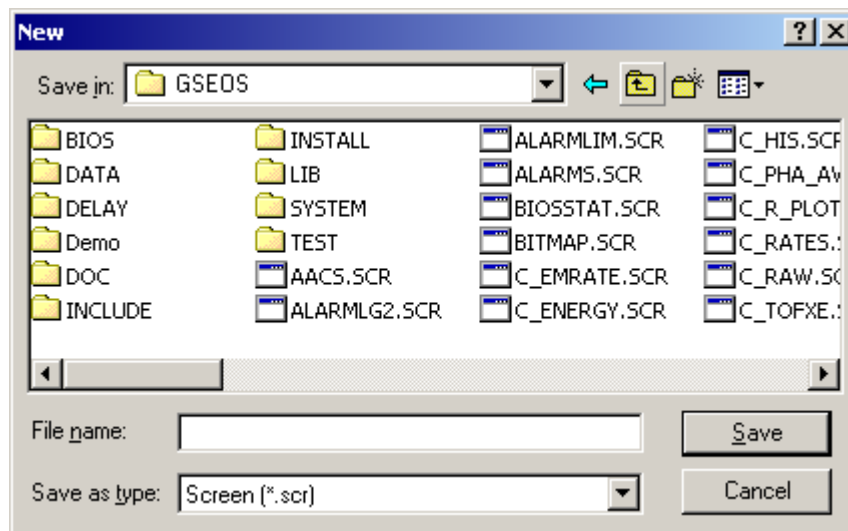
Once you configured your gseos.ini file and set up your block definition file you are ready

start GSEOS. You will see the application like shown below.

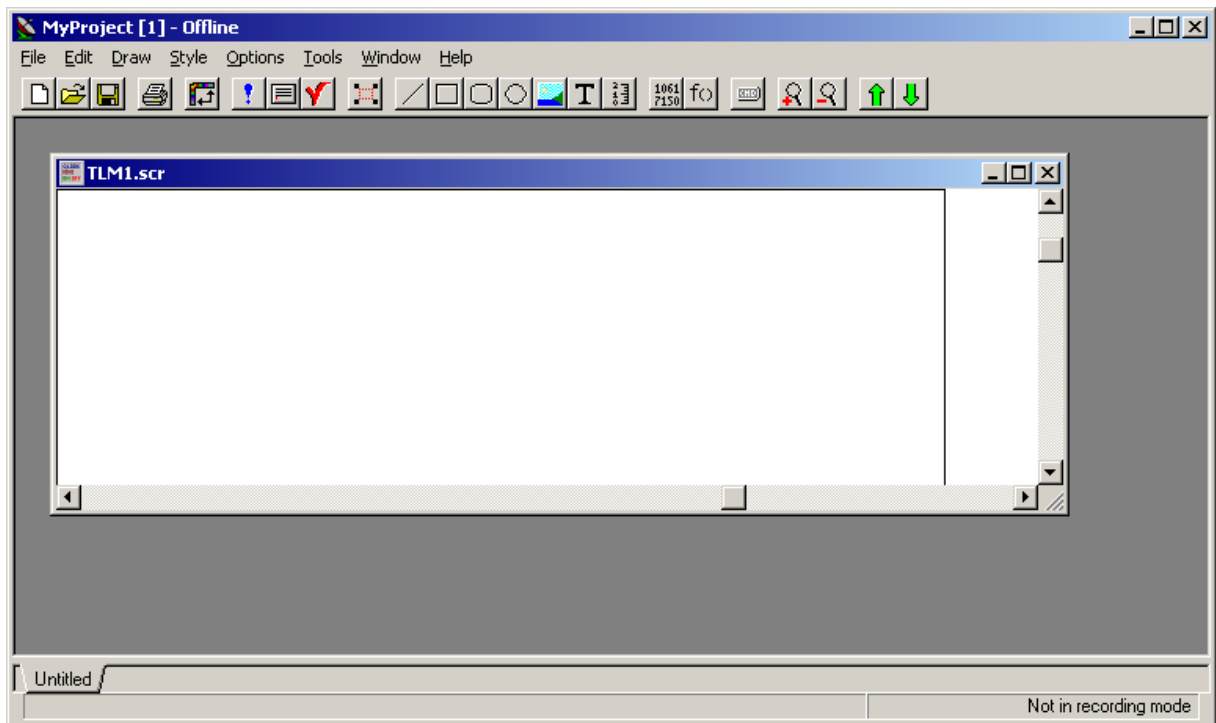


1.1.1 Add a screen window

Now that the system is up and running lets display the TLM block that is defined in our block definition. To create a new screen you select File|New from the main menu or click the File|New toolbar button.



The above dialog opens and prompts you to enter a new file name. Note that in the combo box 'Save as type' you can select various different file types. In order to create a new screen file you have to select 'Screen (*.scr)'. Once we specify the file name e.g.: TLM1.scr an empty screen opens up in GSEOS and you can place objects on the screen. You will notice that the menu and toolbar expand and contain screen editing specific commands. The application now looks like this:

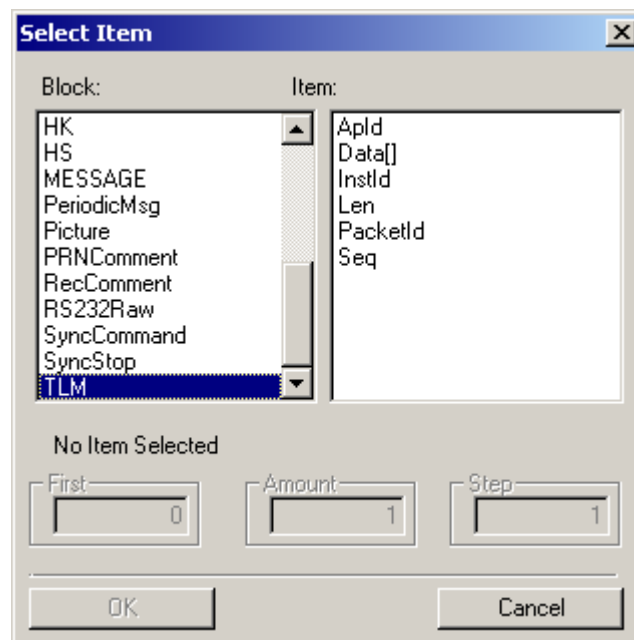


The thin line that shows in the TLM1.scr window is the border of the printing area. Each screen represents a printable sheet. The dimensions of the limiting rectangle are controlled by the printer settings.

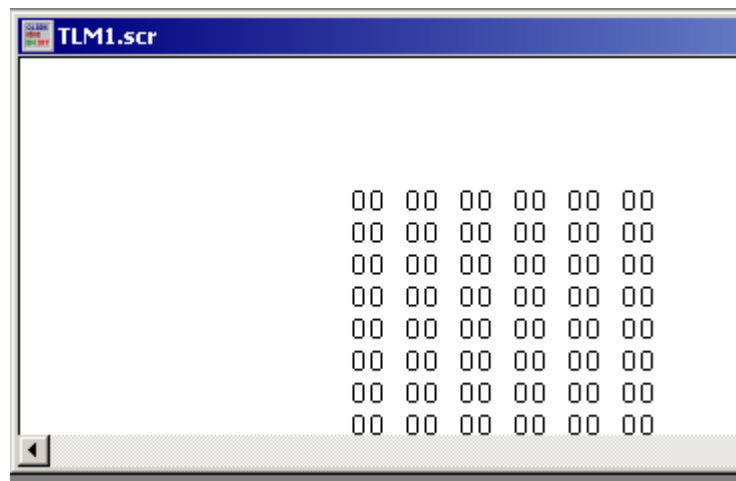
To place a data item on the screen we select the 'Data Item' tool. Hint: If you move the mouse cursor over the toolbar buttons their function is indicated in the status bar. The button shown below invokes the 'Data Item' tool (alternatively you can choose Draw|Data Item from the main menu).



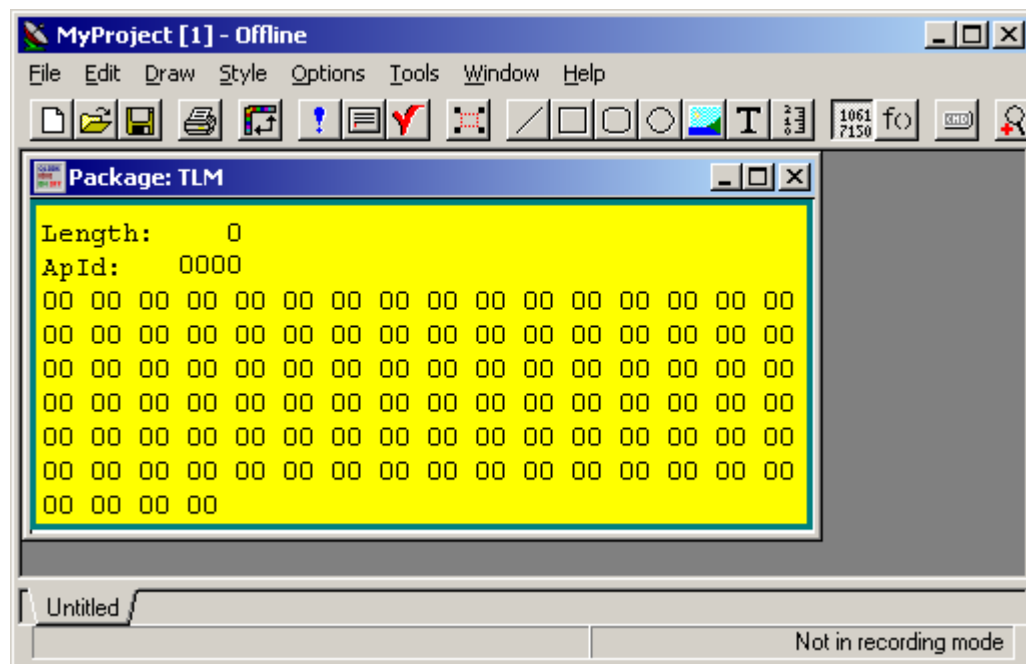
Once you select a tool the according button appears depressed to indicate the tool selected in addition the mouse cursor takes a tool specific shape. Now drag a rectangle on the screen by pressing the left mouse button and moving the mouse while holding the mouse button down. Once you release the mouse button the select data item dialog appears:



Select the TLM block and then the Data item. Select 100 for the amount to restrict the number of elements to display. The screen should now look similar to the following picture and display your TLM.Data item.



The default display type is hexadecimal. You can change the properties of the display object once you placed it on the screen. After placing multiple other items and some static elements like rectangles the TLM.scr file finally looks like this:



1.1.2 Histogram Data

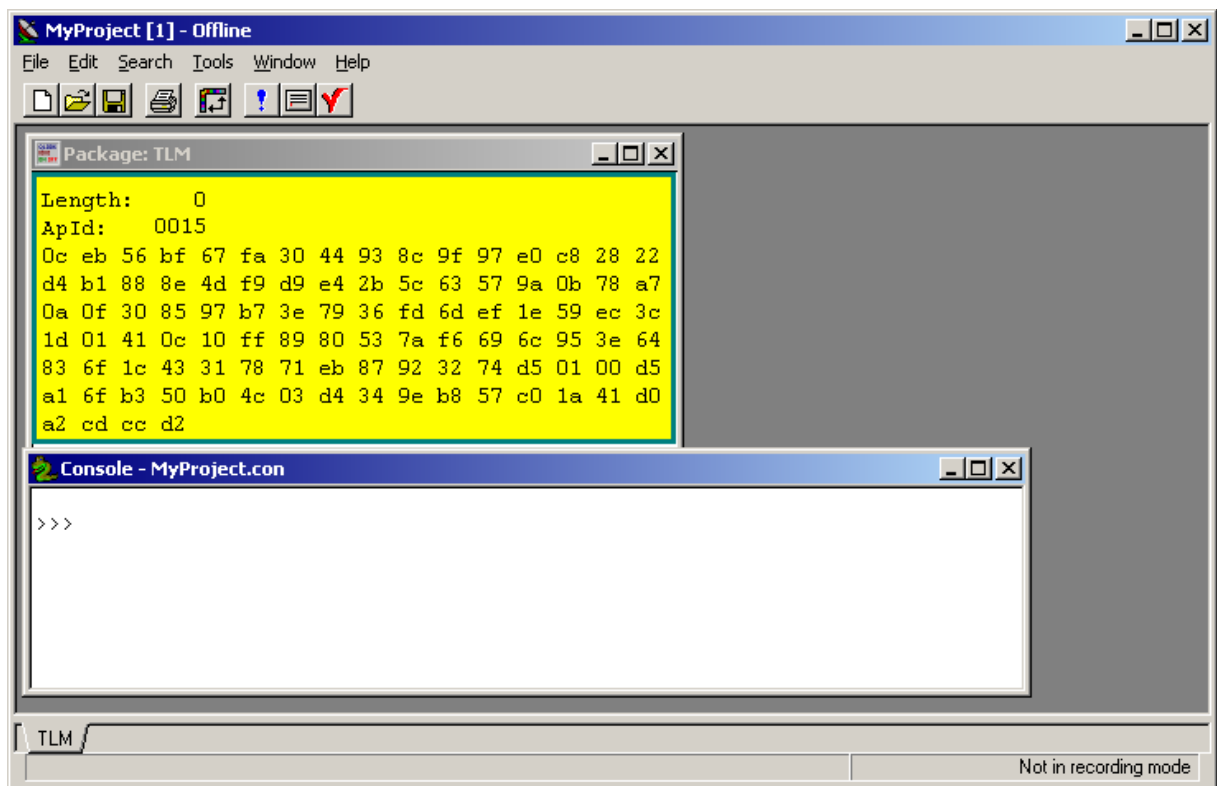
This chapter demonstrates the use of the histogram decoder and displays the TLM data as a histogram. A histogram decoder classifies the input data and maps it into an output block. Let's assume the first 100 items of the TLM block contain event data that you want to histogram and display the distribution as a two dimensional spectrum.

This task involves various steps:

- Create the histogram block definition
- Setup the histogram decoder
- Setup the histogram display
- At first we have to define the destination block. The event values of the TLM block can be in the range [0..255]. We create a TLMHisto block that will take the histogram data and consists of one array item with dimension 256. This way each source value gets mapped onto a destination slot.

```
TLMHisto
{
  Histo[128] 0 , , , 16;
}
```

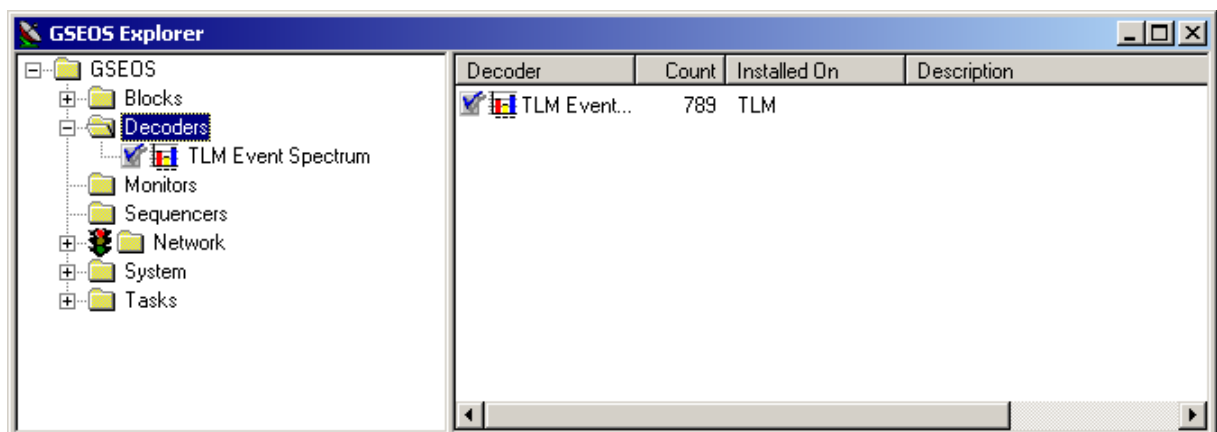
The next step is to create the histogram decoder. We can either write a python script TLMHisto.py which we then load or we can directly type it in the console window. If you don't have a console window open select File|New from the main menu and select file type Console (*.con). This will open a new console window. The application will look like this:



To create the histogram decoder we type into the console:

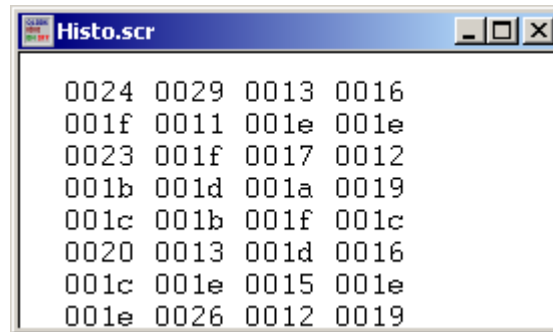
```
>>> import Histogram
>>> histo=Histogram.Histogram1D('TLM Event Spectrum', 'TLM.Data', 'TLMHisto.Histo',
0, 256, 100)
```

This creates the histogram decoder and starts it. We hold on to a reference in the histo variable. This allows us to call various functions on the histogram decoder later. Now lets check out the decoder in the GSEOS Explorer:

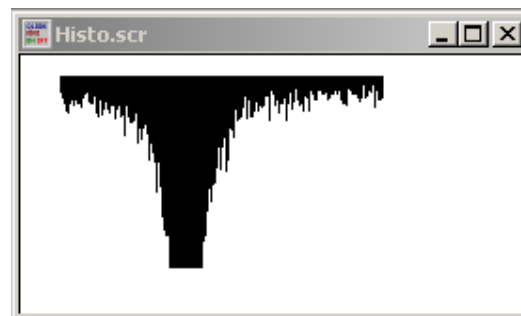


You can see the 'TLM Event Spectrum' histogram in the decoders node. The checkmark indicates that the decoder is running (as you can also tell from the count). The GSEOS Explorer is a useful tool to get a quick system overview and to manage your block definitions, decoders, monitors, histograms, network connections, etc.

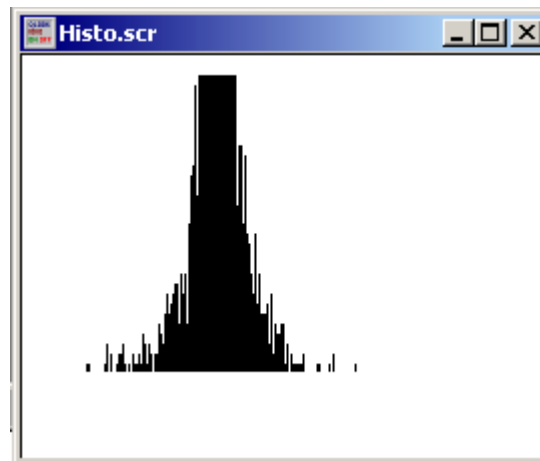
As the last step we have to create the spectrum display. We create a new screen that displays the TLMHisto.Histo item.



Lets change the display style to 'Bargraph' to display the histogram as a spectrum and not just as plain hex data. Right click on the data item and select 'Data Item'. This pops up the Style dialog, choose 'Bargraph' and set the width to 100%. We also have to adjust the range property of the Data Item. By default the range is [0..1]. We want to display a range from [0..1000]. Right click on the data item and select Range. This brings up the Range dialog. Select 1000 for the Y max value. Now a bargraph maps all values from 0 to 1000 into a graphical representation. If the value exceeds 1000 it is simply clipped. By now we have the following display:



One obvious flaw is that the histogram is upside down. All displays have a default orientation of right-handed 4th quadrant (which is what you usually use for a textual representation, that is left to right, top to bottom). For graphical representations we usually want to have the origin in the lower left corner which is the 1st quadrant. To change the orientation right click on the data item and select Orientation. This results in our final display:

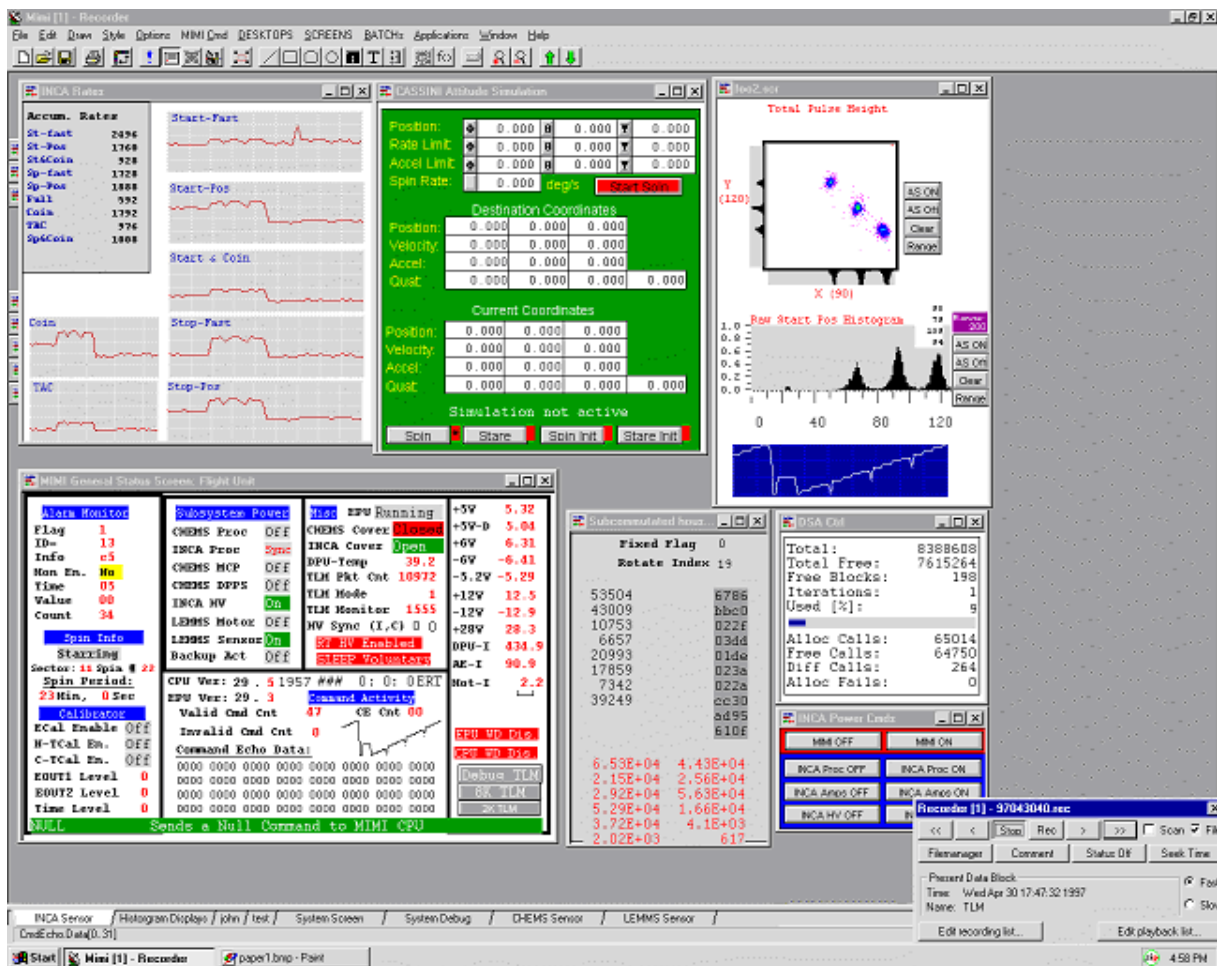


1.2 About this document

Release	Date	Changes
5.0.176	Jan-02-2003	Update Gseos5.0 documentation
5.1.217	Nov-30-2003	Add documentation on Expressions and Conversion Functions.

1.3 Features

- Simple bit-level telemetry format definitions
- Rapid GUI driven display development
- Data display in various numeric and graphical formats
- Easy command interface
- Monitoring of data
- Powerful decoding of telemetry data
- Recording and playback of data and commands
- Built-in networking
- Distributed operations (Remote commanding)
- Easy to script for regression testing
- Batch file capabilities
- Easy to learn generic scripting language
- Open system can easily be extended and customized
- Simple integration of instrument hardware



1.4 Introduction

What is GSEOS?

GSEOS (Ground Support Equipment Operating System) was designed to support the testing and integration of instruments and small spacecraft.

GSEOS meets the need for a low cost, flexible, and maintainable spacecraft ground system which is a common denominator across most space programs.

Low Budget

One of the most common requirements of any spacecraft program is the need to test and operate the hardware prior to launch, and monitor its operation after launch. In recent years, spacecraft and missions have grown more complex while budgets and schedules have grown ever tighter. While such an environment puts pressure on all aspects of a mission, it is particularly difficult on the elements that support ground test and operations, since these elements must be in place to support early hardware development, and yet must last throughout the mission.

Bench Checkout Equipment

Typically, this capability has often been met by developing and supporting several independent systems. The first, often called bench checkout equipment, is designed primarily to help engineers test the flight hardware in a stand-alone configuration. Due to the need to support early testing, the BCE's capability is often limited. Despite these

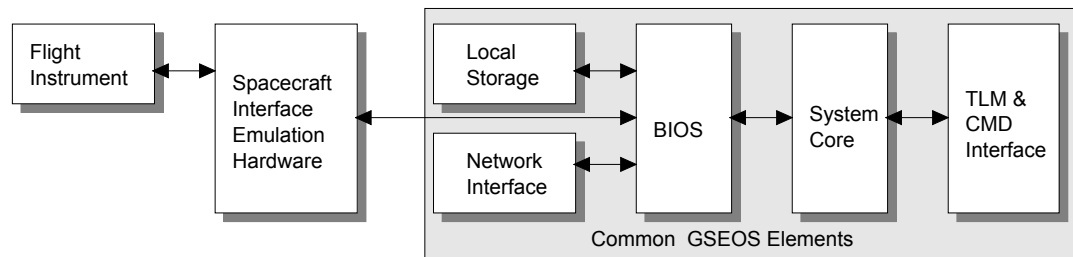
limitations, and the fact that it is not intended for long-term use, the BCE development often requires significant resources to insure that the hardware delivery is not imperiled.

Spacecraft Integration

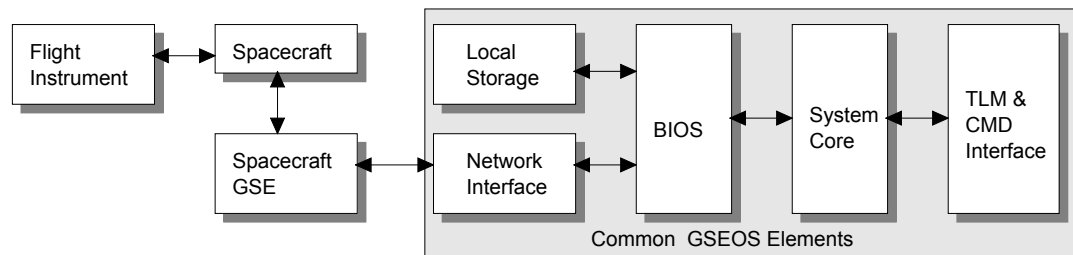
A second system, developed in parallel with the BCE, is intended primarily to support mission operations. Because such systems can not be fully checked until late in the hardware development cycle, significant resources are expended to run simulated hardware interface tests. Such tests are not a complete substitute for testing with real hardware, however, and invariably, problems are discovered in the hardware and/or software. A third system may also be used solely to support system-level testing during spacecraft integration. It shares problems with both of the systems described above; it must be available in time for system integration, but the opportunity for testing may be limited.

GSEOS addresses these problems by providing a common platform that allows to perform bench checkout as well as spacecraft integration and flight operations. The advantages of recycling a system in such a way are manifold. The picture below depicts three possible system configurations:

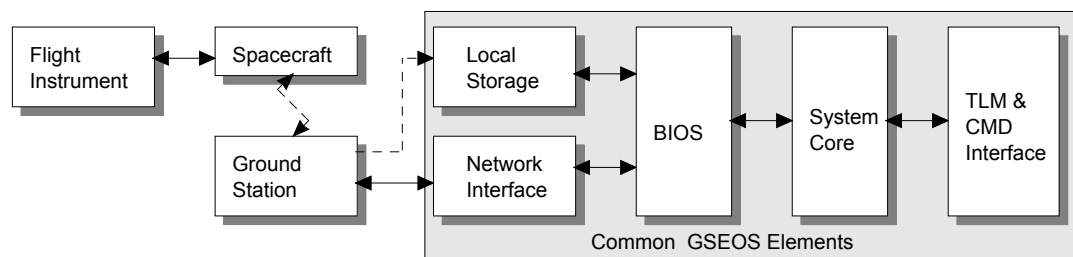
Assembly-Level Test Configuration



System-Level Test Configuration



Mission Operations Configuration



Part



2 Architecture Overview

This chapter gives a brief description of the GSEOS internal structure and of its main features.

During the development of data processing units (DPU) for scientific space experiments, sophisticated test equipment is needed. In particular a spacecraft simulator (S/C Sim) is needed to simulate the electrical interface between the experiment data processing unit (DPU) and the spacecraft data and power system. A computer which is connected to the spacecraft simulator emits control commands to the spacecraft simulator and receives DPU data from the simulator. The whole arrangement (computer & S/C Sim) is referred to as the Experiment Ground Support Equipment (EGSE).

Primary tasks of the EGSE are to send commands and to receive data from the connected hardware. The following tools are supported to achieve the mentioned purpose:

- **Spacecraft simulator and instrument control.** Instrument and spacecraft commands can be send through the BIOS.
- **Data monitoring.** Any data item can be checked automatically.
- **Data display.** Data items can be displayed on the screen in selectable formats (hex, decimal, binary, graphic). The data items can be organized in different windows which in turn can be arranged on various pages. This allows for a quick and easy navigating to access your data.
- **Data logging.** The recorder module allows you to store data and to write command protocols to mass storages in real-time. Conventional hard disks and optical disks can be used as mass storage devices.
- **Automatic test procedures.** End to end testing can be achieved by writing sequencer scripts that issue commands and verify the appropriate behavior of the instrument. This allows to automatic test sequences and regressing testing.

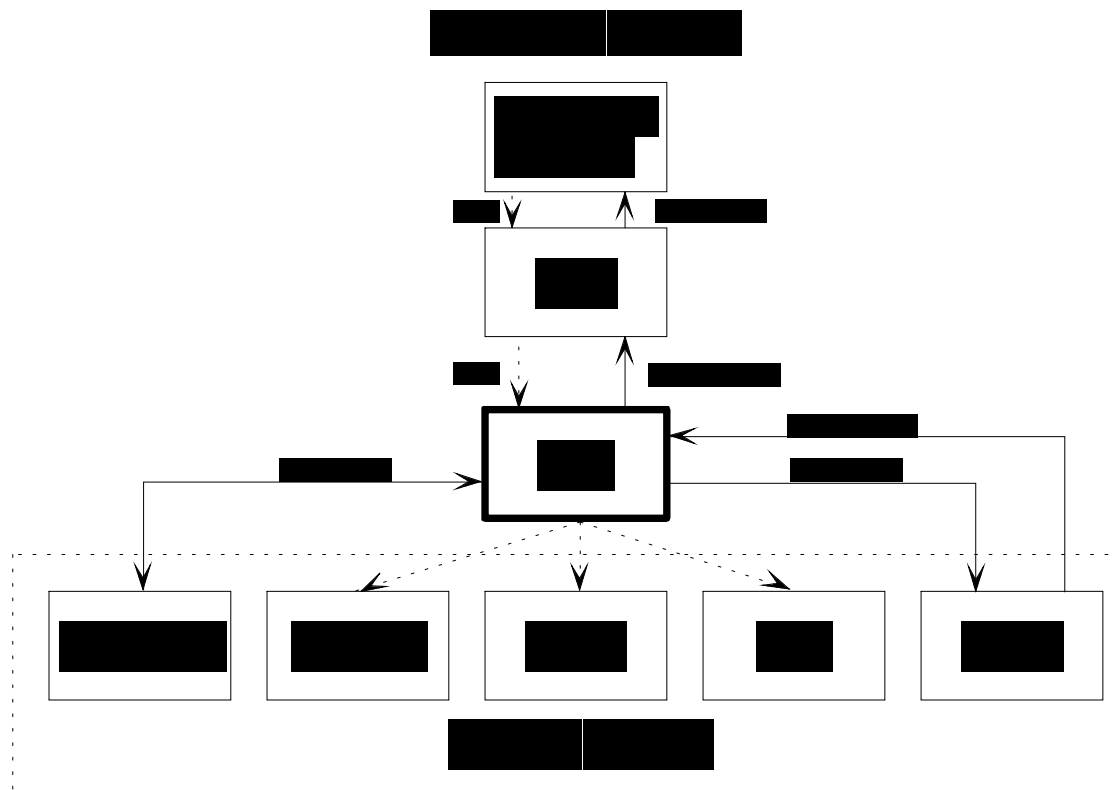
2.1 Data Flow

Data Flow

The data flow in GSEOS is shown below. This figure shows, that the BDM plays a central role for the data flow. When a data block has been received by the active data source, i.e. the S/C simulator BIOS, this module informs the BDM about the "arrival" of new data. The BDM then activates all consumer modules (e.g. Qlook, RECORDER, MONITOR,...) that want to work on the arrived data. For this purpose the BDM keeps a list of consumers for every block definition.

Command Flow

The command flow in GSEOS is outlined in the following picture.



The GSEOS command processor intercepts CMDSTRING blocks that contain command text and generates a binary command representation suitable to your instrument. This translation is defined in a command script. The binary command data is relayed to the BIOS as the BINCOMMAND block. The BIOS then invokes the appropriate hardware to send the command to the spacecraft simulator and/or the instrument.

Each command can be executed by a batch file that contains time information when the command has to be executed. Relative and absolute time tags are possible. "Relative" in this case means for example "100 minutes after the previous command".

With the Recorder tool all commands which arrive at the command processor (in the CMDSTRING block) can be protocolled on a mass storage device. A replay of such a command file can regenerate test scenarios. Together with the time tagging capability a convenient way of implementing time dependent batch jobs is provided.

2.2 Real-time control

After informing a data consumer about the arrival of data, the BDM accesses the whole data block or single data items which are defined in the block definition. The decoder module also allows you to build new data blocks tell the BDM about the "arrival" in the same way the S/C SIM BIOS does. At any given time the system uses one of the mutually exclusive data sources available: BIOS, Recorder, Net, or a custom Python data source (you can set this with the EnableDataSource command). I.e. when data is played back from disk the Recorder is the data source and all data coming in on the other channels is discarded.

Data processing within GSEOS is inherently data driven. Therefore it is easy to provide real time control, i.e. to test whether the whole system keeps up with the incoming data load or is overloaded.

Every data block is assigned to an input buffer memory that is managed with a FIFO

(first in, first out) buffer technique. Several blocks can be read into the FIFO. To detect a real time violation, every consumer has to inform the BDM about finishing its work. This is called the "consumer acknowledge". If the BDM detects, that one of the consumers did not send its acknowledge before a new block arrives, BDM buffers the data in a queue and does not call that consumer again before the acknowledge is received. Generally all consumers of a block "acknowledge" the processing of the data block within the time the next block needs to arrive. However there are situations where GSEOS cannot process its applications, for example when the user interacts in Windows (moving or resizing a window) or when another application has access to the CPU. Then the FIFO queues will fill. You can configure buffer space and threshold values. If FIFO overflow and therefore data loss should occur an error counter is incremented. The system condition can be checked with the GSEOS Explorer.

2.3 System Structure

Within the DPU data items are assembled into Experiment Data Blocks (EDBs) according to experiment specific block definitions. The incoming data blocks will be stored in buffers (exchange buffers, FIFOs) for subsequent decoding in the GSEOS.

The data access is through symbolic lookup as defined in the according the block definitions.

Data items with low update rates are often transferred using a subcommutation scheme. Each time such a block is received, the data items meaning can change and is pointed out by an index. GSEOS has a built-in decoder mechanism that makes it easy to define custom decoders to perform decommutation of arbitrarily subcommutated data. The decoder output will feed into a new block that can then be displayed or used for further decoding. The decoder/block hierarchy can be visualized with the GSEOS Explorer.

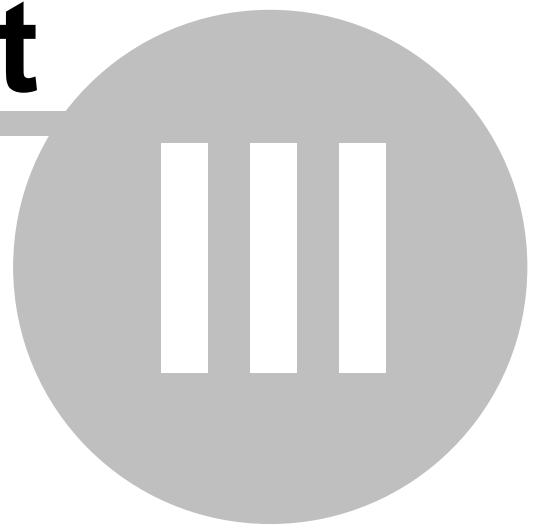
The block disassembly and data processing has to be completed during the average block repetition period, this imposes a rigid real-time condition. GSEOS consists of several kernel modules:

- [BIOS](#), the basic input/output system. This is the experiment specific part of Gseos that needs to be customized to the specific spacecraft/instruments needs. It collects data from the spacecraft simulator box and provides a defined interface to the other GSEOS modules.
- [BDM](#), the block data manager, handles the data blocks and the memory management.
- [GseosCommand](#), the command processor, which interprets and executes commands and runs test procedures without supervision. It interprets all commands for the experiment, the S/C Simulator and commands for GSEOS. The open source programming language [Python](#) is used to control all aspects of the system. It provides a very intuitive syntax, a small learning curve and an efficient implementation to meet the needs of a real-time system. The command processor can execute command macros and time dependent batch jobs. Commands can be assigned to custom menus as well as command buttons.
- [Monitor](#), performs range checks against definable bounds and calculations on the data items.
- [Decoder](#), runs custom decoder scripts to decode dynamic data into derived data products which are GSEOS blocks. These blocks can be accessed like all other blocks in the system, especially they can be used as sources for further decoders. In this manner a decoder hierarchy can be built that allows visibility of decoding layers where

appropriate.

- **Qlook**, an interactive configurable quick-look tool for all data items defined in the block definition file. Data items can be displayed in various textual as well as graphic formats. All the displays are updated in real-time to give a precise view of the system condition.
- **Recorder**, the data logging module enables GSEOS to store data and to write command protocols to mass storages in real-time. Conventional hard disks and optical disks are used as mass storage devices. Since the recorder can store all BDM blocks, an entire test session including commands and system messages can be recorded and replayed at a later time.
- **Message Window**, the message module, which logs all user and system messages with the ability to print out a protocol of a test session.
- **Log**, the log module allows to set up custom logging files that can be accessed through user interface windows.
- **Net**, the network module allows to connect to arbitrary TCP/IP data sources and destinations. Simple configuration of the available client or server connections will set up a data source within GSEOS. This also enables to distribute data to various instances of GSEOS running on the same or separate boxes.

Part



3 User Interface

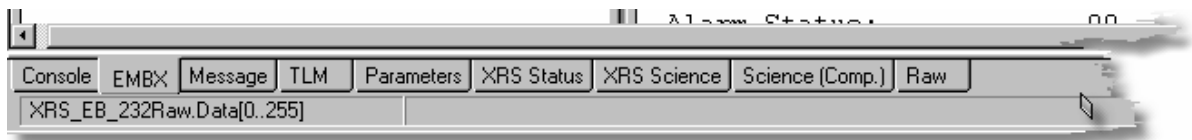
3.1 Desktop Management

GSEOS combines features of a Multiple-Document-Interface (MDI) application with a custom desktop scheme. In order to be able to quickly navigate between various screens you can place the GSEOS screen windows on different desktop pages. Pages can be selected by clicking on the according tab. The entire configuration is saved in a Desktop file. On shutdown of GSEOS the system saves the current desktop into the file AutoDeskxx.dt where xx refers to the instance of GSEOS running. On startup the last desktop file is automatically loaded so you will find your last configuration being restored. You can also save the desktop file at any time with the Save As... menu, or restore a previously saved one.

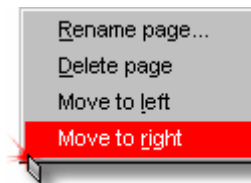
You can override this behavior by loading a desktop file in the gseos.ini Config/Load section. This will cause GSEOS to always appear as determined by the desktop file specified regardless in what state the system was terminated previously.

Appending a desktop file to the currently loaded one will append the desktop pages to the current pages. This allows for easy merging of desktop pages.

The picture below shows a typical desktop layout:



You can create a new desktop page from the File/New Desktop Page... menu. By right-clicking on the **active** desktop tab you can rename, move, or delete the page.



To programmatically set the active page use the following command:

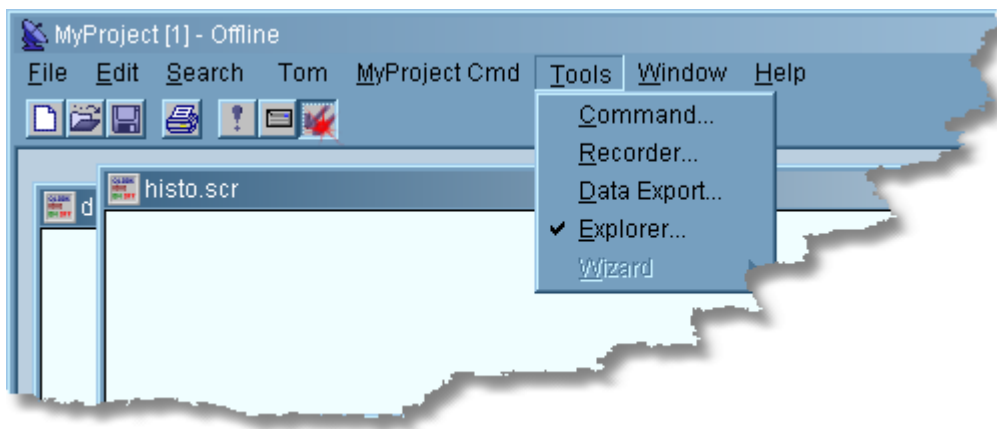
```
>>> Gseos.SetActiveDesktopPage(strPageName)
>>>
```

where strPageName is the name of the page you want to activate.

3.2 GSEOS Explorer

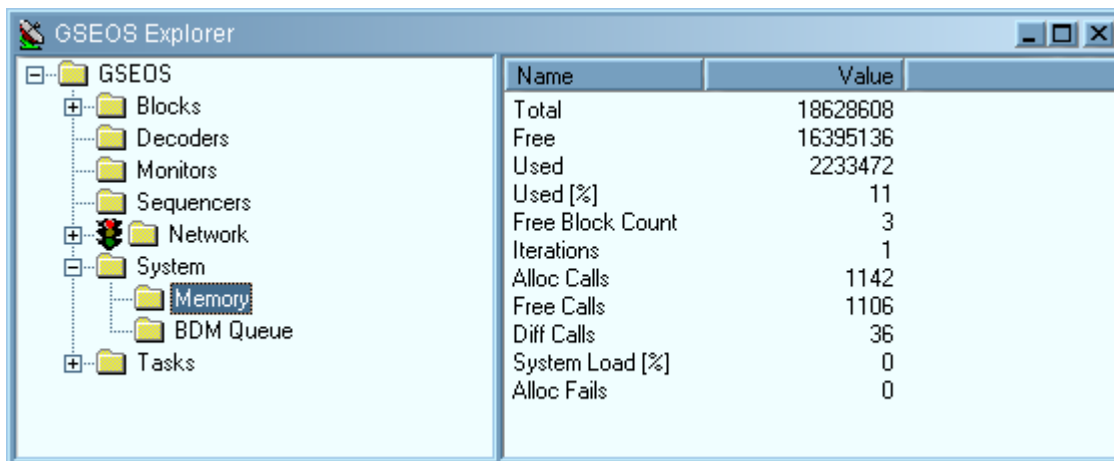
The GSEOS Explorer window allows you to control and monitor the GSEOS system status. You can monitor the currently installed Blocks, Decoders, Monitors, and Sequencers. The System node gives access to system parameters and performance data. In the network section you can monitor and modify your network connections.

You invoke the GSEOS Explorer from the main menu Tools/Explorer, or the Toolbar:



These selections toggle the Explorer window on and off.

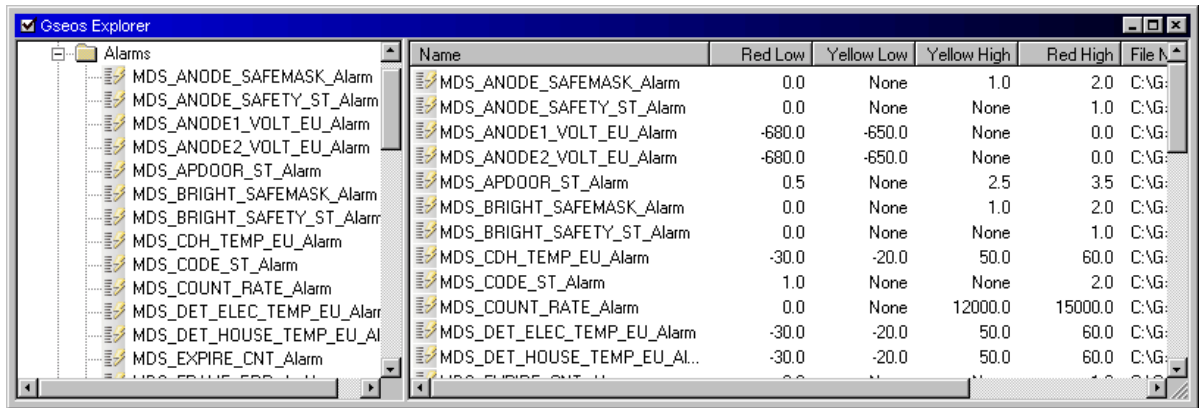
The Explorer window itself consists of two panes, similar to the Windows Explorer interface. On the left hand side you select the specific node you are interested in, on the right hand side you will see the detail information about the selected node. In the figure below the System/Memory node is selected and you can see the memory status on the right hand side.



3.2.1 Alarms

The Alarms node displays all the alarms currently loaded. To change or add new alarms load an alarm file.

The picture below shows the GSEOS Explorer with the Alarms node expanded.

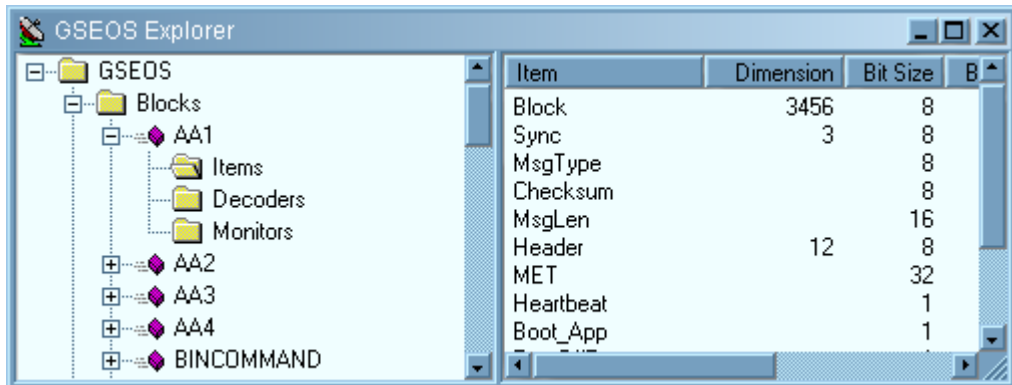


Name	Red Low	Yellow Low	Yellow High	Red High	File Name
MDS_ANODE_SAFEMASK_Alarm	0.0	None	1.0	2.0	C:\G...
MDS_ANODE_SAFETY_ST_Alarm	0.0	None	None	1.0	C:\G...
MDS_ANODE1_VOLT_EU_Alarm	-680.0	-650.0	None	0.0	C:\G...
MDS_ANODE2_VOLT_EU_Alarm	-680.0	-650.0	None	0.0	C:\G...
MDS_APDOOR_ST_Alarm	0.5	None	2.5	3.5	C:\G...
MDS_BRIGHT_SAFEMASK_Alarm	0.0	None	1.0	2.0	C:\G...
MDS_BRIGHT_SAFETY_ST_Alarm	0.0	None	None	1.0	C:\G...
MDS_CDH_TEMP_EU_Alarm	-30.0	-20.0	50.0	60.0	C:\G...
MDS_CODE_ST_Alarm	1.0	None	None	2.0	C:\G...
MDS_COUNT_RATE_Alarm	0.0	None	12000.0	15000.0	C:\G...
MDS_DET_ELEC_TEMP_EU_Alarm	-30.0	-20.0	50.0	60.0	C:\G...
MDS_DET_HOUSE_TEMP_EU_Alarm	-30.0	-20.0	50.0	60.0	C:\G...

