



GSEOS 7.1 User Manual

Copyright GSE Software, Inc. 1998 - 2017

Table of Contents

Foreword	0
Part I Welcome to GSEOS	1
1 Version	1
2 What's New	2
3 Features	2
4 Introduction	3
5 A Quick Tour	5
Add a screen window	6
Histogram Data	10
Part II Architecture Overview	14
1 Publish Subscriber Data Bus	15
2 Real-time control	15
Part III Binary Streamers	15
Part IV Commanding	17
Part V Logging	18
Part VI Graphs and Charting	21
Part VII User Interface	25
1 The GSEOS Main Window	25
2 Desktop Management	26
3 Menus	27
The File Menu	28
The Edit Menu	30
The Help Menu	31
The View Menu	32
The Tools Menu	34
The Data Export Tool.....	34
The Window Menu	37
4 Screen Windows	38
Placing objects	39
Selecting a Drawing Tool..	39
Selecting a Drawing Region.....	40
Selecting Object Properties.....	41
Adjust Display Style.....	43
Selecting Objects	44
Menus	47
Draw	48
Line	49

Rectangle	49
Rounded Rectangle.....	49
Ellipse	49
Image	49
Text	50
Scale	50
Data Item	51
Command Button.....	55
Expression	57
Format	59
Font	60
Color	61
Line	62
Range	62
Orientation	63
Data Item	65
Update Properties.....	66
Alarm Properties.....	68
ASCII	69
Bargraph	69
Bitmap	71
Float	74
Integer	75
RGB Bitmap	77
Status Image	78
Status Text	79
Stripchart	81
Description	85
Window	85
5 Graph Windows	86
Graph Concepts	90
Layout	91
Update and Sampling.....	93
Range	95
Graph Properties	97
Chart Properties	99
Context Menu	103
Stripcharts	105
1D Histogram	110
2D Histogram	113
Bargraph	116
Scatterplot	117
Troubleshooting	119
6 Log Windows	120
7 GSEOS Explorer	121
Alarms	121
Binary Streamers	122
Blocks	123
Commands	124
Conversions	125
Decoders	126
Expressions	127
File Uploads	127

Monitors	127
Network	128
System	131
8 The Command Dialog	133
9 The Console Window	135
10 The Data Export Dialog	138
11 The Message Window	140
12 The Alarm Window	142
13 The Recorder Dialog	145
14 The Options Dialog	147
Part VIII GSEOS Reference	147
1 Command Line Options	147
2 Directory Structure	148
3 Configuration Files	148
Alarm Limit Files (*.alarm)	149
Block Definition Files (*.blk)	150
Command Batch Files (*.cpb)	154
Command Definition Files (*.cpd)	156
Command Menu Files (*.cm)	161
Configuration Files (.cfg)	163
Alarm Monitor Configuration.....	163
Alarm Monitor Sample.....	167
Formula Definition Files (*.qlf)	168
gseos.ini	171
BinaryStreamers.....	174
Buffer	178
Command.....	178
Config	179
Console	179
FileUploads.....	180
GeosGraph.....	182
Instance	182
License	183
Logs	183
Message	186
Net	187
Project	191
PyStartup.....	191
QLook	192
Recorder.....	193
Timebase	194
System	195
Status Definition Files (*.tr)	196
4 GSEOS Python Interface	198
Modules	199
Geos	200
Geos.Alarm	200
Geos.AddEventHandler.....	201
Geos.ConsoleWrite.....	201

Gseos.Exit	202
Gseos.FileAppend.....	202
Gseos.FileMenu.....	202
Gseos.FileNew	203
Gseos.FileOpen.....	204
Gseos.FileOpenDialog.....	204
Gseos.FileSaveDialog.....	204
Gseos.GetActiveDesktopPage.....	205
Gseos.GetIniFileName	205
Gseos.GetInstrumentPath.....	205
Gseos.GetInstance.....	206
Gseos.GetProjectPath.....	206
Gseos.GetSystemLoad.....	206
Gseos.GetWindow Pos.....	206
Gseos.GetWindowSize.....	207
Gseos.Help	207
GseosInputDialog.....	208
Gseos.Idle	208
Gseos.ListStatusTextMaps.....	208
Gseos.ListStatusTextMapItems.....	209
Gseos.Log	209
Gseos.LogReload.....	210
Gseos.LogSave.....	210
Gseos.LookupStatusText.....	211
Gseos.MakePathAbsolute.....	211
Gseos.MakePathRelative.....	211
Gseos.MakePathNormal.....	212
Gseos.Message.....	212
Gseos.MessageBox.....	213
Gseos.PrintDesktop.....	213
Gseos.PumpWaitingMessages.....	214
Gseos.RemoveEventHandler.....	214
Gseos.SetActiveDesktopPage.....	214
Gseos.SetStatusBarText.....	215
Gseos.SetSplashScreenText.....	215
Gseos.SetSplashScreenVersion.....	216
Gseos.ShellExecute.....	216
Gseos.StartApplication.....	216
Gseos.Version	217
Gseos.WaitDialog.....	217
Gseos.Window Close	218
Gseos.Window Maximize.....	218
Gseos.Window Minimize.....	218
Gseos.Window Move.....	219
Gseos.Window Print.....	219
Gseos.Window Resize.....	220
Gseos.PlaySound.....	221
Gseos.Window Restore.....	221
GseosBinaryStreamer.....	222
GseosBinaryStreamer.ListBinaryWriters.....	222
GseosBinaryStreamer.StartBinaryWriter.....	222
GseosBinaryStreamer.StopBinaryWriter.....	222
GseosBinaryStreamer.SetBinaryWriterPathTemplate.....	223
GseosBinaryStreamer.SetBinaryWriterFileTemplate.....	223

GseosBlocks.....	223
TBDMBlock	227
TBDMBlock.GetBlockCountStamp.....	228
TBDMBlock.IsScalarItem.....	228
TBDMBlock.ReadBlockAsString.....	228
TBDMBlock.ReadItem.....	229
TBDMBlock.ReadString().....	230
TBDMBlock.SendBlock.....	231
TBDMBlock.WriteBlockFromString().....	231
TBDMBlock.WritItem.....	232
TBDMBlock.WriteString().....	232
TItemDescriptor	233
GseosBlocks.EnableDataSource.....	235
GseosBlocks.IsEnabled.....	235
GseosBlocks.TThreadSafeBlock.....	235
GseosCmd.....	236
GseosCmd.ListCmdMnemonics.....	237
GseosCmd.GetCmdOpcode.....	237
GseosCmd.EnableRangeCheck.....	237
GseosCmd.ExecCmd.....	238
GseosCmd.GetCmdArgDoc.....	239
GseosCmd.GetCmdArgName.....	239
GseosCmd.GetCmdDoc.....	240
GseosCmd.IsRangeCheckEnabled.....	240
GseosCmd.SendBinCmd.....	240
GseosCmd.StartBatch.....	241
GseosCmd.StopBatch.....	242
GseosConversion.....	242
GseosConversion.dtoll.....	243
GseosConversion.ftol.....	244
GseosConversion.ltol.....	244
GseosConversion.ltolf.....	245
GseosConversion.signed.....	245
GseosDecoder.....	246
GseosDecoder.TDecoder.bEnable.....	247
GseosDecoder.TDecoder.....	247
GseosDecoder.TDecoder.Delete.....	248
GseosDecoder.TDecoder.dw Cnt.....	249
GseosDecoder.TDecoderStatus.....	249
Examples	251
Simple Decoder.....	251
Variable Length Decoder.....	254
GseosDecoder.TDecoder.strName.....	256
GseosError.....	256
GseosExport.....	257
GseosExport.IsEnabled.....	257
GseosExport.Enable.....	257
GseosExport.GetActiveConfig.....	258
GseosExport.SetActiveConfig.....	258
GseosIni	258
GseosIni.fAddEntry.....	259
GseosIni.fGetEntry.....	259
GseosIni.fGetEntries.....	260
GseosIni.fIsEnabled.....	260