3.2.2 Blocks

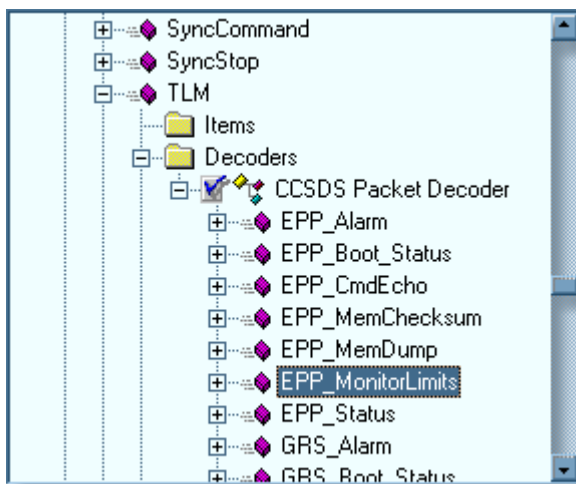
The Blocks node displays all the blocks currently configured in the system. To change or add new blocks modify the block definition files.

When you expand a particular block you can see three folders, Items, Decoders, and Monitors. The Items folder displays all the data items this block is comprised of. The picture below shows the GSEOS Explorer with the Block AA1 expanded and the items folder selected.



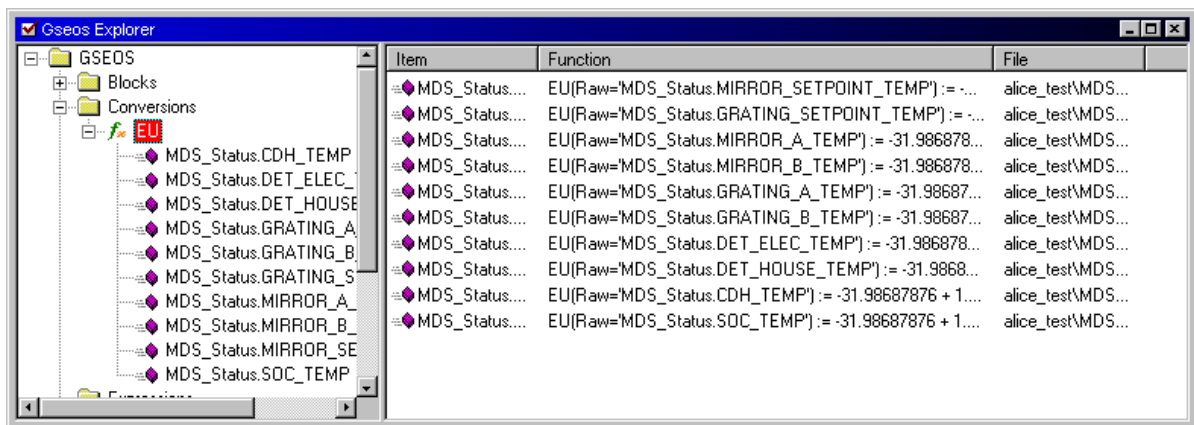
Item	Dimension	Bit Size	Bit
Block	3456	8	
Sync	3	8	
MsgType		8	
Checksum		8	
MsgLen		16	
Header	12	8	
MET		32	
Heartbeat		1	
Boot_App		1	

The Decoders folder lists all decoders that are installed on this block, that is all decoders that fire upon arrival of this block. In the following example you can see that the decoder 'CCSDS Packet Decoder' is installed on the TLM block. The decoder node then lists all the blocks generated by the decoder. The block nodes here have the same properties as the block nodes in the main Blocks folder. That is you can recurse through the decoder hierarchy using the GSEOS Explorer getting a quick overview of the system block and decoder topology. The Decoder subnode of a block node has the same properties as the main Decoder folder, the same is true for the Monitors node. The Monitors subnode of a block node lets you determine what monitors are installed on that particular block. For more details on the Decoder node please refer to GSEOS Explorer/Decoders, for more details on Montiors please refer to GSEOS Explorer/Monitors.



3.2.3 Conversions

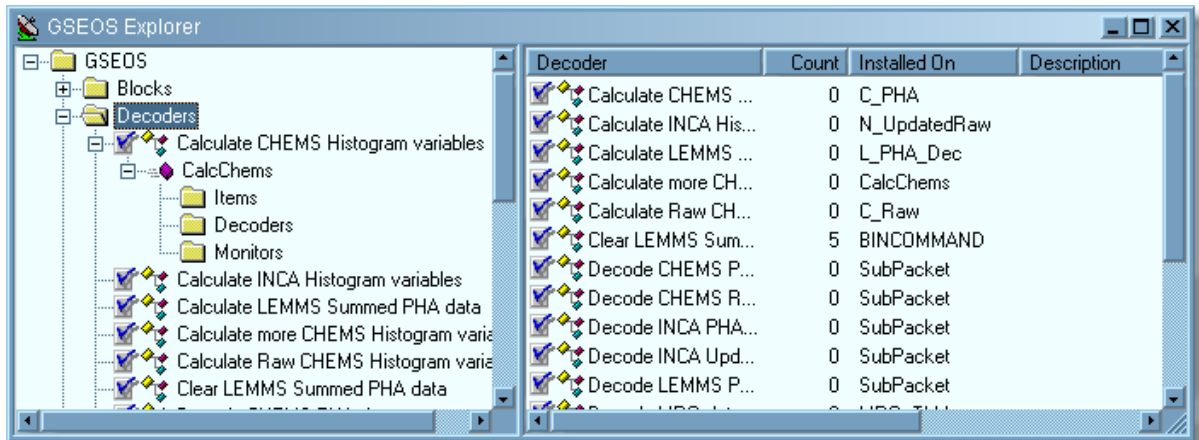
The Conversions node displays all the conversion functions currently loaded. To change or add new conversion functions modify the formula file and load it.



The right hand pane shows the data items the conversion function is defined for. Multiple data items can share the same conversion function name. The function body is listed as well as the formula file the conversion function was loaded from.

3.2.4 Decoders

The Decoders node displays all decoders currently installed in the system. For more information on how to configure a decoder please refer to Modules/Decoder. As you can see in the picture below the Decoders node lists all the decoders currently installed. In the right hand pane you can see the number of invocations, the source block, i.e. the block that triggers the decoder execution. And finally a description which is the doc string that can be specified with the decoder definition.



Upon expanding a particular decoder node in the left pane you will see the output blocks this particular decoder generates. From here on you have the same properties as the Blocks node.

Right clicking on any decoder node will pop up the following menu:

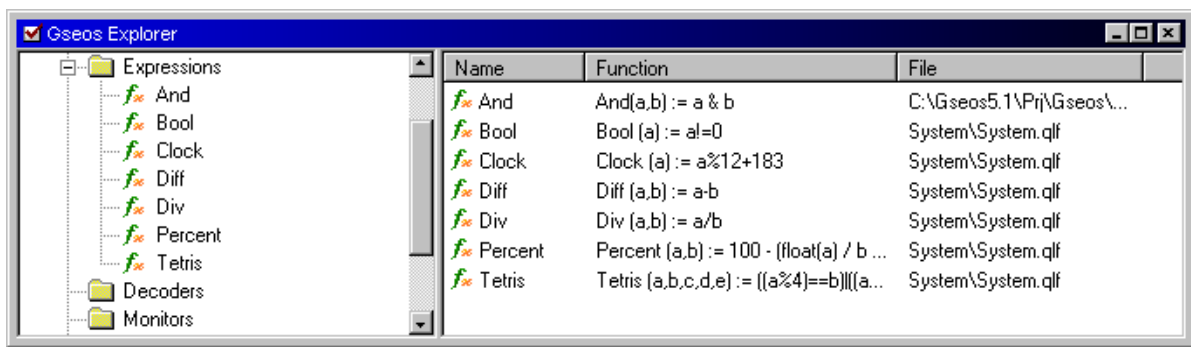


Here you can Enable/Disable the decoder or delete it altogether. Keep in mind, if you assign the decoder to a variable that you maintain, for example in a Python script, the decoder will not be removed until you release this variable. The delete operation only releases the internal reference that is held by the Explorer. If the decoder is a histogram decoder you can additionally set some of the histogram decoder properties here.

In the right hand pane you can select multiple decoders and invoke the operation on all the selected decoders at once. This feature is not possible with the left hand pane since you can only select one node at a time.

3.2.5 Expressions

The Expressions node displays all the expressions currently loaded. To change or add new a new Expression modify the formula file and load it.

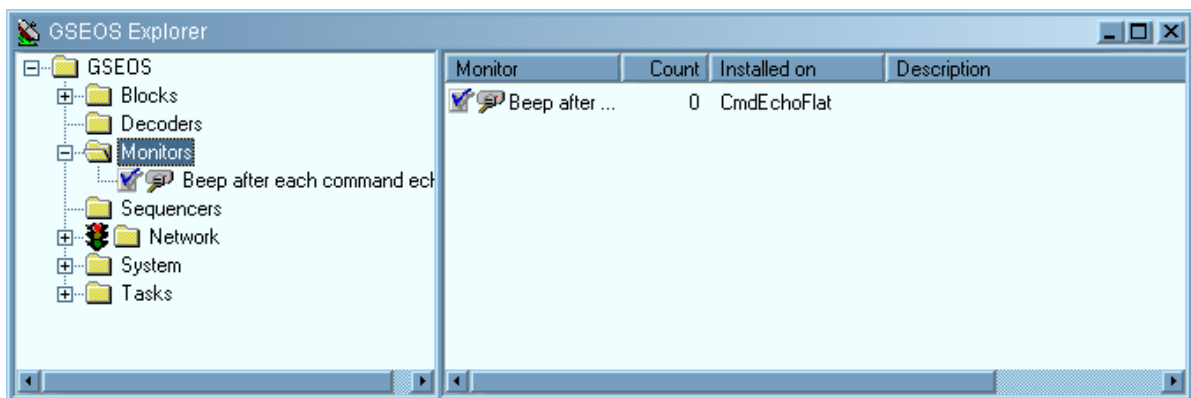


The right hand pane shows the Expression name together with the function body and the formula file the expression was loaded from.

3.2.6 Monitors

The Monitors node displays all monitors currently installed in the system. For more information on how to configure a monitor please refer to Modules/Monitor. The picture below displays a view of the currently installed monitors. They can be displayed by expanding the Monitors node underneath the GSEOS root node.

In the right hand pane you can see the number of invocations, the source block, i.e. the block that triggers the monitor execution, and finally a description which is the doc string that can be specified with the monitor definition.



Right clicking on any monitor node will pop up the following menu:



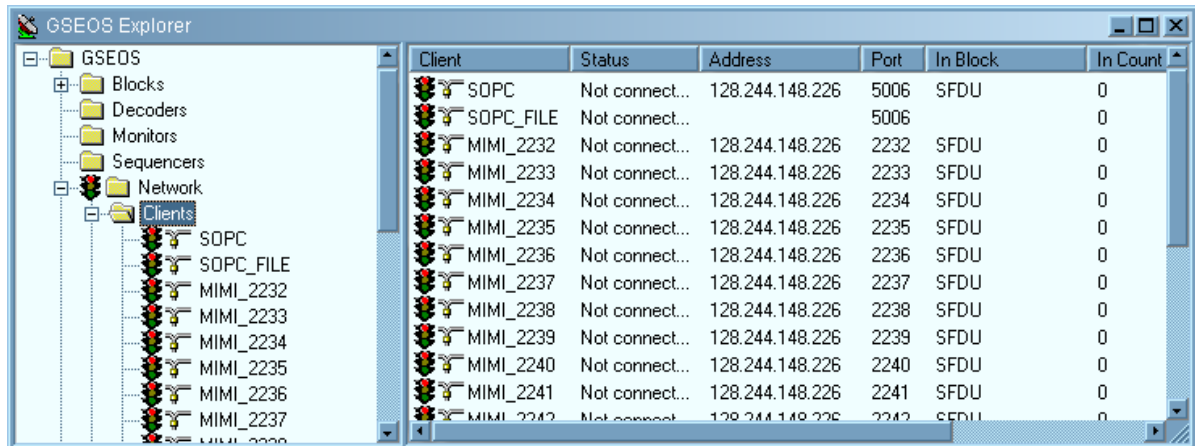
Here you can Enable/Disable the monitor or delete it altogether. Keep in mind, if you assign the monitor to a variable that you maintain, for example in a Python script, the monitor will not be removed until you release this variable. The delete operation only releases the internal reference that is held by the Explorer.

In the right hand pane you can select multiple monitors and invoke the operation on all the selected monitors at once. This feature is not possible with the left hand pane since you can only select one node at a time.

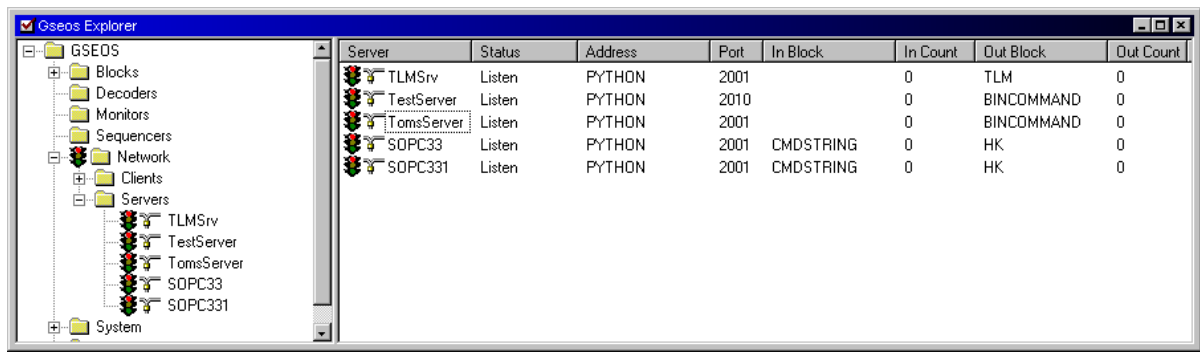
3.2.7 Network

The Network node lets you display and modify the status of your network connections. It is categorized into Servers and Clients. The network connection themselves are configured in the gseos.ini file in the [Net] section.

The Network main node lets you enable or disable the network as a data source. When the network is the active data source all data received on the network connections is forwarded to the system and data coming from any other data sources is discarded. When the network is disabled no data from the network is forwarded to the system. By right clicking on the main Network node you can enable/disable the network as a data source. The current active data source is indicated in the GSEOS main title bar.



Under the Clients node you will find all your network clients as configured in the gseos.ini file. In addition the current connection status as well as the incoming and outgoing block counts are reported. By right clicking on any particular client (or selection of multiple clients) you can connect or disconnect the client, depending on connection status. A green light in the icon associated with the connection indicates an established connection, whereas a red light indicates the client as disconnected.

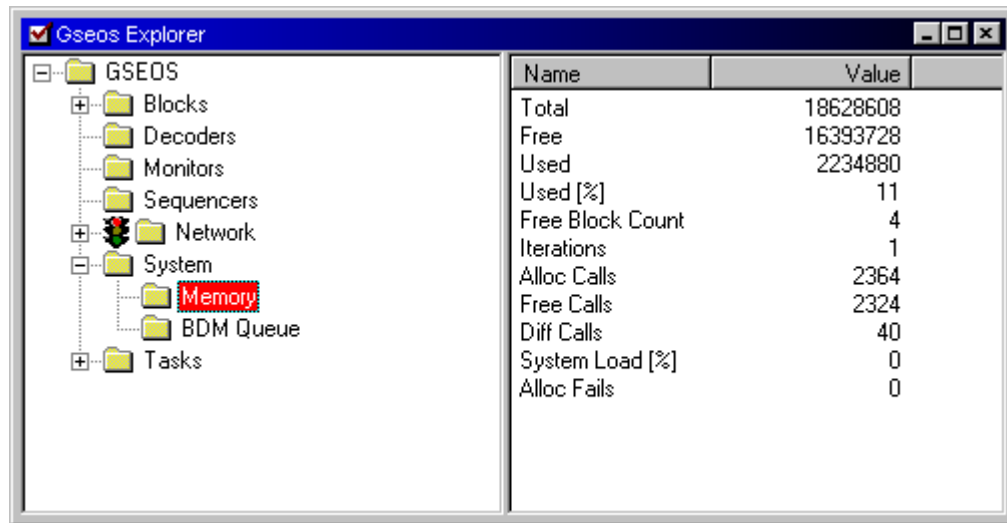


Underneath the Servers node the configured network servers are displayed. In addition to the configuration parameters you can see the connection status and the incoming and outgoing block counts. By right clicking on a server node (or a selection of multiple server nodes) you can bring up the server menu. This menu allows you to reset the connection if it is established. A green light indicates an established connections, whereas a red on indicates the server in listen mode, waiting for a client to connect.

3.2.8 Sequencers

3.2.9 System

The System node give you system performance measures. It lists all memory related performance counters in the Memory node and the status of the Block queue in the BDM Queue node. The following pictures shows a view of the System/Memory node:

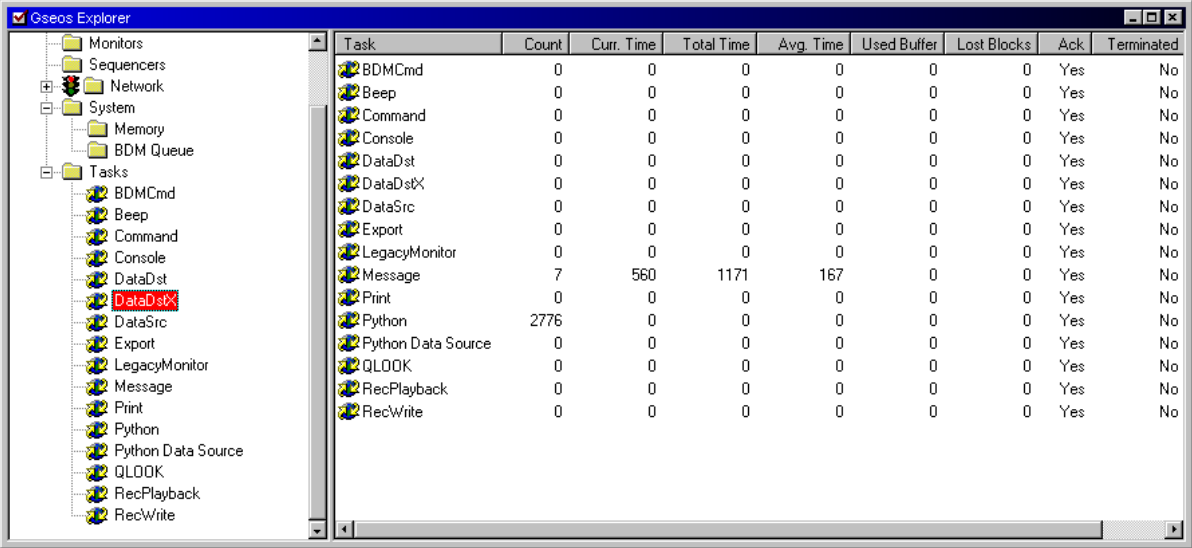


Name	Value
Total	18628608
Free	16393728
Used	2234880
Used [%]	11
Free Block Count	4
Iterations	1
Alloc Calls	2364
Free Calls	2324
Diff Calls	40
System Load [%]	0
Alloc Fails	0

The Total lists the total number of bytes allocated for GSEOS block buffer. This corresponds to the setting in the gseos.ini [Buffer] section. The Free count lists the available memory and the Used count the space that is currently occupied by unprocessed blocks. A percentage counter is provided as well. One very important counter is the Alloc Fails counter. This counter should always be zero. If the system can not accocate any space for a new buffer this counter will be incremented and the received data discarded. This means that incoming data was not processed and lost. If this counter is different from zero you might try to increase the buffer space allocated for the use of GSEOS. If this problem still persists, even with an adequately configured buffer the system is overloaded.

3.2.10 Tasks

All data sources and data consumers in GSEOS are listed in the Tasks node. This list give you an overview of the performance of the individual data sources.



Task	Count	Curr. Time	Total Time	Avg. Time	Used Buffer	Lost Blocks	Ack	Terminated
BDMCmd	0	0	0	0	0	0	Yes	No
Beep	0	0	0	0	0	0	Yes	No
Command	0	0	0	0	0	0	Yes	No
Console	0	0	0	0	0	0	Yes	No
DataDst	0	0	0	0	0	0	Yes	No
DataDstX	0	0	0	0	0	0	Yes	No
DataSrc	0	0	0	0	0	0	Yes	No
Export	0	0	0	0	0	0	Yes	No
LegacyMonitor	0	0	0	0	0	0	Yes	No
Message	7	560	1171	167	0	0	Yes	No
Print	0	0	0	0	0	0	Yes	No
Python	2776	0	0	0	0	0	Yes	No
Python Data Source	0	0	0	0	0	0	Yes	No
QLOOK	0	0	0	0	0	0	Yes	No
RecPlayback	0	0	0	0	0	0	Yes	No
RecWrite	0	0	0	0	0	0	Yes	No

Task

The name of the data source.

Count

The number of blocks that this task has processed.

Curr. Time

The time in ms this task takes to generate one block. This is a sliding average over the last few blocks so you can determine the current performance of the task.

Total Time

The total time in ms this task has been executing.

Avg. Time

The average time in ms needed to generate a block. This is the quotient of total time over count.

Used buffer

The buffer in bytes that is allocated in blocks that have not been processed by this task. This should typically be a small number, ideally zero.

Lost Blocks

The number of blocks that could not be allocated due to insufficient memory. See also the Explorer System node for information on allocation failures. If a task has requested notification of a particular block and this block could not be put on the BDM queue due to allocation failures that lost block counter of this task is incremented. This should not happen and the system needs to be configured to avoid lost blocks. This counter gives you some visibility into the performance of the system.

Ack

A data consumer has to acknowledge the processing of a particular block with the BDM. It will not get notified of new block arrivals until the acknowledgement for the current block is received. If this flag is permanently 'No' the task will not process blocks and there is most likely a bug in the task.

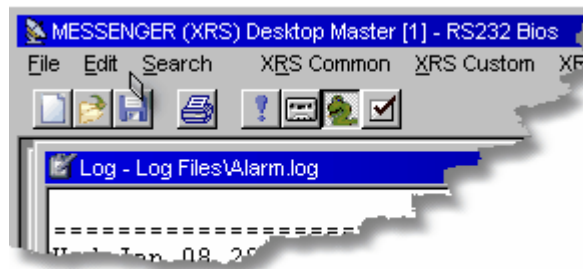
3.3 Log Windows

Log windows can be used as general purpose reporting windows. Each log window is associated with a log file. Content written to the log window will be automatically saved on system exit.

You can write content to a log file without the corresponding log window to be open. Log windows are regular MDI child windows like Qlook screen windows. They are located on a particular desktop page and their layout will be saved with the desktop settings. The caption bar indicates the log file the log window associated with.

There can be only one window open per log file, although you can have several different log files open. You can use the File menu to open and save log windows.

As opposed to the message window you can search and copy and paste in log windows. Once a log window is active the GSEOS main menu merges the Edit and Search menus:



Programmatic Access:

Most of the access to log windows will be programmatic from Python scripts. The Gseos module exports two functions you can use with log windows:

```
Gseos.Log
Gseos.LogSave
Gseos.LogReload
```

3.4 Menus

The GSEOS main menu consists of various different kinds of menus. The base menu consists of:



File:	All file related operations
View:	Open can close various View windows like the GSEOS Explorer or the Console window.
Tools:	Invoke GSEOS tools like configuration Wizards
Window:	Window management. Minimize, maximize, close windows, etc.
Help:	Access to this file and the online GSEOS web site for up to date

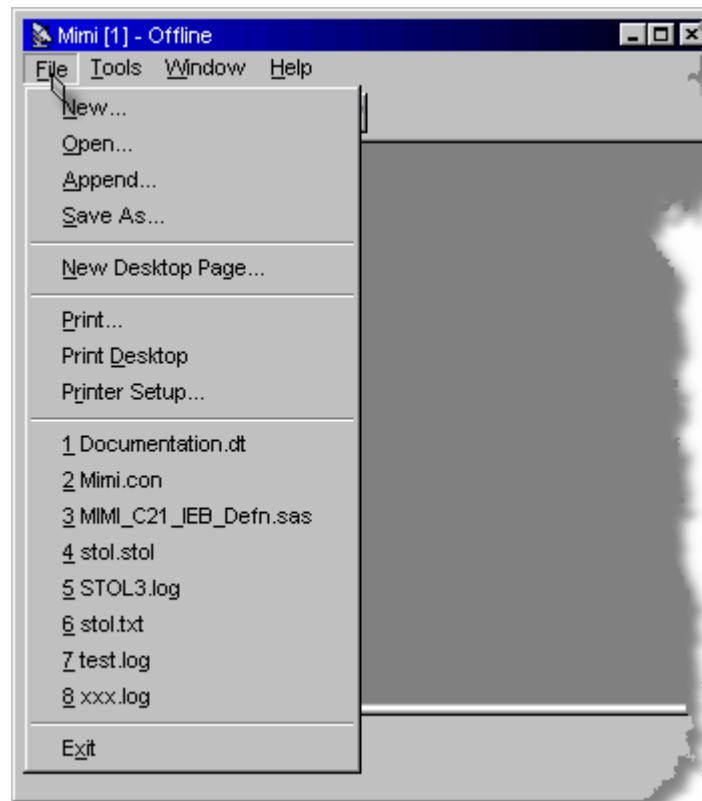
information.

Depending on what kind of child window is active in GSEOS specific menus will be merged into the main menu, between the File and View menus. In the above image a Qlook window is active and the menus: Edit, Draw, Style, and Options are merged into the main menu.

In addition to the automatic merging you can define your own custom menus that will be placed before the View menu. In this case the XRS Common, XRS Custom, and XRS EMBOX menus have been added.

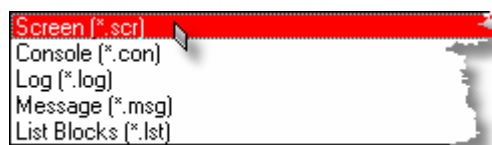
3.4.1 The File Menu

File management is accomplished with the File menu. The following picture shows the File menu.



New...

To create a new file you use the File/New... menu. You have to select the type of file you want to create from the 'Save as type' list box. The following options are available:

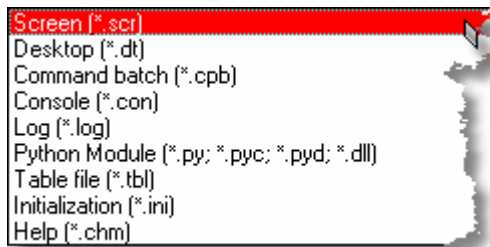


The system only allows one Message window open at the same time. If the Message

window is already open the according entry will not appear in this select list. Screen files will create a new GSEOS screen window. Screens allow to display the different data items in numerical or graphical form. Log will create a new Log window. The List Blocks setting allows you to get more detailed information about your block definitions in form of a list file. You have to enable the block listing capability in the gseos.ini file [System] section.

Open...

To open an existing file use the File/Open... menu. Select the type of file you want to open from the 'Files of type' select list. The following options are available:



You can select multiple files at one time and all of them will be opened, this is especially useful for window based files like Screen files or Log files.

The Screen, and Log files are the same you can find in the File/New... menu.

A Desktop file loads an entire screen configuration. This allows you to store the positions of all windows on all desktop tabs in one convenient format. You can also combine different desktop files by using the File/Append... menu.

A Command Batch file will start a time controlled command file. The start and end of the batch execution will be logged in the Message Window.

If you open an existing Python module (which can be either a .py, .pyc, .pyd, or .dll file) the system will try to reload the file first. If you made any changes to the file after it was imported already these changes will take effect. If the reload operation is not successful an import operation is performed. Any errors that occur during the load process will be reported in the Console window. Initialization files will open any *.ini file in the editor you have configured in the gseos.ini [System] section or Notepad if you have not configured an editor of your choice. This will give you quick access to gseos.ini the main configuration file. Note that most settings in gseos.ini don't take effect until you restart the application.

The file type Help lets you open Windows help files (*.chm) like this file. Finally, the Table file entry is a custom file type. To find out how to register your own custom file types in the GSEOS file management check the Gseos module for more information.

Append...

The Append... menu will append to the currently loaded files of the same file type. The system only supports the Desktop file type for append operations. If you append a desktop file the new desktop will be merged into the existing one. If you want to save this newly merged desktop use the File/SaveAs... menu and select the file type Desktop. Please note that the default name for the desktop will be the one of the desktop file last appended.

SaveAs...

The SaveAs... combines the functionality of the conventional Windows Save and SaveAs

commands. It will default to the current file name for the file type you select. In this way it acts like the Save command found in other applications. If you change the file name you can save the file under a different name and will therefore get the SaveAs functionality. You can save the following file types:

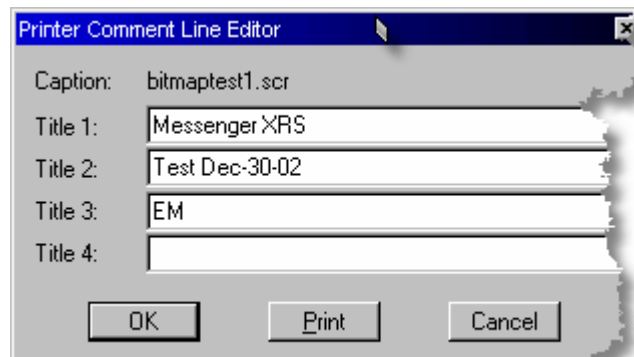


New Desktop Page...

In order to add additional desktop pages to the desktop use this menu. You will have to provide a unique name. The name can contain spaces and special characters. Please refer to the Desktop section for more information on desktop management.

Print...

The print menu opens the print dialog that allows you to print the active window. You can optionally provide comment information for the page. If you click Ok the comment information will be saved and the dialog closed without printing. If you choose Print the active window will be printed and the comment information will be saved. For the various printer options please refer to the gseos.ini Printer section documentation.



Print Desktop...

The Print Desktop menu allows you to print the entire GSEOS application. This feature allows to quickly print the current configuration for documentation purposes.

Printer Setup...

This menu invokes the standard Windows printer setup dialog and lets you configure the current printer settings from within GSEOS.

Most recently used list

The following entries represent the most recently used files and by simply clicking on the name you can load the according file quickly. This is especially useful for loading Python modules that you might change in an editor and want to reload.

3.4.2 The Help Menu

The Help menu allows you to invoke the GSEOS help system. It also provides a link to the GSEOS home page at www.gseos.com, as well as access to the About dialog that gives you detail version information about the system and any modules.



The About dialog shows the current version information as well as per module version information.



The GSEOS version number is the major and minor version, in this case 5.0. The last number represents the release in this case 183.

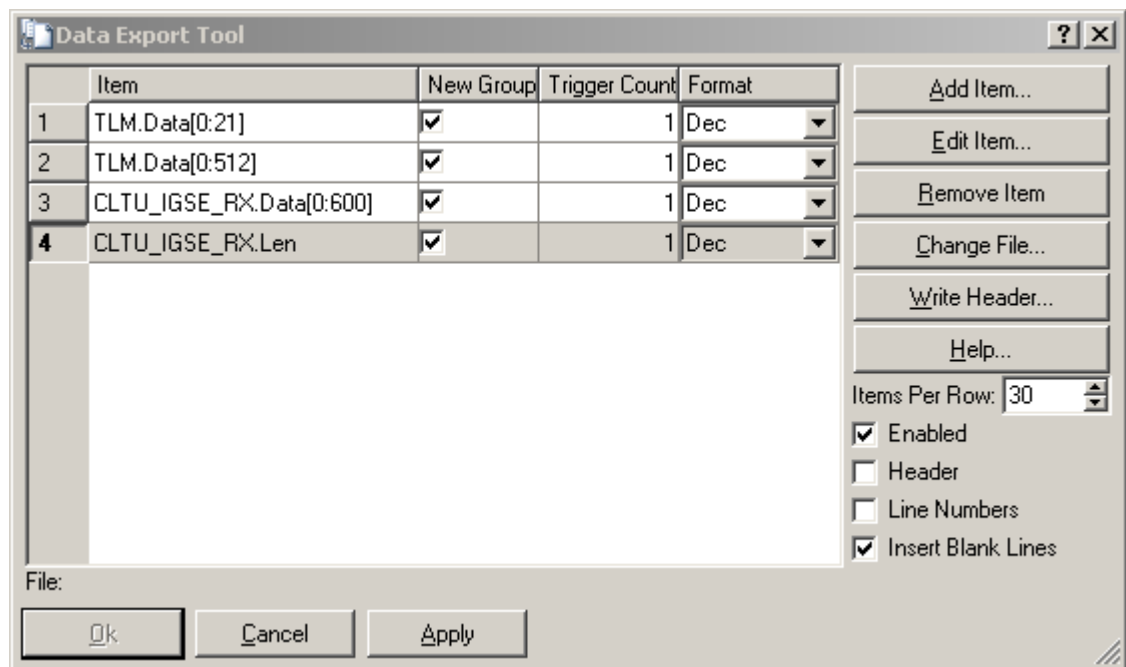
The next line indicates the current Python version number. In the list box below you can see all the modules registered in GSEOS with their according release number.

3.4.3 The Tools Menu

The Tools menu allows you to access various GSEOS tools like the Data Export Tool or configuration wizards. Also any plug-ins you have loaded may add an entry to the Tools menu.

3.4.3.1 The Data Export Menu

You can log any data item to a flat ASCII file with the Export Tool. When you select the Data Export menu from the Tools main menu the Data Export dialog is displayed:



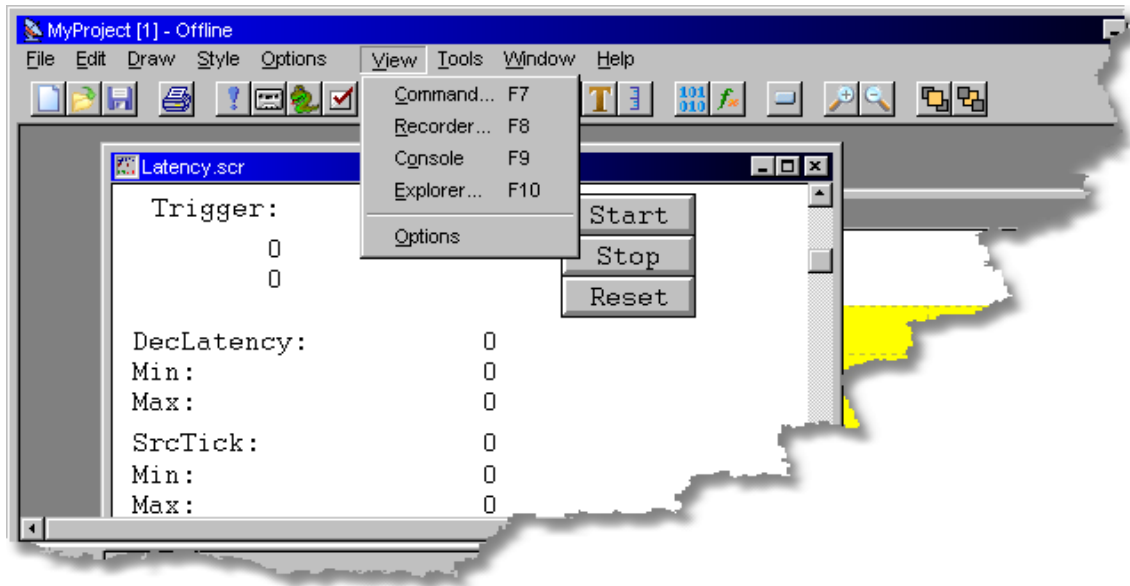
The listbox on the left displays all items to be exported. The Add Item button opens the data item selection dialog and allows you to pick an item and its dimension if it is an array item. The Change File button lets you specify the export file. If you have array items in the export list you can specify in the 'Items on row' edit window how many elements you want to place on one line.

The enabled flag lets you turn on and off the export of the selected items. If the 'Description' check box is checked each item will be reported with its block and item name, otherwise only the data is written.


For more elaborate formatting a GSEOS Monitor will be more appropriate. The monitor function will then write the data items out to a file in the formatting desired. This also has the advantage of programmatic access.

3.4.4 The View Menu


The View menu gives you access to various system windows and dialogs. The following picture shows the View menu. The menu items toggle the display state of the window or dialog referred to by the menu. The state of the dialog or window is indicated with a checkmark by the menu. If the menu entry is checked the dialog or window is open, otherwise it is closed. Most of the windows can be activated with a hotkey (as indicated on the menu) or via the tool bar as well.




Command...

The Command menu entry opens/closes the modeless Command dialog. You can also use the F7 hotkey or the  toolbar button to open or close the Command dialog.


Recorder...

The Recorder menu entry opens/closes the modeless Recorder dialog. You can also use the F8 hotkey or the  toolbar button to open or close the Recorder dialog.

Console...

The Console menu entry opens/closes the Console window. You can also use the F9 hotkey or the  toolbar button to open or close the Console window. Another way to close the Console window is to right click on the window and choose the Hide menu.

Explorer...

The Explorer menu entry opens/closes the GSEOS Explorer window. You can also use the F10 hotkey or the  toolbar button to open or close the GSEOS Explorer window. You can also close the GSEOS Explorer from the system menu of the GSEOS Explorer window itself.

3.4.5 The Window Menu

The Window menu allows you to modify the display status of the child windows on the desktop page. You can also adjust the settings of the tool and status bars of the application. A list of all windows on the current page is appended to the menu so you can quickly activate a particular window. The following image displays the Window menu:



Arrange Icons

The Arrange Icons entry will place all minimized windows to the bottom of the desktop page. Each window has a normal display position and a minimized display position. That is if you minimize a window and move the icon to a particular position on the desktop page it will be restored to that position the next time you minimize the window. If you use the Arrange Icons entry all icons will be laid out at the bottom (even if a window is not minimized, it's minimized position is arranged in line with all other icons).

Display Icons

Minimized windows are displayed as icons. These icons can be covered by other windows. If you select Display Icons the icons will be brought to the foreground so you can select the window of your choice.

Tile Horizontal

Lays out the windows in a horizontal fashion.

Note

This changes the size of the window. If you have sized the window to fit it's contents you will loose this sizing! Be careful using this function.

Tile Vertical

Lays out the windows in a vertical fashion.

Note

This changes the size of the window. If you have sized the window to fit it's contents you will loose this sizing! Be careful using this function.

Minimize All

Minimizes all windows on the current desktop page. The icons will take on their minimized positions. If you want to lay out the icons at the bottom of the desktop page you can use the Arrange Icons menu.

Restore All

Restores all windows on the current desktop page to their normal size. The windows will take on their normal positions and size as configured.

Close All

Closes all windows on the current desktop page.

Tool Bar

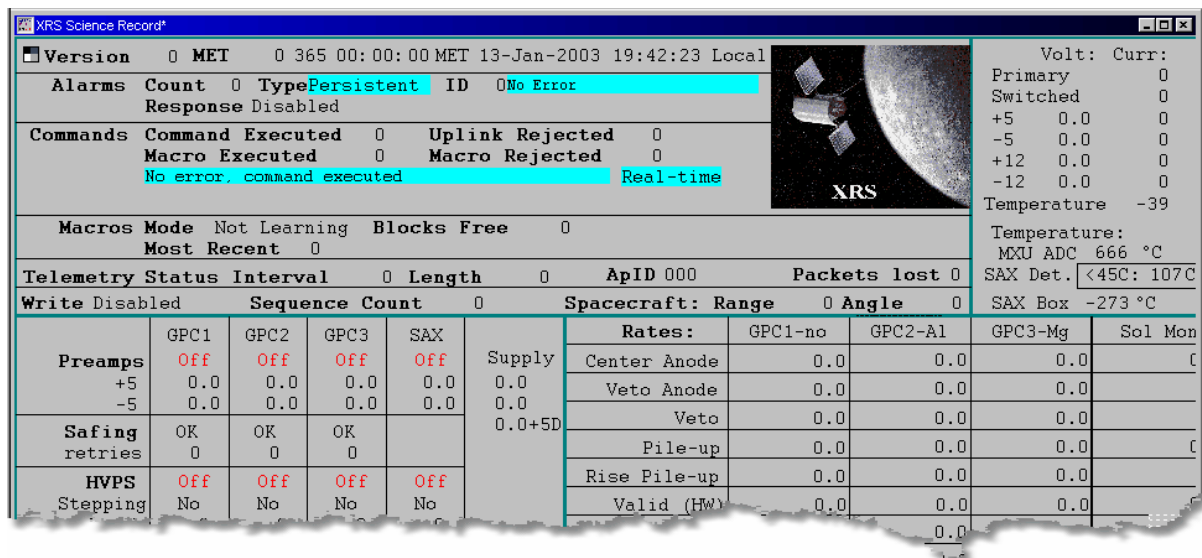
This menu lets you reposition the tool bar to either the left, top, or right edges of the GSEOS main window. However, this setting is not permanent. You can set the default tool bar position in the gseos.ini file [System] section.

Status Bar

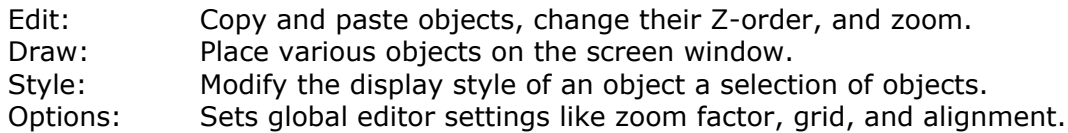
This menu lets display or hide the GSEOS status bar. However, this setting is not permanent. You can set the default status bar display setting in the gseos.ini file [System] section.

3.5 Screen Windows

Screen windows, also referred to as Qlook (Quick Look) windows are the heart of the GSEOS application. They display your data in real-time. The graphical, interactive editor lets you place data items on the screen window while the data is being displayed. You can place data items in various formats, command buttons, bitmaps, various static graphical elements such as lines and rectangles, as well as static text on a screen window. Once configured you can save the window to a .scr file. The following picture shows a screen window:



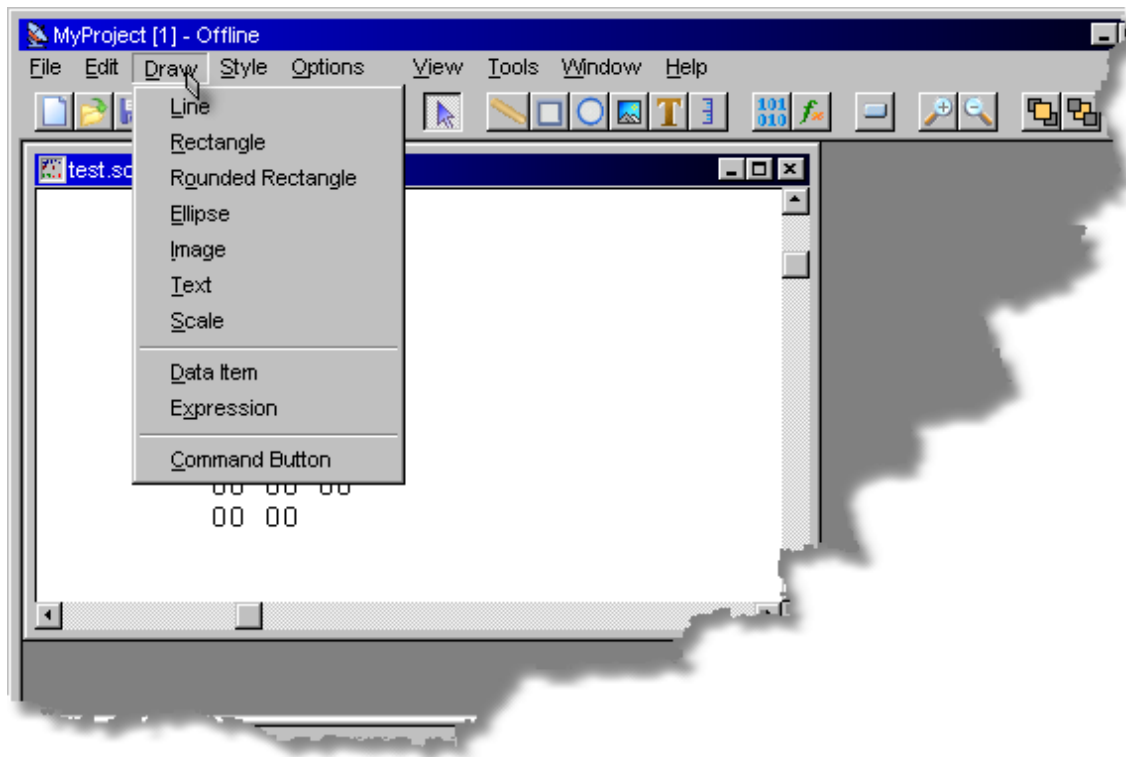
When a screen window is activated the GSEOS main menu will be merged with the Qlook menu. The following menus are added:



3.5.1 Menus

Edit
Draw
Style
Options

The Draw menu can be invoked from the GSEOS main menu when a screen window is active.




The following drawing tools can be selected:

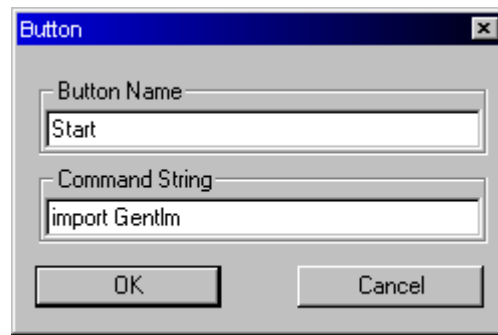
- Line
- Rectangle
- Rounded Rectangle
- Ellipse
- Image
- Text
- Scale
- Data Item
- Conversion Function
- Command Button

3.5.1.1.1 Command Button

The Command Button tool allows you to place a Push-button on the screen. On activation this button will issue the command you configure. A simple text preprocessing module allows you to prompt for command parameters. Commands can be any valid Python command. Another option to use commands in an easy fashion from the user interface is to configure command menus.

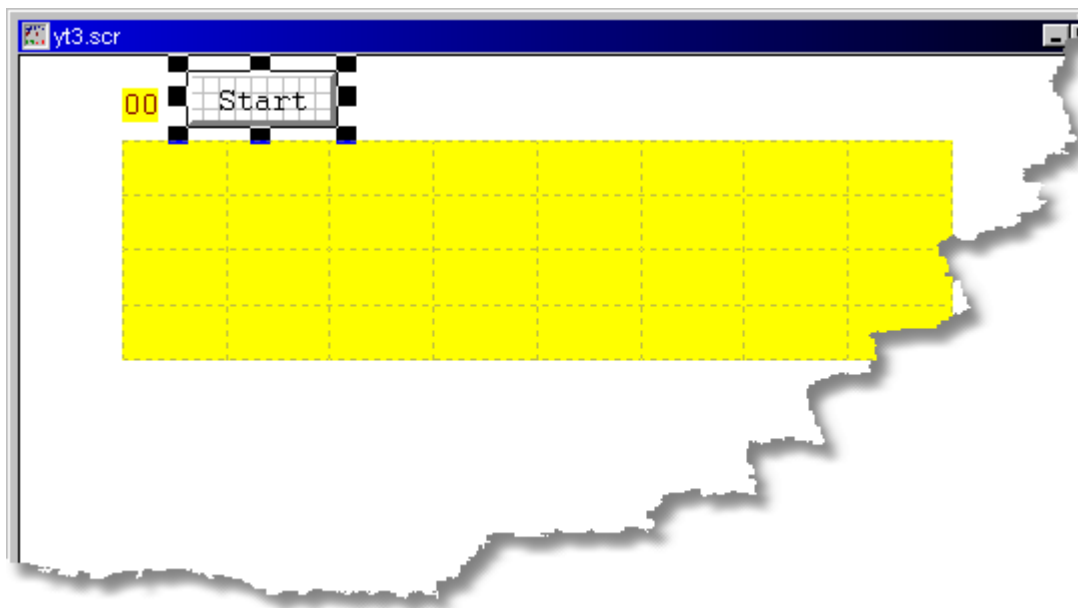
The Command Button tool can also be activated from the toolbar with the  button. After you finish selecting your drawing area the specific dialog to specify your command

parameters pops up:



The Command Button dialog lets you specify the button name, that is that text that is display on the button, as well as the command string to be executed. Upon execution a new CMDSTRING block will be generated with the contents of this command string copied into the CMDSTRING block. Then the CMDSTRING block is processed by the GSEOS command processor and forwarded to the Python interpreter.

The following snapshot shows the resulting button:



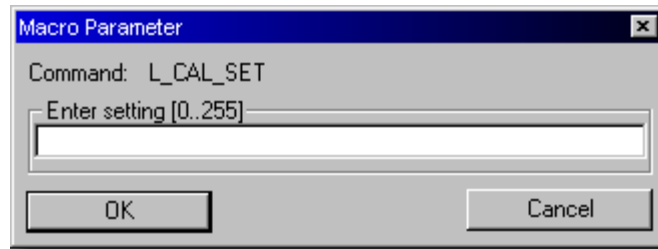
The Command String text allows simple text replacement:

If you specify the '\$' escape sequence a dialog box will be displayed when the command is executed. The dialog will prompt the user for input. Everything between and including the '\$' escape sequence and the terminating ' will be replaced by the users input. You can have multiple replacement strings in one Command String, this will pop up multiple input boxes.

The following example will query the user for a command parameter:

```
L_CAL_SET("E_RANGE1", '$Enter setting [0..255]')
```

On execution the following dialog will be displayed:




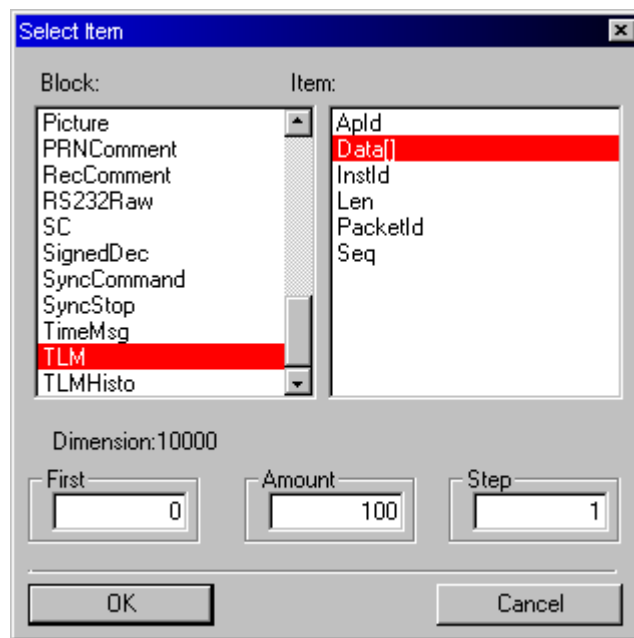
If the user enters 20 the following command will be generated:

```
L_CAL_SET("E_RANGE1", 20)
```

When you move the mouse cursor over the button the Command String will be displayed in the GSEOS status bar. The command button object can be configured with the following attributes: Text, Color.

3.5.1.1.2 Data Item

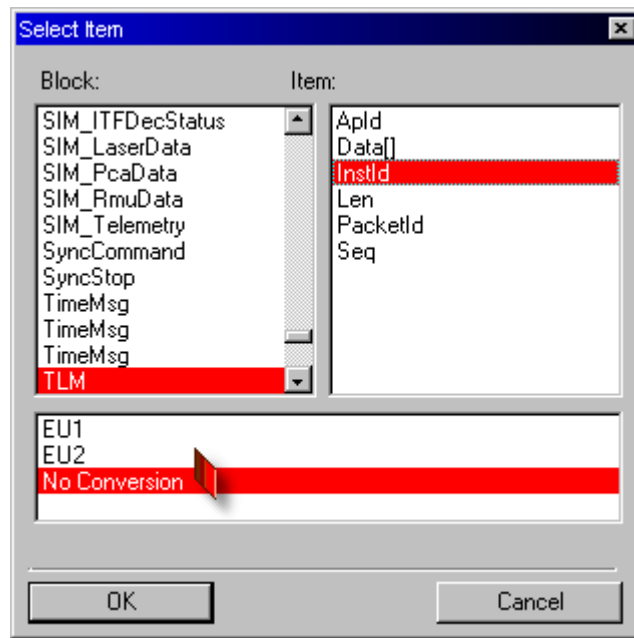
The Data Item tool allows you to place data objects. Data objects are visible representations of the items you define in the block definition file. Data items can be displayed in a variety of formats, numeric as well as graphic formats. Please refer to the Style/Data Item section for a detailed explanation of the various formats available. The Data Item tool can also be activated from the toolbar with the  button. After you finish selecting your drawing area the specific dialog for data items pops up:



This dialog allows you to enter select the data block and the item within the block you want to display. On the left hand pane select the block the item is a part of. Once you select a block the right hand listbox gets populated with the items of the selected block. If you select an array item, which is indicated by the square brackets you have the choice to select the range of the array you wish to display. The First edit box specifies the first index in the array to be displayed. Arrays are zero based. The Amount edit box specifies the number of items to display starting with the First item. The Step edit box can be used to skip over a number of items, i.e. if you want to display every other item you would set Step to 2. If you want to display every tenth item you would set Step to 10 and so on. After you confirm the selection the data item will be displayed on the screen within your selection area. Note that the selection area is adjusted according to your selection and may be enlarged if the data item does not fit into the area selected. If you plan on displaying large array items you might want to select a smaller range at first and verify the display dimensions. It is easy to change to range by simply double-clicking on the data object and adjusting the range.

Once a data item is placed on a screen it will be updated automatically every time the according data block is generated by the system.

If the selected data item has Conversion Functions associated the dialog box will allow you to select the conversion function you wish to apply to the item before displaying it. The image below shows a dialog for a data item that has several conversion functions associated with it:



If you choose 'No Conversion' the raw value of the data item is displayed.

Note

If your default drawing style is a graphic mode (Bargraph or y(t) display modes) and the range is not set properly the display might appear empty.


Note

For fast updating items the font selection for numeric format display is critical to system performance. True Type fonts require a lot of system resources to render. The fastest fonts are fixed pitch fonts, variable pitch and True Type fonts are the slowest to display. This is mainly a problem for fast updating items (refresh time less than 100ms).

The following attributes can be set on a data item object include: Color, Text, Range, Orientation, and Data Item.

The foll


3.5.1.1.3 Ellipse

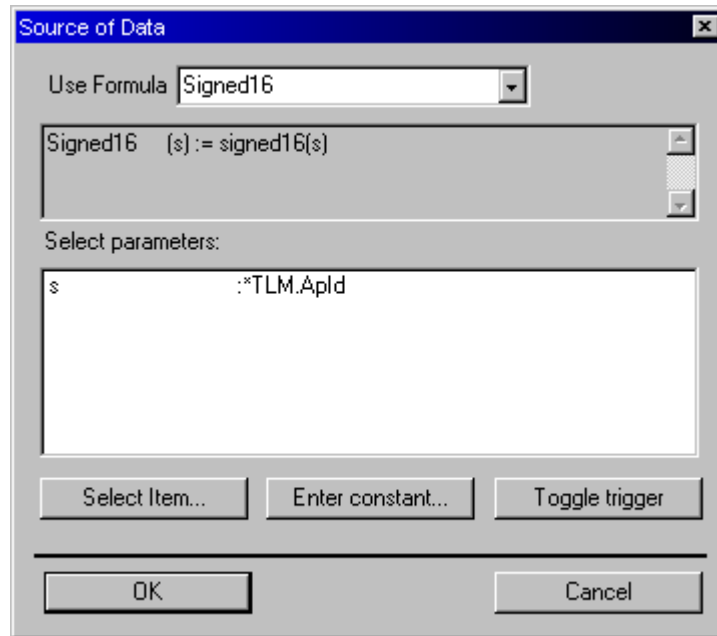
The Ellipse drawing tool allows you to place circles/ellipses on the screen. The Ellipse tool can also be activated from the toolbar with the  button. After you finish selecting your drawing area the ellipse is painted with the default line width and colors. The line width can be adjusted with the line style. The border will be painted in the foreground color and the inside of the ellipse in the background color. Line width and colors can be changed at any time with the Style menu.

3.5.1.1.4 Expression

The Expression tool is similar to the Data Item tool in that it allows you to place dynamic data items. However, the Expression tool lets you combine several data items in a mathematical expression. This is useful if you don't want to decode a specific data block using an according mathematical expression but just want to setup a quick Conversion Function. This way you don't have to set up a decoder that decodes the source block(s)

into a destination block, define the destination block and finally display the result. Once you define an Expression and load the Formula Definition file you can access the function from the dialog displayed below as well as from Python code.

Expression objects can be displayed in the same styles as a simple data item. Please refer to the Style/Data Item section for a detailed explanation of the various formats available. The Expression tool can also be activated from the toolbar with the  button. After you finish selecting your drawing area the specific dialog to specify your Expression and parameters pops up:



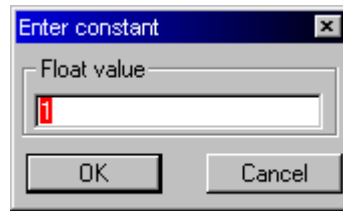
The Expression dialog is structured into two parts. First you have to select a predefined Expression. Expressions are defined in Formula Definition files and have to be loaded before they can be accessed. If you don't find your Expression function in the drop down list you can use the GSEOS Explorer to show you all loaded Expressions. Once you select an Expression, the function is displayed with it's formal parameters. The second part of the dialog assists you with filling in the actual parameters to be evaluated. Actual parameters can either be data items as described in the block definition file or constants. It is not possible to nest Expressions. Although, when defining an Expression you can use other Expression functions and nest them in this way.

Triggers

The actual parameters of an Expression can be data items located in different blocks. This means that these blocks will arrive at different times. The question is: When should the function be evaluated? To allow you to specify the execution sequence the concept of triggers is introduced. If your function takes parameters from different data blocks you can set a trigger on any block to invoke the evaluation of the function. A trigger is indicated with an asterisk. There only needs to be one trigger per block, however, if you set a trigger on multiple items of the same block the function is only evaluated once. There needs to be at least one trigger on one of the data items to evaluate the Expression.

To set a parameter select the according formal parameter in the Select Parameters list box. To select a data item click on the Select Item... button, to select a constant click on the Enter Constant... button.


When you click on the Select Item... button the standard data item select dialog opens and you can specify the data item you want to use as the actual parameter. Note that you can use array items as parameters, however in this case the dimensions of all parameters need to be the same. Constants can be used in conjunction with array item parameters. To specify a constant use the Enter Constant Dialog:




Any valid float number is acceptable as a constant.

Please note that data items are not typed, so if you need a signed or floating point representation of the bit pattern you have to use one of the conversion functions available. For more complex Expressions a specific decoder is the more appropriate solution. As with the data item be careful with displaying large array items. Otherwise, all the stlye options that can be set for a data item are also applicable to Expressions.


3.5.1.1.5 Image

The Image tool places static bitmap images on the screen. The images have to be in the Windows bitmap (*.bmp) format. The Image tool can also be activated from the toolbar with the  button. After you finish selecting your drawing area the Image tool specific dialog pops up. This is a simple file open dialog that allows you to specify the bitmap file you wish to display. The default extension is *.bmp. The image will be scaled to the dimensions of the drawing area you selected. There are no attributes for the image object.

3.5.1.1.6 Line

The Line drawing tool allows you to place lines on the screen. Lines are helpful in separating a screen into logical divisions (sometimes rectangles may be more appropriate). The Line tool can also be activated from the toolbar with the  button. After you finish selecting your drawing area the line is painted with the default line width. The width of the line can be adjusted with the line style. The line width and color attributes are the only ones that can be set for a line object. To move or change the line, simply click on the line itself to select it.

3.5.1.1.7 Rectangle

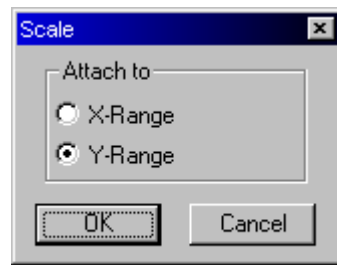
The Rectangle drawing tool allows you to place rectangles on the screen. Rectangles like Lines are helpful in separating a screen into logical divisions. The Rectangle tool can also be activated from the toolbar with the  button. After you finish selecting your drawing area the rectangle is painted with the default line width and colors. The line width can be adjusted with the line style. The border will be painted in the foreground color and the inside of the rectangle in the background color. Line width and colors can be changed at any time with the Style menu.

3.5.1.1.8 Rounded Rectangle

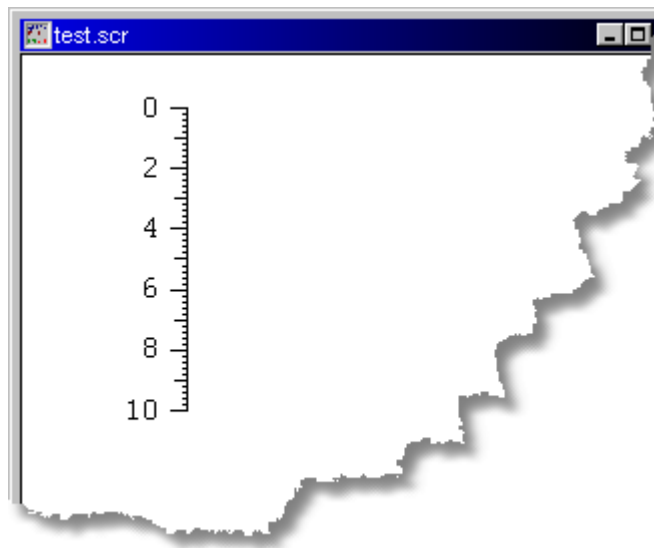
The Rounded Rectangle drawing tool allows you to place rounded rectangles on the screen. The Rounded Rectangle tool only be selected from the Draw menu. This tool is very similar to the Rectangle tool. The radius of the corners can not be adjusted.

3.5.1.1.9 Scale

The Scale tool will place a scale on the screen. Scales are typically used in conjunction with other graphical object like bargraphs or $y(t)$ plots. You usually compose the graphic by placing one or two scales for the horizontal and/or vertical axis and the graphic object itself. After you finish selecting your drawing area the specific dialog for the scale tool pops up:



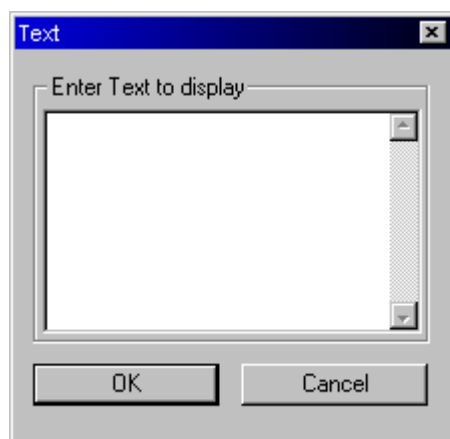
This dialog allows you select the range to use. The range style covers two dimensions, x and y for horizontal and vertical respectively. When selecting multiple objects like the three objects mentioned above (two scales and the graphical object) you can set the range for all three of them at the same time. In this case you want to use the x-range for the scale attached to the horizontal axis and the y-range for the scale attached to the vertical axis. In this dialog you can specify which range to use to dimension the scale. The following image shows a scale attached to the y-range and the range set to x: (0, 1); y: (0, 10):



The attributes you can set on the scale object include: Color, Text, Line, Range, and Orientation.

3.5.1.1.10 Text

The Text tool allows you to place static text on the screen. Static text is useful in describing data items placed on the screen. The Text tool can also be activated from the toolbar with the **T** button. After you finish selecting your drawing area the specific dialog for the text tool pops up:



This dialog allows you to enter the text you want to display. To enter carriage returns in the text use Ctrl-Enter. The text object recognizes the Color and Text attributes. You can change these attributes with the Style menu. Since various fonts render differently on a printer than on the display your layout may be distorted on the printer. To avoid this you can choose printer fonts, these fonts will render the same on the screen and on the printer. To change the text simply double-click on the text object and the Text dialog will pop up and allow you to change the text.

3.5.1.2 Edit

3.5.1.3 Options

The options menu controls the three modeless dialogs Grid, Zoom, and Align that can be used to control the placement of items on a grid, the zoom factor, and the alignment respectively.

3.5.1.3.1 Align

The Align dialog allows you to align screen objects on a screen window. The alignment will adjust the elements according to the alignment selected. The item(s) moved will move off the grid if that is required for the alignment to take effect.

The following image displays the Alignment dialog. The Alignment dialog is a modeless dialog box, this means you can continue to use the GSEOS user interface while the Alignment dialog remains open.



3.5.1.3.2 Grid

The Grid dialog allows to switch the item grid on/off and to set the grid spacing for the active screen window. When you switch windows the current grid settings of the window selected will be displayed in the Grid dialog. The grid settings are saved with the window. So if you load a window the grid settings there were in effect when the screen was saved will be restored.

Display objects are aligned to the current grid if it is enabled. If you change the grid spacing or turn it off the alignment of newly placed items may not be on the grid as defined before. Changing grid spacing within a screen definition should in general be avoided.

The following image displays the Grid dialog. The Grid dialog is a modeless dialog box, this means you can continue to use the GSEOS user interface while the Grid dialog remains open.



3.5.1.3.3 Zoom

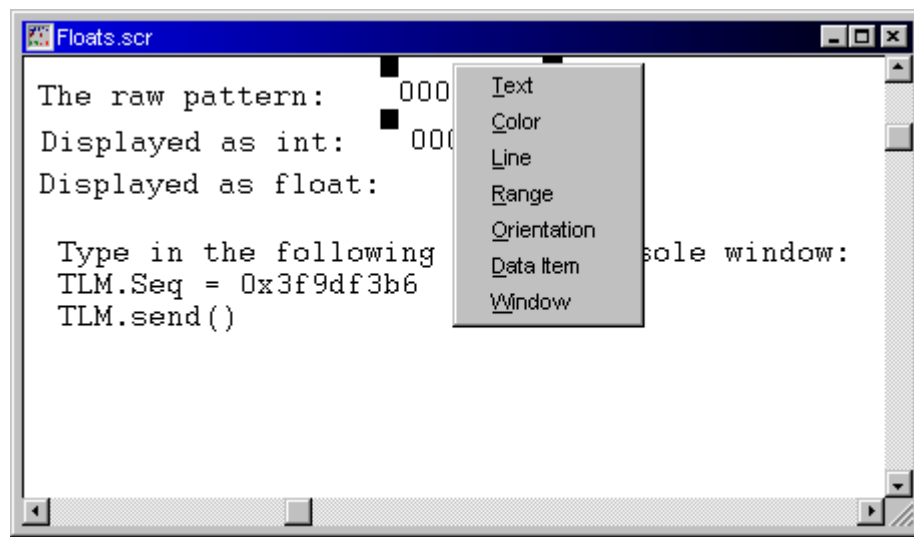
The Zoom dialog allows you to change the zoom factor for the active screen window. When you switch windows the current zoom factor of the window selected will be displayed in the Zoom dialog. The zoom factor is saved with the window. So if you load a window the zoom factor that was active when the screen was saved will be restored.

The following image displays the Zoom dialog. The Zoom dialog is a modeless dialog box, this means you can continue to use the GSEOS user interface while the Zoom dialog remains open.



3.5.1.4 Style

The Style menu is available from the main menu or by right clicking on a screen item while in editing mode.



You use this menu to set display styles of the screen objects. A style specific dialog box will open upon menu selection and you can set the properties accordingly. The style is applied to the object(s) selected. If no object is selected the default style is modified. The default style gets applied to newly created objects. Say you set the background color to red with no object selected. This will set the default background color to red. All objects you create new will have a red background.

The following styles are supported:

- Text
- Color
- Line
- Range
- Orientation
- Data Item
- Window

3.5.1.4.1 Color

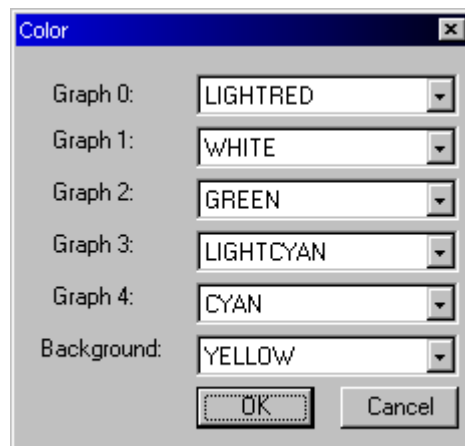
The Color style modifies the foreground and background color of the object. Almost all objects support foreground and background color, the only exception are Data Items or Conversion Functionss that use the casting (text reference style).

You can change the color settings with the following dialog box:



To set the foreground color you use the upper drop down box, to select the background color you use the lower one. The only special 'color' is Transparent. This means that nothing is drawn. This typically only makes sense for graphical objects and only for the background color. In this case the background is not erased and you can stack multiple objects on top of each other. The refresh of one will not erase the other.

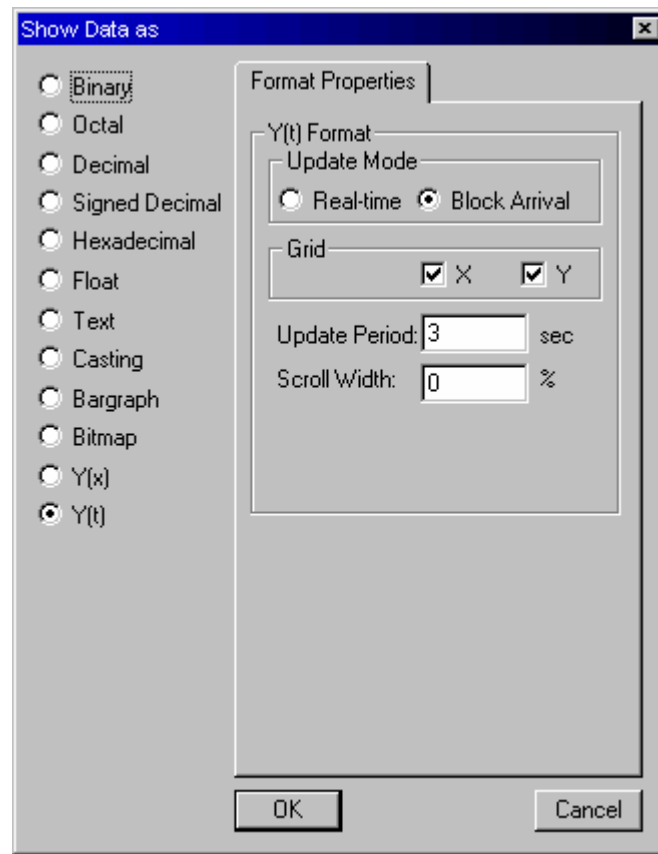
$y(t)$ objects can display multiple graphs and you can set the color of each graph independently. If you select the Color dialog with one or multiple $y(t)$ displays selected you will get the following dialog box:



The color for Graph 0 is equivalent to the foreground color. If you convert a $y(t)$ display to a different type you will lose the settings for the colors of Graph 1 to 4.

3.5.1.4.2 Data Item

The Data Item style dialog box is invoked by selecting Data Item from the Style menu, either on the main menu or from the context sensitive menu which you can pop up by right clicking on a data item object. The Data Item style lets you display the data item in a number of different styles from various numeric formats to textual display to graphic representation like bargraph, $y(t)$ and bitmap. Depending on the style you choose you can set different properties of the display. The following image shows the data item style dialog:



On the left hand pane you can select the format you want the item to be displayed as, the right hand pane allows you to modify the properties of that particular style. In the picture above the style $y(t)$ is selected and you can set the $y(t)$ display properties like update mode, grid, and scroll width which are specific to the $y(t)$ display style on the right hand side of the dialog.

The numeric formats supported are Binary, Octal, Decimal, Signed Decimal, Hexadecimal, and Float. The available text formats are Text and Casting. The graphic formats are Bargraph, Bitmap, $Y(x)$, and $Y(t)$.

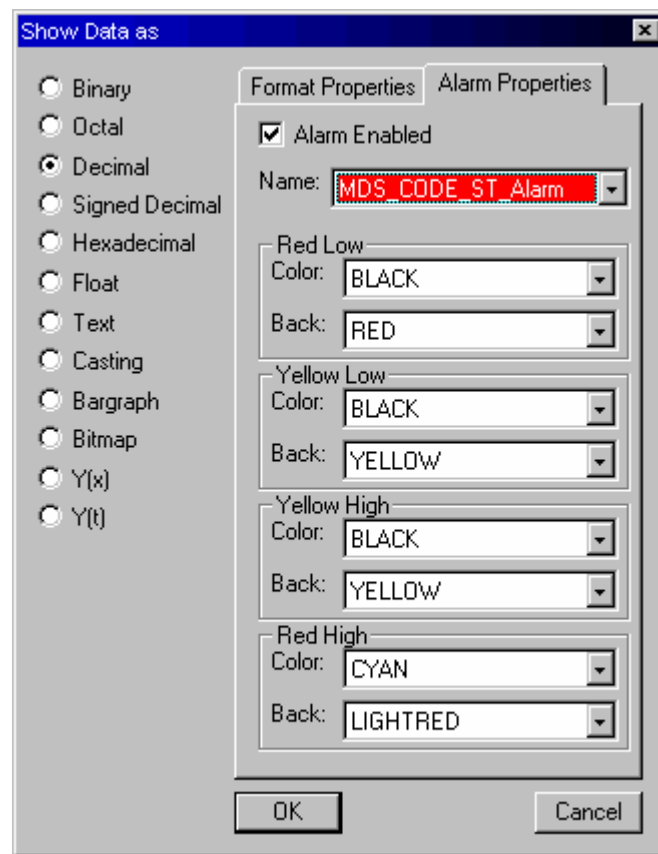
Some formats also allow you to select alarm properties, in this case the dialog shows an additional tab 'Alarm Properties' that controls the alarm settings.

3.5.1.4.2.1 Alarm Properties

The following data item formats support alarmed display mode:

- Binary
- Decimal
- Float
- Hexadecimal
- Octal
- Signed Decimal

If you select one of these formats the right hand pane will display an additional tab with alarm properties:

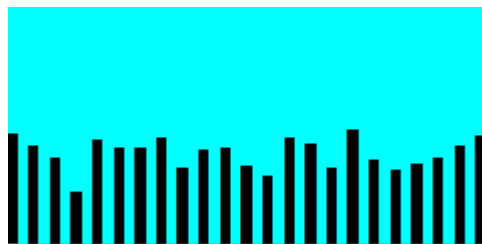


The settings on this tab let you specify the alarm definition to use. Keep in mind that for your alarm definitions to show up in the select box you have to load them first. Once you select a valid alarm definition the Enabled flag is automatically checked for you. If you select "No Alarm" the Enabled box will be unchecked and disabled. You can also manually uncheck the Enabled box even when you have selected a valid alarm. Setting alarm checking on an item adds considerable processing overhead and should be used accordingly. Especially fast updating items and large array items might slow down the system.

The color drop boxes allow you to specify the colors to use for the individual alarm conditions.

3.5.1.4.2.2 Bargraph

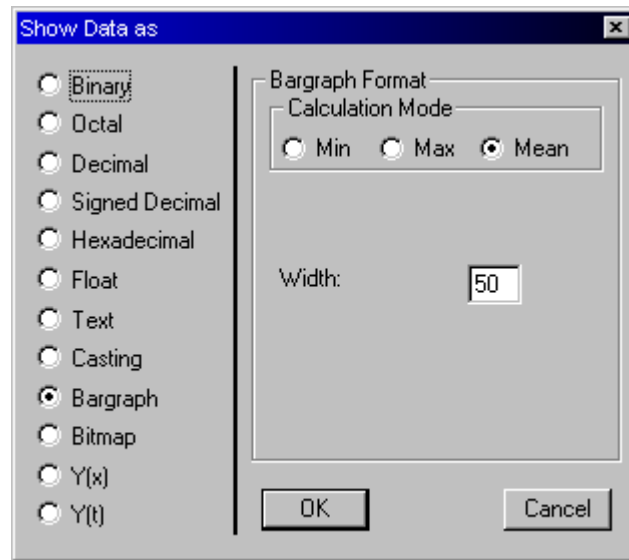
The Data Item style Bargraph displays a data item with the graphical representation of a bar. The following picture shows a Bargraph display:



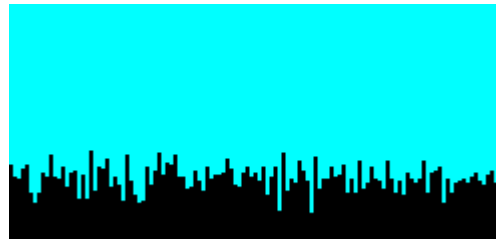
The above display shows an array item in Bargraph format. You can specify the width of the graphs in percent, this allows for a configurable gap between the bars. If you set the

width to 100% there will be no gap. For scalar items the Width setting has no effect.

The Data Item style dialog box for Bargraph displays looks as follows:



The bars are drawn so they spread evenly into the selected display area. If the array can not be displayed in the space available bars will be dropped from the display. The following image shows such a configuration.

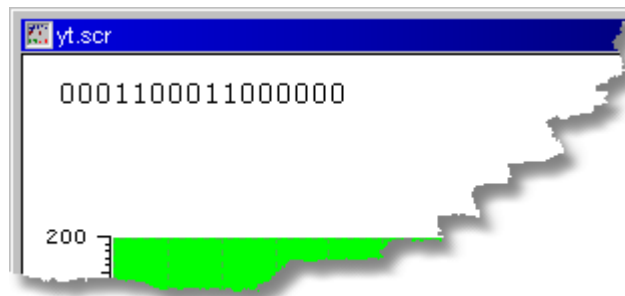


In the above picture an array item with 1000 elements is displayed. The screen can not hold that many values on the display area and bars get dropped. The calculation mode determines how to calculate the value of the resulting bar, if values are dropped. Say there are three values that need to be mapped to one bar. Min picks the minimum of the three, Max the maximum, and Mean the mean value of the three data points. The resulting values is used for the display.

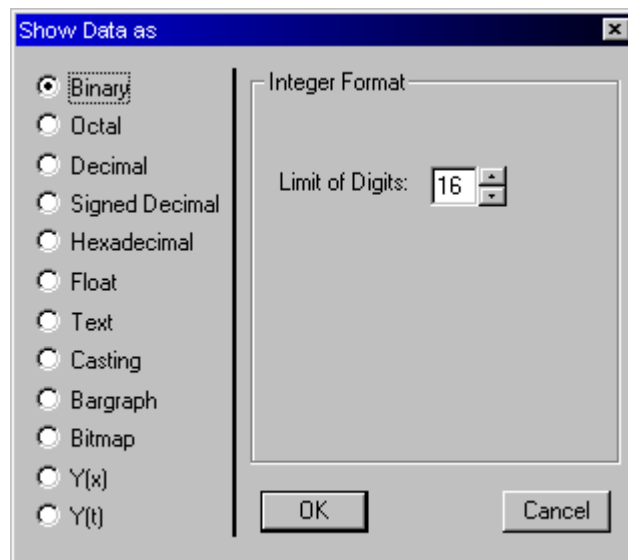
As with other graphical displays the proper setting of the range is important. In the case of the Bargraph display only the y-dimension of the range is used. It maps the values onto the graphical display. Say your y-range is 0..1000. If your data value is 333, the bar will extend to about 1/3 of the total display height. The orientation of the bar depends on the settings of the Orientation property.

3.5.1.4.2.3 Binary

The Data Item style Binary displays a data item in binary format:



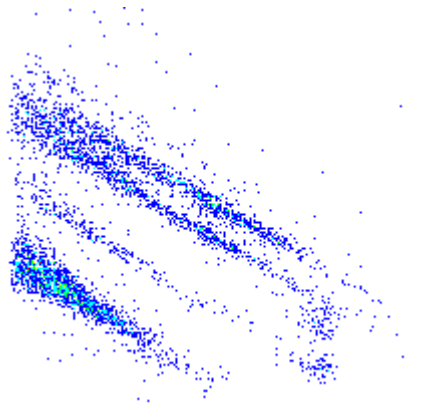
If you want to display status data the Casting display may be the better choice since it lets you assign text labels for the individual status values.
The Data Item style dialog box for Binary displays looks as follows:



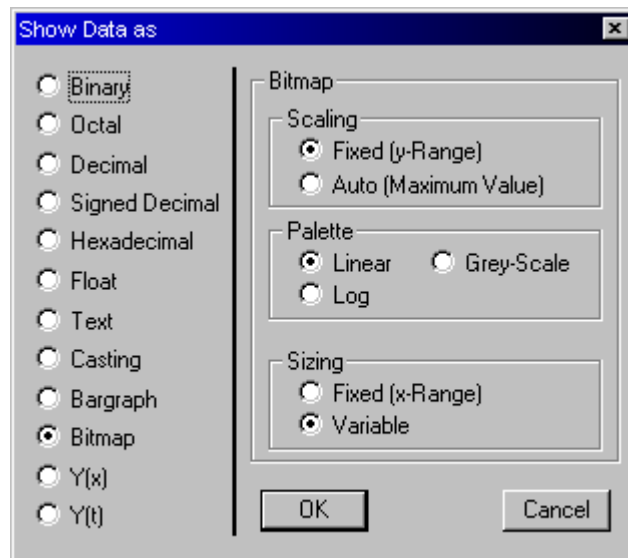
The limit of digits sets the number of digits to display per item. If the item can not be displayed within this limit the display will show as hash characters (#####). The default value of the limit corresponds to the bit length of the item selected.

3.5.1.4.2.4 Bitmap

The Data Item style Bitmap displays a data item as a two dimensional bitmap with the data values encoded as color mappings of the bitmap pixels. This display style is typically used to illustrate two dimensional histograms. See the histogram decoder section for more information on histograms. with the graphical representation of a bar. The following picture shows a Bitmap display:



The Data Item style dialog box for Bitmap displays looks as follows:



Range:

The Bitmap display maps an array item to a two dimensional display. The dimensions of the bitmap are determined by the Range setting. The x-Range determines the width of the bitmap in pixel, the height is the result of the dimension of the array item divided by the x-Range. Say you have an item of dimension 1024 and set the x-Range to 128 the resulting bitmap will be 128 pixel wide and 8 pixel high (The mapping to the horizontal/vertical dimensions can be changed with the Orientation setting). The y-Range is used to map the data values onto colors as outlined in the following paragraph.

Scaling:

The scaling section of the Bitmap properties determines how the colors are mapped to the data values. If you select Fixed (y-Range) the entire y-Range is mapped to the available 256 colors. Say your y-Range is 0..100 and your data value is 10 you would see a pixel with about 10% 'intensity'. This setting will keep a constant mapping of colors to values, depending on the y-Range and the color mapping.

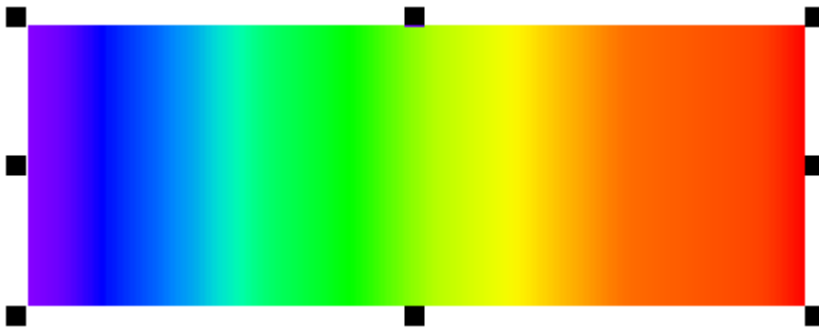
Sometimes it may be desirable to have a relative mapping, that is the largest value on the display should be displayed with the 'highest' color, all other values are mapped onto the interval 0 .. largest value. This Auto mode utilizes the entire color mapping.

However, the mapping will change depending on the currently largest value on the display. This is an appropriate mode for histograms to display the relative distribution of data.

Palette:

The value to color mapping is accomplished with different palettes. Each palette has 256 different colors and the values to be displayed get mapped to that range. Three palettes are available, Linear, Grey-scale, and Log. Please refer to the images below for the color mappings. The linear palette distributes the values linearly over the available RGB range, the grey scale uses a linear distribution as well with $R=G=B$ resulting in a grey mapping. The log palette uses a logarithmic distribution to map 0..256 to the resulting RGB values.

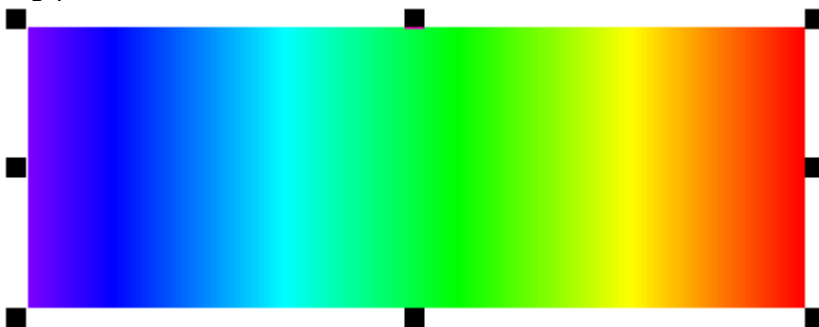
Linear palette:



Grey-scale palette:



Log palette:



Sizing:

If you choose fixed sizing the dimensions of the bitmap are mapped 1:1 to physical pixels

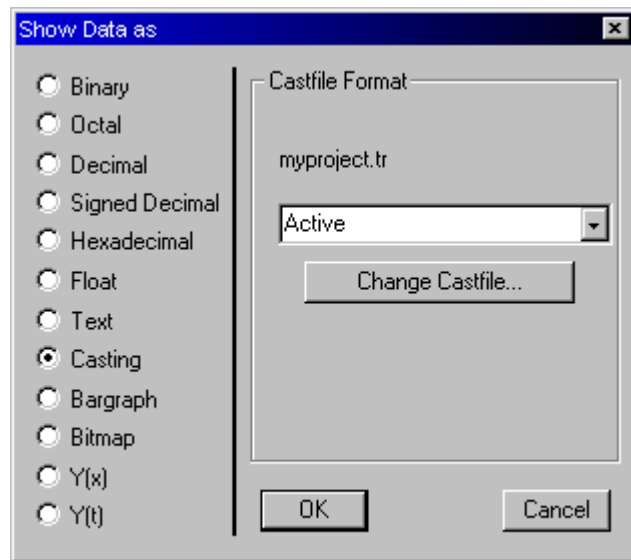
on the screen. In variable mode you can resize the bitmap and it is mapped to physical pixels as appropriate. The fixed sizing mode is useful for getting an undistorted display and map be appropriate for larger bitmaps. The variable mode allows for a zoomed in view.

3.5.1.4.2.5 Casting

The Data Item style Casting displays a data item as text looked up in a reference file. This display mode is suited for status data or non-linear lookup. ASCII text. The lookup file has the extension .tr for text reference. The following example shows an item displayed in casting mode

Inactive

The Data Item style dialog box for the Text display looks as follows:



The text reference file has the following syntax:

```
Reference Name
{
  Item Definition;
  Item Definition;
  .
  .
  .
  Item Definition
}
```

Item Definition:

Value [- Range], "Text", ColorCode

Examples:

```
ErrOk      { 0 ,      "Error", 0x0C;
             1-65355 , "Ok",    0x02 }
```

```
Illegal    { 0 , "Legal",   0x02;
             1 , "Illegal", 0x0C }
```

```
EPU_Status { 0 , "Booting", 0x70;
             1 , "Calc ...", 0x70;
             2 , "Running",  0x70 }
```

The reference name describes the name of the text reference and is selected in the Casting dialog as described above. The item definitions define which values get mapped to what text and in what color representation. The most significant nibble of the color code byte describes the foreground color and the least significant nibble describes the background color. You can either specify an individual value or a range that gets mapped to the display text. The data item displayed on the screen will take up as much space as the longest text in the reference definition.

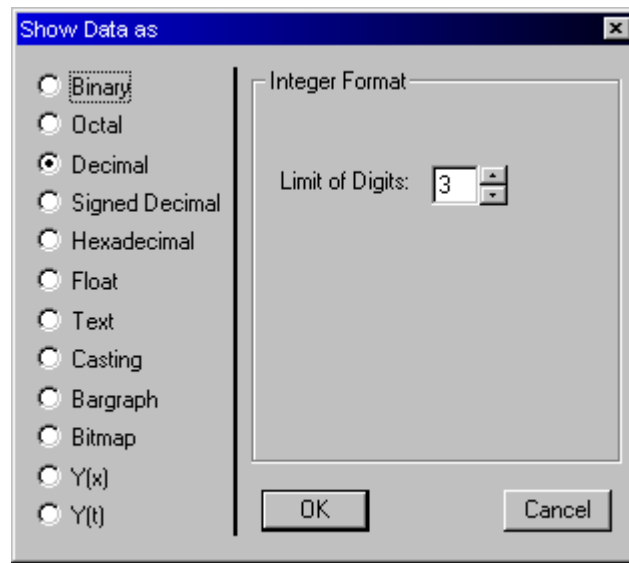
3.5.1.4.2.6 Decimal

The Data Item style Decimal displays a data item in decimal format:

```

■      ■      ■      ■
 133 106 105  74 111 106 126
  85 130 102  86  82 114 106
  88  95  73  93 129  82 118
■      ■      ■      ■
  58  76  80 116 143  93 114
  85 118  81 139 101 114 119
 121  90 120  66  99
■      ■      ■      ■
```

This display style is different from the Binary, Octal, and Hexadecimal styles in that the value will not be zero padded but is padded with spaces. The Data Item style dialog box for the Decimal display looks as follows:



The limit of digits sets the number of digits to display per item. If the item can not be displayed within this limit the display will show as hash characters (#####). The default value of the limit corresponds to the bit length of the item selected. The value will be padded with spaces to the left to the limit of digits specified.

3.5.1.4.2.7 Float

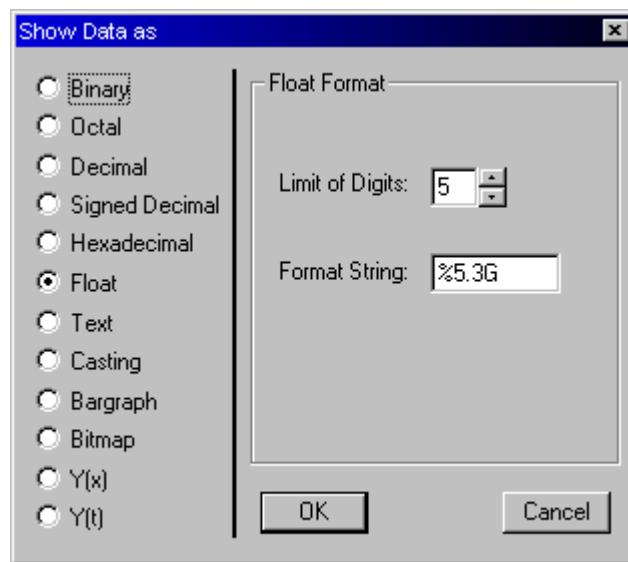
The Data Item style Float displays a data item in float format:

```

■ 235.000 105.000 ■ 226.667 160.000 ■
  130.000 171.667 183.333 195.000
  161.667 126.667 250.000 160.000
  203.333 200.000 173.333 185.000
■ 195.000 166.667 123.333 196.667 ■
  150.000 135.000 185.000 221.667
3 160.000 158.333 191.667 235.000
  138.333 130.000 143.333 196.667
  148.333 155.000 150.000 173.333
  131.667 173.333 118.333 148.333
■                                     ■

```

Since GSEOS does not assign data types the float representation is only useful with an Conversion Functions. The format string you can specify in the properties section of the dialog is a subset of the C printf format parameter. 'G, g' chooses the shortest format possible, 'F, f' displays the item in fixed point notation, 'E, e' uses the exponent, mantissa format. Various precision parameters and padding options can be applied. The Data Item style dialog box for the Float display looks as follows:



The limit of digits sets the number of digits to display per item. This does not include the leading negative sign. If the item can not be displayed within this limit the display will show as hash characters (#####). The default value of the limit corresponds to the bit length of the item selected. The padding depends on the format string specified.

Note:

If you have a data item that is the binary representation of a IEEE float and you want to display as float you have to interpret it as a float value using the `dtof()` or `ltof()` functions before using the float style.

3.5.1.4.2.8 Hexadecimal

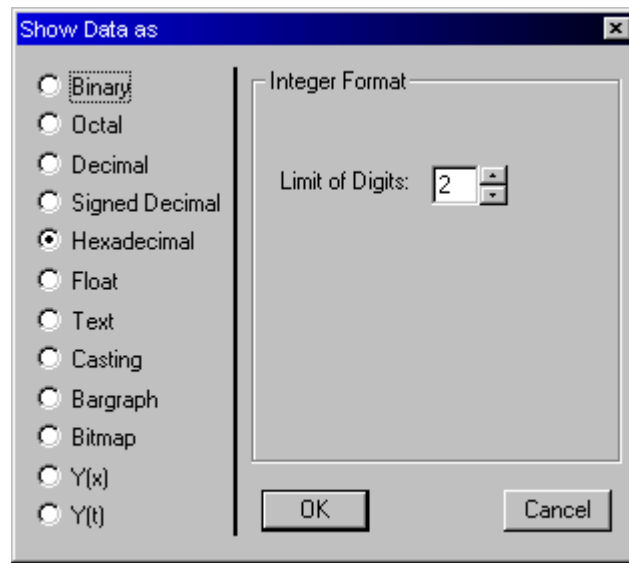
The Data Item style Hexadecimal displays a data item in hexadecimal format:

```

■      ■      ■
f0 79 91 53 54 3b 5f
5b 5e 55 55 58 4e 62
5d 50 87 7a 46 82 4f
■ 65 6d 5c 78 49 65 75 ■
6c 5a 68 6c 7e 78 4e
77 81 70 67 7c
■      ■      ■

```

This display style is similar to the Octal and Binary styles in that the value will be zero padded to the limit specified. The Data Item style dialog box for Hexadecimal displays looks as follows:



The limit of digits sets the number of digits to display per item. If the item can not be displayed within this limit the display will show as hash characters (#####). The default value of the limit corresponds to the bit length of the item selected. The value will be zero padded to the left to the limit of digits specified.

3.5.1.4.2.9 Octal

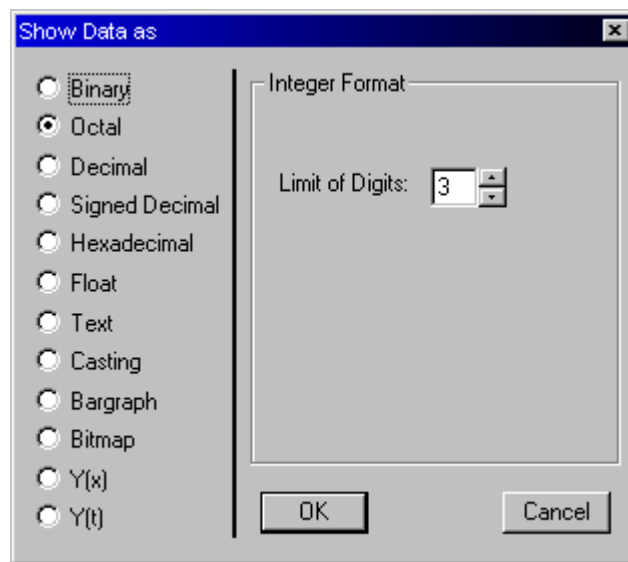
The Data Item style Octal displays a data item in octal format:

```

■      ■      ■
 341 127 137 101 164 212 167
 223 171 132 134 114 124 201
 216 133 152 114 110 127 076
■      ■      ■
146 063 143 117 103 162 111
141 143 157 101 202 167 163
166 141 113 152 203
■      ■      ■

```

The Data Item style dialog box for Octal displays looks as follows:



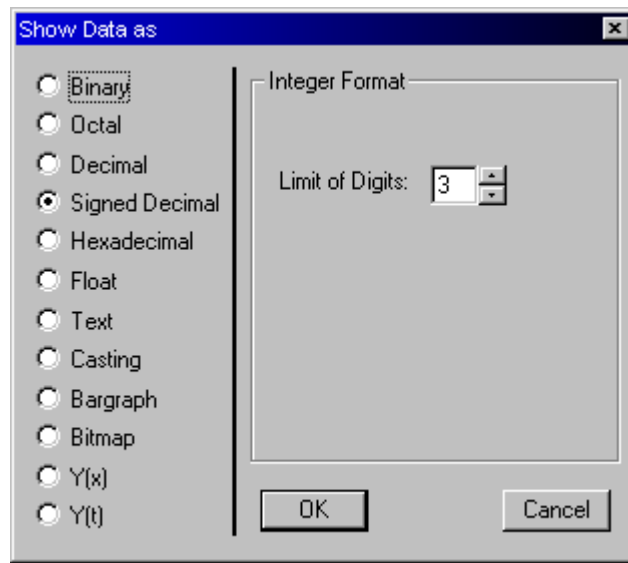
The limit of digits sets the number of digits to display per item. If the item can not be displayed within this limit the display will show as hash characters (#####). The default value of the limit corresponds to the bit length of the item selected. The value will be zero padded to the left to the limit of digits specified.

3.5.1.4.2.10 Signed Decimal

The Data Item style Signed Decimal displays a data item in signed decimal format:

■	-88	85	99	■	84	97	105	109	■
	98	82	84		115	116	100	89	
■	97	84	81		117	111	109	60	■
	96	95	91		101	54	64	104	
	70	97	76		125	99	-107	98	
■	92	88	108	■	112	85			■

GSEOS does not assign data types to the data items specified in the block definition file. That is all values are raw bit patterns. In order to display an item as signed it will be interpreted as if the most significant bit indicates the sign bit. The Data Item style dialog box for the Signed Decimal display looks as follows:



The limit of digits sets the number of digits to display per item. This does not include the leading negative sign. If the item can not be displayed within this limit the display will show as hash characters (#####). The default value of the limit corresponds to the bit length of the item selected. The value will be padded with spaces to the left to the limit of digits specified.