GseosIni.fListEntries	260
GseosIni.fListSectionNames	261
GseosIni.fRemoveEntries	261
GseosIni.fRemoveSection	261
GseosIni.fSetEntry	261
GseosIni.fWriteSection	262
GseosFileUpload	262
StartUpload	262
AbortUpload	263
IsUploadActive	263
GseosHistogram	264
GseosHistogram.Clear	267
GseosHistogram.Dow nYScale	267
GseosHistogram.MoveLeft	268
GseosHistogram.MoveRight	268
GseosHistogram.MoveUp	268
GseosHistogram.MoveDow n	268
GseosHistogram.SetAutoScaleMode	269
GseosHistogram.SetCalcMode	269
GseosHistogram.SetHorzOffset	269
GseosHistogram.SetVertOffset	270
GseosHistogram.SetLogMode	270
GseosHistogram.SetYScale	270
GseosHistogram.SetZoom	271
GseosHistogram.ToggleAutoScaleMode	271
GseosHistogram.ToggleLogMode	272
GseosHistogram.UpYScale	272
THistogram1D	272
Example	274
THistogram2D	275
Example	278
GseosLog	280
GseosLog.Log	280
GseosLog.Reload	281
GseosLog.Print	281
GseosLog.AddAutoLog	282
GseosLog.AutoLogUpdateSettings	282
GseosLog.ListAutoLogNames	283
GseosLog.GetAutoLogInfo	283
GseosLog.AutoLogNew Session	283
GseosLog.AutoLogShow	284
GseosLog.AutoLogPin	284
GseosLog.AutoLogClose	284
GseosMonitor	285
GseosMonitor.TMonitor.bEnable	285
GseosMonitor.TMonitor	286
GseosMonitor.TMonitor.Delete	287
GseosMonitor.TMonitor.dw Cnt	287
Examples	287
CounterCheck	287
LimitCheck	288
GseosMonitor.TMonitor.strName	290
GseosNet	290
GseosNet.ClientConnect	291

GseosNet.ClientDisconnect.....	291
GseosNet.ClientStatus.....	291
GseosNet.Disable.....	292
GseosNet.Enable.....	292
GseosNet.IsEnabled.....	293
GseosNet.ServerReset.....	293
GseosNet.ServerStatus.....	293
GseosRecorder.....	293
GseosRecorder.AddPlaybackBlock.....	294
GseosRecorder.AddRecordBlock.....	294
GseosRecorder.GetDataPath.....	295
GseosRecorder.GetPlaybackBlocks.....	295
GseosRecorder.GetPrefix.....	295
GseosRecorder.GetRecordBlocks.....	296
GseosRecorder.IsPlayingBack.....	296
GseosRecorder.IsRecording.....	296
GseosRecorder.RemovePlaybackBlock.....	296
GseosRecorder.RemoveRecordBlock.....	297
GseosRecorder.SetDataPath.....	297
GseosRecorder.SetPrefix.....	297
GseosRecorder.StartPlayback.....	298
GseosRecorder.StartRecording.....	298
GseosRecorder.StopPlayback.....	298
GseosRecorder.StopRecording.....	298
GseosSequencer.....	299
GseosSequencer.TSequencer.....	303
TSequencer.Delete.....	304
TSequencer.GetStatus.....	304
TSequencerInputDialog.....	305
TSequencerInputDialogModeless.....	305
TSequencer.MessageBox.....	306
TSequencer.MessageBoxModeless.....	306
TSequencer.TriggerBlock.....	307
TSequencer.Sleep.....	308
TSequencer.Start.....	308
TSequencer.Stop.....	309
TSequencer.Wait.....	309
3rd Party Packages	310
5 Recorder File Format	311
Part IX Cross Platform Deployment	324
Part X How do I...	326
1 How do I display alarm information?	326
2 How do I configure an alarm monitor?	326
3 How do I configure the networking module?	326
4 How do I configure startup settings?	329
5 How do I open a screen file programmatically?	331
6 How do I implement a file upload?	332
7 How do I use Expressions and Conversion Functions?	341