3.5.1.4.2.11 Text

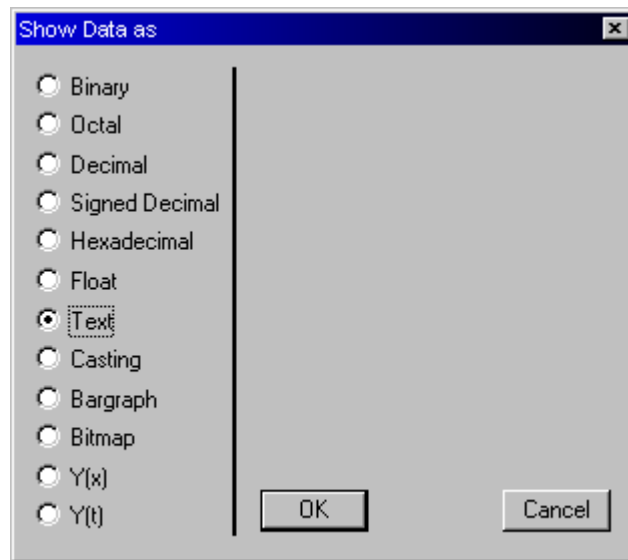
The Data Item style Text displays a data item as ASCII text. It does not interpret carriage return or line feed or other control characters. Only a single line can be displayed at a time. In order to display multiple lines you have to place multiple items.

```

■      This is data in text format.      ■
■                                         ■

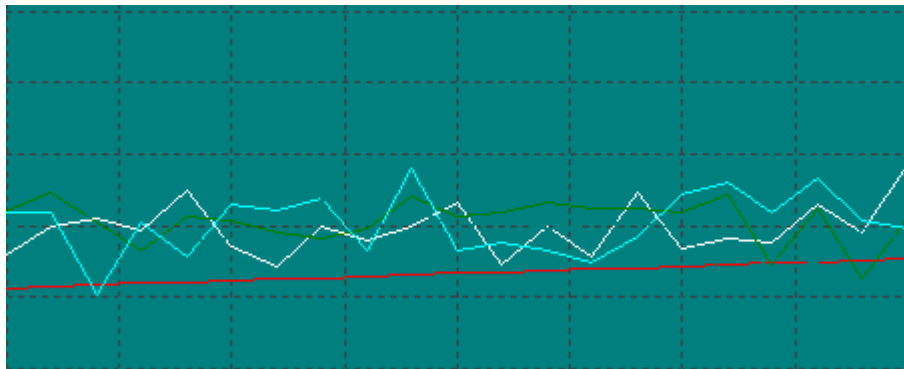
```

The Data Item style dialog box for the Text display looks as follows:



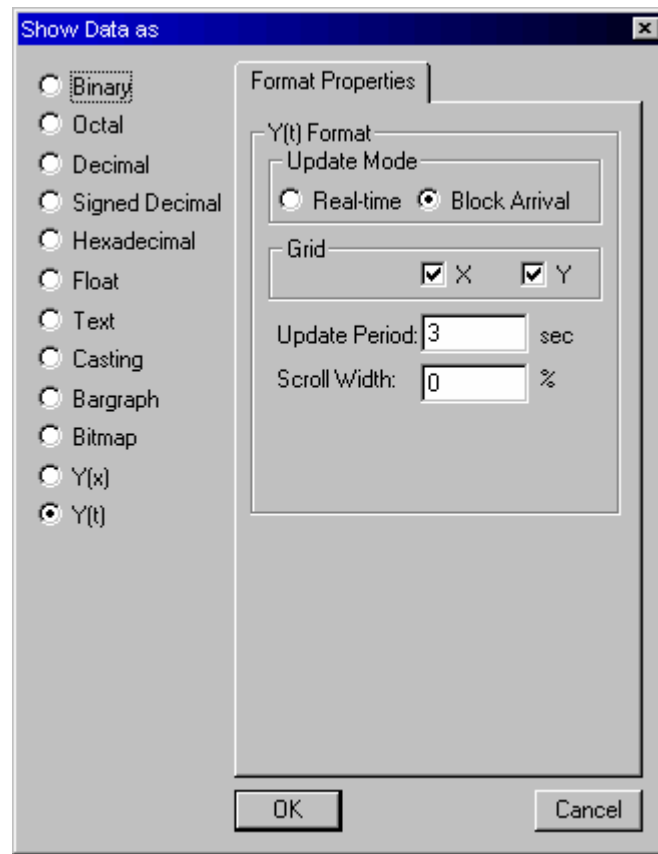
3.5.1.4.2.12 Y(t)

The Data Item style Y(t) displays a data item as a strip chart, that is the data value is displayed over time. The following picture illustrates a typical Y(t) display:



The Y(t) format can display up to five graphs. If you select an array item and choose to display five or more elements the first five elements will be charted in the color specified with the color dialog.

The Data Item style dialog box for Y(t) displays looks as follows:



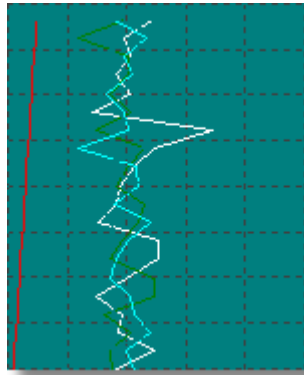
You can configure the update mode to be either Real-time or Block Arrival. In Real-time mode the display update is triggered by clock time. The interval to poll the data is specified in the Update Period field. You can specify fractions of a second.

Typically block arrivals are periodic. If your data is generated on a schedule and you know what the time base of the data is you can use Block Arrival mode. In Block Arrival mode a new value is plotted each time a block arrives. You can specify in the Update Period what the interval between your arrivals is. This way you will get a correct time base. The x-dimension of the Range determines the time interval that is displayed. Say you specify Block Arrival and your period is 2s and that's what you specify in the Update Period field. If you select a x-Range of 0 .. 200, you will see 100 data points on the display which equals 200s. The x-range is specified in seconds.

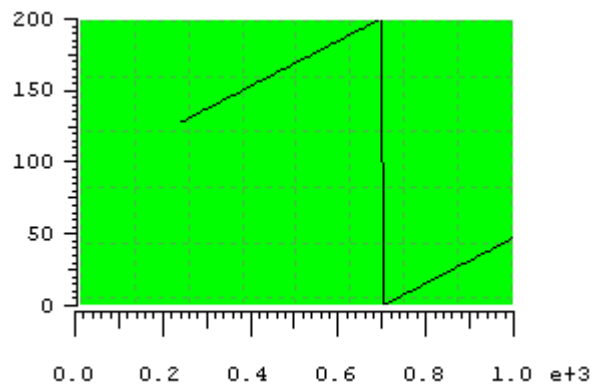
The y-range determines the mapping of the values to be displayed. By default the y-range is 0 .. 1, so you most likely have to adjust it to fit your data accordingly.

The scroll width determines how much of the x-range is displayed at least at any given time. Say you specify 50%, the display will scroll back 50% as soon as it reaches the end of the display. Scroll width 0% equals smooth scrolling but incurs the most overhead. If you have very fast updating data items you might want to at least allow for about 25% scroll width to keep the data update efficient. Every time the display needs to be scrolled the entire display needs to be rendered, whereas if no scrolling is necessary only the according data line needs to be drawn.

As with other graphical displays the Orientation dialog allows you to configure different orientations. The most appropriate for a Y(t) display is probably the 1-L orientation displayed above where the origin is in the lower left corner. The picture below shows the same display with right hand orientation.



The check boxes for the x- and y-Grid let you turn on/off the respective grid. To make the display range visible a scale object can be attached to a $Y(t)$ scale. This will result in a display similar to the one below.



3.5.1.4.2.13 $Y(x)$

3.5.1.4.3 Line

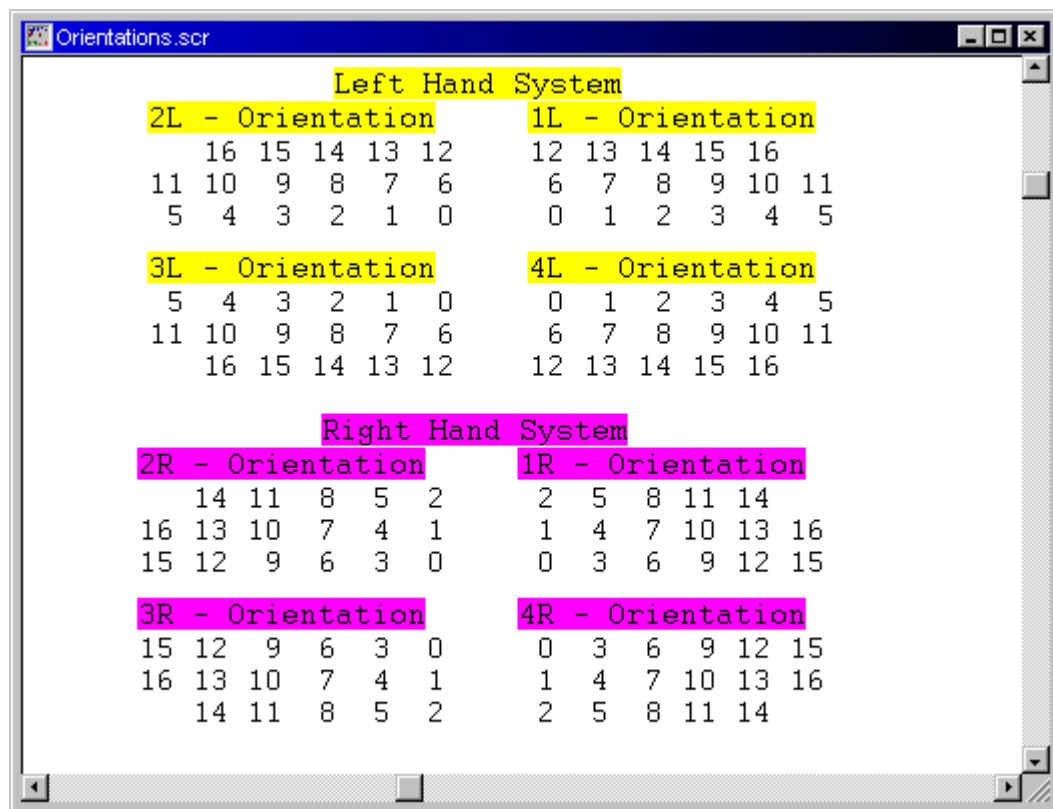
The Line style modifies the line width of static graphic objects like Line, Rectangle, Ellipse and so on.

You can change the line width with the following dialog box:



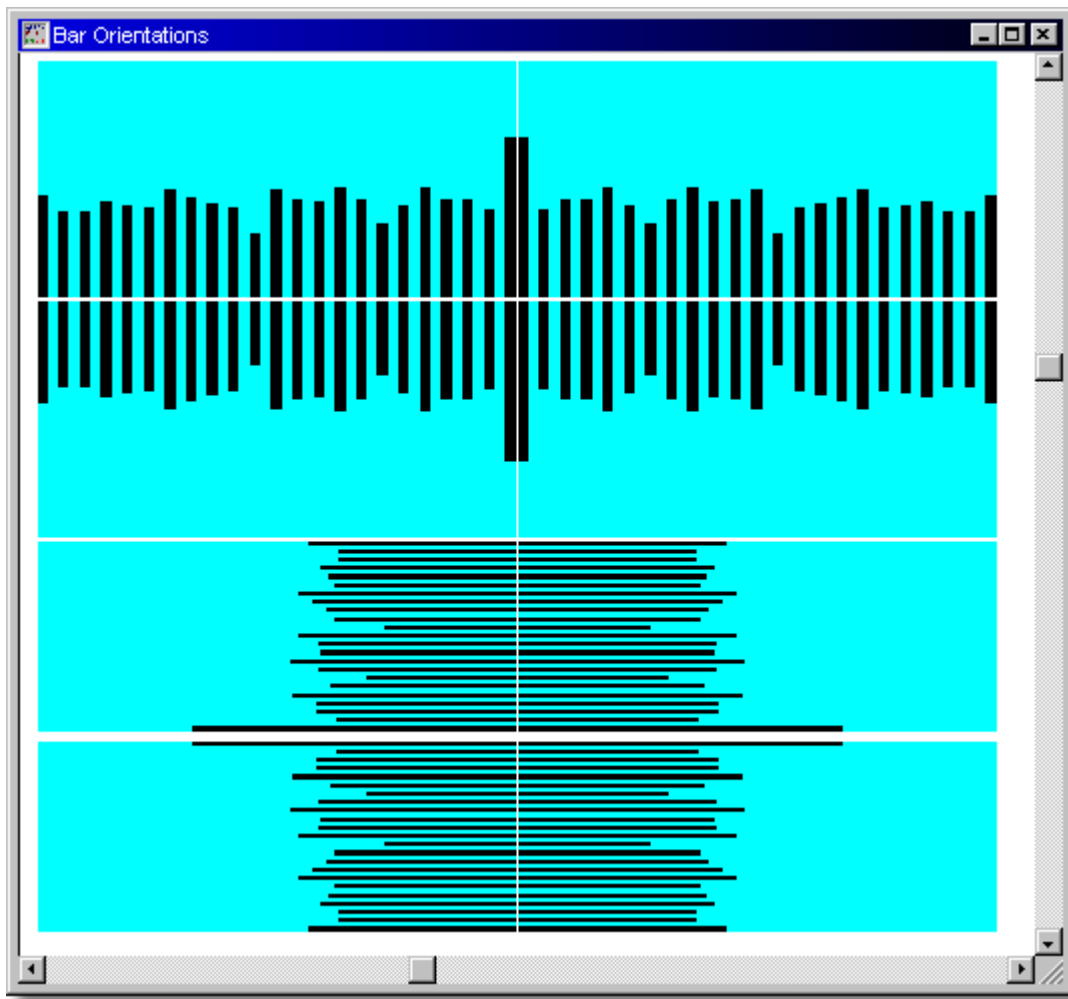
3.5.1.4.4 Orientation

The Orientation style modifies the way the data is displayed. Typically text items are displayed from left to right and top to bottom (this is called 4th quadrant left hand system). There is a total of eight different orientations available. The following example will demonstrate the different orientations. The data in the example is an array item with dimension 17 displayed in all possible orientations:



The default orientation is 4L which is fine for text. For graphical representations like bargraphs, $y(t)$, and bitmaps the origin is usually in the lower left corner which is the 1L orientation. You might want to adjust the orientation for graphical displays. If you use any non standard orientation for any of the items you display you might want to put some text on the screen that indicates on how to interpret the data.

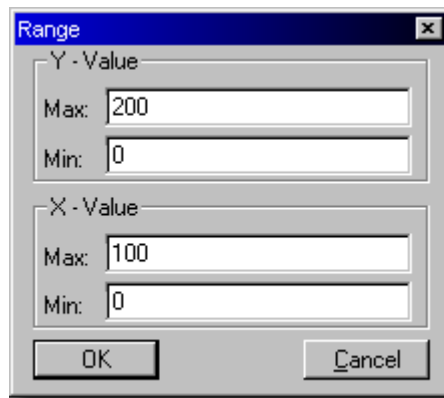
The picture below shows a bargraph display with all possible orientations:



3.5.1.4.5 Range

The Range style modifies the range settings of the display object. Ranges are used for graphical objects like Bargraph, $y(t)$, $y(x)$, and Bitmap. For two dimensional items like Bitmap graphs (2D histograms typically) , $y(t)$, and $y(x)$ both dimensions are used, for Bargraphs only the y dimension is used.

You can change the range settings with the following dialog box:



Bargraph

For Bargraph displays the y dimension determine the minimum and maximum values of the Bargraph. Any values smaller then Min or greater then Max are clipped. When displaying a Bargraph you want to make sure the y range is set properly so the graph displays the range of values of interest.

y(t)

For the y(t) displays both dimensions are used. The y dimension maps the value between the Min and Max limits, the x dimension determines the time base. For the time base only the difference between Min and Max matters. The unit of the time base is 1sec.

Bitmap

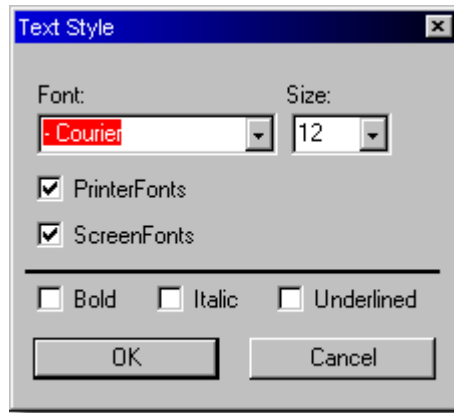
The bitmap display is a 2D display. The one dimensional vector item is mapped into the two dimensional bitmap by means of the x-dimension of the range. Let's assume you want to display a 128 x 64 pixel bitmap the source array item needs to have the dimension of $128 \times 64 = 8192$. Now, when you display the item as a bitmap you have to indicate how wide the image is, that is how many pixels one line holds, this is what is specified by the x-range.

The y-range defines the 'intensity'. The y-range is mapped to the color scheme selected. Say your values are in the range 0 - 10 and you want to map those to the full range of colors available you would set the y-range to 0 - 10.

3.5.1.4.6 Text

The Text style modifies the font of the object. Only numerical and textual objects like Data Item, Conversion Function and so on are effected by this style.

You can change the font settings with the following dialog box:



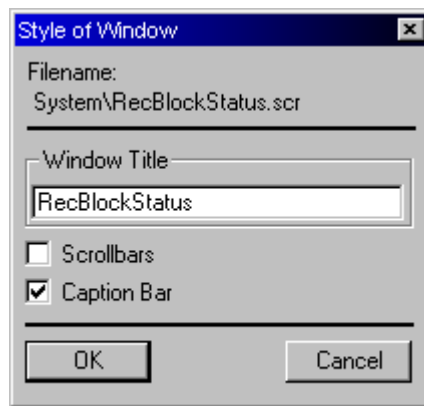
The font drop down list box allows you to select the font. You can choose to display printer fonts or screen fonts or both. Screen fonts are prepended with a dash '-' character.

Printer fonts are guaranteed to render the same on the print output as on the screen. However, due to this property these are mostly true type fonts and therefore slow to display on the screen. If printing the screen is not the main purpose you will achieve faster display with the screen fonts. Especially fixed pitch fonts are to be preferred for system performance. If you select a screen font the printout may be slightly out of alignment.

3.5.1.4.7 Window

The Window style sets properties for the entire screen window as opposed to an individual item on the screen. This dialog allows you to change the window title and controls the display of scroll bars and the window caption.

The following image shows the Window style dialog:



The window title is the text that gets displayed in the caption bar or the window (if any). The Scrollbars check box allows you to turn scroll bars on or off. The Caption Bar check box controls if a caption bar is displayed. If you turn the caption bar of a window off the window can not be moved any longer on its desktop page. This can be useful if you want to create a desktop page that contains various groups but don't want to incur the overhead of the window frames. You would simply place the windows and turn the

caption bars off. The following image shows a desktop page with four screens without caption bars. Note that the windows can not be repositioned until the caption bar is reenabled.



3.5.1.4.7.1 Snapshot



3.5.2 Placing objects

When a screen window is active the GSEOS merges the Qlook menus into the main menu. The toolbar also displays some of the most often used commands to place objects.

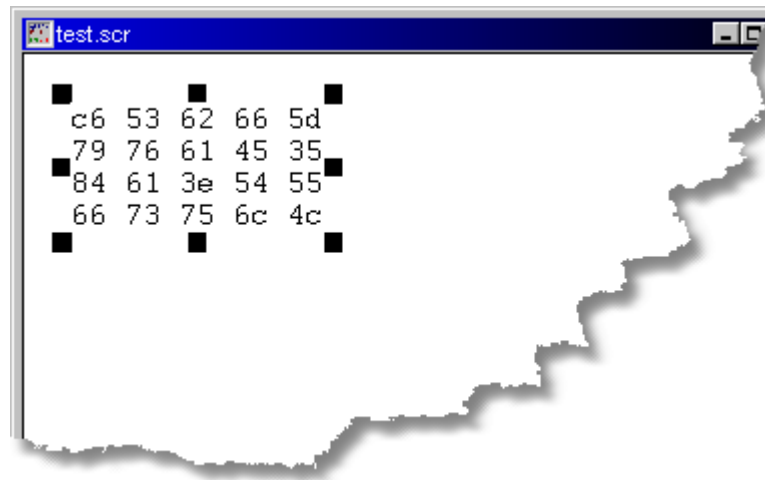


Placing an object involves several steps:

- Selecting a drawing tool
- Selecting a drawing region
- Selecting object properties
- Adjust display style

3.5.2.1 Adjust Display Style

The newly created object is placed with default attributes. The Style menu allows you to change these attributes. To change an objects size, position, or display attributes it needs to be selected. A left mouse button click on the object will select the object and this selection is indicated by chooser pads on the corners and edges of the object.



By double-clicking on a selected object you can invoke the properties dialog for this object and change the objects properties. The chapter on the Draw menu explains the various object properties in detail.

To move an object you can simply drag it with the move, that is you click within the object, hold the left mouse button down move the object to its destination and release the mouse.

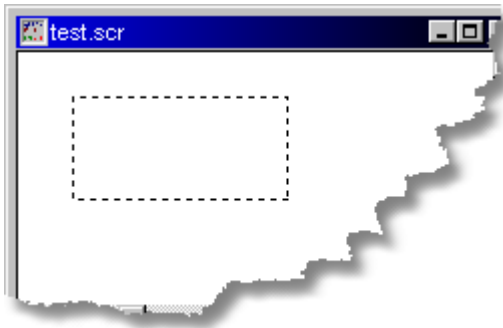
Alternatively you can use the cursor keys to move the object on a pixel-by-pixel (if a grid is enabled then the object will be move on grid steps) basis.

To size the object move the mouse on one of the choose pads and drag the mouse to the desired size of the object. Some objects only support discrete sizes, in this case the objects size will snap back to the closest allowed dimension.

By right-clicking on the object you will open the Style menu that will allow you to set the style of various attributes on the selected object.

3.5.2.2 Selecting a Drawing Region






















The second step in placing a new object on a screen is selecting the area of the screen the object will be placed. Once you move the mouse over the screen window with a drawing tool selected the tool will be indicated by the mouse cursor. To select the drawing region where the object will be placed click the left mouse button, hold it down and move the mouse (drag the mouse) to the destination. You will see a selection rectangle indication your drawing region. To finish the selection release the left mouse button at the destination. Don't worry if the size or placement of the object is not optimal, you can move and resize the object at any time after you created it. See below for a picture of the selection rectangle:



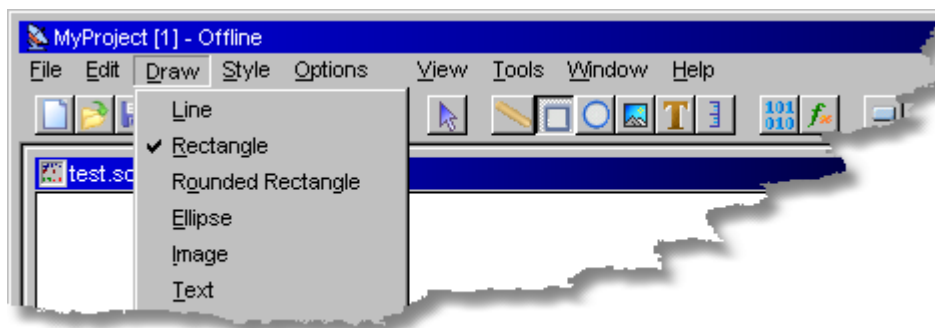
Once you release the mouse button the selection of the drawing region is complete and, depending on the drawing tool, a properties dialog will allow you to specify properties of the object you are about to place.

3.5.2.3 Selecting a Drawing Tool

You can either use the Draw menu or select one of the drawing tools from the tool bar:

Drawing Tool	Toolbar Button	Cursor	Description
Select			Selects one or multiple screen objects.
Line			Draws a line.
Rectangle			Draws a rectangle.
Rounded Rectangle			Draws a rounded rectangle.
Ellipse			Draws an ellipse.
Image			Draws an image from a bitmap file.
Text			Draws static text.
Scale			Draws a scale.
Data Item			Draws dynamic data items as defined in your block definition file.
Conversion Function			Draws the result of a mathematical Conversion Function of data items.
Command Button			Draws a command button.

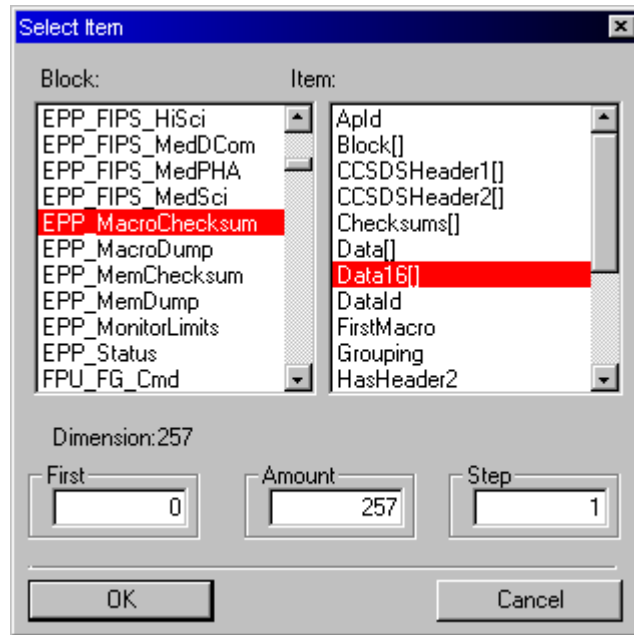
Once you selected a drawing tool the according submenu in the Draw menu will be checked and the toolbar button will be depressed. The following picture shows the drawing tool 'Rectangle' selected:



3.5.2.4 Selecting Object Properties

For most of the simpler drawing tools like line, rectangle, etc. you are done with the object. You can change the size and position or attributes of the object at any time. Refer to the next chapter on how to adjust the display style.

However, for more complex objects you will need to specify object properties. These properties vary from one drawing tool to the next. For a detailed description of all the properties of the various drawing tools refer to the Draw section in the Menus chapter. Here we want to demonstrate how to place a data item. Once you complete the drag operation and release the left mouse button the Select Item dialog specific to the drawing tool Data Item will pop up:

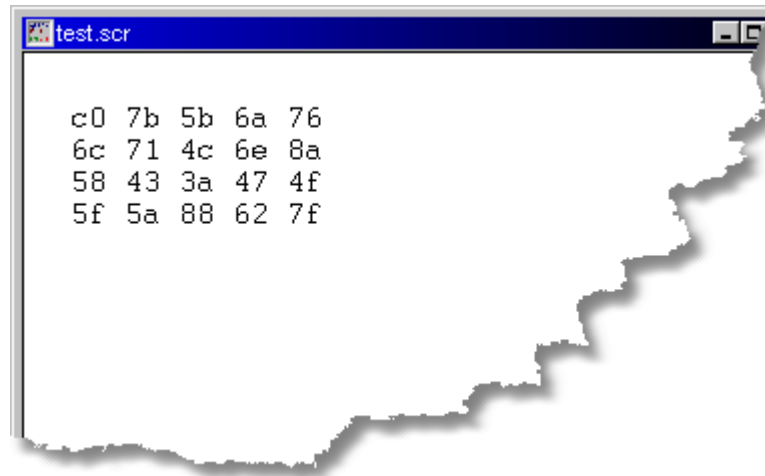


The Select Item dialog allows you to select the data item you want to display. On the left hand side you will see all the blocks defined in your various block definition files as well as the system blocks. Select the block that contains the data item of interest. The right hand list box gets populated with the items contained in the block selected on the left. Now choose the item to display. If the data item you select is an array item (indicated by the square brackets) the First, Amount, and Step fields are enabled and you can choose the part of the array you want to display.

Note:

Be aware that if the data item is very large and you choose a textual presentation the screen may not be able to hold the entire item, in this case it might be a good idea to start out with a subset of the data and increase it later.

Once you select Ok and your selection is valid the new data item will be displayed with the default data style like in the image below. You can adjust the data style with the Data Style menu.



3.6 The Command Dialog

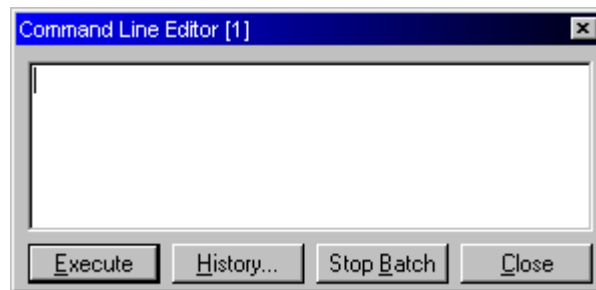
The Command dialog, similarly to the console window, allows you to interact with the built-in Python interpreter. You can issue commands, execute Python scripts, import modules, examine block data, etc.

As opposed to the console window no Python output is written to the command dialog. This dialog is strictly for issuing commands. If you happen to load a script that generates periodic error messages or that writes continuously to the console window the console window is rendered useless for input. In this case you can terminate the offending script or command and continue working with the console window. Besides issuing commands you can also stop batch files and access a command history from the command dialog.

To execute the command you can either use the Enter key or click on the Execute button. If you retrieve a command from the history it will paste the text into the input window, you still have to issue the command.

Note

The command dialog is a line editor so you can only issue simple one line Python commands. For any commands but simple statements you should use the console window.



You can open the command dialog from the View menu, the F7 hotkey, or the  toolbar

button.

3.7 The Console Window

The console window allows you to interact with the built-in Python interpreter. You can issue commands, execute Python scripts, import modules, examine block data, etc.

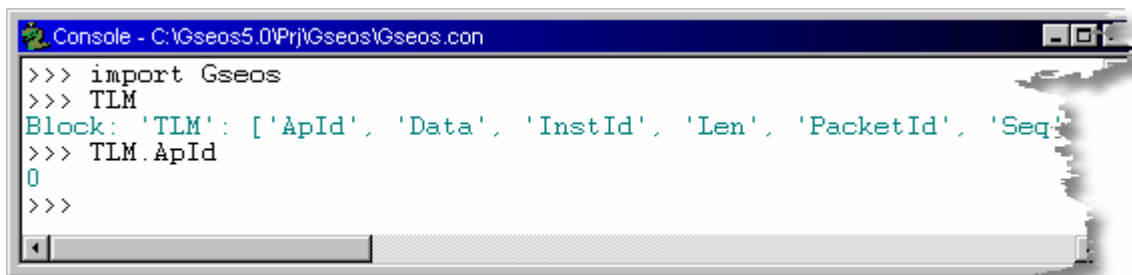
The console window is not a regular MDI child window like for example the screen or log windows. Instead it is a floating window that will be in the foreground no matter what desktop page is selected. The window can be easily displayed and hidden with the View menu, the Console window tool bar button or the F9 hotkey. The console window is associated with a file and all output written to the console window is also logged to this file when the console window is closed or the application is shut down.

The Python icon on the tool bar allows you to open/close the console window:



The file name of the Console window can be specified in the gseos.ini configuration file. The entry FileName in the [Console] section specifies the file the console window contents will be written to. If you do not supply this file name it defaults to ProjectName[Instance].con, where the ProjectName is the name of your project as configured in the [Project] section. When the system restarts the console file from the previous session will be restored and any new text will be appended to the file. The maximum size of this file can be configured in the [Console] section as well. The console file will not grow beyond this size and the oldest content will be discarded. The default value is 512KB.

The picture below shows a typical console window:



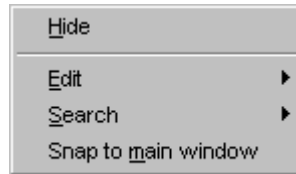
The >>> is the Python command prompt and indicates that the system is ready for input. The caption bar indicates the current console file associated with the console window.

In case the console window is not opened or iconized and new data is written to the console window this activity is flagged to the user by flashing the caption bar of the console window if it is iconized and by highlighting the toolbar button of the Console window:



Context Menu:

A right mouse button click will open the console context menu:



The Hide command will close the console window and automatically save the current content to the associated console file. This is an alternative to the toolbar button, the View menu, or the hotkey F9.

The Edit menu allows you to copy and paste and perform general edit functions in the console window.

The Search menu opens a dialog to search or replace text in the console window.

The Snap to main window function will reposition the console window to the bottom of the GSEOS main window. If the main window is too close to the bottom edge of the screen it will move it up so it will fit on the screen in its entirety, partially covering the GSEOS main window. The console window is not permanently docked to the main window and a move of either window will not move the other.

Command History:

The console window provides two different mechanisms to retrieve previously entered commands. If you position the cursor on any but the current input line and press Enter that line will be copied to the current input line. There you can modify the command as necessary and issue it.

If you have a lot of output in the console window the commands you entered previously may be scrolled out of sight or burried somewhere in the output. In this case you can use the ESC history. If you type the first letters of any command and press the ESC key a menu with all matching previously typed commands pops up. Here you can select the command you are interested in. Upon selection the command is copied into the current input line and you can modify and issue the command from there.



The above example show the history menu on typing: 'im' + ESC.

Programmatic Access:

You can programmatically hook the console output. The system generates the ConsoleOut block (if it is defined in the system.blk block definition file) every time output is written to the console window. Here the block definition:

```
ConsoleOut INTEL
```



```

{
    Len                ,,, 32;
    Truncated          ,,, 8;
    WindowState        ,,, 8;
    Text               [1024] 0,,, 8;
}

```

You can set the length of the Text item to any size appropriate, the default is 1024. If the text that is inserted into the Console window exceeds this length the content in the Text item is truncated and the field Truncated is set to 1. The Len field indicates the number of valid characters in the Text field. To detect if the console window toolbar button has been flagged you can check the WindowState field. The WindowState field is set to 0 if the window is visible, to 1 if it is closed and to 2 if it is open but iconized. So if the WindowState field is not equal to 0 the console window toolbar button is flagged. This block can be used to write a simple Monitor to notify the user of new activity in the console window. The sample below implements a monitor that plays a sound to notify the user of new console activity when the window is not visible or iconized:

```

import Gseos
import GseosCmd
import Monitor
from __main__ import ConsoleOut

# -----
-- #
# - Monitor the ConsoleOut block. If we see one, play a sound if the
- #
# - window is not visible or iconized.
- #
# -----
-- #
def MonConsoleOut(oBlock):
    if oBlock.WindowState <> 0:
        GseosCmd.sound('boing.wav')

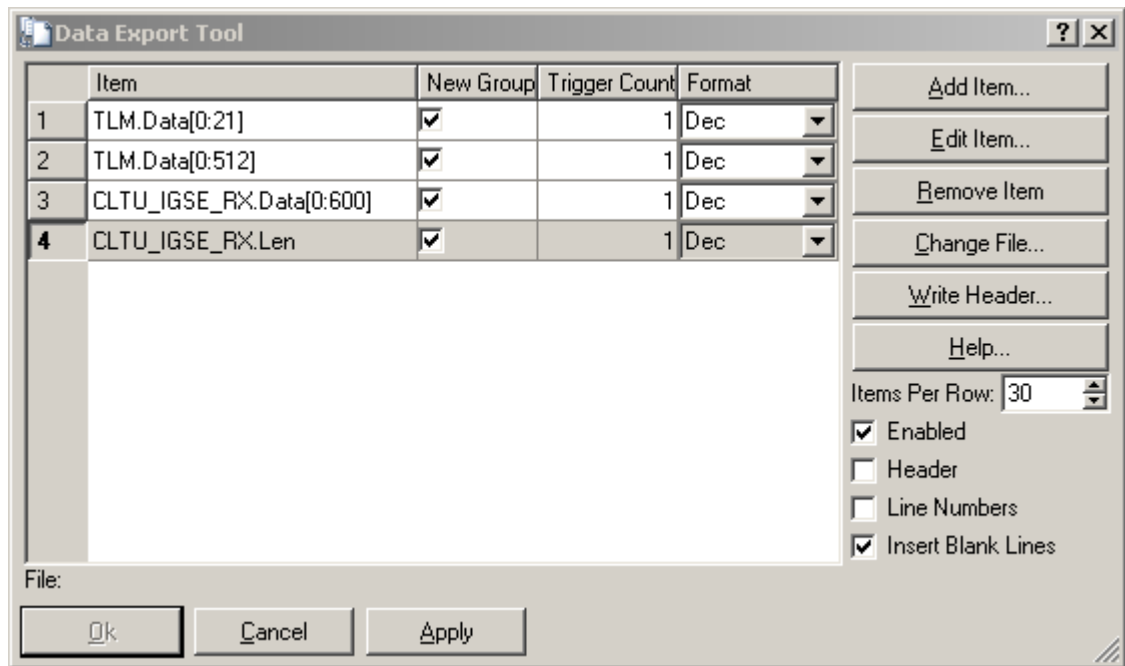
ConsoleOut.Monitors.append(Monitor.Monitor('Console Activity',
MonConsoleOut))

```

For more information about programmatic access to the console window check the GseosConsole module.

3.8 The Data Export Dialog

The Data Export dialog allows you to export GSEOS data to a flat ASCII file. You can configure multiple items to be written to the output file. The following figure shows the Data Export dialog.



You can display the Data Export dialog from the main menu: Tools\DataExport. The items to be exported can be added with the 'Add Item...' button and are displayed on the left hand side of the dialog. Items can be scalar or array items of any dimension. Conversion functions can also be selected to be exported. If you want to select Conversion functions please make sure the proper conversions are loaded. You can group export items which means they are displayed on the same line (except when the line wraps around). This makes for a more compact export.

Enabled

The enabled check box determines if the data export tool is active or inactive. If the Enabled checkbox is not checked not data is written to the output file.

File Header

In order to associate the data with the item that are being output you can check the 'Header' checkbox. This will write a file header every time the export file gets changed or the export settings are modified. Clicking the 'Write Header...' button will also write a header to the file. See below for a sample of the file header:

```
## GSEOS Data Export
## =====
##
## The following items are exported with the corresponding line numbers:
## Line #0: TLM.Len
##           TLM.PacketId
##
## Line #1: Clock.DayOfYear
##
## Line #2: CLTU_IGSE_RX.Len
##
## Line #3: CMDSTRING.abData[0:511]
##
```

```
## Line #4: ConsoleOut.Len  
##
```

The header displays the output line number assignment. If the 'Line Numbers' check box is checked the data is prefixed with the line number per output line like in the following example:

```
#0: 0, 0  
#0: 0, 0  
#4: 4  
#0: 0, 0  
  
#0: 0, 0  
  
#4: 4
```

Line Numbers

You can also choose not to have the line numbers printed by unchecking the 'Line Numbers' check box.

Insert Blank Line

For easier import into Excel you can uncheck the 'Blank Line' checkbox. If this check box is checked a blank line will be inserted after every output line.

Items Per Row

To restrict the line length the 'Items Per Row' setting allows you to select any number of elements per line. An output line as indicated by the file header might be printed on multiple physical lines depending on the 'Items Per Row' setting.

New Group

If the New Group check box is checked a new export group is started. An export group arranges the items on a line until the maximum line length is reached and then wraps the line. Depending on the setting of 'Items Per Row' the number of physical lines may be more than one per output line. However, each trigger set in a group will trigger the entire group.

Trigger

The trigger setting allows you to control when each output line is printed. Each time a block arrives the trigger counter is decremented, if it reaches zero the output line is printed. There can be multiple triggers per output line. If you don't have any triggers set (the trigger count is 0) the line will not be printed.

Format

Currently there are two output formats: Decimal and Hexadecimal. When decimal is selected the necessary number of digits is displayed. In case of conversion functions a floating point value is printed. If Hex is selected conversion function results will be clipped to integer and displayed as 32-bit hex values. For regular items the number of digits is determined by the size of the item.

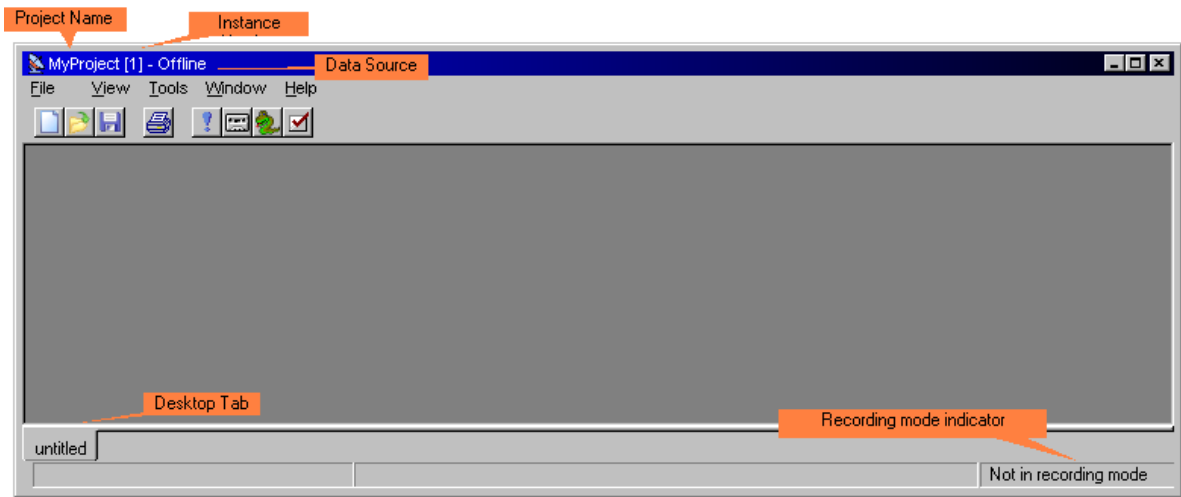
Order of Export Items

You can change the order of the export items by holding down the Ctrl key and dragging the item row number(s) you want to move to a different location. The following picture shows the row numbers to drag.

Data Export Tool	
	Item
1	TLM.Len
2	TLM.PacketId
3	Clock.DayOfYear
4	CLTU_IGSE_RX.Len
5	CMDSTRING.abbyData
6	ConsoleOut.Len

3.9 The GSEOS Main Window

GSEOS is a Multiple Document Interface (MDI) application. Data is organized in child windows within the main application window. Besides just being an MDI application GSEOS also organizes its windows on Desktop tabs. This allows you to configure various window layouts and to easily switch between them. Fig. 1 shows the empty main application.



In the caption bar of the main window you can see: Mimi [1] - Offline. This is the title of the main window and consists of three fields:

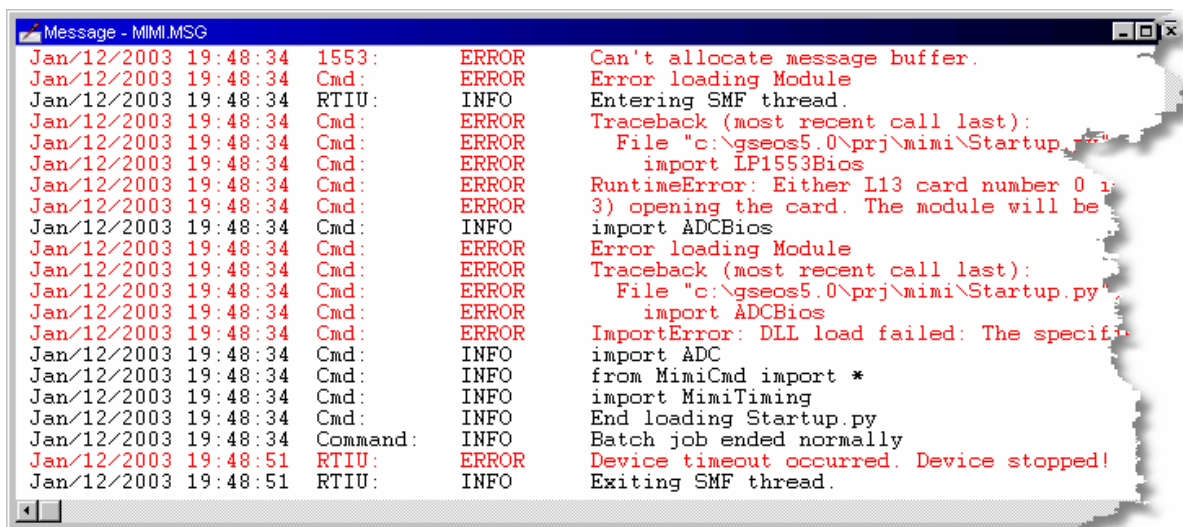
- MyProject: The project name as configured in gseos.ini.
- [1] The number of the current instance. The GSEOS application can be run as multiple concurrent instances. You can specify the instance number on the command line, the \In switch will indicated the instance number. This allows you to configure different application settings for different instances of the application.
- Offline The currently active data source. The system can have different data sources, but only one active one at a given time. These data sources are Offline, Bios (Spacecraft Simulator), Net, Recorder, or your own custom data source.

On the bottom of the application you can see the currently active desktop tab which is 'Untitled' initially. The recorder mode is indicated in the lower right of the status bar.

3.10 The Message Window

The message window is the means for GSEOS to post system message, certain error messages and general purpose messages. You can also post messages to the message window. However, log windows may be the more appropriate place to post user messages.

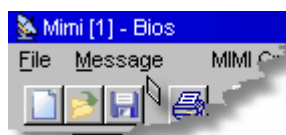
The following snapshot depicts the message window:



The message window is a regular MDI child window and 'lives' on a particular desktop page. In general it is a good idea to reserve a page for the message window so it can be easily located. There can be only one message window open at a time. A message window is associated with a file and all contents are written to the message file. The file will not be truncated and grow over time. You might want to purge it from time to time.

The default display of the message window prints the data and time, the source that generated the message, an error status that can either be INFO, WARNING, or ERROR, and the message itself. Messages are limited to 80 characters and will be truncated after 80 characters and continued on a new line.

When the message window is active the GSEOS main menu changes and offers an additional drop down menu Message:



The Message menu allows you to clear the contents of the message window and to configure the display. Any of the fields, date, time, source, and type can be enabled or disabled.

When no message window is open you can create a new one by selecting the File/New menu and specifying file type Message.

Once a message window is open this choice disappears from the File/New menu.

All messages that get written to the message window get routed through the system block MESSAGE.

Below is the definition of the MESSAGE block that is defined in System.blk:

```
MESSAGE      {MSG_TYPE      , , , 32;
               App_Name      [10] 0 , , , 8;
               Message        [80] 0 , , , 8}
```

You can intercept this block with a GSEOS Monitor or Decoder and perform the appropriate action.

Programmatic Access:

The GseosCmd module offers the method: GseosCmd.msg() which you can call to post your own messages to the message window. This is useful for short, infrequent messages like exceptions or errors you want to report. For more extensive logging a log window might be more appropriate.

It is also conceivable to directly generate a MESSAGE block from a Python script. For more information on how to generate GSEOS data blocks from Python scripts refer to the `__main__` module.

3.11 The Recorder Dialog








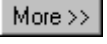


GSEOS is capable of real time data logging to standard Windows storage devices (hard disks, optical disks, etc.). These files can be played back for data evaluation purposes. In playback mode the Recorder module is the active data source and will provide all input data. The Recorder allows to record/play back any block that is defined in the system. The data is written in sequential manner only and no backward references are required, this allows the usage of strictly sequential writeable media like tape drives. The file format is detailed in the Recorder File Format chapter.

The user interface for the Recorder is the Recorder dialog box shown below:



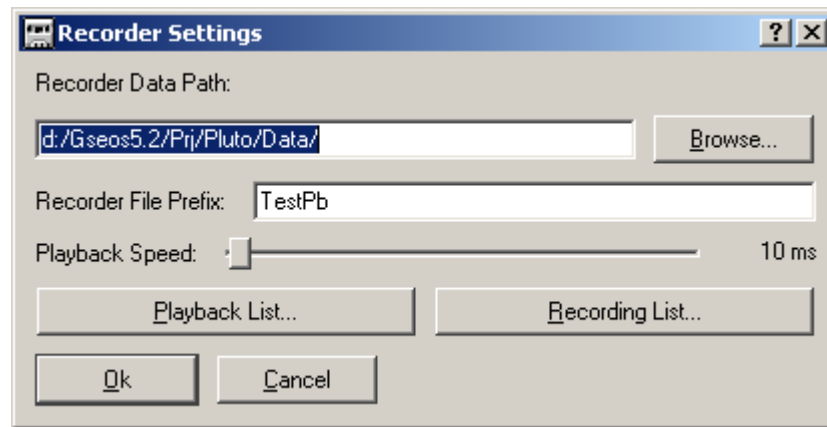
The Recorder dialog can be invoked from the View menu or with the F8 hotkey. The caption bar indicates the instance number of the currently running GSEOS instance.

The recorder can be controlled with the following buttons:

	Playback single step reverse	Play back one data block in reverse direction. Only blocks on the playback list will be played back.
	Playback fast reverse	Start reverse fast playback. Only blocks on the playback list will be played back. The playback speed can be adjusted with the Settings Dialog.
	Stop	Stops the fast playback modes or the recording mode, depending on which is active.
	Record	Start recording of blocks. Only blocks on the record list will be recorded. When in recording mode the available disk space will be displayed in the status bar.
	Playback single step	Play back one data block.
	Playback fast	Start fast playback mode. Only blocks on the playback list will be played back. The playback speed can be adjusted with the Settings Dialog.
	Recorder settings	This opens the Recorder Settings dialog.
	Expand Recorder dialog	The More button expands the Recorder dialog and displays additional status information while recording or playing back.
	Scan mode	In scan mode the recorder moves through the blocks like in regular playback mode, except without generating any data blocks.
	File mode	The file mode checkbox selects between automatic and single file mode. In automatic file mode the recorder automatically generates a file name for the data. The file will be closed once it reaches a threshold that can be set in the gseos.ini file. A new file will be opened automatically and the recording continued. In single file mode you have to specify the file name to record to and no automatic file switch is performed.

Recorder Settings Dialog

The Recorder Settings dialog allows you to specify several recorder options:



The data path edit box displays the current path for files generated in automatic mode. The path can be changed and will be saved in the gseos.ini file in the [Recorder] section. The prefix is the string that will be prepended to all automatically generated recorder blocks.

The playback speed can be set to either fast which is approximately every 50ms or to slow which is approximately once a second.

The playback and recording list buttons open dialog windows to add or remove blocks from the playback or recording list respectively.

Note

The Recorder writes the GSEOS data blocks as binary data to the record medium. This means the interpretation of the data is left to the block definition. Therefore these two files have always be synchronized. Say you record data, later change the block definitions of the already recorded block in a way that is not compatible to the old format. E.g. new item has been inserted somewhere in the block. This will cause the recorder to generate 'wrong' data on playback since now the interpretation of the data has changed over the way it was interpreted when the data was recorded. You should always back up your block definition files (and probably your monitor and decoder files as well) together with your recorder data files so you have a coherent snapshot of the system at a given time. A good version control system like CVS or Visual SourceSafe is highly recommended to support this task.

Programmatic Access:

The GseosRecorder module exports functions to perform most of the actions that can be accomplished with the Recorder dialog. Please refer to the GseosRecorder module for more details on how to script the recorder module.

Part

IV

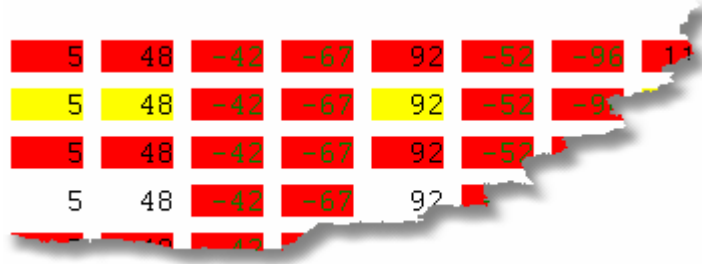
4 GSEOS Reference

4.1 Configuration Files

4.1.1 Alarm Limit Files (*.alarm)

You can configure alarm limits in one or more alarm definition files. The alarm definitions are stored in flat ASCII files with the .alarm extension. You can specify the alarm definition files to load in the Load entry of the [Config] section of the gseos.ini file.

The following snapshot shows data items that use the alarm feature:



Below is a sample alarm definition file demonstrate the syntax:

```
# -----
#
# - Alarm sample definition. -
#
# -----
#
Alarm MDS_RESTART_REQUEST,      0,      ,      ,      1
Alarm MDS_TURNOFF_REQUEST,      0,      ,      ,      1
Alarm MDS_APDOOR_ST,            0.5,      ,      2.5,      3.5
Alarm MDS_COUNT_RATE,           0,      ,      12000,      15000
Alarm MDS_HVPS_SET_VOLT_EU,     -4.5,     -0.25,      ,      0
```

An alarm is defined on a single line. The definition has the following syntax:

Alarm Name, Red Low, Yellow Low, Yellow High, Red High

It starts with the keyword: 'Alarm' followed by a unique name for the Alarm and the four comma separated limits for Red Low, Yellow Low, Yellow High, Red High. Not all limits need to be specified. If you only require some of the limits you can just leave the ones not required blank. However, you have to specify the commas.

The numeric value of the limits has to be in order, that is the following rule must be true for all limits specified:

Red Low < Yellow Low < Yellow High < Red High

In order for the alarm file to take effect you have to load it. This can either be done at startup with a Load entry in the gseos.ini file [Config] section or by manually loading the file at runtime. Select the file type 'Alarm Files (*.alarm)' in the File Open dialog box.

You can verify all loaded alarms with the GSEOS Explorer. Once an alarm is loaded you can use it in your screen definition to apply it to a data item.

4.1.2 Block Definition Files (*.blk)

The block definition file defines the bit level structure of all your data products like telemetry data. A block definition is similar to a structure declaration in the 'C' programming language. A block definition consists of a unique block name and a listing of all the data items that comprise this block.

This section will give a detailed description of data block definitions and the format of data block definition files (*.blk).

If you don't specify any block definition files to load in the gseos.ini configuration file a default file will be loaded. The file has to be located in the GSEOS working directory (e.g. in the same directory as the GSEOS.EXE file). The name of the block definition file consists of the project name specified in the gseos.ini file and the extension '.blk'. You can override this by specifying your block definition files in the config section of the gseos.ini file. In general you want to include the system block definitions that can be found in system\system.blk.

Comments can be inserted anywhere in the file. A comment starts with the '#' character (the old notation is the '*' character) and ends at the end of the line.

Block Definition

```
BlockName TypeModifier
{
    Item1Name [Dimension] Gap, StartByte, StartBit, BitLength,
TypeModifier;
    Item2Name [Dimension] Gap, StartByte, StartBit, BitLength,
TypeModifier;
    Item3Name [Dimension] Gap, StartByte, StartBit, BitLength,
TypeModifier;
    Item4Name [Dimension] Gap, StartByte, StartBit, BitLength,
TypeModifier;
    Item5Name [Dimension] Gap, StartByte, StartBit, BitLength,
TypeModifier;
}
```

A block definition starts with a unique block name. The maximum length of the block name is 32 characters. After the block name an optional block type modifier can follow. The two types allowed are MOT (the default) for Motorola (big) endianness or INTEL for Intel (little) endian. Motorola endianness means that the most significant bit resides on the highest address while the Intel endianness is defined with the least significant bit on the highest address. All items defined in this block are handled with the block default that is specified here. You can override individual items if the endianness changes on item boundaries. The maximum size of a block must not exceed 500kB.

Item definition

The item definitions are enclosed in curly brackets '{' '}'. One block may contain any number of item definitions.

Push address

The optional push address operator '(' pushes the address of the item onto a stack. It

can be retrieved through the pop operator ')'. This can be used to access a single address with multiple items, i. e. have multiple names for the same address space. The push and pop operators can be nested to any level.

Item name

An item definition starts with a block unique item name. The maximum length of the item name is 32 characters. The item name can be omitted. If no item name is specified this item can not be accessed but it inserts a gap of its according size in the block definition. This way you can gap areas of no interest without using an absolute addressing mode.

Array specifier

An optional array specifier indicates an array. The dimension of the array, declared in square brackets, is determined by the dimension field. Array elements do not have to be contiguous but may be subdivided by a bit gap of a constant length. The bit gap length must be less than 65500 bits. If a data item is defined as an array and no bit gap is defined, the array elements are contiguous (packed). If the array specifier is omitted the item is assumed to be a scalar.

Start byte

The offset of the beginning of the block for this data item. The first byte in a block has offset 0. If the start byte is omitted the current byte position, contiguous to the previous defined item, is used as start byte position. Using start bytes defines an absolute addressing scheme which gives you less flexibility in changing your block definitions. Defining a start byte allows you to insert an item into the middle of a block definition without changing all the start addresses of the items defined after the inserted one. However, if you specify the start byte the current position is set to this address. It is not necessary to order the item definitions in the order of the start byte. You can use this feature to overlap multiple items. I.e. you can specify the same start byte for multiple items.

Start bit

The first bit covered by the data item. For type INTEL the first bit in a byte is defined to bit 7, the most significant bit. The last bit in a byte is bit 0, the least significant bit. For type MOT the first bit is bit 0, and the last bit is bit 7. If the start bit is omitted the item is packed behind the previous item.

Bit length

The length of the data item in bits. This can be a number from 1 to 32. If the data item was defined as an array, this is the length of one single array element. The bit length must be specified!

Item type modifier

By default the item type is the block type (MOT or INTEL). This default can be overridden for an item by MOT or INTEL.

Pop startaddress

The optional pop address operator ') ' pops an address from the stack. This address has to be pushed onto the stack with the push operator '(' before. It restores the start byte address of the matching pop operation.

Comments

To be more flexible you should omit the start byte and start bit specifiers. The

disadvantage of this approach is that you cannot directly tell at what absolute address an item is positioned. GSEOS allows you to print out a detailed map of your block definitions which contains the absolute byte and bit positions of every data item. You have to create a file of type 'Block Listing' from the File|New menu.

Example

The following block definition shows a simple block with all items on byte boundaries. This block is a GSEOS system block:

```
# -----
#      Status block for dynamic storage allocation
# -----
DSACtrl      INTEL
{
    dwTotal          ,,, 32;
    dwTotalFree      ,,, 32;
    wFreeCnt         ,,, 16;
    wIterations      ,,, 16;
    dwAllocCalls     ,,, 32;
    dwFreeCalls      ,,, 32;
    dwFailCnt        ,,, 32;
}
```

The block listing of the above block shows the absolute offsets of all data items and the endianness of each individual item.

```
=====
Block Name:      DSACtrl
Size:           24 Byte
dNumber of items: 7

Item Name          Offset      Size  Gap    Endian
                   [Byte/Bit] [Bit] [Bit]
-----
dwTotal            0/0        32    -    Intel
dwTotalFree        4/0        32    -    Intel
wFreeCnt           8/0        16    -    Intel
wIterations        10/0       16    -    Intel
dwAllocCalls       12/0        32    -    Intel
dwFreeCalls        16/0        32    -    Intel
dwFailCnt          20/0        32    -    Intel
```

The next example demonstrates the push address and pop address operators. The start address of the item Block is pushed. In this case the start address is 0 (since Block is the first item in the SimTmMode block definition). Before defining the item State the address is popped. This means that the offset of State is 0 and therefore overlaps Block[0]. If we had not used the push and pop operators the byte offset (start byte) of State would have been 128. You can verify the positioning of the items in the block listing below.

```
# -----
#      telemetry mode change control
# -----
```

```

-----
SimTmMode  { (Block      [128] 0      , , , 8;)
              State      , , , 8;
            }

```

```

=====
Block Name:      SimTmMode
Size:            128 Byte
Number of items: 2

```

Item Name	Offset [Byte/Bit]	Size [Bit]	Gap [Bit]	Endian
Block[128]	0/0	8	0	Motorola
State	0/0	8	-	Motorola

The next example shows a block definition with items which are not aligned on byte boundaries. It also uses absolute addressing by specifying the start position of every item.

```

# -----
#
# S/C telemetry inter-experiment data packet
# -----
SimIePkt  {
    Ident          , 0, 7, 16;
    Segmentation   , 2, 7, 2;
    Tag            , 2, 5, 14;
    Length         , 4, 7, 16;
    OBT            [6] 0, 6, 7, 8;
    Data           [2] 0, 12, 7, 16;
    Validity       [2] 15, 12, 7, 1;
    Master          , 12, 6, 4;
    Event          , 14, 6, 4;
    Type           [2] 15, 12, 2, 1;
    x              , 12, 1, 10;
    y              , 14, 1, 10;
}

```

Oftentimes it is necessary to define arrays of structures as opposed to single items. E.g. assume we have an Event block that defines 10 events with each event consisting of two items: Time-of-Flight (ToF) and Mass. Lets further assume that ToF and Mass have a resolution of 14-bit and are packed like this:

```

ToF1
Mass1
ToF2
Mass2
ToF3
Mass3
ToF4
Mass4
...

```



```
...
...
```

In this scenario we can define two array times which overlap in the address space. Here is the block definition:

```
Event
{
    ToF [10] 28, , , 14;
    Mass [10] 28, 1, 1, 14;
}
```

The first element of ToF starts at byte offset 0 bit offset 7 (remember big endian is MSB on high address). You don't have to specify any of this. But we have to indicate a spacing of 28 bits between the individual elements. 14 bit for the ToF item and a gap of 14 bit where the Mass item will reside.

The Mass item has the same layout as ToF with the exception that it is shifted by 14 bit. We specify the start position as byte 1, bit 1. This is 8 bit for the first byte plus 7-1 = 6 bit in the second bit. Resulting in 14 bit offset from the start of the ToF item.

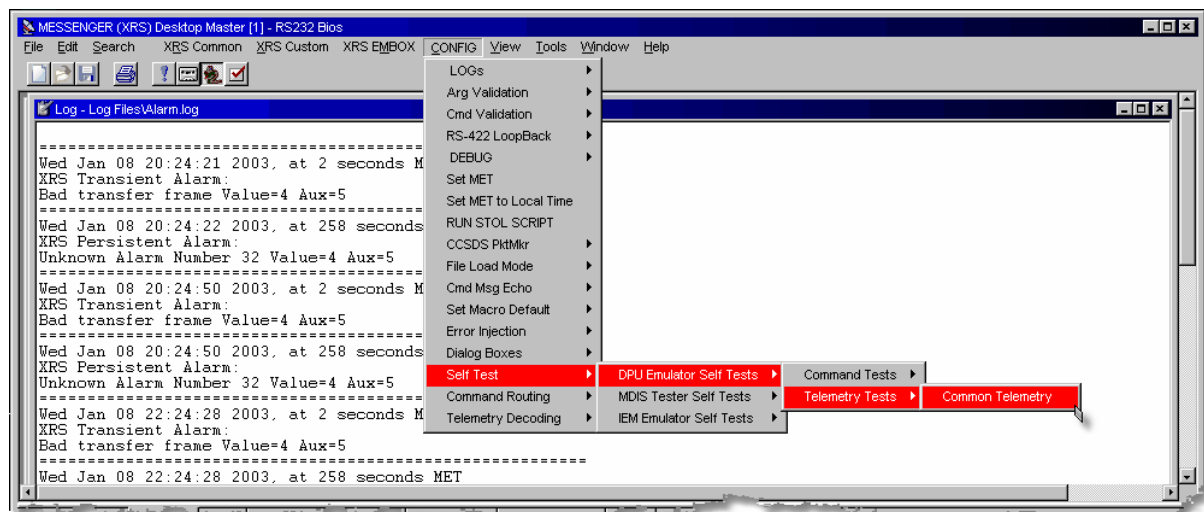
4.1.3 Command Batch Files (*.cpd)

Enter topic text here.

4.1.4 Command Menu Files (*.cm)

GSEOS allows you to define custom menus to easily access any of your commands. The command menu definitions are stored in flat ASCII files with the .cm extension. You can specify the command menu files to load in the Load entry of the [Config] section of the gseos.ini file.

The following image shows a custom menu:



Below is a sample command menu file to demonstrate the syntax:

```

/* -----
*/
/* - GSEOS sample command menu file. -
*/
/* -----
*/
Menu &Config
{
    Popup &Logs
    {
        Popup S&TOL Log
        {
            Menuitem Enable, fEnableLog()
            Menuitem Disable, fDisableLog()
            Separator
            Menuitem Comment, fEnterLogComment("$'Enter STOL LOG comment'")
        } /* End of Popup STOL Log */

        Menuitem &Start, fStart()
    }
    Include MyIncludeFile.cm
}

```

The command menu file recognizes the following keywords:

Menu, Popup, Menuitem, Separator, and Include.

A command menu file can have multiple main menu entries. Each main menu entry is defined with the Menu keyword followed by the menu name. In the above case &Config. The ampersand character '&' can be used to define keyboard shortcuts for the menu. The character following the ampersand character can be used to activate the command.

The Menu body as well as the Popup body is enclosed in curly braces '{', '}'. The Menu body and contain any number of Popup, Separator, and Menuitem entries. A Menuitem entry lets you specify a command to be excuted. The syntax is:

```
Menuitem Menu Name, Command
```

where the Menu Name is the name as it appears on the menu and the command is any valid command you have defined. Make sure you use the proper namespaces. I.e. if you have your commands defined in a module called InstCmds and you import the module with `import InstCmds` you would need to use `InstCmds.MyCommand()` to issue the `MyCommand()` command.

A Separator places a separator between menu items. The Popup keyword allows you to set up a command hierarchy by nesting menus. You can place and number of Popup, MenuItem, and Separator keywords within the body of a Popup statement. You can nest to any level.

As with the command button definition you have a simple text preprocessor available that lets you prompt for parameters and does simple text replacement. If you have Python commands that expect strings you have to make sure to embed the parameter in quotation marks. You can do this in the menu definition so the user does not have to

supply the quotation marks. See the example above.

The Include statement lets you include other command menu files. The file to be included will be inserted in place of the Include statement. The included command menu file must be itself a well-formatted command menu file. The Menu statements in the included file will be converted into Popup statements in the including file. The included file may have multiple Menu statements.

4.1.5 Configuration Files (.cfg)

Configuration files are used to store GSEOS system configuration parameters. They are specified in a modified XML syntax. A configuration file can hold configuration information for various different modules of GSEOS. Currently the only module implemented is the Alarm Monitor configuration. These configuration files are usually generated and modified from the GSEOS user interface. However, they can be edited with a regular ASCII editor as well. Any changes to the configuration file will be recognized by the GUI tools and displayed accordingly. In order for the configurations to take effect in GSEOS they have to be loaded either at startup with an entry in the gseos.ini initialization file or at runtime from the File/Open menu.

Note:

The Alarm Monitor configuration Wizard is currently not implemented, so the only way to configure an Alarm Monitor is to manually create the configuration file.

The following subchapters explain the different configuration settings:

Alarm Monitor configuration

4.1.5.1 Alarm Monitor Configuration

Alarm Monitors scan a data item for a certain condition. Once the condition is met the alarm fires and executes one or many actions. The trigger condition as well as the actions are defined in a Alarm Monitor section of a GSEOS configuration file.

The format is similar to XML but not well-formed XML. Typically the trigger condition contains special characters like (<, >, etc.). For easy configuration these characters can be embedded in the configuration file in clear text and therefore violate the XML definition. If you prefer you can escape these characters yourself and therefore generate well-formed XML. So for example to specify the trigger condition: Value < 20 you would write:

```
<Trigger Condition="Value &lt; 20"/>
```

This way you can run your configuration file through an XML checker and it will be parsed successfully given your other XML definitions are correct.

If you want to check out your config file in Internet Explorer or some other XML display tool you will have to make sure you escape any special characters you use in conditions or other text items. For example:

```
<Trigger Condition="Value < 20"/>
```

has to be converted to

<Trigger Condition="Value < 20"/>

in order to display your config file in an XML display tool.

You don't have to escape if you use the config file with GSEOS, this conversion happens automatically.

Please use the sample file as a template and documentation of the various elements.

<AlarmMonitor>

An alarm monitor is defined in an <AlarmMonitor> element within a GSEOS configuration file. There can be any number of <AlarmMonitor> elements in any given configuration file.

The <AlarmMonitor> element has the following attributes:

Name: The unique name of the alarm monitor. If the name exists already and this file gets loaded the old monitor will be replaced with the new one.

Subelements of the <AlarmMonitor> node are <DataItem>, <Trigger>, and <Actions>.

<DataItem>

There must be exactly one <DataItem> element per <AlarmMonitor> element. This element specifies the GSEOS data item to monitor. This node takes two attributes: Name and Conversion.

Name: The name of the GSEOS data item to monitor. If the item is an array item you have to specify the element of the array you wish to monitor. Currently you can only monitor scalar items or individual elements of an array item.

Conversion: If the data item has a conversion function associated you can specify the conversion function to apply before evaluating the trigger condition. Note that the conversion function does not need to be loaded at the time when you load the alarm monitor configuration, however. As soon as the monitor is installed and gets evaluated the proper formula file (*.qlf) has to be available, otherwise an exception will be raised. This behavior allows you to specify the *.qlf and *.cfg files in any order in the gseos.ini file without creating dependencies.

<Trigger>

There must be exactly one <Trigger> element per <AlarmMonitor> element. This element specifies the trigger condition that will determine when the alarm monitor fires. GSEOS data item to monitor. This node takes three attributes: Condition, Count, and Timeout. The condition is evaluated every time the data item specified in the <DataItem> element arrives. Once the outcome is positive the timeout conditions are applied to determine if the alarm fires or not. The Count and Timeout attributes control the dynamic behavior.

Condition: This element contains the actual trigger condition that gets evaluated every time the monitored data block arrives. As mentioned above you don't need to escape special characters when defining your condition. The

special variable 'Value' represents the current value of your data item under investigation. You should use this variable to refer to the data item value and use it in your condition. Another special variable name is 'Delta'. Delta represents the difference from the previous value to the current value. E.g. if the previous value was 27 and the current value is 11 Delta would be -16. You can use Delta to check for differentials. You can use both Value and Delta in the same expression if required. The condition statement should evaluate to a boolean value. Keep in mind that the data item element defines if a conversion function should be applied to the data item. If so you want to compare against the engineering units instead of the raw count.

- Conversion:** If the data item has a conversion function associated you can specify the conversion function to apply before evaluating the trigger condition. Note that the conversion function does not need to be loaded at the time when you load the alarm monitor configuration, however. As soon as the monitor is installed and gets evaluated the proper formula file (*.qlf) has to be available, otherwise an exception will be raised. This behavior allows you to specify the *.qlf and *.cfg files in any order in the gseos.ini file without creating dependencies.
- Count:** The value for count has to be numeric. If not specified it defaults to 1. The Count determines how often the condition has to evaluate to True before the alarm fires. The condition has to evaluate to True consecutively, that is once it evaluates to False the count will be reset. Also not the Timeout attribute that applies a timing condition to the count.
- Timeout:** Specifies the timeout in seconds. If not specified or 0 no timeout is applied. The timeout determines the interval in seconds in which the condition has to evaluate to True Count number of times. The timeout is implemented as a sliding window. Every time the Count condition is met the timeout is evaluated from the first item that made the Count condition successful.

<Actions>

The <Actions> element configures the actions that can be executed when the alarm fires. You can configure one or more actions within the one and only <Actions> element.

- Message:** This element configures the text that gets written to the Message window. The text you specify between the sub-elements <Text> </Text> is prepended with some Alarm information.

```
<Message>
  <Text>
    A red high
    alarm has occurred.
  </Text>
</Message>
```

- LogFile:** This element configures the text that gets written to a log file. The file name is specified with the attribute FileName. The log text is configured as in the Message element.

```
<LogFile FileName="MIMIAAlarmLog.log">
  <Text>
    A red high alarm has occurred.
  </Text>
</LogFile>
```

Command: The <Command> element issues a command. This is it generates a CMDSTRING block with the contents you specify here. The attribute Name is the command string that gets executed.

```
<Command Name="PS_DECON OFF"/>
```

Python: This element allows you to call an arbitrary Python function. The function is specified with the Function attribute.

```
<Python Function="fMyFunction(4 < 55)"/>
```

Email: The <Email> element configures the email action. This action can send out an email to one or more recipients. The SMTPHost attribute of the Email element specifies the SMTP host you are sending your email from. The sub-element <From> specifies the sender name and email address. The sub-element <To> specifies one recipient, there can be multiple <To> tags within one <Email> node. The <Subject> and <Body> nodes set the subject line and the email body respectively.

The attributes Quota and QuotaPeriod control the number of emails that can get send out. The value of Quota has to be an integer value. It represents the maximum number of emails that can get send out during the time period QuotaPeriod. QuotaPeriod specifies the time the Quota applies to. After the period expires another Quota emails can be sent within the next period. The value of QuotaPeriod has to be a number (floating point is fine) that ends in either m (minutes), h (hours), or d (days) to indicate the dimension. It is usually recommended to set a quota since an ill-behaved alarm monitor could possibly generate emails on a basis of a fraction of a second.

```
<Email SMTPHost = "smtp.jhuapl.edu" Quota="5" QuotaPeriod="2h">
  <From>SOPC@jhuapl.edu</From>
  <To>hauck@gseos.com</To>
  <To>Joe.User@jhuapl.edu</To>
  <Subject>MIMI Alarm</Subject>
  <Body>
```

by This is the message body. In addition this text will be prefixed
some general information about the alarm.

```
</Body>
</Email>
```

4.1.5.1.1 Alarm Monitor Sample

```

<!-- =====
-->
<!-- This configuration file defines one or more GSEOS Alarm Monitors.
-->
<!-- In order to install the Alarm Monitors in GSEOS load the file from
-->
<!-- the File/Open menu or specify it in the Load entry in gseos.ini
-->
<!--
-->
<!-- GSE Software, Inc.
-->
<!-- Author: Thomas Hauck
-->
<!--
-->
<!-- History: Apr-08-2004 th R001 First implementation.
-->
<!--
-->
<!-- =====
-->
<GSEOSConfig Version = "1.0">
  <AlarmMonitor Name="AlarmTest1">
    <DataItem Name="AlarmMonitorTestBlk.X1"/>
    <Trigger Condition='4.0 > Value >= 5.0' Count="2" Timeout = "6.3"/>
    <Actions>
      <Message>
        <Text>
          A red high
          alarm has occurred.
        </Text>
      </Message>

      <LogFile FileName="MIMIAAlarmLog.log">
        <Text>
          "A red high alarm has occurred."
        </Text>
      </LogFile>

      <Command Name = "PS_DECON1 OFF"/>
      <Command Name = "PS_DECON2 OFF"/>

      <Python Function = "GseosCmd.sound('alarm.wav') " />
      <Python Function = "GseosCmd1.sound('alarm.wav') " />

      <Email SMTPHost = "mail.gseos.com" Quota="4" QuotaPeriod= "2h" >
        <From>SOPC@jhuapl.edu</From>
        <To>aaa@bbb.ccc</To>
        <To>xxx@yyy.zzz</To>
        <Subject>AlarmMonitorTest1</Subject>
        <Body>
          This is the message body. In addition this text will be prefixed
          by
          some general information about the alarm.
        </Body>
      </Email>
    </Actions>
  </AlarmMonitor>
</GSEOSConfig>

```

```

    </Actions>
  </AlarmMonitor>
</GSEOSConfig>

```

4.1.6 Formula Definition Files (*.qlf)

Formula Files allow you to define Expressions and Conversion Functions.

Expressions

Expressions are mathematical expressions that can take any number of parameters and you can select Expressions as display objects on the screen. Expressions can be used to perform a mathematical operation on a data item before displaying it. Expressions can also be accessed from Python script as explained later in this chapter.

A simple example of an expression definition would be:

```
Linear(m, x, t) := m*x+t
```

Conversion Functions

Conversion functions are similar to Expressions in that they allow you to use a define a mathematical function. However, they are closely related to a specific data item in that they represent a conversion for this particular item. The following example shows the engineering unit conversion for the data item HK.HTR_5V

```
EU (Raw="HK.HTR_5V") := (Raw*5.0) / 256
```

A Conversion function can have only one parameter and it needs to define a default value which is the name of a data item (Note that you have to specify the name of the item as opposed to the value). You can then use the formal argument name in the function definition to refer to the data item. As opposed to expressions you can define multiple conversion functions with the same name (the data items these functions are tied to should be different though). Usually this is exactly what you want to do. This way you can define for example a EU engineering unit conversion and can call this function on various data items. In this case the appropriate conversion function (which may differ from data item to data item) will be invoked. This feature is used for the STOL emulator to map to the correct conversion function depending on telemetry point.

Both Conversion functions and Expressions are defined in a .qlf formula file. The formulas need to be loaded either at runtime or from the gseos.ini file. Once the formula is loaded it can be accessed from GSEOS when displaying an Expression or in the item select dialog when selecting a data item.

The loaded formulas are also displayed in the GSEOS Explorer underneath the nodes Conversions and Expressions.

If you change the formula file and load it into GSEOS the formula definitions will be updated accordingly and reflected in the display.

Expressions and Conversion functions are mapped into the Conversion module. The two examples defined above can be accessed from Python in the following way:

```

Conversion.Linear(2.3, PHA.TOF1, 20)
Conversion.EU("HK.HTR_5V")

```

This feature now allows you to use the same Expressions that you have used in the past for display purposes only from Python script. It effectively can be used instead of going

through the effort and writing a separate decoder. Since Expressions and Conversions functions are limited to one line statements and can not contain flow control directives any complex decoding tasks are still better managed in a separate decoder generating a new output block that then in turn can get displayed.

On the other hand, if the expression needed is very simple instead of writing Decoders to convert data items you can use Expressions to display items modified by a simple function. The latter approach allows you to quickly assemble display screens without going through the overhead of defining a separate block and writing a Decoder.

Formula files must have the extension .qlf and contain one function per line. The syntax of a Conversion Function definition is:

```
FunctionName(Parameter1, Parameter2, ... ParameterN) := Expression
```

or

```
ExpressionName(Param1="Block.ItemName") := Expression
```

The formal parameter names can be used in the expression. These parameters will be replaced with GSEOS data items when an Expression is displayed on a screen. For Conversion functions you have to provide exactly one parameter that has a default value which is the name of a data item.

The body of the Expression/Conversion function can be any valid Python expression. The expression must not span multiple lines and it can not contain flow control statements. For backward compatibility the old Qlook Formula File format is still supported. However, it is strongly recommended to only use valid Python constructs, i.e. use 0xE43F instead of 0E43Fh for hexadecimal numbers.

In order to use Expressions/Conversion Functions you have to load them. This can either be done with the user interface File/Open function or at startup time in the gseos.ini Config/Load section. Once the functions are loaded they are available to be placed on display screens as Expressions or Conversion Function items. They can also be accessed from Python via the Conversion module. The module Conversion is automatically imported and is updated when new Formula files are loaded. You can load any number of Formula files, the Expressions and Conversion Functions contained in these files will be added to the pool of available Conversion Functions.

You should avoid using the same name for different Expressions since they will overwrite each other. As explained above for Conversion functions you might want to do exactly that and choose the same name for multiple functions (using different data items). Comments can be inserted anywhere in a Formula file and are started with the '#' character.

If a formula raises an exception during runtime the result of the conversion will be set to 0.0. and the exception will be displayed in the status line when moving the cursor onto the data item.

The following functions are available for use in a formula file:

sin	Sine
cos	Cosine
tan	Tangent
max	The maximum of the two values
min	The minimum of the two values
ftol	Float interpreted as signed long
ltof	Signed long interpreted as float
swap16	Swap bytes in a 16-bit item
swap32	Swap bytes in a 32-bit item
signed8	Interpret 8-bit value as signed
signed16	Interpret 16-bit value as signed
signed32	Interpret 32-bit value as signed
log	Log
exp	Exponent
year	Get the year (the full 4-digit year) from the number of seconds as of Jan-01-1958 00:00:00
month	Get the month (Jan=1, Feb=2, ...) from the number of seconds as of Jan-01-1958 00:00:00
day	Get the day from the number of seconds as of Jan-01-1958 00:00:00
hour	Get the hour from the number of seconds as of Jan-01-1958 00:00:00
minute	Get the minute from the number of seconds as of Jan-01-1958 00:00:00
sec	Get the second from the number of seconds as of Jan-01-1958 00:00:00
pow	Power

Besides the functions mentioned above all the functions from the standard Python math module are available.

The sample file below shows a simple formula file:

```
#
# Sample Expressions.
#
Analog      (a,b)           := ltof(a)*b
Div          (a,b)           := a/b
Equal       (a, b)           := a == b
Ever        (a)              := 1
Lin         (m,x,t)           := m*x+t
Percent     (Total, Used)     := Used / Total * 100

Year        (Time)           := year(Time+378691200)
Month       (Time)           := month(Time+378691200)
Day         (Time)           := day(Time+378691200)
Hour        (Time)           := hour(Time+378691200)
Minute      (Time)           := minute(Time+378691200)
Second      (Time)           := sec(Time+378691200)
Mod         (a,b)            := a % b
f           (x,y)            := 2*sin(x) - 2*cos(y)
```

```

NadirCycle    (flNadirCycle)           := ltof(flNadirCycle)
CheckInstNPacket (byInstIs, byInstWant, byPacketIs, byPacketWant) :=
  (byInstIs == byInstWant) && (byPacketIs == byPacketWant)
LongToFloat    (l)    := ltof(l)
FloatToLong    (f)    := ftol(f)
TicksToMs      (Ticks) := (Ticks*1000)/515625
Signed8        (s)    := signed8(s)
Signed16       (s)    := signed16(s)
Signed32       (s)    := signed32(s)

#
# Sample Conversion Functions.
#
EU(Raw='MDS_Status.MIRROR_SETPOINT_TEMP') := -31.98687876 + 1.23958063*Raw
+ (-.01357386)*pow(Raw, 2) + 8.788e-005*pow(Raw, 3) + (-2e-007)*pow(Raw, 4)

EU(Raw='MDS_Status.GRATING_SETPOINT_TEMP') := -31.98687876 + (-1.7654)*Raw
+ 0.0065*pow(Raw, 2) + 8.788e-005*pow(Raw, 3) + (-2e-007)*pow(Raw, 4) +
0.0*pow(Raw, 7)

EU(Raw='MDS_Status.MIRROR_A_TEMP') := -31.98687876 + 2.9833*Raw + (-
0.00333)*pow(Raw, 2) + 8.788e-005*pow(Raw, 3) + (-2e-007)*pow(Raw, 4)

```

4.1.7 gseos.ini

You can configure various parts of GSEOS with options you set in the configuration file `gseos.ini`. The `gseos.ini` file is an ASCII file organized like a typical Windows configuration file (e.g. `WIN.INI`). It contains sections and keys with associated values in the various sections. Sections are delimited by `'[Section]'`. The sample below shows a sample entry:

```

[Project]
Name=MyProject
Title=MyProject

```

This entry defines the section 'Project' which has two keys: 'Name' and 'Title'.

Command line

You can specify a different ini file on the command line with the `/ini` switch. The argument following the `/ini` switch must be the path to a valid GSEOS configuration file (although it does not need to be named `gseos.ini`). This allows to manage several projects independently of each other. Also see the `[ChooseConfig]` section later in this chapter.

The syntax of the GSEOS ini handling is a superset of the Windows format. Please refer to the following paragraphs for more details:

Section and entry names are not case sensitive.

Section

The ini file consists of a sequence of sections. A section is identified by a section name embedded in square brackets. There can be only whitespaces leading the open bracket and only whitespace or comment after the section name. Comment characters are hash '#', and semi-colon ';'.

```
[Section]
Key = Value
```

```
[Section] # This is a comment, this is still a valid section.
Name = Image
```

Section Entries

A section can have a body consisting of Name = Value entries.

```
Load = System.cpd
```

It is also possible (unlike with regular Windows style ini files) to have multiple entries with the same name. In this case all values are added to the multi value list for that name.

```
[Config]
Load = System.cpb
Load = Rtiu\Rtiu.py
```

Instances

The special section [Instance] allows to map various different instances of GSEOS to start with different settings. The command line switch /Ixxx lets you specify the instance number you want to start. The entries in the [Instance] section have to list the section mappings you want to apply to the instance to start.

```
[Instance]
Project = ProjMOC PrjXRS PrjGRS
```

In the above example the section [Project] gets mapped to section [ProjMOC] for /I1, to [PrjXRS] for /I2, and to [PrjGRS] for /I3.

Note that the values have to be specified in the numerical order of the instance number. I.e. if you want to configure an instance /I5 you have to specify all instance mappings from 1 to 5. If you don't have any custom settings for a particular instance you can map this to the original section:

```
[Instance]
Project = Project Project PrjGRS Project Project5
```

If a particular instance mapping is not specified a section with the name of [InstanceNNN.Section] is looked up. If you run /I8 on the above configuration and you have defined a section [Instance8.Project] this section will be used.

The /I instance switch also takes a name argument. This name will be used to look up the instance section in the following way: InstanceName.Section
So if you specify /I MOC on the command line all sections will be mapped to MOC.Section.

```
[MOC.Project]
[MOC.Config]
```

If a section with that name doesn't exist it falls back to the original section name. If you have multiple configurations that share common sections like: Master1, Master2, Master3 all use the same [Config] section you can map the [Master1.Config], [Master2.Config],

[Master3.Config] sections to another section (probably [Config]) with the following entries in the [Instance] section:

Master1.Config=Config

Master2.Config=Config

Master3.Config=Config

Choosing a configuration

The section [ChooseConfig] acts as an interactive configuration selector. You can configure various command line options and they are displayed in a dialog for the user to choose a configuration.



The [ChooseConfig] section has two entries: Option and CmdLine. The format of the entries is somewhat special. The entries have to be specified in pairs, Option and CmdLine together. The Option key specifies the name that is listed in the list box and the CmdLine is the command line switches with which GSEOS will be invoked. The following settings generate the dialog box above:

```
[ChooseConfig]
Option  = P-ALICE
CmdLine = /ini i_ALICE/gseos.ini

Option  = LORRI Master (Connect to Emulator Box)
CmdLine = /I Master /ini i_LORRI/LORRI_ConfigFiles/gseos.ini

Option  = LORRI Slavel (Connect to Master)
CmdLine = /I Slavel /ini i_LORRI/LORRI_ConfigFiles/gseos.ini

Option  = LORRI Slave2 (Connect to Master)
CmdLine = /I Slave2 /ini i_LORRI/LORRI_ConfigFiles/gseos.ini

Option  = PEPSSI Master (Connect to Emulator Box)
CmdLine = /I Master /ini i_PEPSSI/PEPSSI_ConfigFiles/gseos.ini

Option  = PEPSSI Slavel (Connect to Master)
CmdLine = /I Slavel /ini i_PEPSSI/PEPSSI_ConfigFiles/gseos.ini

Option  = PEPSSI Slave2 (Connect to Master)
CmdLine = /I Slave2 /ini i_PEPSSI/PEPSSI_ConfigFiles/gseos.ini

Option  = RALPH
CmdLine = /ini i_RALPH/gseos.ini

Option  = SDC
CmdLine = /ini i_SDC/gseos.ini

Option  = SWAP
CmdLine = /ini i_SWAP/gseos.ini
```

The entries are listed in the order in which they are specified in the ini file. Note that you can specify the /I and /ini switches especially which allows you to redirect to other configuration files. Processing of the [ChooseConfig] section is recursive. That is if you

have another [ChooseConfig] section in another instance or ini file you direct to you can display child dialogs and can therefore build a hierarchy for more complex configurations.

__include__ Directive

The __include__ directive allows you to reference information from other ini files and therefore decentralize management of the gseos.ini file. The __include__ directive has the following syntax:

```
__include__ Filename [Section [Entry]]
```

__include__ directives can be placed either at top level or within a section. Depending on the arguments specified in the __include__ directive the amount of data to be included can be controlled. If only the file name is specified the entire file is added. If a section name is specified the section is added. If a section and entry is specified only the particular entry (or if multiple entries exist for the same name, those entries) will be added.

The process of adding is a merging process. If the section in question does not exist a new one is created and the contents added to the new section. If the section does exist the entries from the source section are added to the existing section. If an __include__ is specified on section level no new section is created but the entries from the source section are added (at the position where the include is located) to the existing section. If the __include__ directive specifies individual entries only those are added.

Writing sections takes the __include__ directive into account and writes the section back to the proper file. However, if single entries are __include__ed those are not written back to the include file, only entire included sections will be written to the source file.

The following sections list the configuration options recognized by GSEOS.

- [Buffer]
- [Command]
- [Config]
- [Console]
- [Instance]
- [Net]
- [Printer]
- [Project]
- [PyStartup]
- [Recorder]
- [System]

4.1.7.1 Buffer

The [Buffer] section manages the GSEOS data buffers. This is a critical system parameter and determines the amount of data buffer available for GSEOS. You can monitor this value as well as the currently used buffer in the GSEOS Explorer.

Entry	Description
FixedBuffer	This entry specifies the total amount of memory (in kB) available to GSEOS. The amount of memory corresponds directly to the time the data can be buffered by GSEOS. When there is no more memory available the system will lose data. The default is 512 (512 kB).

Example

The following example sets the total amount of available memory to 10 MB.

```
[Buffer]
Fixedbuffer      = 10240
```

4.1.7.2 Command

The [Command] section sets some values for the COMMAND module. Especially for the batch handling.

Entry	Description
SendLateCmd	This entry specifies what to do when a late command is detected in a command batch. If 'Yes' is specified the command is send immediately. If 'No' is entered the command is not send at all. For the definition of a late command see MaxDelay.
MaxDelay	MaxDelay specifies when a command is to be considered as late command. The entry must be a positive integer and specifies a time in seconds. During processing of a batch file it is possible that a time mark is missed due to system overload. If the time is within the limit specified by MaxDelay the command is considered as 'not late' and processed normally. Otherwise it is a late command and processed as specified by SendLateCmd.
CMDPEnable	Deprecated! Enable or disable the legacy command processor. The default is 1 meaning the legacy command processor is enabled. Unless upgrading an old system you should not use the old command processor, it will not be supported in future versions. The Python GseosCmd module replaces the command processing functionality. If the command processor is enabled you can still issue python commands but the error reporting will be crippled. If the legacy command processor is enabled all commands are first routed to the Python cmd module. If the command can be executed successfully the command is not propagated. If the command could not be executed successfully it will be forwarded to the legacy command processor. If the command is still not executed without error the error will be reported according to the legacy command processor. Assume to try to issue a python command but make an error, in this case the legacy command processor will intercept the command and issue an error message with has nothing to do with the actual python error. If you do not use the legacy command processor you should set this entry to 0. In this case an erroneous command will properly be reported in the console window.

Example

The following example consideres a command as late when it is processed at least 11 seconds past its time mark and sends a late command immediately.

```
[Command]
SendLateCmd = Yes
MaxDelay    = 11
```

4.1.7.3 Config

The [Config] section defines the configuration files that are loaded when the system is started.

Entry	Assignment
BlkFiles	The BlkFiles entry specifies the block definition files to be loaded. If you don't specify a file here the block file with the project name is loaded. You usually want to include the system block definitions from system\system.blk. In addition you want to provide your specific block definitions in one or multiple additional files. Multiple file names are separated by spaces.
Load	<p>This entry specifies the configuration files to be loaded when the system comes up. The file names specified in this entry must be separated by spaces. The path names can be absolute or relative to the working directory the system starts in. It is important that the files specified have well known extensions. The recognized extensions are:</p> <ul style="list-style-type: none"> *.cpd: Command definition file (legacy module) *.cpb: Command batch file *.cm: Command menu file *.dt: Desktop *.scr: Screen file *.log: Log file

It is possible to specify more than one configuration files for all types except the desktop. If no desktop file is specified the desktop configuration that was active when the last session was closed is loaded (autodskN.dt). When a desktop file is specified this desktop is loaded on every start of the system. This enables the same appearance of the system with every start.

Example

The following example installs a desktop, two command files, one monitor condition file and one monitor check file.

```
[Config]
BlkFiles =system\system.blk MyProject.blk demo\mon1\mon1.blk
Load     = general.dt mimi.cm startup.cpb
```

4.1.7.4 Console

The [Console] section defines the configuration settings for the Console window.

Entry	Assignment
MaxFileSize	Specifies the maximum size of the console file in KB. The file will be truncated once it reaches the MaxFileSize limit. The oldest content will be discarded. The default value is 512KB.
FileName	The name of the console window file. Defaults to ProjectName[Instance].con if not specified. If the console window file is not writeable (read-only) the contents of the console window are not

logged.

4.1.7.5 Instance

The [Instance] Section allows you to run multiple instances of GSEOS at the same time on the same machine. Each instance picks up its assigned configuration information from the gseos.ini file by using the [Instance] section as a lookup table.

Entry	Description
[Section]	A list of reference sections to look up for the according instance. The first entry identifies the section referenced by the first instance, the second entry identifies the section referenced by the second instance and so on. If no entry is found for the current instance, the default section is taken. If no instance dependent handling is used for a predefined section the default section is assumed. If you don't plan using this feature you don't need to supply the [Instance] section.

Example

The following example shows how to install different [Recorder] sections depending on the current instance. The first instance for example is used for recording data, whereas the second instance is mainly used to replay data from an archive.

```
[Instance]
;          1st      2nd      3dr      4th      ... instance
Project    = Prj1    Prj2
Bios       = Bios1   Bios2
Recorder   = Rec1    Rec2

[Project]
Name       = CELIAS
Title      = CELIAS

[Prj1]
Name       = CELIAS
Title      = CELIAS Sim-I

[Prj2]
Name       = CELIAS
Title      = CELIAS Sim-II

[Bios1]
IOBaseAddress = 0x300
HSSIInterrupt = 11

[Bios2]
IOBaseAddress = 0x140
HSSIInterrupt = 10

[Recorder]
DataPath     = ..\data
FileSize     = 1024

[Rec1]
DataPath     = ..\write
```

```

FileSize          = 1024

[Rec2]
DataPath          = ..\read
FileSize          = 1024

```

4.1.7.6 Net

GSEOS network support is very flexible and accommodates various different networking configurations. The basic concept of the network module is to import/export data blocks via the TCP/IP protocol. The network module functions as a data source when importing data. You can configure any number of network connections. Each network connection can be associated with at most two blocks. One that gets exported on this connection and one that gets imported. From a network perspective GSEOS can act as a server or a client. For each connection you have to specify if you want GSEOS to act as a server or a client. This does not necessarily determine if you export or import blocks on that connection. A common scenario is to configure a server connection and export a block on that connection. However you may as well configure a client connection and export a block on that connection.

The [Net] section determines the network configuration. The keys you specify in this section are the names of the connections you want to configure. The value can be either Server or Client for a network server or a network client respectively.

```

[Net]
TLMSrv=Server
TestServer1=Server
TomsServer=Server
SOPC33=Server
TLMClnt=Client
CmdSrc=Client
TestClient=Client

```

The above example configures four server connections and three client connections. You can manage these connections from within the GSEOS Explorer. In order to configure the individual connections you have to create new sections with the connection name as the section name, e.g.:

```

[TLMSrv]
Port=2001
Source=TLM

```

The section above specifies the setting for the TLMSrv server connection you defined in the [Net] section. This particular example configures the server to listen on port 2001 and export the TLM block.

The next paragraphs explain the various options you can specify for a network connection. The settings that only apply to client connections are indicated.

Entry	Description
IP-Address	Only for client connections. The IP address of the remote server. Specify the IP address in 4-byte dotted format, e.g. 150.144.103.23
Port	The port number of the remote machine in case of a client connection, the listen port in case of a server connection. There must be a server listening on this port at the IP-Address specified in order for a client

	connect attempt to be successful.
Source	The data block you want to export on this connection. Every time the system encounters this block it will send the contents of the block to the remote machine. The actual amount of data sent depends on the VariableLen setting.
Destination	The data block you want to import on this connection. All data received from the server will be written to this block. Once the number of bytes specified in the block definition is received the block is submitted to the system. This is the default behavior and can be modified with the VariableLen setting. The default behavior is appropriate for fixed length data packets. For variable length packets you want to use the VariableLen flag.
AutoConnect	Only for Client connections. Allows to automatically connect to a server. Specify a number of seconds that will elapse before an attempt is made to connect to the remote machine. If the connection is already established no attempt to connect will be made. If you set this value to 0 (default) automatic connection is disabled.
VariableLen	This setting controls the amount of data sent over the network connection. The default is 'No'. For the source block the amount of bytes specified in the block definition file is sent. for the destination block the amount of bytes specified in the block definition has to be received before a block is generated. This setting is preferred for inter GSEOS connections or connections that generate fixed length data. If you specify 'Yes' for this setting the connection uses variable length packets. The blocks specified in either Source or Destination have to have a 32-bit field called 'Len' as the first data item. For Source blocks the Len field specifies how many bytes of data are transferred. The Len field itself is not sent, only the data immediately following the Len field. For Destination blocks the Len field is filled with the amount of data read from the network connection. When more data is received than can be placed in the block multiple blocks are generated.
Exclusive	The network module is considered a data source. The default behavior for the network will be to discard all data received on the network connection unless the network is enabled. There are some circumstances where this is not desirable. E.g. consider the case of remote commanding. In this case we may have incoming data from the Bios but want to be able to feed in command data over the network. If we were to enable the network the Bios data would be discarded, not an option. However if the Bios is enabled (and the network therefore disabled since all data sources are mutual exclusive) all command data from the network would be discarded. To enable network input while getting data from another data source set this value to 'No' and do not enable the network. The default is 'Yes' which means all incoming data from the network is discarded unless the network is enabled.