Index

343

1 Welcome to GSEOS

Ground Support Equipment Operating System



1.1 Version

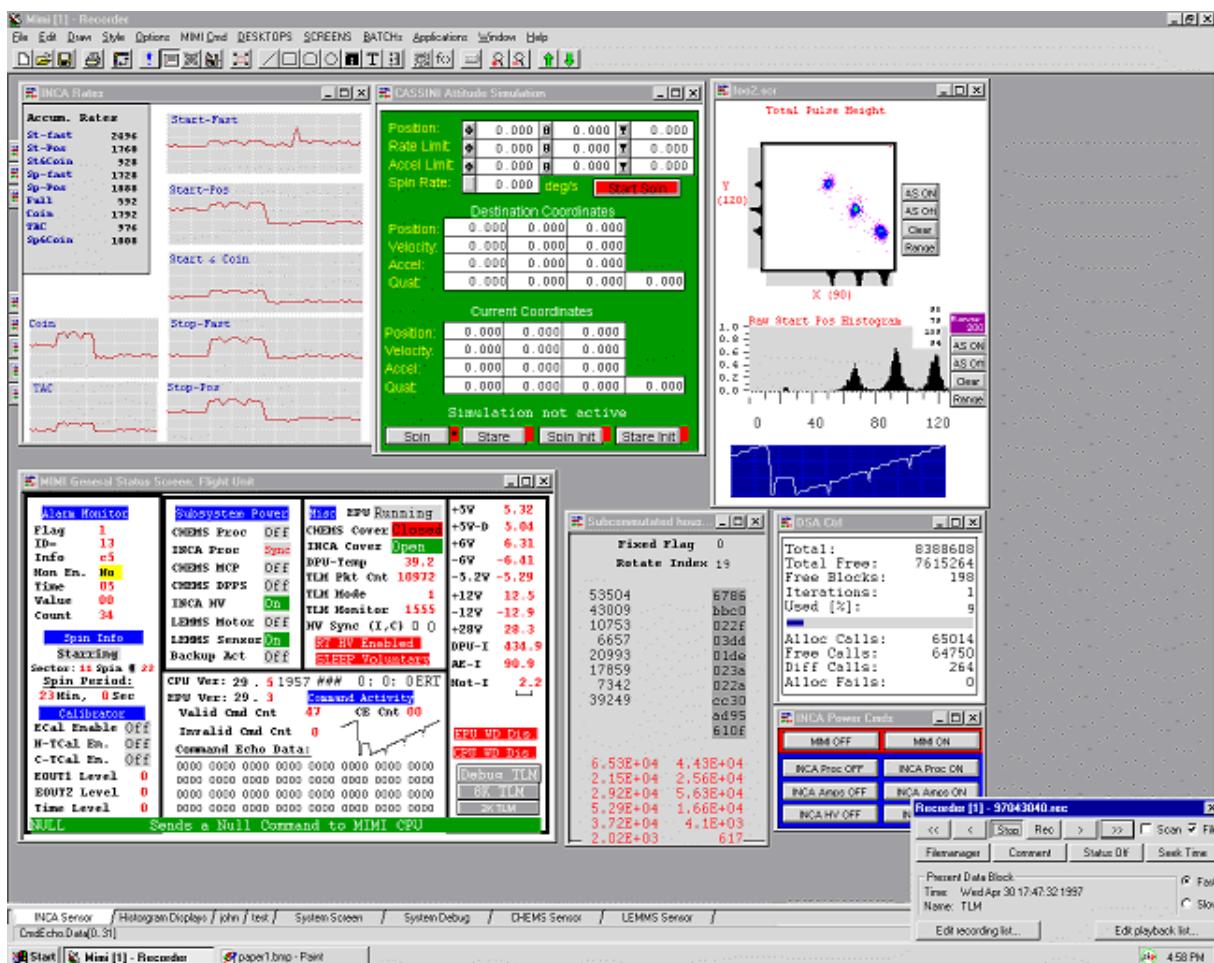
Document	GSEOS	Date	Notes
ation			
R160	7.1.1125	May/13/2 017	Completed GseosGraph package.
R159	7.1.1117	Feb/23/2 017	Rename GseosPlot to GseosGraph.
R158	7.1.1115	Feb/06/2 017	GSEOS 7.1 Installation of 3rd party packages. Add documentation for GseosGraph Add documentation for GseosIni Update documentation for GseosCmd.ExecCmd
R157	7.0.1075	Aug/06/2 016	Add functions GseosRecorder.StartPlayback() and GseosRecorder.StopPlayback()
R156	7.0.1053	Jun/07/20 15	Add function TBDMBlock.WriteString().
R155	7.0.1052	May/16/2 015	Added little endian command arguments.
R154	7.0.1049	Apr/22/20 15	Add BinaryStreamer functions to set path and file templates.
R153	7.0.1037	Dec/12/2 014	Add Logging functions and GseosLog module.
R152	7.0.1031	Sep/11/2 014	Add Command setting: CmdHistoryDialogOnly.
R151	7.0.1025	May/24/2 014	Added new GseosCmd API functions.
R150	7.0.1000	Jan/20/20 12	Updated documentation for GSEOS 7.