Connecting two GSEOS machines

Oftentimes it is desirable to distribute the data decoding/display to various machines. This can easily be done by having one machine exporting a data block and the other importing the same block. The default behavior of a connection is to export/import the entire block. This is a fixed size packet based on the block definition for the block you

import or export. This is what you need to interconnect two GSEOS machines (given of course that the block definitions on both machines are the same!). The decision which machine to configure as server and which one as client pretty much depends on where you want to initiate the connection from. The client machine has to initiate the connection. Lets assume we have two machines, the Lab machine with the physical data connection to the instrument and an Office machine where we want to run remote display. The Lab machine will be configured as server and the Office machine as client so we can start the remote display from the Office machine. The block exported by the Lab machine and imported into the client machine is TLM. We also want to enable commanding from the Office machine. This means we have to set the Exclusive setting on the Lab machine to 'No'. If we don't want to enable commanding we would not need to set the Exclusive flag to 'No' and we would not need to specify the CMDSTRING block in either configuration. Here the configuration for the Lab machine:

```
[Net]
TLMsrv=Server

[TLMSrv]
Port=2020
Source=TLM
Destination=CMDSTRING
Exclusive=No
```

Here the configuration for the Office machine:

```
[Net]
TLMClient=Client

[TLMSrv]
IP-Address=150.134.123.87
Port=2020
Source=CMDSTRING
Destination=TLM
```

4.1.7.7 Printer

The [Printer] section allows to configure the printer settings.

Entry	Description
PrinterInit	Allows to send an initialization file to the printer when GSEOS is started. The name of this file is specified with this entry. If you don't want to send a file to the printer specify 'None'.
PageInit	The entry PageInit allows to send a file to the printer before a page is printed (e.g. a printer macro). The name of this file is entered here. If you don't want to send a file to the printer before every page specify 'None'.
PageExit	The entry PageExit allows to send a file to the printer after a page has been printed. The name of this file is entered here. If you don't want to send a file to the printer after every page specify 'None'.
MetaFile	The entry Metafile allows to send a Windows Metafile to the printer before a page is printed. The name of this file is entered here. If you don't want to send a file to the printer after every page specify 'None'.
FontSize	Specifies the requested height, in logical units, for the font. If this

parameter is greater than zero, it specifies the cell height of the font. If it is less than zero, it specifies the character height of the font. (Character height is the cell height minus the internal leading. Applications that specify font height in points typically use a negative number for this member.) If this parameter is zero, the font mapper uses a default height. The font mapper chooses the largest physical font that does not exceed the requested size (or the smallest font, if all the fonts exceed the requested size). The absolute value of the nHeight parameter must not exceed 16,384 after it is converted to device units. The font size is used for the fields Date, Time, Page, Caption and Title1 to Title4.

Font	Specifies a font. The font must be available in the windows environment. The default font is Courier. The font is used for the fields Date, Time, Page, Caption and Title1 to Title4.
UserOrigin	Specifies the user origin of the paper. The entry is entered as 'x,y', where x is the x-positon in mm and y is the y-position in mm relative to the upper left corner of the paper. All following position are relative to the user origin.
UserArea	Specifies the area GSEOS can print in. The entry is entered as 'x,y', where x is the width in x-direction in mm and y is the height in y-direction in mm. This means that the top left corner is the UserOrigin and the bottom right corner is UserOrigin + UserArea.
DatePos	Specifies the position where the date will be printed. The entry is entered as 'x,y', where x is the x-positon in mm and y is the y-position in mm relative to the UserOrigin. If 0,0 is specified no date is printed.
TimePos	Specifies the position where the time will be printed. The entry is entered as 'x,y', where x is the x-positon in mm and y is the y-position in mm relative to the UserOrigin. If 0,0 is specified no time is printed.
PagePos	Specifies the position where the page number will be printed. The entry is entered as 'x,y', where x is the x-positon in mm and y is the y-position in mm relative to the UserOrigin. If 0,0 is specified no page number is printed.
Title1Pos	Specifies the position where the title 1 will be printed. The entry is entered as 'x,y', where x is the x-positon in mm and y is the y-position in mm relative to the UserOrigin. If 0,0 is specified no title is printed. The titles can be entered in the printer dialog box.
Title2Pos	See Title1Pos.
Title3Pos	See Title1Pos.
Title4Pos	See Title1Pos.
CaptionPos	Specifies the position where the caption title of the window will be printed. The entry is entered as 'x,y', where x is the x-positon in mm and y is the y-position in mm relative to the UserOrigin. If 0,0 is specified no caption title is printed.

Example

The following example sets up a typical printer environment.

```
[Printer]
PrinterInit      = none
MetaFile         = celiass.wmf
FontSize         = 60
Font             = Helv
UserOrigin       = 30,70
```

```

UserArea      = 150,200
DatePos       = 154,13
TimePos       = 154,22
PagePos       = 154,31
CaptionPos    = 40,65
Title1Pos     = 80,13
Title2Pos     = 80,22
Title3Pos     = 80,31
Title4Pos     = 60,50

```

4.1.7.8 Project

The [Project] section holds some project specific attributes like the project name and main window caption bar title.

Entry	Description
Name	This entry specifies the name of the project. The name of the project must match the name of the block definition file. (e.g. for the project 'image' the blk-file must be named 'image.blk'). If you specify block definition files in the Config section the default block file is not loaded by default but the files you specify in the BlkFiles entry of the [Config] section.
Title	The title entry specifies the entry in the main window caption bar (usually the same entry is used for Name and Title). The default is GSEOS.
Version	This takes an arbitrary string and is reported in the version information section on system startup.
SplashBitmap	You can specify a .bmp file (256 colors) that is displayed as the splash screen on system startup. If you don't specify an entry a file with the name of the project (as specified with the Name entry) with the extension .bmp is opened. If no such file exists the GSEOS internal bitmap is displayed.

Example

The following example shows the project setup for the experiment Image.

```

[Project]
Name      = Image
Title     = Image
Version   = 4.2.2

```

4.1.7.9 PyStartup

The [PyStartup] section allows you to import Python modules and packages and to execute Python statements at startup.

Entry	Assignment
Import	The module or package to import. Do not specify the file extension. The module name specified should be the same that you would use in a Python import statement. The module or package must be in the Python search path to be loaded successfully.
Exec	This entry executes the Python statement listed. This is especially useful to import startup files into the __main__ namespace. Please check the

FAQ for more details on how to configure startup behavior.

Example

The following example lists a typical startup scenario. The package TC_TLM_Load is imported and accessible from the console window as TC_TLM_Load. The Startup.py module is loaded into the __main__ namespace and all attributes defined in the startup module will be available from __main__.

```
[PyStartupMaster]
Import = TC_TLM_Load
Exec   = from Common.Startup import *
Exec   = from i_LORRI.StartupMaster import *
```

4.1.7.10 Recorder

The [Recorder] section is used to control the recorder module. Another section that is used in conjunction with the recorder is [Timebase].

Entry	Description
DataPath	This entry specifies the path where the recorder files are stored. The path can be relative to the project path, or absolute. If an absolute path is specified only physical drives are allowed. If you have a substituted drive g: on the path c:\users\gseos, for example, and you want to store recorder data in g:\data you have to specify c:\users\gseos\data. Note that changing this setting does not take effect until you restart GSEOS. This setting is also modified using the Recorder Settings dialog.
FileSize	The entry FileSize specifies the length of a recorder file in kB. When the size limit is reached the current recorder file is closed and a new file is opened.
Prefix	In automatic mode the automatically generated file name will be prefixed with the value under this entry.
IdleCheckPeriod	When playing back data in the fast forward or fast backward mode the system throttles the playback speed depending on the system load. By default an idle check is performed every 10 blocks. If the system is still busy no data is played back until the timer expires again and the idle check is performed again. You can adjust the number of blocks to play back before performing this idle check. The larger the number the faster the playback will be at the expense of system responsiveness. The default value is 10.
FastPlaybackBlockCnt	When playing back data in the fast forward or fast backward mode one block gets played back every time the playback timer expires. You can adjust this setting to play back more blocks and therefore speeding up the playback of data. As with the IdleCheckPeriod setting this will increase playback speed at the expense of system responsiveness. The default value is 1.

Compress	If this setting is 'Yes' compression is turned on. Depending on the data contents you will be able to save significantly more data (3-20 times) within the same file size. You might want to adjust the FileSize setting accordingly. By default Compression is turned on, if you want to turn it off set it to 'No'. The default is 'Yes'.
Record	The value has to be the block name to be recorded. You need one Record entry per block you want to record. Typically these entries are set interactively from the Recorder Settings Dialog.
Playback	The value has to be the block name to be played back. You need one Playback entry per block you want to put on the playback list. Typically these entries are set interactively from the Recorder Settings Dialog.
Timebase	This setting allows you to configure a time base for the recorder data. You can specify multiple different time bases. You need one Timebase entry per time base you want to configure. The value of this entry is the name of the time base section that configures this time base. See also in the [Timebase] chapter for the individual time base configuration options and a sample.

Example

The following example sets up a recorder environment.

```
[Recorder]
DataPath=Data
Prefix=EM
FileSize=1000
Record=RecComment
Record=SFDUCmd
Record=PeriodicMsg
Record=NoData
Record=CIDPRequest

Playback=RS232Raw
Playback=SFDUCmd
Playback=HK
Playback=CMDSTRING
Playback=TLM
```

4.1.7.10.1 Timebase

The [Timebase] section configures different time bases you can use to save in your recorded data. The section name is actually not [Timebase] but a name you specify in the Timebase entry in the [Recorder] section. Please see the example below. Once you specify a time base you can navigate the recorder data by this time base. See also the [Recorder] section for other recorder specific configuration settings.

Entry	Description
-------	-------------

Name	The name is used as a label in the user interface. Please keep it relatively short since it will determine the space of the other user elements. This name is also saved in the recorder file together with the actual times recorded.
Time	A Python expression that returns a long value that represents the current time of you time base. Python long values are not restricted in length so you can use any precision required.
Format	A Python expression that results in a string representing a given time. The time variable you will get passed to this function is 'Time'. The value of 'Time' is the long you saved earlier with your Time function. You should return a string that is a human readable representation of your time.

Example

The following example defines a Timebase Clock and configures it in the time base section [Clock]:

```
[Recorder]
...
TimeBase = Clock
...

[Clock]
Name      = Realtime
Time      = long(time.time())
Format    = time.strftime("%Y/%m/%d %H:%M:%S", Time)
```

4.1.7.11 System

The [System] section allows to setup some GSEOS system specific attributes.

Entry	Description
Configurable	This entry can be 'Yes' or 'No'. If you specify 'Yes' (the default) the system allows the full flexibility and is interactively configurable by the user. This involves modifying screens, and desktop files. If the system is setup as not configurable no files can be modified by the user. This may be useful as safety precaution and prevent accidental system misconfiguration.
Editor	This entry specifies the default editor to be used.
Net	'Enable' sets the network as the default data source. By default the network is no enabled. You can also enable the network once the system is running using the GSEOS Explorer.
BlockListing	This setting enables or disabled the block listing capability. See also the Block Definition File section for further information on block definitions. If you set this entry to 'Yes' the File/New dialog will contain file type: Block listing (*.lst). This will allow you to generate a detailed report about the bit allocations of your blocks.

StatusBar	Display the GSEOS status bar. 'Yes' (default) will display the bar, 'No' will hide it.
ToolBar	The position of the ToolBar, you can specify 'Top' (default), 'Left', or 'Right'.
ConfirmTermination	If this entry is set to 'Yes' a dialog will pop up to confirm termination of GSEOS. This may be useful when you run important tests and don't want to accidentally shut down GSEOS. The default is 'No' and the system terminates without prompting.
DesktopAutosave	GSEOS by default saves the current desktop as AutoDskN.dt where N is the current instance. If you load a saved desktop any changes to this desktop are not saved automatically and the system does not prompt for confirmation if you want to save the changed desktop. If you set DesktopAutoSave to 'Yes' the currently active desktop will be saved on system exit. If you specify this file in the Load entry of the [Config] section this will automatically restore the last desktop configuration (similar as the AutoDskN.dt behavior). If you set the DesktopAutosave entry to 'No' (the default) the file AutoDskN.dt will still be written to disk.

Example

The following example shows a fully configurable system that uses notepad for editor purposes.

```
[System]
Configurable    = Yes
Editor          = notepad.exe
```

4.1.8 Text Reference Files (*.tr)

Text reference files let you map numeric values into text. This is especially useful for status values. Besides the text representation a color code can be stored with the text reference. If you choose to display a data item represented as a text reference (casting) the colors will be displayed accordingly.

File Format

Each text reference must have a unique name, enclosed in curly brackets. Then the lookup items are listed. The value to be looked up or a range indicated by low limit - high limit is followed by a comma and the text to be displayed. The last parameter is a color code, alternatively the color code can also be spelled out by listing the color by name in the order:

foreground color on background color

The colors are only used for display purposes, if you access the text reference through Python you will only retrieve the string but you can also request the color code if you want to implement your own color display.

Color Names

The following strings are valid color names (the names are case insensitive):

- BLACK
- BLUE
- GREEN

- CYAN
- RED
- MAGENTA
- BROWN
- LIGHTGRAY
- DARKGRAY
- LIGHTBLUE
- LIGHTGREEN
- LIGHTCYAN
- LIGHTRED
- LIGHTMAGENTA
- YELLOW
- WHITE

Python Access

All text references can be accessed through the GseosTextRef module.

Here is how you query for the text reference:

```
GseosTextRef.fGetText('TextReferenceName', Value)
```

The return value is the string defined for the value that was passed in or the empty string if the value doesn't match any defined lookup item or range. If the text reference specified in the function call is not loaded a KeyError exception will be raised.

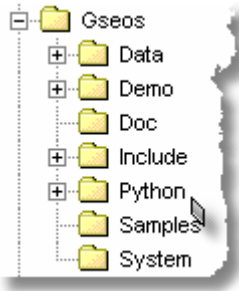
The fGetItem() function returns the text and the color code as a tuple:

```
strText, iTextColor, iBackColor =  
GseosTextRef.fGetItem('TextReferenceName', Value)
```

```
#  
# Sample text reference file.  
#  
DigSnsr      { 0, "OK ", 0xA0;  
               1, "Err", 0xC0}  
  
PriSec       { 0, "Primary ", 0xb0;  
               1, "Secondary", 0xb0}  
  
AliceState   { 0, " Off      ", 0x0C;  
               1, " Checkout ", 0xB0;  
               2, " Safe     ", 0xB0;  
               3, " Acquire  ", 0xB0}  
  
AcqMode      { 0, " Pixel ", 0x70;  
               1, " Histo ", 0x70}  
  
StateMode    { 0, " Off          ", BLACK on WHITE;  
               1, " Checkout      ", MAGENTA on LIGHTRED;  
               2, " Safe          ", GREEN on WHITE;  
               3, " Acquire-Pixel ", GREEN on WHITE;  
               4, " OFF (4)       ", GREEN on WHITE;  
               5, " Checkout      ", RED on WHITE;  
               6, " Safe          ", RED on WHITE;  
               7, " Acquire-Histo ", RED on WHITE}
```

4.2 Directory Structure

The GSEOS application consists of various system as well as configuration files. These can be moved to any location within the file system as long as the internal structure remains the same. No Registry settings are necessary or need to get changed in order to move the GSEOS directory to another location. The main directory that contains the gseos.exe executable and the GSEOS DLLs (*.pyd files) is referred to as GSEOS root directory.



The GSEOS root directory contains all the GSEOS executable files like gseos.exe, *.pyd files, as well as various configuration files, gseos.ini being the most important one. Block definition files as well as other configuration files can be located in the main directory as well. Typically this directory is named after your instrument or spacecraft, i.e.: Mimi

The Data directory is just by convention and receives the recorded files. The Demo and Sample directories contain sample files.

The Doc directory contains this file which will be invoked from the GSEOS help. The Include directory contains all the include and library files necessary to build Python extension modules for GSEOS. The Python directory and subdirectories contain all the Python files for the current distribution.

4.3 Gseos Python Interface

GSEOS uses the scripting language Python as its command and control language. You can interact with the GSEOS Python interpreter directly using the Console window. You can issue ordinary Python commands in this window and interface to GSEOS. The rest of this chapter describes the GSEOS Python interface. For further information on Python in general please refer to the Python Documentation on the Python home page at <http://www.python.org>.

The GSEOS interface to Python is relatively small. It provides support for commanding as well as Decoder, Monitor, and Sequencer modules. Your block definitions are exported as Python classes which allows you to access your real-time data easily through Python scripts.

Let's assume you defined a block called TLM in the block definition file with the following layout:

```
TLM
{
    Length      , , , 32;
    ApID        , , , 16;
    Data[800]   , , , 8;
}
```

To access the value of the ApID item in the TLM block you simply type `TLM.ApID` in the console window. This prints out the current value of the ApID item. Note, that if you would issue the same command again you may get a different value since your instrument may have generated a new TLM block. Besides reading items you can also write items. This is useful when you write a decoder script to generate derived data (e.g. de-subcommutation). To write an item you simply assign a value to it, e.g.:

```
PHA.Data[10:20] = 2
```

The above line would set all elements between 10 and 20 (not including 20) of the Data item in the TLM block to 2. However, at this point you have not generated a new data block! If you would read the data back you would not get 2! In order for the block to be generated you have to forward it to the system with the `send()` command:

```
PHA.send()
```

Python structures its modules in namespaces. The default namespace is `__main__`, this is the namespace you will see when using the console window. All the default GSEOS modules are imported into the `__main__` namespace. The command handling is implemented in the `GseosCmd` namespace which in turn is imported into the `__main__` namespace. E.g. to issue a command you would use the `send()` function. To send the byte sequence 1, 2, 3 to the command channel `DEV_POWER` you would simply type `cmd.send(DEV_POWER, 1, 2, 3)` in the console window. To use the `cmd` module in your own scripts you have to import it, e.g.:

```
import cmd
cmd.send(DEV_POWER, 1, 2, 3)
```

The following subchapters describe the various interface modules in detail.

4.3.1 Modules

4.3.1.1 `__main__`

The `__main__` module imports the GSEOS block and item definitions. You can access data items with the usual `Block.Item` notation. All the block definitions from the block definition files are imported so you can directly access them from the console window. If you want to use the block definitions in your scripts you have to include an `import __main__` statement at the top of your script. When you read data items you get their current value, that means two consecutive reads may result in two different values if the data changes in the meantime. To access arrays you use Python's slicing syntax. Keep in mind that writing a data item does not result in the block being posted to the GSEOS, nor can you read the value you just wrote! In order to make the data available to the system you have to call the `send()` function on the block once you have written all the necessary data items of that block. Check the Decoder module for more detailed information on how to create data blocks. Keep in mind that the block variables are shared resources, if you want to generate a block from multiple threads you have to synchronize access to the block variables. Refer to the Python thread module for more detailed information about threading and synchronization.

To write a Decoder or Monitor you want to take action whenever a new data block arrives in the system. You can create an instance of the Decoder or Monitor objects and add them to the list of Decoders or Monitors for that particular block. Your Decoder or Monitor will be called and evaluated every time a new block of that type arrives. Please refer to the documentation of the Decoder and Monitor modules for more detailed information.

GSEOS supports the concept of data sources. The Bios, Recorder, Net, etc. are all data sources. They are mutually exclusive, i.e. if you turn on the recorder for playback the Bios data source will be stopped and not generate any data (which otherwise could interfere with the recorder output). By default all blocks you generate from your Python scripts will be handled as decoder output (not as a data source). This means all your decoders will perform no matter what the input data source is. This is usually the behavior you would expect if you write a Decoder. If you plan on using Python as a data source you will have to first enable the Python data source by calling `GseosBDM.EnableDataSource(strName, bEnable)`. Once you call this function with the `bEnable` parameter set to 1 all data sources but Python will be disabled. All the blocks you want to generate from the data source take an additional parameter in the `send()` call. If you pass `TRUE` as the `bDataSource` parameter of the `send()` call your blocks will be generated from the Python data source and participate in the mutually exclusive data source scheme.

4.3.1.1.1 `GseosBDM.EnableDataSource`

`EnableDataSource(strName, bEnable)`

This function allows you to generate your blocks from the Python data source. Set `bEnable` to `TRUE` if you want to enable this data source, set it to `FALSE` if you want to disable it. Make sure to generate your blocks with the `bDataSource` parameter set to `TRUE` in the `send()` function.

Parameter	Description
<code>strName</code>	The name of your data source. This name will be displayed in the caption bar if the data source is active.
<code>bEnable</code>	<code>TRUE</code> to enable the data source, <code>FALSE</code> to disable it.

Returns

None

4.3.1.1.2 send

send([bCopyMode], [bDataSource])

The send member function is a method of all BDM blocks. You use this function to generate blocks of that particular type.

Parameter

bCopyMode

Description

Optional. If TRUE the contents of the block will be copied into the next block. The default is FALSE and you will get an uninitialized block after you sent off the current one.

bDataSource

Optional, defaults to FALSE if not specified. If TRUE the block is generated from the Python data source and is mutually exclusive with all other data sources in the system. To enable the data source call GseosBDM.EnableDataSource(). You usually specify this parameter only if you write a data source in a Python module.

Returns

None

Example

```
TestDec.Data[0:100] = TLM.Data[0:100]
TestDec.send()
```

4.3.1.2 Decoder

A decoder is a function that gets triggered on arrival of a specific block. It then reads data from this block and maybe other blocks and generates new blocks. In order for the decoder to be called on the arrival of a specific block it has to be registered with this block. To do this you have to create a decoder object and add it to the list of decoders for that block. Every time the block arrives your decoder will be executed. The decoders will be executed in the order they appear in the list of decoders for the block. Usually as the system grows you will need to refine your data products more and more. This will naturally lead to a layered system of blocks and decoders. You can display the decoder hierarchy in the GSEOS Explorer.

To construct a decoder you have to instantiate a decoder object of the decoder class. The constructor of the decoder object takes the unique decoder name, the decoder function you want executed and a list of blocks the decoder outputs. The decoder function itself takes one parameter. The block that triggers the decoder is passed into the decoder function. Therefore when you register one decoder for multiple blocks you will be able to distinguish which block arrived. In the list of output blocks you specify all blocks you may generate. This allows the system to give you an overview of the decoder hierarchy. You can manage your decoders with the GSEOS Explorer.

The decoder can be disabled by setting bEnable to false. It can be reenabled by setting bEnable to true. The variable dwCnt holds the number of times the decoder has been executed. You can reset this value if desired.

If the decoder raises an exception it will be disabled. You should catch all exceptions you

don't want to terminate the decoder.

It is advised that you give your decoder a documentation string. This documentation will be displayed in the GSEOS Explorer.

For more information please refer to the examples:

- Simple dispatch decoder
- Variable length spin decoder

4.3.1.2.1 bEnable

Property: bEnable

Enable or disable the decoder.

Example

Disable the decoder from a script:

```
oMyDec.bEnable = 0
```

4.3.1.2.2 constructor

Decoder(strName, fDecoder, ctChildBlocks)

The Decoder constructor creates a new decoder. You have to specify a unique name, a decoder function, and a container of child blocks generated by the decoder. In order to start the decoder you have to add it to the list of decoders for the block(s) you want to register your decoder for.

Parameter	Description
strName	The unique name identifies the decoder. This is the name you will see in the GSEOS Explorer.
fDecoder	Your decoder function. It takes one parameter which is the block that triggers the decoder (if you register the decoder for more than one block you can use this parameter to determine the source block). If this function throws an exception the decoder is terminated.
ctChildBlocks	The data blocks your decoder is going to generated. List all blocks your decoder may generate. Given this information the decoder hierarchy can be examined with the GSEOS Explorer.

Returns

The new decoder object.

Comments

In order for the decoder to be useful you have to hook it on the arrival of a block. You do this by adding it to the list of decoders for the block you are interested in. See the example below.

Exceptions

TypeError	The decoder name has to be a string.
DecoderError	The fDecoder parameter has to be a callable object. The child list contains invalid blocks.

Example

The following example defines a decoder function that simply converts the input data and generates one output block for every input block. It then sets the arrival hook for the input block.

```
#
# Simple conversion decoder
#
def fDec(oInputBlock):
    for i in range(len(oInputBlock.Data)):
        OutBlock.Data[i] = oInputBlock.Data[i] * 23
        OutBlock.send()

#
# Hook the decoder to the input block
#
oMyDec = Decoder.Decoder('Convert', fDec, [OutBlock])
InputBlock.Decoders.append(oMyDec)
```

4.3.1.2.3 Delete

Delete()

Deletes a decoder.

Returns

None

Comments

The GSEOS Explorer holds a reference to every decoder created. Even if your object goes out of scope the Explorer will hold on to the decoder. To release the decoder you have to call Delete() first and then destroy the decoder object itself (e.g. by letting it go out of scope). Usually there is no need to delete a decoder.

Example

Release our current decoder and then destroy the decoder object.

```
oMyDec.Delete()
oMyDec = None
```

4.3.1.2.4 dwCnt

Property: dwCnt

The number of invocations of the decoder. You can set this value if desirable.

4.3.1.2.5 Examples

4.3.1.2.5.1 Simple Decoder

This simple decoder dispatches one incoming block into different output blocks depending on the item:ApID. The ApID determines the type of the block. Lets say we

have four different data packets which are all received through one common block: RawTLM. The block definition of the RawTLM block is given below:

```
RawTLM
{
    ApId      , , , 16;
    Length    , , , 32;
    Data[4000] , , , 8;
}
```

The Length item indicates the number of valid data bytes in the Data item. Our four destination blocks are Sc, Hk, Pha, Rates with the following block definitions:

```
Sc
{
    Length    , , , 32;
    Data[4000] , , , 8;
}
```

```
Hk
{
    Length    , , , 32;
    Data[4000] , , , 8;
}
```

```
Pha
{
    Length    , , , 32;
    Data[4000] , , , 8;
}
```

```
Rates
{
    Length    , , , 32;
    Data[4000] , , , 8;
}
```

The ApIDs for the various blocks are listed in the table below:

ApId Block

1	Sc
2	Hk
3	Pha
4	Rates

The decoder function has to determine which destination block to generate depending on the ApID and then copy the data depending on the Length field to the destination block. Finally we have to set the Length field of the destination block and forward it to the system. Here the decoder function:

```
#
# Import block definitions, Decoder class and other stuff
#
from __main__ import *
import Decoder
import Gseos
```

```
#
# RawTLM Block decoder Function
#
def fDecRawTLM(oBlock):
    "RawTLM decoder dispatches raw data into high level packets"

    wLength = oBlock.Length

    #
    # If we have any clue of the distribution of the blocks
    # order the following switch by frequency.
    #

    if oBlock.ApID == 1:
        oDest = Sc

    elif oBlock.ApID == 2:
        oDest = Hk

    elif oBlock.ApID == 3:
        oDest = Hk

    elif oBlock.ApID == 4:
        oDest = Hk

    #
    # Oops, invalid type id. Log error to message window and ignore.
    #
    else:
        GseosCmd.msg('DecTLMRaw: Invalid ApId %d received' % oBlock.ApID)
        return

    #
    # Copy the data to the destination block and send it off.
    #
    oDest.Data[:wLength] = oBlock.Data[:wLength]
    oDest.Length = wLength
    oDest.send()
```

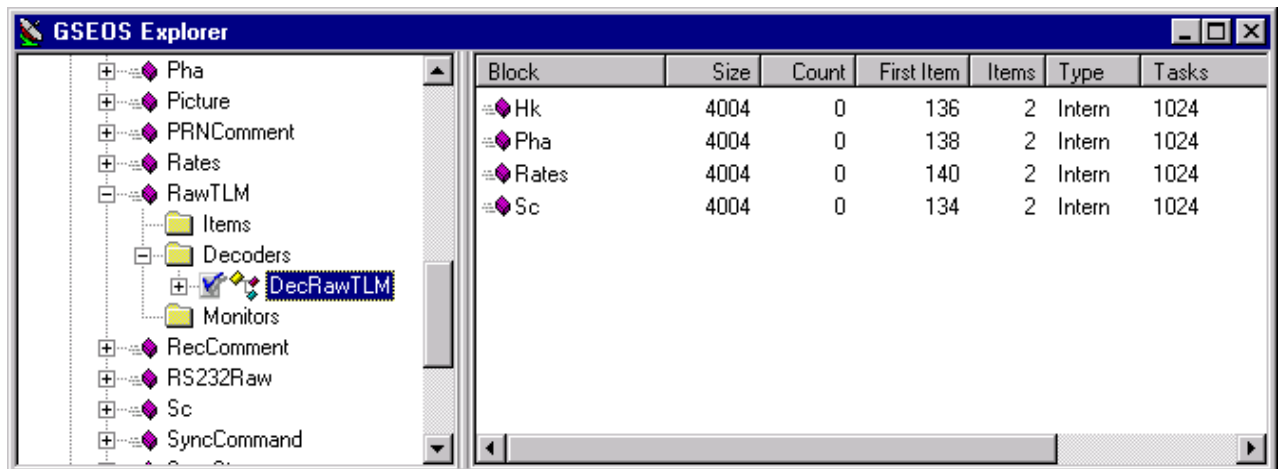
Once we have the decoder function defined we have to assign it to the RawTLM block to get executed:

```
#
# Create a new decoder.
#
oDecRawTLM = Decoder.Decoder('DecRawTLM', fDecRawTLM, [Sc, Hk, Pha, Rates])

#
# Assign it to the RawTLM block.
#
RawTLM.Decoders.append(oDecRawTLM)
```

After we execute this script we can examine the decoder in the GSEOS Explorer. If we choose the Block node and list all the blocks in the system we should see our RawTLM source block. Extending the Decoders node beneath the RawTLM block lists our decoder and indicates that it is running. On the right hand pane we can also see the output blocks generated by the RawTLM decoder.

By right-clicking on the decoder node you can also disable or delete the decoder.



If we now further expand the DecRawTLM decoder node we can traverse the entire decoder hierarchy. In our case we don't have any child blocks after the Sc, Hk, Pha, Rates blocks.

To run this example please install the dec1.blk block definition file in the demo/dec1 folder, restart GSEOS and load the dec1.py decoder script from the same directory.

4.3.1.2.5.2 Variable Length Decoder

Oftentimes the data is not formatted at fixed boundaries but 'variable length'. The following decoder gives an example of how to deal with this kind of data. Here we get a block called Event that contains 8 spins with time-of-flight data. The spins variable length and 'packed' in the Events TOFData item. The length of the following spin is indicated in the word preceding the spin data. Our job is to 'walk' the linked list of spin data fields, and copy the data from the source block into the TOFSpin destination block. Below are the definitions of the Event source block and the TOFSpin destination block. If you plan to install this example please load the Dec2.blk block definition file by entering it in the gseos.ini file in the Config/BlkFiles section. Run the dec2.py sample decoder to install the decoder.

```

*
=====
*      Event Data Blocks
*
=====
Event {
    (Data[6008]      , , , 8;)
    Met              , , , 32;
    SpinVern         , , , 8;
    TOFData[3000]    , , , 16;
    Cksum            , , , 8;
}

```

```

TOFSpin {
    Cnt[8]          ,,,16;
    DataSp0[260]    ,,,16;
    DataSp1[260]    ,,,16;
    DataSp2[260]    ,,,16;
    DataSp3[260]    ,,,16;
    DataSp4[260]    ,,,16;
    DataSp5[260]    ,,,16;
    DataSp6[260]    ,,,16;
    DataSp7[260]    ,,,16;
}

```

The following decoder script gets a local copy of the destination data items for quicker access during decoder runtime. It takes advantage of the naming of the items in the TOFSpin block and loops over the dictionary extracting the items by name. It then defines the decoder function fDecEvent and registers it for the Event block. As you can see it is not necessary to create a temporary decoder object which you then append to the decoder list of the Event block. Instead you can simply pass the result of the constructor into the append function directly.

```

#
# Decode Event Data
# The Event Data consists of 8 spins with variable length data. Each spins
# data is prepended with a count indicating the length of the following
# data
# field. Walk the list of spins and extract the event count and event data
# for every spin and assign it to the appropriate items of the TOFSpin
# Block.
#
from __main__ import Event, TOFSpin
import Decoder

#
#
# Get the SpinData items from the TOFSpin block. Instead of assigning
# one array item at a time we loop over the items accessing the
# dictionary directly.
#
SpinData = []
for wSpin in range(8):
    SpinData.append(TOFSpin.__dict__['DataSp'+str(wSpin)])

def fDecEvent(Event):
    "TOF Event Decoder"
    wLengthPos = 0

    #
    # Loop over all spins, extract the length of the following spin, and
    # copy the spin data. The advance to the next spin.
    #
    for wSpin in range(8):
        wSpinLen = Event.TOFData[wLengthPos]
        TOFSpin.Cnt[wSpin] = wSpinLen

        #
        # Get the spin data

```

```

#
wPos = wLengthPos+1
SpinData[wSpin][:wSpinLen] = Event.TOFData[wPos:wPos+wSpinLen]

wLengthPos = wPos + wSpinLen
#
# Now we have filled in all our spins, ship the block to the system.
#
TOFSpin.send()

#
# Hook the decoder on arrivals of Event blocks.
# Don't bother creating a temporary decoder object.
#
Event.Decoders.append(Decoder.Decoder('TOF Event Decoder', fDecEvent,
[TOFSpin]))

```

4.3.1.2.6 strName

Property: strName

The unique name of the decoder. You should not change this name after you created the decoder. Rather delete the current one and create a new one with the desired name.

4.3.1.3 Gseos

The Gseos module implements several helper functions related to the GSEOS application. You should `import Gseos` whenever you need to access application specific services. The function `help()` which is implemented by this module is also directly available in the `__main__` namespace. It displays this helpfile. Most of the functions also have a doc string which you can access with: `Gseos.function.__doc__`

The `FileMenu()` function allows you to hook your own file types into the GSEOS file handling. The `Log()` and `LogSave()` functions are used to append text to a GSEOS log and to save the log file respectively. The `MessageBox()` and `InputDialog()` functions give you simple input dialog boxes. The function `PumpWaitingMessages()` has a special meaning: If you plan on performing time consuming computations in your scripts you will effectively block the GSEOS user interface. To give the user interface a chance to operate you should intersperse calls to `PumpWaitingMessages()` into your script (preferably into the inner loop).

4.3.1.3.1 Gseos.FileMenu

FileMenu(fCallback, strFilterName, strExt, wFlags, [bPriority])

The `FileMenu()` function allows you to hook into the GSEOS file handling and use the common File/Open, File/SaveAs, etc. menus for your own file types. When you register for file handling your callback routine will be called with the file name the user selected and the operation he wants to perform on the file. You can register for all supported file modes: New, Open, Append, SaveAs.

Parameter fCallback

Description

The callback function should take two parameters: `fCallback(strFile, wMode)` Where `strFile` is the file name the user selected and `wMode` is one of the file modes specified in the flags parameter.

strFilterName	Description of the filter name. It should contain the extension, e.g. "Image Files (*.img)".
strExt	The file extension including ".*", e.g. "*.img".
wFlags	Specifies the file mode to register for. If this parameter is 0 this filter is removed from the file handling. wFlags can be one or more of the following constants. If you specify just one parameter you can simply use the constant, for more than one parameter you have to supply the values in a tuple. Here the constants valid for wFlags: REG_FILENEW, REG_FILEOPEN, REG_FILEAPPEND, REG_FILESAVEAS
bPriority	Optional, specifies the order in the list. Smaller numbers occur higher up in the list. You should not install your custom filters before or in between the standard GSEOS filters. If you don't specify this parameter the filter is appended at the end of the list.

Returns

None

Comments

When your callback function is called all it gets passed is the name of the file the user wants to operate on and the mode flag. The file is not opened or touched in any way. The file operations you want to perform are up to you. However, if the callback function is called with REG_FILEOPEN it is guaranteed that the file exists.

Example

The following sample defines a callback function that processes the file a user selects from the File/Open menu. It then registers the filter with GSEOS:

```
from Gseos import *

def fOnImgFile(strFileName, wMode):
    if (wMode == REG_FILEOPEN):
        fProcessMyFile(strFileName)

    elif (wMode == REG_FILESAVEAS):
        fSaveMyResult(strFileName)

# Register with GSEOS
FileMenu(fOnImgFile, 'Image Files (*.img)', '*.img', (REG_FILEOPEN,
REG_FILESAVEAS))
```

4.3.1.3.2 Gseos.FileOpenDialog

FileOpenDialog([strDirectory], [strFileName])

The FileOpenDialog() function opens a Windows file open dialog that allows the user to select a file.

Parameter	Description
strDirectory	Optional. The initial directory to set the dialog box to.
strFileName	Optional. The file name to preset the dialog box to.

Returns

The file name of the file the user selected, None if the user canceled the selection.

4.3.1.3.3 Gseos.GetInstance

GetInstance()

Gets the instance number of the currently running GSEOS instance. It is possible to start multiple instances of GSEOS at the same time. If you specify the \I command line parameter you can specify an instance number. This allows for different configurations being launched for different instances. Check the gseos.ini file for more information on instance settings.

Returns

The instance number.

Example

```
Gseos.GetInstance()
```

4.3.1.3.4 Gseos.GetProjectPath

GetProjectPath()

The GSEOS project path is the directory the gseos.exe application resides in. This is the default directory for all file operations. This function returns the project path if you need to access files relative to the GSEOS project path.

Returns

The project path.

Example

Get the project path:

```
>>> Gseos.GetProjectPath()  
'c:\\gseos5.0\\prj\\messenger\\'
```

4.3.1.3.5 Gseos.help

help()

Shows the online help (this file).

Returns

None

Comments

This command is also imported into the __main__ namespace. That means that you can simply type 'help' in the console window to get the help pages.

Example

The following examples displays this file.

```
help()
```


4.3.1.3.6 Gseos.InputDialog

InputDialog(strText, [strTitle])

Displays an input dialog box. The text strTitle is displayed in the title bar of the dialog box. The text strText is displayed as the input prompt.

Parameter	Description
strText	The input dialog prompt text.
strTitle	Optional, The caption bar text. Default: GSEOS.

Returns

The text entered by the user or None if the user selected cancel.

Example

Display an input box asking for a HV level:

```
Gseos.InputDialog('Please enter the new HV level in [kV]', 'HV Level')
```

4.3.1.3.7 Gseos.InputDialogModeless

InputDialogModeless(strText, [strTitle])

Displays a modeless input dialog box. The text strTitle is displayed in the title bar of the dialog box. The text strText is displayed as the input prompt. This function is the modeless counterpart of the InputDialog() function. It allows you to perform other operations within GSEOS while the dialog box is open.

Parameter	Description
strText	The input dialog prompt text..
strTitle	Optional, The caption bar text. Default: GSEOS.

Returns

The text entered by the user or None if the user selected cancel.

Example

Display an input box asking for a HV level:

```
Gseos.InputDialogModeless('Please enter the new HV level in [kV]', 'HV Level')
```

4.3.1.3.8 Gseos.Log

Log(strLogFile, strText, [sColor], [byAttr])

Append text to a log file. The text strText is appended to the file strLogFile. The Log command can even be issued if the corresponding Log window is not open. In that case the text will be written directly to the file (a LogSave() is not necessary). If the file can't be written the function throws an IOError.

Parameter	Description
strLogFile	The file name of the log file. You can specify a relative file name. The

	default extension for log files is *.log.
strText	The text to be appended to the the log file..
sColor	Optional, an RGB tuple (iii) specifying the color of the text to insert. The module defines several color definitions you can use instead of the RGB representation: BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, LTGRAY, GRAY, LTBLUE, LTGREEN, LTCYAN, LTRED, LTMAGENTA, YELLOW, WHITE.
byAttr	Optional, specifies the text attribute. One or more of the following attributes can be specified: BOLD, ITALIC, UNDERLINE. The default turns all attributes off.

Returns

None

Comments

The color and text attributes are only displayed as long as the window is not closed. After you reopen the window all these attributes will be gone. The reason for that is that the text is stored in plain ASCII format which makes it easier for standard text editors and processing software to handle the text. The text appended to the file is not written to disk immediately. If you want to access the log file from another program you should use the LogSave() function first.

Example

Log a comment:

```
Gseos.Log('testprotocol.log', 'Start of test procedure A', gseos.BOLD)
```

4.3.1.3.9 Gseos.LogReload

LogReload(strLogFile)

The log file window is updated with the contents of the file from disk. This is useful when the contents have been changed from another process.

Parameter**Description**

strLogFile

The file name of the log file to reload.

Returns

None

Example

Reload the log file from the previous example:

```
LogReload('testprotocol.log')
```

4.3.1.3.10 Gseos.LogSave

LogSave(strLogFile)

Save the log file to disk. The log file is not written to file with every Log() update but when the window is closed. In order to access the log contents from outside the application you have to save it.

Parameter	Description
strLogFile	The file name of the log file. The default extension for log files is *.log.

Returns

None

Example

Save the log file from the previous example:

```
LogSave('testprotocol.log')
```

4.3.1.3.11 Gseos.MakePathRelative

MakePathRelative(strPath)

Returns a path relative to the GSEOS project path. To get the GSEOS project path use the function GetProjectPath(). If strPath is not a subdirectory of the GSEOS project path an absolute path will be returned.

Parameter	Description
strPath	The absolute path.

Returns

The path relative to the GSEOS project path.

Example

Make the path relative to the project path:

```
>>> Gseos.MakePathRelative('c:\\gseos5.0\\prj\\messenger\\xrs')  
'xrs'
```

4.3.1.3.12 Gseos.MessageBox

MessageBox(strText, [strTitle], [cButtons], [cIcon])

Displays a message box. The text strTitle is displayed in the title bar of the message box. The text strText is displayed as the message prompt. The standard windows buttons can be specified in the cButtons parameter. This message box is a modal window, that means you can't access any other GSEOS user interface elements until you close the message box. If you need a modeless message box use the MessageBoxModeless() function.

Parameter	Description
strText	The message prompt text.
strTitle	Optional, The caption bar text. Default: GSEOS.
cButtons	Optional, specifies the buttons to display. One of the following attributes can be specified: MB_OK, MB_OKCANCEL, MB_RETRYCANCEL, MB_YESNO, MB_YESNOCANCEL. The default is MB_OK.
cIcon	Optional, specifies an icon to display. MB_ICONEXCLAMATION, MB_ICONINFORMATION, MB_ICONQUESTION, MB_ICONSTOP.

Returns

One of the following constants depending on the user action: IDCANCEL, IDNO, IDOK, IDRETRY, IDYES.

Example

Display a message box that prompts the user to confirm issuing a dangerous command:

```
import Gseos
Gseos.MessageBox('You are about to increase the HV level\n Do you want to
continue?',
                 'Confirm hazardous command', MB_YESNO, MB_ICONEXCLAMATION)
```

4.3.1.3.13 Gseos.MessageBoxModeless

MessageBox(strText, [strTitle], [cButtons])

Displays a modeless message box. This is useful if you want to prompt the user for input while still being able to interact with the GSEOS main window. If you need a modal message box please use Gseos.MessageBox(). In general you want the behavior of a modal message box and stop the execution of your script until the user provides the requested input.

The text strTitle is displayed in the title bar of the message box. The text strText is displayed as the message prompt. The standard windows buttons can be specified in the cButtons parameter.

Parameter	Description
strText	The message prompt text.
strTitle	Optional, The caption bar text. Default: GSEOS.
cButtons	Optional, specifies the buttons to display. One of the following attributes can be specified: MB_OK, MB_OKCANCEL, MB_RETRYCANCEL, MB_YESNO, MB_YESNOCANCEL. The default is MB_OK.

Returns

One of the following constants depending on the user action: IDCANCEL, IDNO, IDOK, IDRETRY, IDYES.

Example

Display a modeless message box.

```
import Gseos
Gseos.MessageBoxModeless('You are about to increase the HV level\n Do you
want to continue?', 'Confirm hazardous command', MB_YESNO)
```

4.3.1.3.14 Gseos.PumpWaitingMessages

PumpWaitingMessages()

The function PumpWaitingMessages() allows all message queues to be processed and therefore enables GSEOS to run while a lengthy script is being processed. Call this function several times (probably within a time consuming loop) to keep the system responsive. An alternative to using this cooperative preemption scheme is the use of threads.

Returns

None

Example

The following example does some lengthy processing and enables the foreground program to execute by calling `PumpWaitingMessages()`.

```
for i in range(0, 2000):  
    fThisTakesAbout200ms()  
    G  
seos.PumpWaitingMessages()
```

4.3.1.3.15 Gseos.SetActiveDesktopPage

SetActiveDesktopPage(strPageName)

Sets the active Desktop page to the one specified by `strPageName`.

Parameter	Description
<code>strPageName</code>	The name of the desktop page to activate.

Returns

None

If a page with the name `strPageName` does not exist a `ValueError` exception is thrown.

Example

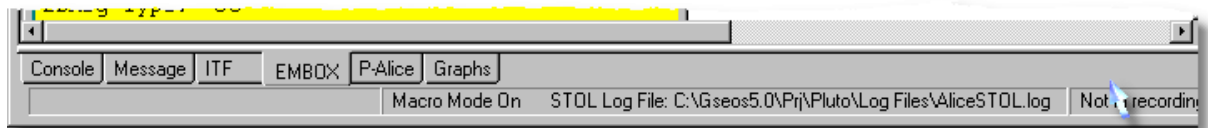
Switches to the 'Test' page:

```
import Gseos  
Gseos.SetActiveDesktopPage('Test')
```

4.3.1.3.16 Gseos.SetStatusText

SetStatusText(strText)

Sets the text in the status bar of the GSEOS main window.



Parameter	Description
<code>strText</code>	The text to display in the status bar.

Returns

None

Example

```
import Gseos
Gseos.SetStatusText('STOL Log File: C:\\Gseos\\Prj\\Pluto\\Log')
```

4.3.1.3.17 Gseos.ShellExecute

ShellExecute(strPageName)

Invokes the default application that is associated with the document passed in. I.e. if your htm documents are associated with Internet Explorer and you call ShellExecute("index.htm") the browser will be launched with index.htm loaded.

Parameter	Description
strDocument	The document to view.

Returns
None

If an error occurs, like the file can not be found, a RuntimeError exception will be thrown.

Example
Open this file in HTML help:

```
import Gseos
Gseos.ShellExecute("doc\\gseos.chm")
```

4.3.1.3.18 Gseos.WaitDialog

WaitDialog(iTimeout, strText, [strTitle])

Displays a wait dialog box. The text strTitle is displayed in the title bar of the DialogBox. The text strText is displayed as the message prompt. A timeout value iTimeout specifies the timeout in seconds. The timeout is counted down and can be interrupted with the 'Skip wait' button. The 'Abort' button returns also but with a result value of IDCANCEL. If the timeout expires the function returns and the dialog box closes.

Parameter	Description
iTimeout	The timeout value in seconds.
strText	The prompt text.
strTitle	Optional. Specifies the title bar of the dialog, if not specified defaults to 'GSEOS'.

Returns
One of the following constants depending on the user action: IDCANCEL, IDOK. If the Abort button was pressed IDCANCEL is returned, otherwise IDOK.

4.3.1.4 Gseos.Blocks

The GseosBlocks module replaces the old Blocks module. It allows easy access to the GSEOS block definitions.

It provides to containers that allow access to blocks:

Blocks: This dictionary is keyed by block name and contains objects of type GBlockProps.

BlocksByHandle: A list that is keyed by block handle. The values contained are objects of type GBlockProps. See below for a description of these objects.

GBlockProps

You can access the following attributes of a GBlockProps object:

Block: The actual block object
Items: A list with all the item names of this block
EnableSelect: This attribute controls if the block is displayed in the block and item select dialog boxes. This lets you simplify the user interface for complex configurations (i.e. multiple instruments defining all their blocks). By default EnableSelect is True, if you set it to False the block will be taken of the select list. However, the block is still accessible from Python and is otherwise not changed at all.

The following sample removes all blocks with block names starting with 'XRS_' from the select block and select item dialog:

```
#
# Remove all blocks starting with 'XRS_'
#
import GseosBlocks

for strBlockName in GseosBlocks.Blocks.keys():
    if strBlockName[:4] == 'XRS ':
        GseosBlocks.Blocks[strBlockName].EnableSelect = False
```

To simulate the old Blocks module Blocks list you can use the following code:

```
#
# The old code: Blocks.Blocks[hBlock] is equivalent to:
#
GseosBlocks.BlocksByHandle[hBlock].Block
```

4.3.1.5 GseosCmd

The GseosCmd module implements the GSEOS command interface. The commands available to you highly depend on your particular instrument and the hardware interface to your instrument. In general all commands are byte sequences. The command module implements the notion of command channels. You can use the command channels from 100 to 250 for instrument specific purposes. The assignment of command channels to your instrument hardware depends on your BIOS module that handles all low level commanding and hardware abstraction. The most important function in the cmd module is send(). This function allows you to send byte sequences to various command channels. You will usually implement a higher level of commanding on top of this function instead of using it directly. Lets assume you have a power channel that allows you to power various subsystems on the instrument. You then may want to define the following command definitions:

```
#
# Define the power command channel
```

```
#
DEV_POWER = 102

#
# Define subsystem states
#
PRW_ON  = 1
PRW_OFF = 0

#
# Define subsystem IDs
#
SUB_HTR1 = 1
SUB_HTR2 = 2
SUB_HV   = 3

#
# Define the subsystem power command
#
def SubSystemPower(SubSystem, bPower):
    send(DEV_POWER, (SubSystem, bPower))
```

4.3.1.5.1 GseosCmd.batchstart

batchstart(szFileName)

The batchstart function starts the execution of a GSEOS batch file. To stop a running batch file issue batchstop. See batch files for a description of the GSEOS batch file format.

Parameter	Description
szFileName	The name of the batch file you want to start. Batch files have the file extension .cpb.

Returns

None

Comments

If the file specified can not be opened no exception is thrown. The command processor handles this error and pops up a dialog box. If you use this function in a script you may want to open the file to make sure it exists before calling batchstart if this behavior is not desirable.

Example

```
GseosCmd.batchstart("mybatch.cpb")
```

4.3.1.5.2 GseosCmd.batchstop

batchstop()

The batchstop function stops the currently executing batch file.

Returns

None

Comments

As opposed to the batch stop function of the command processor the cmd.batchstop function does not ask for confirmation.

Exceptions

RuntimeError No batch file is currently executing.

Example

Stop the currently executing batch job.

```
GseosCmd.batchstop()
```

4.3.1.5.3 GseosCmd.msg

msg(szMessage)

The msg function logs a message to the message window.

Parameter	Description
szMessage	The message to print.

Returns

None

Comments

This function emulates the msg command of the command processor and can be used the same way as for the legacy command processor.

Example

Print a message into the message window.

```
GseosCmd.msg("Hello, World")
```

4.3.1.5.4 GseosCmd.send

send(byChannel, ctData)

The send function issues a command to the underlying hardware. Refer to your instrument documentation regarding the commands supported.

Parameter	Description
byChannel	The command channel to send the command to. The channel number allows to address different hardware devices (depending on the BIOS). The range from 100 to 200 is available for your own command devices you may want to define. 0 to 99, and 201 to 255 are reserved for system purposes.
ctData	The data you want to send. This parameter accepts a single value or a sequence. A sequence can contain nested sequences. A sequence is a

tuple, a list, or a string. If you use nested sequences the structure is resolved depth first left to right. Strings are resolved character by character, a terminating zero is not appended. A number or a character is a terminal node and has to be in the range -127 to 255 (it has to be a byte sequence). If you need to send words you can use a function that takes a word and returns a tuple of bytes (using your required endianness). Since only bytes are accepted the issue of endianness does not arise. The total number of terminal nodes (bytes) in ctData can not be larger than 256.

Returns

None

Comments

This is the lowest level function available for commanding. You generally want to wrap it into higher level command functions which you then in turn assign to command buttons, command menus and so on.

Exceptions

- | | |
|---------------|--|
| ValueError | One or more values of the ctData parameter are out of bounds (-127 to 255) or can not be converted into an integer value. The byChannel parameter is out of bounds (0 to 255). |
| TypeError | A value passed into the ctData parameter is not of type sequence (tuple, list, string) or number. |
| OverflowError | The total number of bytes in the ctData sequence exceeds 256. |

Example

The following examples show different sequences passed into cmd.send().

```
#
# Just a simple integer
#
cmd.send(22, 3)

#
# A tuple
#
cmd.send(22, (1,2,3))

#
# A string
#
cmd.send(22, "Hello, World")

#
# A nested sequence
#
GseosCmd.send(22, ("This goes out first", 2, [3, 4, 5, "Another string",
(6, 7, 8)]))
```

4.3.1.5.5 GseosCmd.sound

sound(szWaveFile)

The sound function plays a wave file.

Parameter	Description
szWaveFile	The wave file to play.

Returns

None

Comments

This function emulates the sound macro defined in the system.cpd file. This macro is obsolete, use the sound function instead.

Exceptions

RuntimeError	The wave file could not be played.
--------------	------------------------------------

Example

Play a wave file.

```
GseosCmd.sound("boing.wav")
```

4.3.1.5.6 GseosCmd.winexec

winexec(szApplication, [iShow])

The winexec function launches an application.

Parameter	Description
szApplication	The application to start.
iShow	The initial state of the application. This parameter is optional. The default value is 1 (show normal). 0: Hide the application 1: Show normal 2: Show minimized 3: Show maximized

Returns

None

Comments

This function implements the Windows API function WinExec().

Exceptions

RuntimeError	The application could not be started.
--------------	---------------------------------------

Example

Start notepad.

```
GseosCmd.winexec("notepad.exe", 1)
```

4.3.1.6 GseosConsole

The console module implements the GSEOS console interface. You usually don't need to access the console module directly. In fact, tampering with this module may disable all console output. This module is not imported into the `__main__` namespace by default. In order to use the console module directly you have to import it.

4.3.1.6.1 GseosConsole.off

off()

Disables the console window.

Returns

None

Comments

Be careful with this command. If you issue this command from the console window it will be the last one! To turn the console window back on you can use the command line editor and issue `GseosConsole.on()`.

Example

The following examples disables the console window.

```
GseosConsole.off()
```

4.3.1.6.2 GseosConsole.on

on()

Enables the console window.

Returns

None

Comments

This command enables the console output. To issue this command from the console window the console has to be on to begin with! Therefore if the console window is disabled you can not switch it on from within the console window. You can use the command line editor to issue the command.

Example

The following examples enables the console window.

```
GseosConsole.on()
```

4.3.1.6.3 GseosConsole.write

write(szText, [iRed, iGreen, iBlue])

The write function writes text to the console window. All output (from print commands

and so on) is redirected to this function. You can use it directly. The RGB parameters are optional and allow you to specify any text color. The default color is black.

Parameter	Description
szText	The text to write to the console window.
iRed	Optional parameter which specified the red component of the text color. This value can be in the range 0 to 255.
iGreen	Optional parameter which specified the green component of the text color. This value can be in the range 0 to 255.
iBlue	Optional parameter which specified the blue component of the text color. This value can be in the range 0 to 255.

Returns

None

Comments

If this function actually outputs any data depends on the enable state of the console window. If the console is disabled all output will be suppressed. You usually don't use this function directly, however all standard Python functions you invoke that output any text will use this function indirectly. After your text is output a prompt will be displayed. In order to start the prompt on a new line you should include a carriage return character at the end of your output.

Example

The following examples prints some text in red to the console window.

```
GseosConsole.write("Hello, World", 255, 0, 0)
```

4.3.1.7 GseosConvert

The block definitions in GSEOS are not typed. This means all items are interpreted as raw bit storage with no sign bit or floating point format. You may run into the problem that your items are not raw binary data but signed values or floating point data. The GseosConvert module offers functions to convert between various data representations:

ftol():	Performs a float to long conversion.
ltof():	Performs a long to float conversion.
signed():	Performs an unsigned to signed conversion.

4.3.1.7.1 GseosConvert.ftol**ftol(flValue)**

Performs a float to long conversion. The binary representation of the IEEE floating point value is returned as a long. This function does not coerce to a long but interprets the floating point value as a signed long.

Parameter	Description
flValue	The IEEE floating point value to convert.

Returns

The binary representation of the passed in floating point value as a signed long.

Example

The following example converts a floating point value to a binary representation that can be assigned to a 32-bit data item:

```
flTemp = 72.34

HK.Temp = GseosConvert.ftol(flTemp)
```

4.3.1.7.2 GseosConvert.ltof

ltof(IValue)

Performs a long to float conversion. The binary representation of the long value is returned as a floating point value. This function does not coerce to a float but interprets the long parameter as a floating point value.

Parameter	Description
IValue	The long value to convert.

Returns

The corresponding IEEE floating point representation.

Example

The following example converts a long value (which is the binary representation of an IEEE float) into a float:

```
flTemp = 72.34
lTemp = GseosConvert.ftol(flTemp)

print GseosConvert(.ltof(lTemp))

>>> 72.34
```

4.3.1.7.3 GseosConvert.signed

signed(IValue, wSignBit)

Interpretes an unsigned BDM item as a signed value. The wSignBit parameter specifies the bit to interpret as sign bit. (This must be the most significant bit in the item, all bits to the left of the sign bit will be overwritten with the sign expansion).

Parameter	Description
IValue	Unsigned long value to convert into a signed value.

Returns

The sign converted value.

Example

The following example assigns -2 to a 9 bit long data item. It then prints the results after it posted the block to the system. The value is 510. To get the signed value we use the signed conversion function and specify bit 8 as sign bit (the MSB).

```
HK.Temp = -2
HK.send()

print HK.Temp

>>> 510

print GseosConvert.signed(HK.Temp, 8)

>>> -2
```

4.3.1.8 GseosMsgWindow

The GseosMsgWindow module gives access to the GSEOS Message window. You can open a new message window with the New() function. You can close the message window using Close(), to clear the contents of the message window you use Clear(). BringToTop() brings the window to the foreground, and Print() prints it.

4.3.1.8.1 BringToTop

Enter topic text here.

4.3.1.8.2 Clear

Enter topic text here.

4.3.1.8.3 Close

Enter topic text here.

4.3.1.8.4 New

Enter topic text here.

4.3.1.8.5 Print

Enter topic text here.

4.3.1.9 GseosNet

The GseosNet module exposes the Python interface to the GSEOS networking functionality.

The status of the network module as a datasource can be queried with IsEnabled(). Enable() and Disable() will activate or deactivate the network as a data source.

The client and server connections can be queried and managed with the client and server methods. These methods take the connection name as a parameter to identify the connection to manipulate. ClientConnect() will try to establish a connection and ClientDisconnect() will terminate a connection if it is connected. ClientStatus() returns the status of the connection.

ServerStatus() returns the current status of a server connection and ServerReset() terminates a server connection if it is connected and goes back to listening on the configured server port.

4.3.1.9.1 GseosNet.ClientConnect

ClientConnect(strClientName)

Connect the client connection. This function returns instantly without blocking for the connection to be established. In order to find out if the connect call was successful use the ClientStatus() function. Establishing a connection may take several seconds so polling in a separate thread is one solution to determine the success or failure of the connect call. Performs a float to long conversion. The binary representation of the IEEE floating point value is returned as a long. This function does not coerce to a long but interprets the floating point value as a signed long.

Parameter	Description
strClientName	The name of the client connection to connect.

Returns
None.

4.3.1.9.2 GseosNet.ClientDisconnect

ClientDisconnect(strClientName)

Disconnect the client connection.

Parameter	Description
strClientName	The name of the client connection to disconnect.

Returns
None.

4.3.1.9.3 GseosNet.ClientStatus

ClientStatus(strClientName)

Returns the current connection status. It can be one of the following constants defined in this module:

GseosNet.NOTCONNECTED
GseosNet.CONNECTING
GseosNet.CONNECTED

Parameter	Description
strClientName	The name of the client to get the status of.

Returns
The connection status.

4.3.1.9.4 GseosNet.Disable

Disable()

Disable the network as the current data source. All data coming from the network is discarded.

Note:

Network sources can be configured to be non-exclusive (see the last setting in the configuration section below). This means that the data will be generated even if the network is not enabled. For these data sources the Disable() function does not discard the incoming data.

```
[CmdSrc]
IP-Address=127.0.0.1
;IP-Address=128.125.143.61
Source=Event
Port=2002
;AutoConnect=0
Exclusive=No
```

Returns

None.

4.3.1.9.5 GseosNet.Enable**Enable()**

Enable the network as the current data source. All data coming from other data sources than the network (i.e. Recorder, Bios, etc.) is discarded.

Note:

Network sources can be configured to be non-exclusive (see the last setting in the configuration section below). This means that the data will be generated even if the network is not enabled. For these data sources the Enable() function has no effect (they will generate data blocks regardless of the network being enabled as a data source or not).

```
[CmdSrc]
IP-Address=127.0.0.1
;IP-Address=128.125.143.61
Source=Event
Port=2002
;AutoConnect=0
Exclusive=No
```

Returns

None.

4.3.1.9.6 GseosNet.IsEnabled**IsEnabled()**

Returns the status of the network. If the network is enabled all other data sources in the system are disabled. However, even if the network is disabled it can generate data blocks if the network connection is tagged as Non-Exclusive (see Disable() and Enable() functions also).

Returns

True if the network is enabled, false otherwise.

4.3.1.9.7 GseosNet.ServerReset

ServerReset(strServerName)

Reset the server connection. The server will go back into listen status. If a connection is established it will be disconnected before the server goes back to listen mode.

Parameter	Description
strServerName	The name of the server to reset.

Returns
None.

4.3.1.9.8 GseosNet.ServerStatus

ServerStatus(strServerName)

Returns the current connection status. It can be one of the following constants defined in this module:

GseosNet.NOTCONNECTED
GseosNet.CONNECTING
GseosNet.CONNECTED
GseosNet.LISTEN

Parameter	Description
strServerName	The name of the server to get the status of.

Returns
The connection status.

4.3.1.10 GseosRecorder

The GseosRecorder module exposes the Python interface to the GSEOS Recorder functionality.

The status of the Recorder module can be queried with `IsRecording()` and `IsPlayingBack()`. Use `StartRecording()` and `StopRecording()` to start and stop the Recorder respectively.

To modify the blocks to be recorded or to be played back you can use the following functions:

`AddRecordBlock()`, `RemoveRecordBlock()`, `AddPlaybackBlock()`, and `RemovePlaybackBlock()`. `GetRecordBlocks()` and `GetPlaybackBlocks()` return lists with the blocks currently on the record/playback list.

To access and change the file prefix and the Recorder data path you can use `GetPrefix()`, `SetPrefix()`, and `GetDataPath()`, and `SetDataPath()`. The `SetPrefix()` and `SetDataPath()` functions will also write the new value to the `gseos.ini` file so it will be preserved across invocations of the software.

Getting status information on playback data

In order to get status information about played back data blocks you can refer to the RecBlockStatus block. If you select this block in the playback list it will be generated for every played back block (except the RecBlockStatus block itself, of course).

The RecBlockStatus block should be defined in the system.blk file, if you don't have this block defined the mechanism to get status information on recorded data is disabled.

The RecBlockStatus is not a block that is recorded but a block that gets generated for every played back block and contains meta-data for that block. The time field contained in the RecBlockStatus block indicates the time the block was recorded and can be decoded with the Python time.localtime() function. It denotes the number of seconds since Jan, 1 1970. The Name item holds the name of the block that was played back. If you write a script to evaluate the RecBlockStatus block you can either use the name or the Handle item. The handle item will result in the same number for every block as long as the block definition is the same. So you should cache the handle and map the proper name to the handle the first time you encounter a new handle. From that point on you can use the handle instead of the block name which should make for faster processing. Keep in mind that the association between block name and handle can be different for every run of GSEOS.

4.3.1.10.1 GseosRecorder.AddPlaybackBlock

AddPlaybackBlock()

This function adds a block to the Recorders playback list. If the recorder is in playback mode this takes effect immediately and the block won't be played back any more. To check if a block is on the playback list you can use GetPlaybackBlocks(). To remove blocks from the playback list use RemovePlaybackBlock().

Parameters

strBlockName: Name of the block to put on the playback list.

Returns

None.

4.3.1.10.2 GseosRecorder.AddRecordBlock

AddRecordBlock()

This function adds a block to the Recorders recording list. If the recorder is in recording mode this takes effect immediately and if the block arrives in the system it will be recorded. To check if a block is on the record list you can use GetRecordBlocks(). To remove blocks from the record list use RemoveRecordBlock().

Parameters

strBlockName: Name of the block to put on the record list.

Returns

None.

4.3.1.10.3 GseosRecorder.GetDataPath

GetDataPath()

Returns the currently selected data path. The data path is the directory where the automatically generated files are stored. In single file mode you specify the location of the recorder file. To change the data path use SetDataPath().

Returns

The current data path.

Example

```
>>> GseosRecorder.GetDataPath()  
'c:\\temp'
```

4.3.1.10.4 GseosRecorder.GetPlaybackBlocks

GetPlaybackBlocks()

This function returns a tuple with all the names of the blocks that are on the playback list. Note, these are not the actual block objects but the names of these blocks. Any block that is on the playback list will be played back when the Recorder is in playback mode. To get a list of the blocks on the playback list use GetPlaybackBlocks(). To add or remove blocks from the playback list you can use the Recorder interface or the functions AddPlaybackBlock() and RemovePlaybackBlock().

Returns

Tuple of the block names of blocks on the recording list.

Example

```
>>> GseosRecorder.GetPlaybackBlocks()  
( 'CMDSTRING', 'RS232Raw', 'TLM', 'HK' )
```

4.3.1.10.5 GseosRecorder.GetPrefix

GetPrefix()

The file prefix is a string that gets added to the automatically generated Recorder file name. This takes effect only in multi file mode. That is if the Recorder automatically generates the file names. In single file mode you determine the file to record to. GetPrefix() returns the current file prefix. You can use SetPrefix() to modify the prefix.

Returns

The current file prefix.

Example

```
>>> GseosRecorder.GetPrefix()  
'EM 1 '
```

4.3.1.10.6 GseosRecorder.GetRecordBlocks

GetRecordBlocks()

This function returns a tuple with all the names of the blocks that are on the recording list. Note, these are not the actual block objects but the names of these blocks. Any block that is on the record list will be recorded when the Recorder is in record mode and the block arrives. To get a list of the blocks on the record list use GetRecordBlocks(). To add or remove blocks from the record list you can use the Recorder interface or the functions AddRecordBlock() and RemoveRecordBlock().

Returns

Tuple of the block names of blocks on the recording list.

Example

```
>>> GseosRecorder.GetRecordBlocks()  
( 'RecComment', 'DSACtrl', 'RS232Raw', 'PeriodicMsg' )
```

4.3.1.10.7 GseosRecorder.IsPlayingBack

IsPlayingBack()

Determine if the Recorder is in playback mode.

Returns

True if the Recorder is in playback mode, false otherwise.

4.3.1.10.8 GseosRecorder.IsRecording

IsRecording()

Determine if the Recorder is in recording mode. In order to put the Recorder into recording mode use StartRecording(), to stop it use StopRecording().

Returns

True if the Recorder is in recording mode, false otherwise.

4.3.1.10.9 GseosRecorder.RemovePlaybackBlock

RemovePlaybackBlock()

This function removes a block from the Recorders playback list. If the recorder is in playback mode this takes effect immediately and the block will no longer be played back. To check if a block is on the playback list you can use GetPlaybackBlocks(). To add blocks to the playback list use AddPlaybackBlock().

Parameters

strBlockName: Name of the block to remove from the playback list.

Returns

None.

4.3.1.10.10 GseosRecorder.RemoveRecordBlock

RemoveRecordBlock()

This function removes a block from the Recorders recording list. If the recorder is in recording mode this takes effect immediately and the block will no longer be recorded. To check if a block is on the record list you can use GetRecordBlocks(). To add blocks to the record list use AddRecordBlock().

Parameters

strBlockName: Name of the block to remove from the record list.

Returns

None.

4.3.1.10.11 GseosRecorder.SetDataPath

SetDataPath()

Set the data path of the recorder. This is to be used for multi file mode only, in single file mode you will provide the file and directory. See also GetDataPath(). The Recorder can't be in recording mode while changing the data path. You have to stop recording first. To determine if the Recorder is in recording mode use IsRecording(), to stop the Recorder from recording use StopRecording(). The changes to the prefix are made permanent by writing them to the gseos.ini configuration file. On the next invocation of GSEOS you will have the prefix preserved.

Parameters

strDataPath: The new data path.

Returns

None.

4.3.1.10.12 GseosRecorder.SetPrefix

SetPrefix()

Set the file prefix to be used for multi file mode, see also GetPrefix(). Make sure this string does not contain any forward or backward slashes or the colon character since this will lead to errors in the file name generation. The changes to the prefix are made permanent by writing them to the gseos.ini configuration file. On the next invocation of GSEOS you will have the prefix preserved.

Parameters

strPrefix: The new file prefix.

Returns

None.

4.3.1.10.13 GseosRecorder.StartRecording

StartRecording()

Switch the Recorder into recording mode. All blocks that arrive after the Recorder is in recording mode and that are on the recording list (to get a list of blocks that are on the recording list use `GetRecordBlocks()`) will be recorded. To determine if the Recorder is in recording mode use `IsRecording()`.

Returns

None

4.3.1.10.14 GseosRecorder.StopRecording

StopRecording()

Stop recording. To determine if the Recorder is in recording mode use `IsRecording()`.

Returns

None

4.3.1.11 GseosSys

The GseosSys module hosts several system classes and mappings mostly used for internal purposes. It also supports some useful utility functions.

`FileOpen()` allows you to open any GSEOS file. `systemRecorder` module exposes the Python interface to the GSEOS Recorder functionality. The window functions `WindowPrint()`, `WindowMinimize()`, `WindowMaximize()`, `WindowRestore()`, and `WindowClose()` allow you to control some of the window behavior. `StartApplication()` lets you invoke external programs.

4.3.1.11.1 FileAppend

Enter topic text here.

4.3.1.11.2 FileOpen

Enter topic text here.

4.3.1.11.3 StartApplication

Enter topic text here.

4.3.1.11.4 WindowClose

Enter topic text here.

4.3.1.11.5 WindowMaximize

Enter topic text here.

4.3.1.11.6 WindowMinimize

Enter topic text here.

4.3.1.11.7 WindowPrint

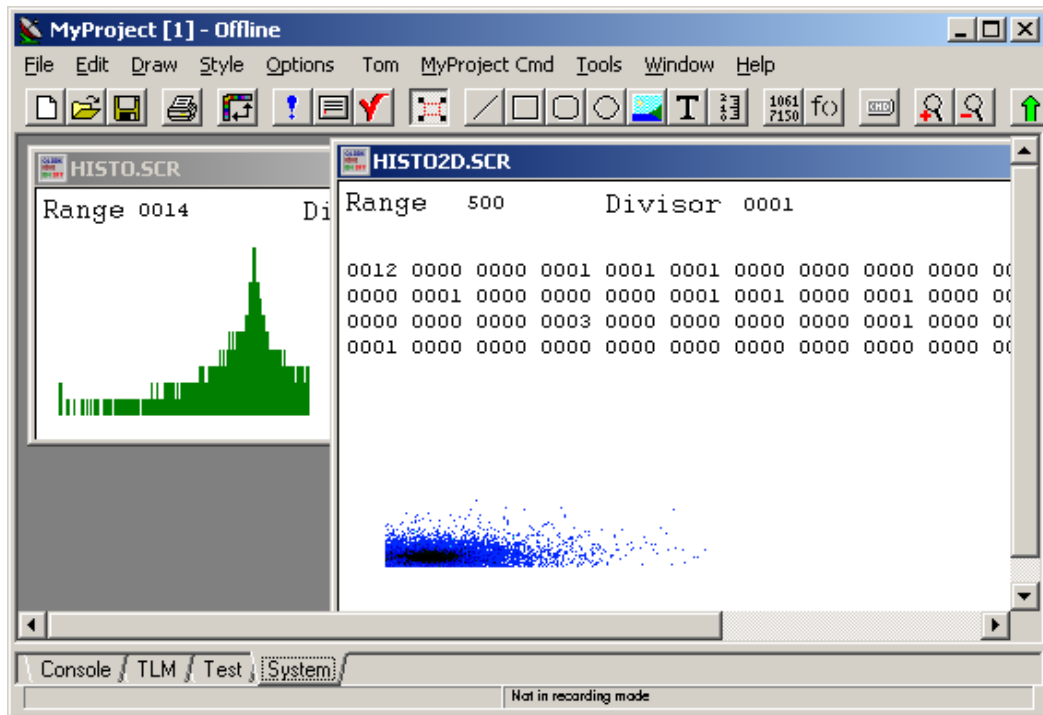
Enter topic text here.

4.3.1.11.8 WindowRestore

Enter topic text here.

4.3.1.12 Histogram

The histogram module allows you to easily histogram/classify your data. A histogram decoder is a specialized Decoder. There are two types of histogram decoders a 1D histogram decoder and a 2D histogram decoder. The purpose of a histogram decoder is to classify or histogram the source data. This can be used to display a spectrum of the distribution of events. The picture below depicts samples of a 1D histogram and a 2D histogram.



The histogram decoders take input array items and generate an output block containing the histogrammed data.

In the 1D case a single value will be mapped to a destination array. If we for example get a value of 5 in our source data we would increment the destination item at index position 5 by one. In the 2D case a pair of source values (x,y) will be mapped to a destination entry in the following way:

$$\text{Destination Position} = x + \text{Width} * y$$

where Width is the width of the matrix. Since in GSEOS all array items are vectors and not two-dimensional matrixes the Width is used to determine the x-dimension of the matrix. The destination array item can then in turn be displayed as a two dimensional bitmap with various color mappings to indicate the distribution of 'intensity' in the two

dimensional space.

Autoscaling

The histogram decoders also allow for automatic rescaling of the histogram. If the maximum value in the histogram reaches a certain limit the histogram will be scaled down by simply dividing all values in the histogram by 2. This process is repeated over every time we reach the upper limit. If a 'Divisor' item exists in the destination block the current scale value will be exported to that item. This way you can determine what the absolute values for the histogram are (approximately only since we lose resolution every time we scale own).

Configuration

The histogram decoders can be configured with the following parameters most of which can be specified in the according constructor:

1D: Name
 SourceX
 Destination
 Left
 Right
 Range
 EventCnt

2D: Name
 SourceX
 SourceY
 Destination
 Left
 Top
 Right
 Bottom
 Range
 EventCnt

The SourceX and SourceY parameters specify the item names of the source items. In the case of a 1D histogram only the X source is needed. For the 2D histogram a pair of values is required, therefore SourceX and SourceY. Usually these items have to be array items. If you only have a single value declare an array item with dimension 1. The Destination parameter specifies the destination item name, its dimension must be at least Right-Left in the 1D case and (Right-Left)*(Bottom-Top) for the 2D histogram.

Be aware that the histograms require according memory. Especially for the 2D histograms if you chose for example (0, 1024, 0, 1024) as the dimensions of your destination histogram the internal histogram buffer alone will occupy $1024 \times 1024 \times 4$ byte = 4MB of memory!!!

The Left, Top, Right, Bottom parameters specify the dimensions of the clipping area. For the 1D histogram this is a simple interval for the 2D histogram this is a rectangle. The source values are only mapped into the destination item if they fall within the clipping area. E.g. if the clipping interval is Left=6, Right=12 a source value of 5 would be discarded, a source value of 7 would cause to increment the destination item at position $7-6 = 1$.

Therefore the clipping area gives you the opportunity to map a certain area of your source data onto the destination. You could also think of it as a kind of zoom in/zoom out feature. The Right and Bottom values are not included in the range. E.g.: If you specify 0 for Left and 256 for Right the indices from 0..255 will be covered.

The Range parameter allows you to set an initial range for the histogram. This is the maximum value in the histogrammed data. In the case of autoscaling the entire histogram is scaled down by a factor of 2 if any destination value reaches the range value. If autoscaling is off the destination item is simply not incremented.

The EventCnt parameter is the name of a data item that is read at runtime to determine the range of input values to read. By default, that is if no EventCnt item is specified all values in the SourceX item will be added to the histogram. This may not always be the appropriate behavior. In case not all values are valid or the number of valid values changes you may want to use the EventCnt item. It is assumed that the EventCnt will hold the currently valid number of items in the SourceX data item. Only these items contribute to the histogram.

If your destination block has the scalar items 'Divisor' and 'Range' these items will be updated every time a new histogram block is generated. This feature gives you some feedback as to what Range you have selected for autoscaling. The Divisor will be set to the current value of the scaling factor used for autoscaling. This allows you to find out how often the histogram has been scaled down.

For more information please refer to the 1D and 2D histogram decoders:

- 1D Histogram
- 2D Histogram

4.3.1.12.1 Histogram.bAutoscale

Property: bAutoscale

Enable or disable the auto scale feature. When auto scale is enabled the entire histogram is scaled by .5 if any value in the histogram reaches the dwRange value.

4.3.1.12.2 Histogram.Clear

Clear()

The Clear() function resets all values in the destination histogram to 0.

4.3.1.12.3 Histogram.dwRange

Property: dwRange

The dwRange property determines the maximum value your destination histogram will accumulate. Once this value has been reached this particular point is not further incremented if auto scaling is not enabled. If auto scaling is on, all values in the histogram are divided by 2 and the destination point is incremented by the appropriate amount.

4.3.1.12.4 Histogram.Histogram1D

Histogram1D(strName, SourceX, Destination, Left, Right, [Range], [EventCnt])

The Histogram1D constructor creates a new 1D histogram decoder. You have to specify a

unique name, the source data item, the destination data item, and the dimensions of the clipping area. Optionally you can also specify a range and an item that specifies the event count.

Parameter	Description
strName	The unique name identifies the histogram decoder. This is the name you will see in the GSEOS Explorer.
SourceX	The SourceX parameter specifies the item name of the source item. You have to specify the name of a data item here and not the data item itself. This parameter is a string! Usually the source item has to be an array item. If you only have a single value declare an array item with dimension 1.
Destination	This parameter specifies the destination item name (note again that this is a string, e.g.: 'Histo.Data'). The dimension of the destination item must fit in the clipping area (Right-Left).
Left	The Left and Right parameters specify the dimensions of the clipping area. The source values are only mapped into the destination item if they fall within the clipping area. E.g. if the clipping interval is Left=6, Right=12 a source value of 5 would be discarded, a source value of 7 would cause the destination item at position $7 - \text{Left} = 7 - 6 = 1$ to be updated (incremented). Therefore the clipping area gives you the opportunity to map a certain area of your source data onto the destination. You could also think of it as a kind of zoom in/zoom out feature.
Right	The Right and Bottom values are not included in the range. E.g.: If you specify 0 for Left and 256 for Right the indices from 0..255 will be covered.
Range	See Left
EventCnt	The range specifies the maximum value that can be accumulated in the histogram. If autoscaling is enabled the histogram will be scaled down, otherwise it will be clipped at the range value.
	The EventCnt parameter is the name of a data item that is read to determine the range of input values to use. By default, that is if no EventCnt item is specified all values in the SourceX item will be added to the histogram. This may not always be true. In case not all values are valid or the number of valid values changes you may want to use the EventCnt item. It is assumed that the EventCnt will yield the currently valid number of items in the SourceX data item. Only these items contribute to the histogram. The EventCnt item does not have to be located in the source block but can be any item.

Returns

The new histogram object.

Comments

The histogram is automatically registered for its source block. There is no need to append it to the Decoder list of the source block like you have to do for an ordinary decoder.

If your destination block has the scalar items 'Divisor' and 'Range' these items will be updated every time a new histogram block is generated. This feature gives you some feedback as to what Range you have selected for autoscaling. The Divisor will be set to the current value of the scaling factor used for autoscaling. This allows you to tell how often the histogram has been scaled down.

Be aware that the items specified in the constructor are **names** of data items and therefore strings. Do not pass in the item directly. The item would be resolved and therefore an integer would be passed in the constructor.

E.g.: Use: `Histogram.Histogram1D('Block.SourceX',)`
 Do NOT use: `Histogram.Histogram1D(Block.SourceX,)`

This applies to all items you specify in the constructor. In addition to the special histogram functionality you can use all the methods a normal decoder offers like `bEnable` to enable or disable the histogram.

Example

The following example defines a simple 1D histogram decoder.

```
Histogram.Histogram1D('My1DHistogram',
                      'SrcBlock.SrcItemX',
                      'DestBlock.DestItem',
                      0, 128,
                      20,
                      'Block.EventCnt')
```

4.3.1.12.5 Histogram.Histogram2D

Histogram2D(strName, SourceX, SourceY, Destination, Left, Top, Right, Bottom, [Range], [EventCnt])

The Histogram2D constructor creates a new 2D histogram decoder. You have to specify a unique name, the source data items (one for the x- and one for the y-axis), the destination data item, and the dimensions of the clipping area (in the 2D case a rectangle). Optionally you can also specify a range and an item that specifies the event count.

Parameter	Description
strName	The unique name identifies the histogram decoder. This is the name you will see in the GSEOS Explorer.
SourceX	The SourceX parameter specifies the item name of the source item for the x-coordinate. A point in the destination histogram is determined by

	the tuple (x, y). The SourceX item supplies the x-part, the SourceY item the y-part. You have to specify the name of a data item here and not the data item itself. This parameter is a string! Usually the source item has to be an array item. If you only have a single value declare an array item with dimension 1.
SourceY	See SourceX. This item supplies the y-value.
Destination	This parameter specifies the destination item name (note again that this is a string, e.g.: 'Histo.Data'). The dimension of the destination item must fit in the clipping area (Right-Left)*(Bottom-Top).
Left	The Left, Right, and Top, Bottom parameters specify the dimensions of the clipping area. The source values are only mapped into the destination item if they fall within the clipping area. The Right and Bottom values are not included in the range. E.g.: If you specify 0 for Left and 256 for Right the indices from 0..255 will be covered. Be aware that the histograms require according memory. Especially for the 2D histograms if you chose for example (0, 1024, 0, 1024) as the dimensions of your destination histogram the internal histogram buffer alone will occupy 1024*1024*4 byte = 4MB of memory!!!
Right	See Left.
Top	See Left.
Bottom	See Left.
Range	The range specifies the maximum value that can be accumulated in the histogram. If autoscaling is enabled the histogram will be scaled down, otherwise it will be clipped at the range value.
EventCnt	The EventCnt parameter is the name of a data item that is read to determine the range of input values to use. By default, that is if no EventCnt item is specified all values in the SourceX item will be added to the histogram. This may not always be true. In case not all values are valid or the number of valid values changes you may want to use the EventCnt item. It is assumed that the EventCnt will yield the currently valid number of items in the SourceX data item. Only these items contribute to the histogram. The EventCnt item does not have to be located in the source block but can be any item.

Returns

The new histogram object.

Comments

The histogram is automatically registered for its source block. There is no need to append it to the Decoder list of the source block like you have to do for an ordinary decoder.

If your destination block has the scalar items 'Divisor' and 'Range' these items will be updated every time a new histogram block is generated. This feature gives you some feedback as to what Range you have selected for autoscaling. The Divisor will be set to the current value of the scaling factor used for autoscaling. This allows you to tell how often the histogram has been scaled down.

Be aware that the items specified in the constructor are **names** of data items and therefore strings. Do not pass in the item directly. The item would be resolved and therefore an integer would be passed in the constructor.

E.g.: Use: `Histogram.Histogram2D('Block.SourceX',)`
Do NOT use: `Histogram.Histogram2D(Block.SourceX,)`

This applies to all items you specify in the constructor. In addition to the special histogram functionality you can use all the methods a normal decoder offers like `bEnable` to enable or disable the histogram.

Example

The following example defines a simple 2D histogram decoder that expects its source values to be in the range (0..127/0..127).

```
Histogram.Histogram2D('My2DHistogram',  
                      'SrcBlock.SourceXItem',  
                      'SrcBlock.SourceYItem',  
                      'DestBlock.DestItem',  
                      0, 128,  
                      0, 128,  
                      20,  
                      'Block.EventCnt')
```

4.3.1.13 Monitor

The main purpose of the Monitor module is to verify the integrity of the incoming data and to report alarm conditions. It works similar like the Decoder in that it hooks on the arrival of a data block. However, opposed to the Decoder it does not generate any child blocks but reports alarms or issues commands depending on the outcome of the monitor condition. In most cases a simple limit check will be sufficient, but even complex monitor conditions are easy to manage.

In order for the monitor to be called on the arrival of a specific block it has to be registered with this block. To do this you have to create a monitor object and add it to the list of monitors for that block. Every time the block arrives your monitor condition is evaluated.

To construct a monitor you have to instantiate a monitor object of the monitor class. The constructor of the monitor object takes the unique monitor name and the monitor condition you want evaluated. The monitor function itself takes one parameter. The block that triggers the monitor is passed into the monitor function. Therefore when you register one decoder for multiple blocks you will be able to distinguish which block arrived.

The monitor can be disabled by setting `bEnable` to false. It can be reenabled by setting `bEnable` to true. The variable `dwCnt` holds the number of times the monitor has been executed. You can reset this value if desired.

If the monitor raises an exception it will be disabled. You should catch all exceptions you don't want to terminate the monitor.

It is advised that you give your monitor function a documentation string. This documentation will be displayed in the GSEOS Explorer.

For more information please refer to the examples:

- Limit Check
- Counter Check

4.3.1.13.1 bEnable

Property: bEnable

Enable or disable the monitor.

Example

Disable the monitor from a script:

```
oMyMon.bEnable = 0
```

4.3.1.13.2 constructor

Monitor(strName, fMonitor)

The Monitor constructor creates a new monitor. You have to specify a unique name and the monitor function. In order to start the monitor you have to add it to the list of monitors for the block(s) you want to register your monitor for.

Parameter	Description
strName	The unique name identifies the monitor. This is the name you will see in the GSEOS Explorer.
fMonitor	Your monitor function. It takes one parameter which is the block that triggers the monitor (if you register the monitor for more than one block you can use this parameter to determine the source block). If this function throws an exception the monitor is terminated.

Returns

The new monitor object.

Comments

In order for the monitor to be useful you have to hook it on the arrival of a block. You do this by adding it to the list of monitors for the block you are interested in. See the example below.

Exceptions

TypeError	The monitor name has to be a string.
MonitorError	The fMonitor parameter has to be a callable object.

Example

The following example defines a simple limit check monitor. In case of an out of range condition a message is posted to the GSEOS message window:

```
#*
*****
*#
#* * Import the block names into our namespace.
* *#
#*
*****
*#
from __main__ import *
import Monitor
import Gseos
```

```
def MyMonitor1(oBlock):  
    if oBlock.Temp1 > 200:  
        cmd.msg('Monitor1: Sensor temp out of range')  
  
HK.Monitors.append(Monitor.Monitor('Sensor Temp Check', MyMonitor1))
```

4.3.1.13.3 Delete

Delete()

Deletes a monitor.

Returns

None

Comments

The GSEOS Explorer holds a reference to every monitor created. Even if your object goes out of scope the Explorer will hold on to the monitor. To release the monitor you have to call Delete() first and then destroy the monitor object itself (e.g. by letting it go out of scope). Usually there is no need to delete a monitor.

Example

Release our current monitor and then destroy the monitor object.

```
oMyMon.Delete()  
oMyMon = None
```

4.3.1.13.4 dwCnt

Property: dwCnt

The number of invocations of the monitor. You can set this value if desirable.

4.3.1.13.5 Examples

4.3.1.13.5.1 CounterCheck

Besides limit checking the verification of a sequence of events may be important. This example demonstrates how to set up a monitor that checks the proper sequence of a counter. Let's assume we get a block TLM that contains a sequence number. This sequence number should be incrementing by one with every sample we receive. If we get a number that is not in sequence we want to report this in a separate log file and set the current sequence number to the one we just received. The block definition of the TLM block is given below:

```
TLM
{
    ApID                ,,,16;
    Seq                 ,,,32;
    Data[2000]          ,,,16;
}
```

In this example the monitor has to maintain state information which is a simple value holding the last sequence number seen. The monitor function has to compare the value of the latest sequence number with the one stored and check if the sequence is in proper order. If so our local data needs to be updated. If not report an error and update the sequence number with the last sample received. Here the monitor function:

```
#
# Import block definitions, Monitor class and other stuff
#
from __main__ import *
import Monitor
import gseos

#
# Define our local sequence number and initialize to 0.
#
dwSeq = 0

#
# TLM Sequence Monitor
#
def fMonSeq(oBlock):
    "TLM Sequence Monitor"

    global dwSeq

    if oBlock.Seq != dwSeq+1:
        gseos.Log('error.log', 'TLM Sequence out order. Expected sequence
number: %d, \
                                'received sequence number: %d' % dwSeq,
oBlock.Seq)
        dwSeq = oBlock.Seq
```

Now assign it to the TLM block:

```
#
# Create a new monitor.
#
oMonSeq = Monitor.Monitor('MonSeq', fMonSeq)
```

```
#
# Assign it to the TLM block.
#
TLM.Monitors.append(oMonSeq)
```

4.3.1.13.5.2 LimitCheck

The most straightforward use of the monitor is as a limit check on individual items. This simple limit check example demonstrates how to set up such a simple monitor. Let's say we have a HK block with some voltage items. We want to log a message in the GSEOS message window when the voltage of Plus_12V_Mon exceeds 15. The block definition of the HK block is given below:

```
HK
{
    Plus_12V_Mon          ,,,8;
    Plus_28V_Mon          ,,,8;
    Plus_5V_Mon           ,,,8;
    Plus_6V_Mon           ,,,8;
    Minus_5V_Mon          ,,,8;
    Plus_5DPU_Mon         ,,,8;
    Minus_12V_Mon         ,,,8;
    Minus_6V_Mon          ,,,8;
    I_Stop_MCP_V          ,,,8;
    I_Start_MCP_V         ,,,8;
}
```

The monitor function has to compare the value of the item Plus_12V_Mon to the limit of 15. If the condition holds a message is issued. Here the monitor function:

```
#
# Import block definitions, Monitor class and other stuff
#
from __main__ import *
import Monitor
import gseos

#
# 12V Monitor
#
def fMon12V(oBlock):
    "12V Monitor"

    if oBlock.Plus_12V_Mon > 15:
        cmd.msg('12V Monitor: Limit exceeded, current voltage %d' %
oBlock.Plus_12V_Mon)
```

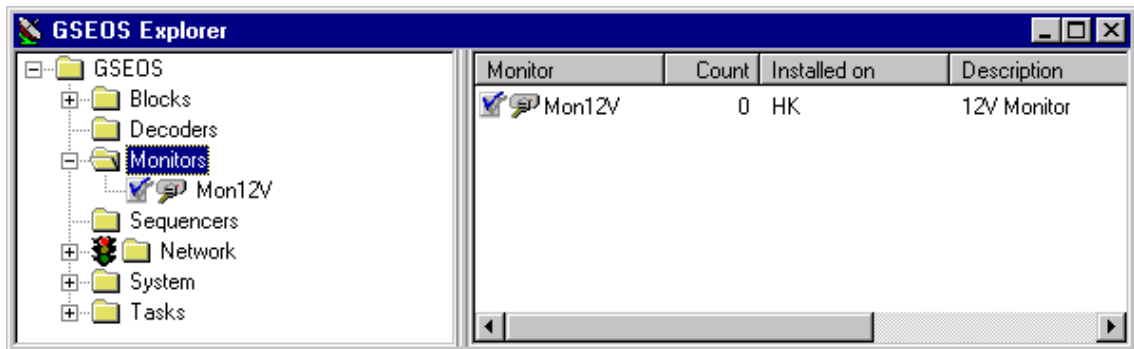
Once we have the monitor function defined we have to assign it to the HK block to get executed:

```
#
# Create a new monitor.
#
oMon12V = Monitor.Monitor('Mon12V', fMon12V)
```

```
#
# Assign it to the HK block.
#
HK.Monitors.append(oMon12V)
```

After we execute this script we can examine the monitor in the GSEOS Explorer. If we choose the Block node and list all the blocks in the system we should see our HK source block. Extending the Monitors node beneath the HK block lists our monitor and indicates that it is running. You can also get to all monitors from the Monitors node beneath the main GSEOS node.

By right-clicking on the decoder node you can also disable or delete the decoder.



To run this example please install the mon1.blk block definition file in the demo/mon1 folder, restart GSEOS and load the mon1.py monitor script from the same directory.



Keep in mind that the block definitions are not typed. This means your data items are treated as unsigned binary fields. Assume the most significant bit of the 8-bit Minus_5V_Mon item in the HK block is a sign bit a -5 would be represented as $256 - 5 = 251$. To convert this number to a signed value which you can compare to -5 instead of 251 in your monitor function use the `GseosConvert.signed()` function e. g.:

```
#
# -5V Monitor
#
def fMonNeg5V(oBlock):
    "-5V Monitor"
    Minus5V = GseosConvert.signed(oBlock.Minus_5V_Mon, 7)

    if Minus5V < -7:
        cmd.msg('-5V Monitor: Limit exceeded, current voltage %d' % Minus5V)
```

4.3.1.13.6 strName

Property: strName

The unique name of the monitor. You should not change this name after you created the monitor. Rather delete the current one and create a new one with the desired name.

4.3.1.14 Sequencer

The Sequencer module allows you to run and control test sequences. Using the Sequencer module it is easy to write test scripts for closed-loop instrument control. You can command the instrument and verify that certain values are met within a certain time. Depending on the outcome you can issue further commands to log the success of failure in a test procedure log. This allows you to set up test sequences that can be run to autonomously verify the proper performance of the system which is especially useful as a regression testing tool. Although it may seem that the Monitor and Sequencer have overlapping functionality they have an entirely different focus and concept. The Sequencer module allows you to write synchronous test control sequences meaning that you can write down a sequential test procedure whereas the Monitor module works asynchronous (event driven) and makes it therefore hard to write a sequential control script. The central functionality of the Sequencer is provided by the Wait() function. You provide a list of blocks (events), a condition and an optional timeout value. Your condition is evaluated whenever one of the blocks in the list arrives. If your condition evaluates to true the Wait() returns otherwise the condition will be reevaluated with the next block arrival. There are two possible scenarios that the Wait() function can return before your condition returns true. If you specify a timeout value and the timeout timer expires before your condition returns true the Wait() function throws a TimeoutError exception. If you stop the Sequencer the Wait() function will throw an AbortError exception. For trivial condition functions (one liners) you can make use of the lambda function of Python. Let's just start with a simple example to demonstrate the use of the Sequencer:

```
import Sequencer

# A Sequencer script turning on the heater circuit, waiting for the
# temperature to
# rise to 20 and turn the heater off.
def fWarmUp(oSeq):
    """
    It is always a good idea to document your Sequencer scripts. These
    document strings
    are also accessible from the Sequencer user interface. Here we go:
    Fire up the heater, wait until we reach 20C and turn the heater off.
    """
    # Start the heater with our heater command
    fStartHeater()

    # Wait until we reach 20C
    oSeq.Wait(HK, lambda: HK.Temp == 20)

    # Stop the heater
    fStopHeater()

# Start the Sequencer
Sequencer.Sequencer('Heater Control 1', fWarmUp)
```

This simple Sequencer turns on a heater circuit, waits for the temperature to rise (the heater presumably changes the temperature which is reflected in the HK.Temp item) to a desired value and stops the heater.

Well, this script is far from perfect, for example if the temperature is higher than 20 to begin or we don't get a block with HK.Temp equal to 20 with we will never exit the Wait().

The last line then finally starts the Sequencer. To start the Sequencer you have to provide a unique name, the Sequencer function, an optional argument, and an optional flag that indicates if you want to create the Sequencer in the stopped state. By default the Sequencer starts running as soon as you create it. The Sequencer runs in it's own thread (in the background) and the call to create the Sequencer returns immediately. Now let's assume we have three similar heaters and we want to control all of them. Instead of defining three Sequencer scripts we can parameterize the sequencer function `fWarmUp()`. When we start a Sequencer we can pass an optional argument which then in turn is passed to the sequencer function. Here our example for multiple heaters:

```
import Sequencer

# Parameterized heater script. Pass two arguments, the heater number and
# the
# destination temperature.
def fWarmUp(oSeq, (wHeater, wDestTemp)):
    "Heater Control 2. Takes the heater number and the destination temp."

    # Start the heater with our heater command
    fStartHeater(wHeater)

    # Wait until we reach our destination temperature
    oSeq.Wait(HK, lambda Heater=wHeater, Temp=wDestTemp: HK.Temp[Heater] ==
Temp)

    # Stop the heater
    fStopHeater(wHeater)

# Fire up the Sequencer for heater 2 and 30C.
Sequencer.Sequencer('Heater Control 2', fWarmUp, (2, 30))

# Fire up the Sequencer for heater 3 and 25C.
Sequencer.Sequencer('Heater Control 3', fWarmUp, (3, 25))

# Fire up the Sequencer for heater 4 and 45C.
Sequencer.Sequencer('Heater Control 4', fWarmUp, (4, 45))
```

The above script expects a second tuple parameter besides the Sequencer reference that is passed in as the first parameter. In our case we pass two different values in this tuple, the heater number and the destination temperature. When we invoke the Sequencer we have to provide the parameters as the third parameter of the `Sequencer()` call. Now we can start as many Sequencers as needed (three in the above example) using just one sequencer function.

The above examples both terminate when the desired temperature is reached. To restart them we can just call the `Start()` function of the Sequencer. (This can easily be done from the GSEOS Explorer).

In order to programmatically restart the sequencer we of course have to keep a reference of the Sequencer when we create it. Here is how you would do it:

```
# Fire up a Sequencer
oHeaterCtrl = Sequencer.Sequencer('Heater Control 5', fWarmUp, (5, 10))
...
# At some later point restart the Sequencer to bring
# the heater 5 back up to it's temperature.
oHeaterCtrl.Start()
```

The other functions of interest are `Stop()` to stop a running Sequencer, and `Delete()` to delete it. Besides the `Wait()` function there are three other functions the Sequencer provides: `MessageBox()`, `InputDialog()`, and `Sleep()`. Let's look at one more example which simulates a two-point control for the temperature:

```
# The Sequencer script of the two-point heater control
def fTwoPointHeater(oSeq, (wLowTemp, wHighTemp)):
    "Heater Control. A two-point heater control."

    # Control our heater circuit until we get terminated
    while 1:
        if HK.Temp >= wHighTemp:
            fStopHeater()
            oSeq.Wait(HK, lambda wTemp=wLowTemp: HK.Temp <= wTemp)

        elif HK.Temp <= wLowTemp:
            fStartHeater()
            oSeq.Wait(HK, lambda wTemp=wHighTemp: HK.Temp >= wTemp)

        # If we are in the right range just wait for a while
        else:
            oSeq.Sleep(300)

    # Fire up the two-point controller.
    Sequencer.Sequencer('Two-point controller', fTwoPointHeater, (20, 30))
```

Although this functionality could also be implemented with a Monitor it demonstrates some of the Sequencer functions.

The next paragraph covers some Sequencer internals:

To understand how the Sequencer works here some details about it's implementation: Every Sequencer you start runs in it's own thread of execution and is therefore very efficient in a `Wait()` state. Whenever one of the trigger blocks arrives the Sequencer preserves the state of all the blocks (e.g. there won't be any block arrivals before the condition is completely evaluated) and runs your condition function. The condition function is therefore guaranteed to see a consistent view of the system. Once your condition returns however the blocks are released and if there are any new arrivals they will overwrite the contents of the old blocks. This means that if you would perform the same condition right after you exit out of a `Wait()` function it may not evaluate to true any more. E.g.:

```
# Check for the counter value to be 10
def fCheckCount():
    return Block.Count == 10

# The Sequencer script waiting for the counter to turn 10
# and log this event
def fLogCounterEvent(oSeq):
    while 1:
        oSeq.Wait(Block, fCheckCount)
        Gseos.Log('Count.log', 'The counter is ' + str(Block.Count))
```

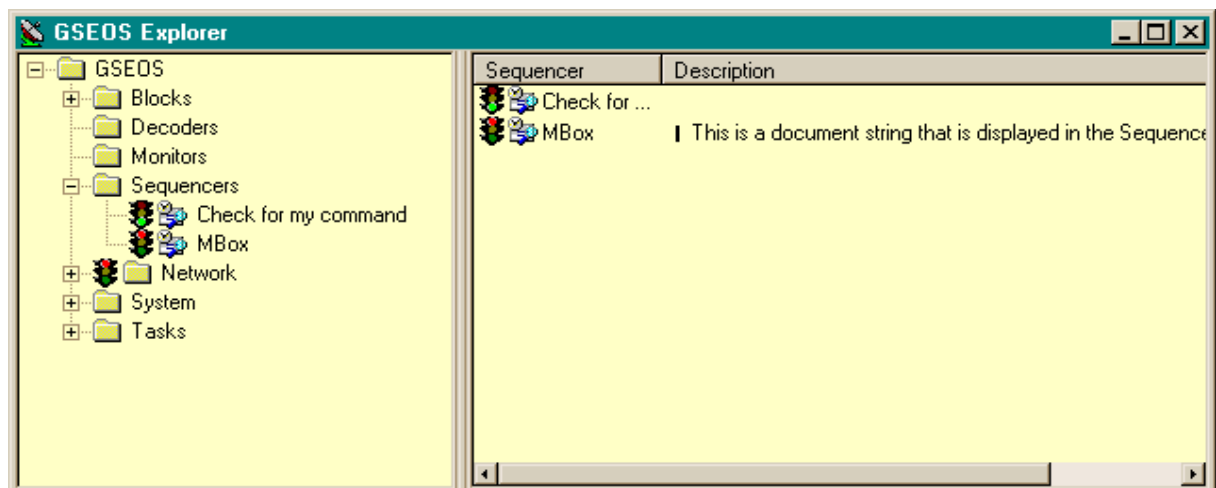
Besides the fact that a Monitor would be more appropriate for this kind of checking the above Sequencer waits (in a loop) for the item `Block.Count` to become 10. If this is the case the `Wait()` function returns and the value of the counter is logged to a log file. However when we fetch the counter value after the `Wait()` function returned the count

may have already changed. This is because the Sequencer runs in it's own thread a new block may arrive while we perform the Log function. If you were to log the event from within the condition it is guaranteed that the data does not change. This is exactly the reason why for example limit checking is better done with the Monitor module where you get a notification of every block no matter if you happen to be in a Wait() state or not. You should keep this behavior in mind while designing Sequencer scripts. However for a Sequencer script this is perfectly okay since we want to trigger on a certain event and then take action. In general you want to use the Sequencer if you have a sequence of events to check, perform some sort of verification and move on to the next part of the sequence. The beauty of the Sequencer is that you can just write down your script in a sequential (hence the name) manner:

```
# Sample sequence
oSeq.Wait(BINCOMMAND, fWaitForMyCommand)
oSeq.Sleep(2000)
oSeq.Wait(HK, fIsTempOk)
fSetLimits()
oSeq.Wait(TLM, fWaitForImage)
fMoveCamera()
# etc, etc.....
```

Tip: It is a good idea to document the progress of the Sequencer in for example a log file with the Log() function.

Now that we know how to use the Sequencer from a script let's have a brief look at the Sequencers in the GSEOS Explorer. You will find the Sequencers beneath the GSEOS main node:



The Explorer lists all Sequencers in existence and their respective status. The docstring of the selected Sequencer is displayed in the left pane (it is always a good idea to provide a short description). If you right-click on the Sequencer you get a menu that allows you to Start/Stop/Delete the Sequencer depending on its state. You can also highlight multiple sequencers and right-click on the selection to invoke the operation for a number of sequencers at once. The Start/Stop button allows you to start or stop the selected Sequencer(s). You can also delete the Sequencer from the control dialog. This tool gives you a quick overview of the state of your Sequencers. One important implementation detail: In order to manage the Sequencers with the Explorer a reference to each created

Sequencer is held by the Sequencer manager. This means if you create a Sequencer and assign it to a variable, e.g.:

```
oMySeq = Sequencer.Sequencer('MySeq', fMyCond)
```

There are two references to the sequencer. To remove the sequencer you have to delete the reference from the Sequencer manager, you do that by issuing the Delete command from the menu. If then your object goes out of scope the sequencer is removed from the system. After you call Delete() either from within a script or from the control dialog the Sequencer is marked for deletion and can not be started any more. As soon as the last reference to the Sequencer is removed the Sequencer is deleted from the system. In general it should not be necessary to keep a reference to a Sequencer around, in this case the Sequencer will be deleted as soon as you select Delete from the Explorer window.

The sequencer module exports the following Methods and Properties:

Sequencer

Start

Stop

Delete

Wait

Sleep

MessageBox

InputDialog

Properties:

wStatus

Most of the above functions also have a doc string which you can access with:

```
Sequencer.function.__doc__
```

4.3.1.14.1 Sequencer.Delete

Delete()

Delete a Sequencer.

Returns

None

Comments

The call to Delete() removes the reference held by the control dialog. It marks the sequencer as deleted and renders it unusable for all other references. As soon as the last reference terminates the Sequencer is deleted. If the Sequencer is still running it will be stopped first.

Example

Delete the Sequencer from the above example:

```
oMySeq.Delete

# Now remove our object reference
del oMySeq
```

4.3.1.14.2 Sequencer.InputDialog

InputDialog(strText)

Displays an input dialog box. The text strText is displayed as the input prompt. If the user pressed Cancel or does not provide any input an AbortError exception is raised.

Parameter	Description
strText	The input dialog prompt text.

Returns

The text entered by the user or None if the user selected cancel.

Example

Display an input box asking for a HV level:

```
def fMySeq(oSeq):
    oSeq.InputBox('Please set a new HV level in the range 1 .. 4!')
    oSeq.Wait([HK], fCheckHV)
```

4.3.1.14.3 Sequencer.MessageBox

MessageBox(strText, [cButtons], [cIcon=MB_ICONEXCLAMATION])

Displays a message box. The text strText is displayed as the message prompt. The standard windows buttons can be specified in the cButtons parameter. If the user choses Cancel the function throws an AbortError exception.

Parameter	Description
strText	The message prompt text..
cButtons	Optional, specifies the buttons to display. One of the following attributes can be specified: MB_OK, MB_OKCANCEL, MB_RETRYCANCEL, MB_YESNO, MB_YESNOCANCEL.
cIcon	Optional, specifies an icon to display. MB_ICONEXCLAMATION, MB_ICONINFORMATION, MB_ICONQUESTION, MB_ICONSTOP.

Returns

One of the following constants depending on the user action: IDNO, IDOK, IDRETRY, IDYES.

Example

Display a message box that prompts the user to confirm a parameter and continue with the sequence:

```
def fMySeq(oSeq):
    oSeq.Wait([HK], fSetHV)
    oSeq.MessageBox('Please make sure the HV value is in the range 1 .. 4!',
MB_OKCANCEL)
```

4.3.1.14.4 Sequencer.Sequencer

Sequencer(strName, fSequencer, [Args=None], [bStart=1])

Create a Sequencer instance and start the Sequencer.

Parameter	Description
strName	A unique name for the Sequencer. This name will be used to identify the Sequencer in the control dialog.
fSequencer	The Sequencer function. It takes one or more arguments (depending if you supply arguments in the Args parameter). The first argument is an instance of the Sequencer itself. You will use this to invoke the Sequencer methods on this object. You have to provide an additional argument for every argument you pass in the Args tuple.
Args	Optional, any arguments you want to pass to your Sequencer function. If you need more than one argument you have to wrap them in a tuple. Make sure you have a matching parameter in your fSequencer function for every argument you list. The default is no arguments.
bStart	Optional, start flag. If you set this flag to false the Sequencer is not started until you explicitly call the Start() method. The default is true and therefore runs the Sequencer as soon as you create it.

Returns

Instance of the Sequencer class. See the comments section.

Comments

As mentioned in the section above you usually dont have to hold on to a reference to the Sequencer object since the control dialog will allow you to manage the Sequencer. However if you need to control the Sequencer from within a program you have to explicitly call Delete() if you want to get rid of the Sequencer (to delete the reference the control dialog holds).

Example

Create a Sequencer:

```
def fMySeq(oSeq):
    oSeq.MessageBox('Hello from your sequencer')
    oMySeq = Sequencer.Sequencer('MySeq', fMySeq)
```

4.3.1.14.5 Sequencer.Sleep

Sleep(dwTimeout)

Put the Sequencer to sleep for the specified number of seconds. You should always use this function to put the Sequencer to sleep. This Sleep() function reacts to the control dialog as opposed to the sleep() function of other modules. If the function gets aborted by the user through a Stop() command an AbortError is raised.

Parameter	Description
dwTimeout	Timeout value in seconds.

Returns
None

Comments
The Sleep() function puts the Sequencer thread in an effective state where it consumes only very little processor time.

4.3.1.14.6 Sequencer.Start

Start()

Start (restart) a Sequencer.

Returns
None

Comments
If the Sequencer is still running or if it is deleted a RuntimeError exception is thrown.

Example
Restart the Sequencer from the above example:

```
oMySeq.Start()
```

4.3.1.14.7 Sequencer.Stop

Stop()

Stop a Sequencer.

Returns
None

Comments
A Sequencer can only be stopped at certain points during its execution. In a Wait(), MessageBox(), InputDialog(), Sleep() function. If the Sequencer is not in any of these states while the Stop() is issued it will be stopped as soon as it executes one of the above functions. If the Sequencer is already stopped or if it is deleted a RuntimeError exception is thrown.

Example
Stop the Sequencer from the above example (this will throw an exception since the Sequencer terminates right away):

```
oMySeq.Stop()
```

4.3.1.14.8 Sequencer.Wait

Wait(ctBlocks, fCondition, [oArgs=()], [dwTimeout=INFINITE])

Wait for one of the events (block arrivals) specified in the event list (ctBlocks) to be signaled. Evaluate the condition and return if the condition evaluates to true. If the condition evaluates to false go back and wait for the next block arrival, if it evaluates to true return from the function call. If the function gets aborted by the user through a Stop() command raise an AbortError. If an optional timeout value is specified and the timer expires before the condition evaluates to true raise a TimeoutError.

Parameter	Description
ctBlocks	The block(s) to trigger on. If one of these blocks arrives the condition is evaluated. A single block can simply be passed in as is. Multiple blocks have to be wrapped in a tuple.
fCondition	The condition to evaluate when any of the above blocks arrives. If true the Wait() function finishes, otherwise it will block until the next arrival of a trigger block. The condition function must take an according number of parameters depending on oArgs. I.e. if you pass three values in the oArgs tuple fCondition() needs to have three argument parameters.
oArgs:	Optional. A tuple with command arguments that get passed to the condition function.
dwTimeout	Optional, timeout value in seconds. A TimeoutError is raised if the timeout value is met before the condition evaluates to true.

Returns

None

Comments

Oftentimes it is convenient to use a lambda function for a simple condition instead of defining a separate condition handler.

Example

Delete the Sequencer from the above example:

```
def fMySeq(oSeq):
    try:
        oSeq.Wait([BINCOMMAND], lambda: BINCOMMAND.byChan == 123, (), 20)

    except sequencer.TimeoutError, e:
        print 'Did not receive a command on channel 123 in the last 20 seconds'
        return

    print 'Got a command on channel 123'

# Now remove our object reference
del oMySeq
```

4.3.1.14.9 Sequencer.wStatus

Property: wStatus

Get the status of a Sequencer. Do not modify this value!

Comments

The property can have three states: RUNNING, STOPPED, or DELETED.

Example

Query the state from the Sequencer from the above example:

```
if oMySeq.wStatus == sequencer.RUNNING:
    print 'The sequencer is running'

elif oMySeq.wStatus == sequencer.STOPPED:
    print 'The sequencer is stopped'

else:
    print 'The sequencer is deleted'
```

4.4 Recorder File Format

The Gseos Recorder can record the data blocks defined in the block definition file (*.blk file). The data blocks contained in the binary recorder files can be accessed via a simple filter. The following description shows how to make the GSEOS Recorder data available for further evaluation by writing a specific filter.

The files consist of three different record types: File Header, Block Header and Block Body.

Every file starts with a File Header that allows you to check whether this file is a GSEOS Recorder file for your project. It has only one single file header.

Every GSEOS Recorder file may contain different block types. The first time a specific block is recorded in that file a Block Header is written. It contains some block specific data like the block name. The Block Header is followed by one or more Block Bodies, which contain the data. The next time a data block of that type is recorded only the Block Body is written to the file. This means that there is always exactly one Block Header and one or more Block Bodies for a block type in the recorder file. The following picture shows a possible structure of File Header, Block Headers and Block Bodies:

File Header
Block Header for Block "EDB"
Block Body for Block "EDB"
Block Body for Block "EDB"
Block Body for Block "EDB"
Block Header for Block "HK1"
Block Body for Block "HK1"
Block Body for Block "EDB"
Block Body for Block "EDB"
Block Body for Block "EDB"
Block Body for Block "HK1"
Block Body for Block "EDB"
Block Body for Block "EDB"
Block Body for Block "EDB"
Block Header for Block "HK2"
Block Body for Block "HK2"
Block Body for Block "EDB"
Block Body for Block "EDB"
Block Body for Block "EDB"
Block Body for Block "HK1"

Block Body for Block "HK2"
 Block Body for Block "EDB"
 Block Body for Block "EDB"
 Block Body for Block "EDB"

The layout of the FileHeader, Block Header and Block Body structures is as follows:

1. File Header

Name	Size	Contents	Description
byID[5]	5	0xEB 0x90 "GSE"	Unique identification
wVersion	2	0x0100	Version of recorder format
szProjectName[19]	19	Name of Project (zero terminated)	Identification of project
dwTime	4	Time in seconds since Jan/01/70 00:00:00	File creation time
dwNull	4	0	Spare

2. Block Header

Name	Size	Contents	Description
wID	2	"DE"	Block Header identification
wBlockID	2	File local block ID	Link to block bodies
szName[32]	32	Name of data block (zero terminated)	Name of data block
dwLen	4	40	Length of this structure

3. Block Body

Name	Size	Contents	Description
wID	2	"TA"	Block Body identification
wBlockID	2	File local block ID (link to header)	Link to block header
dwStamp	4	Stamp number	The stamp number must be consecutive for each block type
dwSize	4	Size of data area in bytes	
dwTime	4	Reception time in seconds since Jan/01/70 00:00:00.	
abyData	dwSize	data bytes	Block data
dwLen	4	wSize+20	Length of this structure

The following source code is a sample on how to parse the recorder data:

```

/*****
***/
/*
**/
/* DUMPDAT32.CPP
**/
/* Dumps the recorder contents of GSEOS recorder files for GSEOS 4.0 and
**/
/* higher.
**/
/*
**/
/* Copyright (C) 1997-1999, GSE Software, Inc.
**/
/* Author: Thomas Hauck (th)
**/
/*
**/
/* History:
**/
/* Feb-10-1999 th V 2.0 Adapted to new 32-bit structures.
**/
*****/

#include <stdio.h>
#include <stdlib.h>
#include <dir.h>
#include <dos.h>
#include <io.h>
#include <fcntl.h>
#include <time.h>

#define MAX_FILES 700
#define MAX_BLOCKS 400
#define MAX_ERRORS 100

#define BYTE unsigned char
#define WORD unsigned short
#define DWORD unsigned long

#define BOOL int
#define TRUE 1
#define FALSE 0

#define BLOCKNAME_LEN 32 // length of blockname

typedef struct DATABLOCK
{
    DWORD dwLastStamp ;
    int iBlockID ;
    char szName[BLOCKNAME_LEN] ;
} DATABLOCK ;

typedef struct DATAFILE
{

```

```
        char szName[16] ;
        DWORD dwTime ;
    } DATAFILE ;

typedef struct FILEHEADER
{
    BYTE byID[5];
    WORD wVersion;
    char szProjectName[19];
    DWORD dwTime;
    DWORD dwNULL;
} FILEHEADER ;

typedef struct BLOCKHEADER
{
    WORD wID;
    WORD wBlockID;
    char szName[BLOCKNAME_LEN];
    DWORD dwLen;

} BLOCKHEADER ;

typedef struct BLOCKBODY
{
    WORD wID;
    WORD wBlockID;
    DWORD dwStamp;
    DWORD dwSize;
    DWORD dwTime;
} BLOCKBODY ;

static BLOCKHEADER sBlkHdr ;
static BLOCKBODY sBlkBody ;
static FILEHEADER sFileHdr ;

static DATABLOCK sBlock[MAX_BLOCKS] ;
static WORD wBlockCount = 0;

static DATAFILE sFile[MAX_FILES] ;
static int iFileCount = 0 ;

static BYTE abyBuffer[0x20000] ;

int fCompTime(DATAFILE * p1, DATAFILE * p2)
{
    if (p1->dwTime < p2->dwTime)
        return -1 ;
    else if (p1->dwTime == p2->dwTime)
        return 0 ;
    else
        return 1 ;
}

int main(int argc, char * argv[])
{
    BOOL bWriteDirectory;

    BYTE abyID[2] ;
```



```
int i,j,
    iDone,
    iErrCount = 0,
    iBlockNumber,
    iFirstArg;

char szPath[64],
    szFileMask[64],
    szFileName[64],
    szProjectName[64] ;

DWORD dwLen;

struct ffbldk ffbldk ;

FILE * pFile ;

bWriteDirectory = FALSE ;

iFirstArg = 1 ;

while ((iFirstArg < argc) && (argv[iFirstArg][0] == '-'))
{
    switch(argv[iFirstArg][1])
    {
        case 'l' : bWriteDirectory = TRUE ;
                    break ;

        default : printf("Wrong option switch : %s\n",argv[iFirstArg]) ;
                    exit(-1) ;
    }
    iFirstArg++ ;
}

if (argc-iFirstArg != 2)
{
    printf("usage :\n") ;
    printf("dumpdat32 [switches] path projectname\n\n") ;
    printf("This program dumps GSEOS 4.0 data files from the specified\n\n") ;
    printf("path\n") ;
    printf("to standard output.\n") ;
    return -1 ;
}

strcpy(szProjectName,argv[iFirstArg+1]) ;
strcpy(szPath,argv[iFirstArg+0]) ;
if (strlen(szPath))
{
    if (szPath[strlen(szPath)-1] != '\\')
        strcat(szPath,"\\") ;
}

strcat(strcpy(szFileMask,szPath),"*.*)" ;

/* -----
   Build list of data files and sort them by file
   ----- */
```

```

*/

iDone = findfirst(szFileMask,&ffblk,0) ;

while (!iDone)
{
    fprintf(stderr,"\r%13s ...",ffblk.ff_name) ;

    /* -----
       open the file and read the header
       ----- */

    strcat(strcpy(szFileName,szPath),ffblk.ff_name) ;

    pFile = fopen(szFileName,"r+b") ;

    if (!pFile)
    {
        fprintf(stderr,"\n Error : '%s' could not be opened.\n",szFileName)
;
        exit(-1) ;
    }
    else
    {
        if (fread(&sFileHdr,sizeof(sFileHdr),1,pFile) != 1)
        {
            fprintf(stderr,"\n Warning : file header could not be read from
'%s'\n",szFileName) ;
        }
        else
        {
            if ( (sFileHdr.byID[0] == 0xEB ) &&
                (sFileHdr.byID[1] == 0x90 ) &&
                (sFileHdr.byID[2] == 'G' ) &&
                (sFileHdr.byID[3] == 'S' ) &&
                (sFileHdr.byID[4] == 'E' ) &&
                ((sFileHdr.wVersion == 0x0100) || (sFileHdr.wVersion ==
0x0200)) &&
                (strcmp(strupr(sFileHdr.szProjectName),strupr(szProjectName))
== 0) )
            {
                if (iFileCount < MAX_FILES)
                {
                    strcpy(sFile[iFileCount].szName,ffblk.ff_name) ;
                    sFile[iFileCount].dwTime = sFileHdr.dwTime ;
                    iFileCount++ ;
                }
                else
                {
                    fprintf(stderr,"\nWarning : too many files, can open only %d
files !\n",MAX_FILES) ;
                    fclose(pFile) ;
                }
            }
            else
            {
                fprintf(stderr," no valid data file\n") ;
            }
        }
    }
}

```

```

    }

    fclose(pFile) ;

}

iDone = findnext(&ffblk) ;

}

/* -----
   sort the files found by time
   ----- */

qsort(sFile,iFileCount,sizeof(DATAFILE),(int (*)(const void *a, const
void *b))fCompTime) ;

/* -----
   if a Directory is requested, push it out now
   -----
*/

if (bWriteDirectory)
{
    printf("Directory of %s data files\n\n",strupr(szProjectName)) ;
    for (i=0; i<iFileCount; i++)
    {
        printf("%12.12s  %s",sFile[i].szName, ctime(&(sFile[i].dwTime))) ;
    }
    exit (-1) ;
}

/* -----
-   start the output
-   -----
- */

printf("GSEOS 4.0 data dump\n") ;
printf("-----\n\n\n") ;
printf("Data path : %s\n",szPath) ;

/* -----
   loop over all files
   ----- */

for (i=0; i< iFileCount; i++)
{
    WORD w;

    /* -----
       open file and write message if not successfull
       ----- */

    iBlockNumber = 1 ;
    for (w=0; w<wBlockCount; w++)
    {
        sBlock[w].iBlockID = -1 ;
    }
}

```

```

    }

    strcat(strcpy(szFileName,szPath),sFile[i].szName) ;
    pFile = fopen(szFileName,"r+b") ;
    printf("\nFile : %s of %s\n",sFile[i].szName,ctime(&(sFile[i].dwTime)))
;

    if (!pFile)
    {
        extern char * sys_errlist[] ;
        extern int errno ;

        fprintf(stderr,"\n Error : '%s' could not be opened
(%s).\n",sFile[i].szName,sys_errlist[errno]) ;
        iErrCount++ ;
        if (iErrCount > MAX_ERRORS)
            goto TOO_MANY_ERRORS ;
    }
    else
    {
        /* -----
        read the header again and test the data
        ----- */

        if (fread(&sFileHdr,sizeof(sFileHdr),1,pFile) != 1)
        {
            printf(">> FileHdr could not be read from '%s'\n",sFile[i].szName)
;

            iErrCount++ ;
            if (iErrCount > MAX_ERRORS)
                goto TOO_MANY_ERRORS ;
        }
        else
        {
            if (sFileHdr.dwNULL != 0)
            {
                printf(">> FileHdr.dwNULL is not 0 but %4.4X\n",sFileHdr.dwNULL)
;

                iErrCount++ ;
                if (iErrCount > MAX_ERRORS)
                    goto TOO_MANY_ERRORS ;
            }

            /* -----
            read records from file until file is empty
            ----- */

            while (fread(abyID,sizeof(abyID),1,pFile) == 1)
            {
                if ( (abyID[0] == 'D') &&
                    (abyID[1] == 'E') )
                {
                    /* -----
                    block definition record
                    ----- */

                    if (fread(&(sBlkHdr.wBlockID),sizeof(sBlkHdr)-2,1,pFile) != 1)
                    {
                        printf(">> BlkHdr could not be read from

```

```

'%s'\n",sFile[i].szName) ;
        iErrCount++ ;
        if (iErrCount > MAX_ERRORS)
            goto TOO_MANY_ERRORS ;
    }
    else
    {

        /* -----
        test if block ID has already been used
        ----- */

        for (j=0; j<wBlockCount; j++)
        {
            if (sBlkHdr.wBlockID == sBlock[j].iBlockID)
            {
                printf(">> BlockID %d used twice.\n",sBlkHdr.wBlockID) ;
                printf("    Block '%s' will not be
recognized.\n",sBlkHdr.szName) ;
                iErrCount++ ;
                if (iErrCount > MAX_ERRORS)
                    goto TOO_MANY_ERRORS ;
                break ;
            }
        }

        if (j == wBlockCount)
        {
            WORD w = 0;
            while ( (w<wBlockCount) &&
                    (strcmp(sBlock[w].szName,sBlkHdr.szName)) )
                w++ ;

            sBlock[w].iBlockID = sBlkHdr.wBlockID ; // set a new
block ID

            if (w == wBlockCount)
            {
                strcpy(sBlock[w].szName,sBlkHdr.szName) ;
                wBlockCount++ ;
                sBlock[w].dwLastStamp = 0 ;
            }

            if (sBlkHdr.dwLen != sizeof(sBlkHdr))
            {
                printf("BlkHdr.dwLen is incorrect for definition of block
'%s' (%d)\n",
                    sBlkHdr.szName,sBlkHdr.dwLen) ;
                iErrCount++ ;
                if (iErrCount > MAX_ERRORS)
                    goto TOO_MANY_ERRORS ;
            }
        }
    }
}
else if ( (abyID[0] == 'T') &&
          (abyID[1] == 'A') )
{
    /* -----

```

```

        block data record
        ----- */

1)      if (fread(&(sBlkBody.wBlockID), sizeof(sBlkBody)-2, 1, pFile) !=
        {
            printf(">> BlkBody could not be read from
'%s'\n", sFile[i].szName) ;
            iErrCount++ ;
            if (iErrCount > MAX_ERRORS)
                goto TOO_MANY_ERRORS ;
        }
        else
        {
            for (j=0; j<wBlockCount; j++)
            {
                if (sBlkBody.wBlockID == sBlock[j].iBlockID)
                {
                    if (sBlock[j].dwLastStamp+1 != sBlkBody.dwStamp)
                    {
                        printf("\n>> Missing sample #%d of block %s. Next
recorded sample is #%d\n\n",
                            sBlock[j].dwLastStamp+1,
                            sBlock[j].szName,
                            sBlkBody.dwStamp) ;
                    }
                    printf("\nBlock %3d : %18.18s #%d:[%d] %s\n",
                        iBlockNumber++,
                        sBlock[j].szName,
                        sBlkBody.dwStamp,
                        sBlkBody.dwSize,
                        ctime(&(sBlkBody.dwTime))) ;
                    sBlock[j].dwLastStamp = sBlkBody.dwStamp ;
                    break ;
                }
            }
            if (j == wBlockCount)
            {
                printf("\n\n>> Block with ID %d is not defined
!\n\n", sBlkBody.wBlockID) ;
                iErrCount++ ;
                if (iErrCount > MAX_ERRORS)
                    goto TOO_MANY_ERRORS ;
            }

            if (fread(aByBuffer, sBlkBody.dwSize, 1, pFile) != 1)
            {
                printf(">> Data could not be read\n") ;
                iErrCount++ ;
                if (iErrCount > MAX_ERRORS)
                    goto TOO_MANY_ERRORS ;
            }
            else
            {
                int iPos ;
                int k ;

                iPos = 0;

```

```

for (j=0; j<sBlkBody.dwSize; j++)
{
    if (iPos == 0)
    {
        printf("%4.4X ",j) ;
    }

    printf("%2.2X ",(WORD)abyBuffer[j]) ;
    iPos++ ;

    if (iPos == 16)
    {
        printf(" ") ;
        for (k=j-15; k<=j; k++)
        {
            if (abyBuffer[k] >= 0x20)
                printf("%c",abyBuffer[k]) ;
            else
                printf(".") ;
        }
        printf("\n") ;
        iPos = 0 ;
    }
}

if (iPos != 0)
{
    j = j - iPos ;
    for (k=iPos; k<16; k++)
        printf(" ") ;
    printf(" ") ;
    for (;j<sBlkBody.dwSize; j++)
    {
        if (abyBuffer[j] >= 0x20)
            printf("%c",abyBuffer[j]) ;
        else
            printf(".") ;
    }
    printf("\n") ;
}
printf("\n") ;
}

if (fread(&dwLen,sizeof(dwLen),1,pFile) != 1)
{
    printf(">> Back Pointer could not be read\n") ;
    iErrCount++ ;
    if (iErrCount > MAX_ERRORS)
        goto TOO_MANY_ERRORS ;
}
else
{
    if (dwLen != sBlkBody.dwSize+sizeof(sBlkBody)+sizeof(DWORD))
    {
        printf(">> Back Pointer has wrong value (0x%4.4X), expected
0x%4.4X\n",

```

```
                dwLen,
sBlkBody.dwSize+sizeof(sBlkBody)+sizeof(DWORD)) ;
                iErrCount++ ;
                if (iErrCount > MAX_ERRORS)
                    goto TOO_MANY_ERRORS ;
            }
        }
    }
    else
    {
        printf(">> Record Header has wrong format : 0x%4.4X\n",
abyID[0], abyID[1]) ;
        iErrCount++ ;
        if (iErrCount > MAX_ERRORS)
            goto TOO_MANY_ERRORS ;
    }
}
}
fclose(pFile) ;

}
}

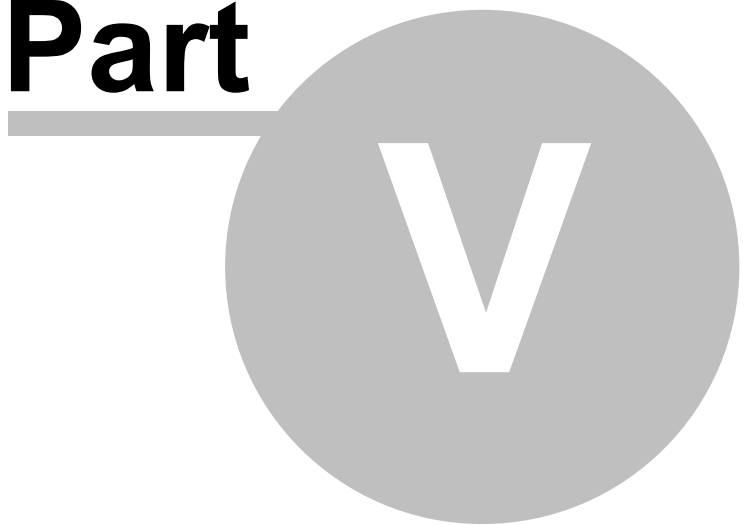
return 0 ;

TOO_MANY_ERRORS :
printf("\nToo many errors, program aborted.\n") ;

return -1 ;

}
```


Part



5 How do I...

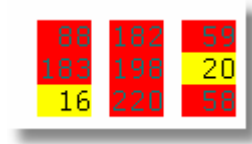
5.1 How do I display alarm information?

Setting up an alarm item is a three step approach:

- Create your alarm definition and load an alarm file.
- Place the data item on a screen.
- Select the alarm and configure the alarm properties.

Once you have the alarm loaded you will be able to verify it in the GSEOS Explorer.

The image below shows a screen shot of a data item that has an alarm set up.



5.2 How do I configure an alarm monitor?

Alarm monitors are useful to inform you of certain conditions in your telemetry data.

An Alarm Monitor monitors one data item with a trigger condition. The alarm trigger condition can be any operation that returns a boolean result. You can define the actions that you want to invoke once the alarm fires. The possible actions are:

- Write message to the message window.
- Write message to a log file.
- Send command.
- Execute Python function.
- Send email.

You configure the alarm monitor in a GSEOS configuration file. For more information on the alarm monitor configuration file please see the Alarm Monitor Configuration section.

5.3 How do I configure the networking module?

GSEOS network support is very flexible and accommodates various different networking configurations. The basic concept of the network module is to import/export data blocks via the TCP/IP protocol. The network module functions as a data source when importing data. You can configure any number of network connections. Each network connection can be associated with at most two blocks. One that gets exported on this connection and one that gets imported. From a network perspective GSEOS can act as a server or a client. For each connection you have to specify if you want GSEOS to act as a server or a client. This does not necessarily determine if you export or import blocks on that connection. A common scenario is to configure a server connection and export a block on that connection. However you may as well configure a client connection and export a

block on that connection.

Configuration options

The network module is configured in the gseos.ini configuration file. Specify all your connections in the [Net] section (or an according instance specific name). The keys you specify in this section are the names of the connections you want to configure. The value can be either Server or Client for a network server or a network client respectively.

```
[Net]
TLMSrv=Server
TestServer1=Server
TomsServer=Server
SOPC33=Server
TLMClnt=Client
CmdSrc=Client
TestClient=Client
```

The above example configures four server connections and three client connections. You can manage these connections from within the GSEOS Explorer. In order to configure the individual connections you have to create new sections with the connection name for the section name, e.g.:

```
[TLMSrv]
Port=2001
Source=TLM
```

The section above specifies the setting for the TLMSrv server connection. This particular example configures the server to listen on port 2001 and export the TLM block. The following sections discuss the various options you can specify. The settings that only apply to client connections are indicated.

Key	Assignment
IP-Address	Only for Client connections. The IP address of the remote server. Specify the IP address in 4-byte dotted format, e.g. 150.144.103.23
Port	The port number of the remote machine. There must be a server listening on this port in order for a connect attempt to be successful.
Source	The data block you want to EXPORT on this connection. Every time the system encounters this block it will send the contents of the block to the remote machine. The actual amount of data sent depends on the VariableLen setting.
Destination	The data block you want to IMPORT on this connection. All data received from the server will be written to this block. Once the number or bytes specified in the block definition is received the block is submitted to the system. (This is the default behavior and can be modified with the VariableLen setting.)
AutoConnect	Only for Client connections. Allows to automatically connect to a server. Specify a number of seconds that will elapse before an attempt is made to connect to the remote machine. If the connection is already established no attempt to connect will be made. If you set this value to 0, the default value, automatic connection is disabled.
VariableLen	This setting controls the amount of data sent over the network connection. The default is "No". For the source block the amount of bytes specified in the block definition file is sent. for the destination block the amount of bytes specified in the block definition has to be received before a block is generated. This setting is preferred for inter GSEOS connections or connections that generate fixed length data. If you specify "Yes" for this setting the connection uses variable length packets. The blocks specified in either Source or Destination have to have a 32-bit field called "Len" at the beginning of their block definition. For Source blocks the Len field specifies how many bytes of data are transferred. The Len field itself is not sent, only the data immediately following the Len field. For Destination blocks the Len field is filled with the amount of data read from the network. When more data is received than can be placed in the block multiple blocks are generated.
Exclusive	The network module is considered a data source. The default behavior for the network will be to discard all data received on the network connection unless the network is enabled. There are some circumstances where this is not desirable. E.g. consider the case of remote commanding. In this case we may have incoming data from the Bios but want to be able to feed in command data over the network. If we were to enable the network the Bios data would be discarded, not an option. However if the Bios is on all command data from the network would be discarded. To enable network input while getting data from another data source set this value to "No" and do not enable the network. The default is "Yes" which means all incoming data from the network is discarded unless the network is enabled.

Connecting two GSEOS computers

Oftentimes it is desirable to distribute the data decoding/display to various machines. This can easily be done by having one machine exporting a data block and the other importing the same block. The default behavior of a connection is to export/import the

entire block. This is a fixed size packet based on the block definition for the block you import or export. This is what you need to interconnect two GSEOS machines (given of course that the block definitions on both machines are the same!). The decision which machine to configure as server and which one as client pretty much depends on where you want to initiate the connection from. The client machine has to initiate the connection. Lets assume we have two machines, the Lab machine with the physical data connection to the instrument and an Office machine where we want to run remote display. The Lab machine will be configured as server and the Office machine as client so we can start the remote display from the Office machine. The block exported by the Lab machine and imported into the client machine is TLM. We also want to enable commanding from the Office machine. This means we have to set the Exclusive setting on the Lab machine to "No". If we don't want to enable commanding we would not need to set the Exclusive flag to "No" and we would not need to specify the CMDSTRING block in either configuration.

Here the configuration for the Lab machine:

```
[Net]
TLMSrv=Server
[TLMSrv]
Port=2020
Source=TLM
Destination=CMDSTRING
Exclusive=No
```

Here the configuration for the office machine:

```
[Net]
TLMClient=Client
[TLMSrv]
IP-Address=150.134.123.87
Port=2020
Source=CMDSTRING
Destination=TLM
```

5.4 How do I configure startup settings?

When GSEOS starts up your custom configuration files can be loaded to setup GSEOS for your environment. There are two categories of startup files that can be loaded:

- Configuration Files
- Python scripts

Loading of Configuration files at startup

Configuration files are screen files (*.scr), command menu files (*.cm), desktop files (*.dt), log files (*.log), formula files (*.qlf), text reference files (*.tr), and any custom files you register with GSEOS. These can be loaded automatically by specifying the file name in the gseos.ini [Config] section. See the example below for a typical configuration entry:

```
[Config]
Load = System\System.qlf Common\Pluto.qlf i_LORRI\LOR_cust.qlf
Load = system\system.cpd Common\sce.cm
Load = Common\common.tr Common\basic.tr Common\Common_ltgray.tr
Load = i_LORRI\LOR_cust.tr
```

```
Load = i_LORRI\CMD_n_TLM\ApId_601.qlf
Load = i_LORRI\CMD_n_TLM\ApId_601.alarm
Load = i_LORRI\CMD_n_TLM\ApId_601.tr
```

It is also possible to load Python files (*.py, *.pyd, *.dll) here, this is only recommended for simple configurations, complex (mission or multiple instrument) configurations should use the alternate approach listed below.

Loading of Python scripts at startup

It is often it is necessary load Python scripts at startup time. There are several different approaches with different consequences:

1. Specify the scripts to load in the gseos.ini [Config] section.
2. Specify the scripts to load in a command batch file that is specified in the [Config] section of the gseos.ini file.
3. Use a Python startup script that imports all the necessary configuration scripts.

1. Using the [Config] section

The first approach is the simplest and most straightforward, you can just specify the Python scripts to load at startup time in the gseos.ini file [Config] section. You have to use the Load entry. You can specify multiple files for one Load entry and you can also specify as many Load entries in the [Config] section as you like. Please see the sample below:

```
[Config]
BlkFiles=system\system.blk Messenger.blk common\pkt_tlm.blk rtiu\rtiu.blk
Load = LP1553Bios.dll common\common_cmds.py core\test_arg.py
Load = core\RTIUDec.py common\embx_cmds.py core\core.py common\config.py
Load = system\system.cpd common\com_DPU\instrument.cm
common\com_DPU\embox.cm
Load = Instrument\i_DPU\startup_dpu.cpb
```

Multiple entries on one Load line are separated by white-spaces. In general it is probably the best approach to just specify one file per line.

Two things happen when these files get loaded:

The system checks if the path specified is already in the sys.path (the Python search path). If it is not the path gets prepended to sys.path to load the file. Once the file is successfully loaded the sys.path is restored to its previous contents.

If the module has been loaded before it will get reloaded, otherwise it will be imported.

The import is run in the __main__ namespace context. Say you specify xyz.py, this means you can access the module xyz directly from the __main__ namespace, i.e. the console window: `dir(xyz)`.

However, one limitation of this approach is that you can't do the equivalent of:

```
from xyz import *
```

and therefore import the contents of the module into the __main__ namespace. Say you define a command script MyCmds.py and you want to issue the commands directly from a button or the console window. You would need to invoke a command like so: `MyCmds.POWER_ON()`. The next approach will show you how to address this problem. Also, loading a package is not possible only Python modules can be loaded with this approach.

For simple configurations this is the recommended approach. More complex

configurations for entire mission support or multiple instrument configurations should use approach 3.

Note:

In general it is a good idea to explicitly use namespaces and only import an entire module. If you load everything directly into the `__main__` namespace this namespace gets cluttered and it is difficult to track down where individual attributes are defined.

2. Using a command batch (*.cpb) file

With this approach we can address the problem we encountered with directly specifying the Python scripts in the Using batch files for startup configuration is strongly discouraged since it might introduce time dependent behavior that is hard to control. Please refer to the next section for Python script based startup configuration.

3. Python script based startup configuration

The problem with the simple approach of loading Python scripts directly from the [Config] section is that you don't have the equivalent of:

```
from xxx import *
```

To facilitate this you can use the [PyStartup] section in the gseos.ini file. The [PyStartup] section has two keys: 'Import' and 'Exec'. 'Import' lets you import Python modules as well as Python packages. The module/package has to be on the search path to be loaded successfully.

'Exec' lets you execute an Python statement. We will use this feature to import the contents of the startup script into the `__main__` namespace. Please see the example below for a startup configuration using the 'Exec' approach:

```
[PyStartup]
Import = TC_TLM_Load
Exec   = from Common.Startup import *
Exec   = from i_LORRI.StartupMaster import *
```

The statements are executed in the order listed. You can enter any number of startup scripts you need. However, it is recommended to perform all imports and other configuration within your startup script.

GSEOS startup order

The following lists the order in which GSEOS bootstraps the configuration files:

- Load block definition files from [Config] BlkFiles entries.
- Execute [PyStartup] Import/Exec entries.
- Load configuration files from [Config] Load section.

Since batch files are time depended they will be executed when loaded from the [Config] Load section but may be preempted by other scripts depending on the processing of the message queue. Therefore it is not recommended to load batch files at startup.

5.5 How do I open a screen file programmatically?

Sometimes it is desirable to open or close GSEOS screen files from a script. To open a screen window you use the FileOpen() method of the GseosSys module. You specify the file name of the screen file you wish to open and the file gets loaded and the window

displayed.

The window is displayed on the active desktop page. If you want to locate the screen on a different desktop page you have to activate that desktop page. You can do this with `Gseos.SetActiveDesktopPage()`.

Once a screen is open you can change it's appearance with `WindowMinimize()`, `WindowMaximize()`, or `WindowRestore()`. You can also close it with `WindowClose()`. All these functions are located in the `GseosSys` module. Note that the `Window...` functions take the window caption as their parameter to identify the screen you want to operate on. So if you have assigned a title to the window that is different from the file name you want to use that title to access the correct window.

Oftentimes you might want to activate or restore a window if it exists, and if it doesn't you want to load it. This will make sure you don't load the same window multiple times. The following example defines a command that will open the window if it exists and load it otherwise. `fOpenScreen()` assumes that the title of the window is that same as the file name. This is also the parameter you have to pass into the function. `fOpenScreen()` take advantage of the fact that GSEOS throws a `RuntimeError` when it can't locate the window you try to restore. It then tries to load the file with `FileOpen()`.

Example

```
import GseosSys

def fOpenScreen(strScreenFileName):
    try:
        GseosSys.WindowRestore(strScreenFileName)

    except RuntimeError:
        GseosSys.FileOpen(strScreenFileName)
```

5.6 How do I use Expressions and Conversion Functions?

GSEOS offers two different kinds of mathematical conversions for data items and display purposes. Expressions are general purpose formulas that can take data items as well as constants as parameters and that can be displayed on a screen. Conversion functions are bound to a particular data item and perform a conversion for this specific data item, usually an engineering unit conversion.

Both Expressions and Conversion functions are defined in a formula file. This formula file needs to be loaded for the formulas to be accessible (as opposed to GSEOS 5.0 and earlier where the formula file was referenced directly from the screen file).

Once you have defined and loaded your Expressions and Conversion functions you can access them by placing an Expression object on a screen. If you place a simple data item that has a conversion function associated you can select the conversion function with the item select dialog.

Expressions and Conversions can also be accessed from Python and are available in the Conversion module. In order to use your functions all you have to do is import the

Conversion module:

Example

```
import Conversion
Conversion.Calibrate(3, Rates.Mass[0], 0.9899)
```

5.7 How do I write a GSEOS extension DLL?

Dynamic linking of Python extension DLLs is supported as of Version 4.1. It is pretty straightforward as explained in the Python extension manual. However there are a couple of things to watch out for. You need a compiler (C/C++) that is capable of generating Windows 32-bit DLLs, you also have to be familiar with writing Python extension modules. The Python documentation provides a good overview. When you extend GSEOS you have to link the appropriate export library. For Borland use gseos45bc.lib, for Microsoft use gseos45ms.lib. You should also use the GSEOS specific Python header files. The following source demonstrates how to write a simple extension module. You can use this file as a skeleton to build your own modules. Once you have created your DLL (by convention all compiled Python DLLs have the extension .pyd) and placed on the GSEOS path (type sys.path in the console window to learn about the search path, you can also add a path right there) you should be able to import your module with:

```
import dynaload
```

Here the dynaload source code:

```
/*
*****
*/
/* *
* */
/* * DYNALOAD.CPP
* */
/* *
* */
/* * Demonstrate loading of dynamic python modules.
* */
/* *
* */
/* * History:
* */
/* * Sep-08-98 th V1.0 File creation
* */
/*
*****
*/

/*
*****
*/
/* * Imports
* */
```

```

/*
*****
*/
#include "python.h"

/* -----
- */
/* - Data
- */
/* -----
- */
static PyObject* pPyModuleError = NULL;

/* -----
- */
/* - Implementation.
- */
/* -----
- */
#pragma argsused
PyObject* fPyDLAdd(PyObject* pSelf, PyObject* pArgs)
{
    int iLeft;
    int iRight;

    /* -----
- */
    /* - Get the arguments.
- */
    /* -----
- */
    if (!PyArg_ParseTuple(pArgs, "ii", &iLeft, &iRight))
        return(NULL);

    /* -----
- */
    /* - Return the sum of the arguments.
- */
    /* -----
- */
    return(PyInt_FromLong(iLeft+iRight));
}

/*
*****
*/
/* * DL exported methods
* */
/*
*****
*/
static PyMethodDef sDLMethods[] =
{
    {"Add",          fPyDLAdd, 1},
    {NULL,           NULL}
};
/*

```

```
*****
*/
/* *   void initdynaload()
* */
/* *
* */
/* *   Description : The module initialization function.
* */
/*
*****
*/
extern "C" void __declspec(dllexport) initdynaload(void)
{
    PyObject* pPyModule;
    PyObject* pPyDict;

    /* -----
- */
    /* - Create the module and add the functions
- */
    /* -----
- */
    pPyModule = Py_InitModule("dynaload", sDLMETHODS);

    /* -----
- */
    /* - Add some symbolic constants to the module
- */
    /* -----
- */
    pPyDict      = PyModule_GetDict(pPyModule);
    pPyModuleError = PyErr_NewException("dynaload.error", NULL, NULL);
    PyDict_SetItemString(pPyDict, "error", pPyModuleError);
}
```

Index

- A -

About this document 14
Alarm 98
Alarm Limits 98
Application 90
Architecture 18
Architecture Overview 18

- B -

Block 99
Block Definition 99

- C -

Change Report 14
Command 84
Command Window 84
config file 113
configuration file 113
Console Window 85
Conversion Functions 110

- D -

Data Export 87
Data Flow 18
DataExport 87
Decoder Explorer 27

- E -

Export 87
Expression 110

- F -

Features 14
File Open 141
FileOpenDialog 141
Formula 110

- G -

Gseos Main Window 90
gseos.ini 113

- L -

log 34
Log windows 34
logging 34

- M -

Main Window 90
message 34
Message Window 91
Modules 20, 131

- P -

Python 131

- R -

Recorder 92
Recording 92
Red Alarm 98

- S -

Screen 42
System Structure 20

- Y -

Yellow Alarm 98