1.2 What's New

GSEOS 7.1 upgrades the built-in Python interpreter from Python 2.7.3 to Python 2.7.12. This and other changes to GSEOS 7.1 make it significantly easier to install 3rd party packages. GSEOS 7.1 also features a new plot feature with the GseosGraph extension. This allows you to quickly design stripcharts.

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

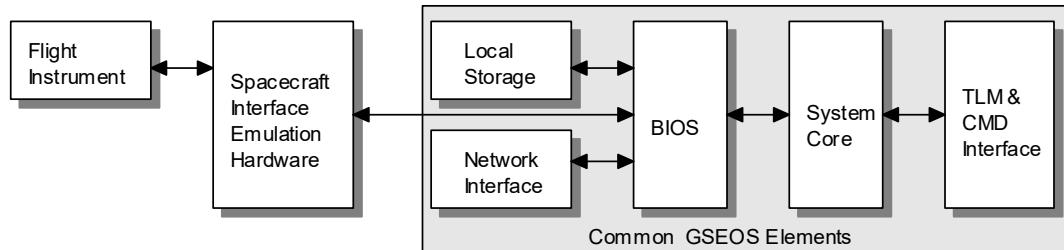
limitations, and the fact that it is not intended for long-term use, the BCE development often requires significant resources to insure that the hardware delivery is not imperiled.

Spacecraft Integration

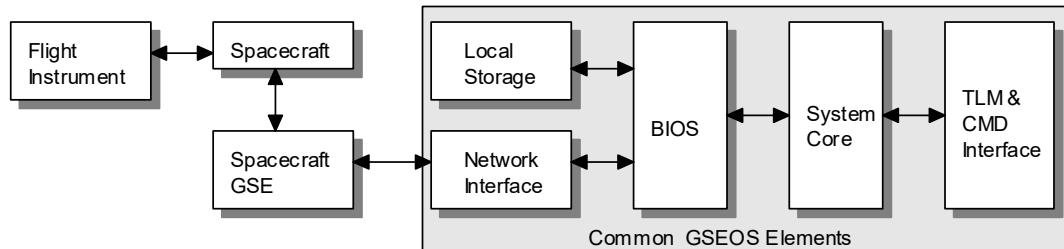
A second system, developed in parallel with the BCE, is intended primarily to support mission operations. Because such systems can not be fully checked until late in the hardware development cycle, significant resources are expended to run simulated hardware interface tests. Such tests are not a complete 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 manyfold. The picture below depicts three possible system configurations:

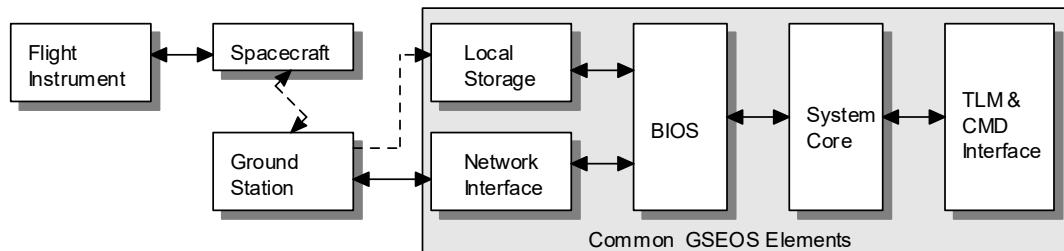
Assembly-Level Test Configuration



System -Level Test Configuration



Mission Operations Configuration



1.5 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](#)^[147] 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 describe the various files you will have to create/modify in order to set up a working system.

[gseos.ini](#)^[171]

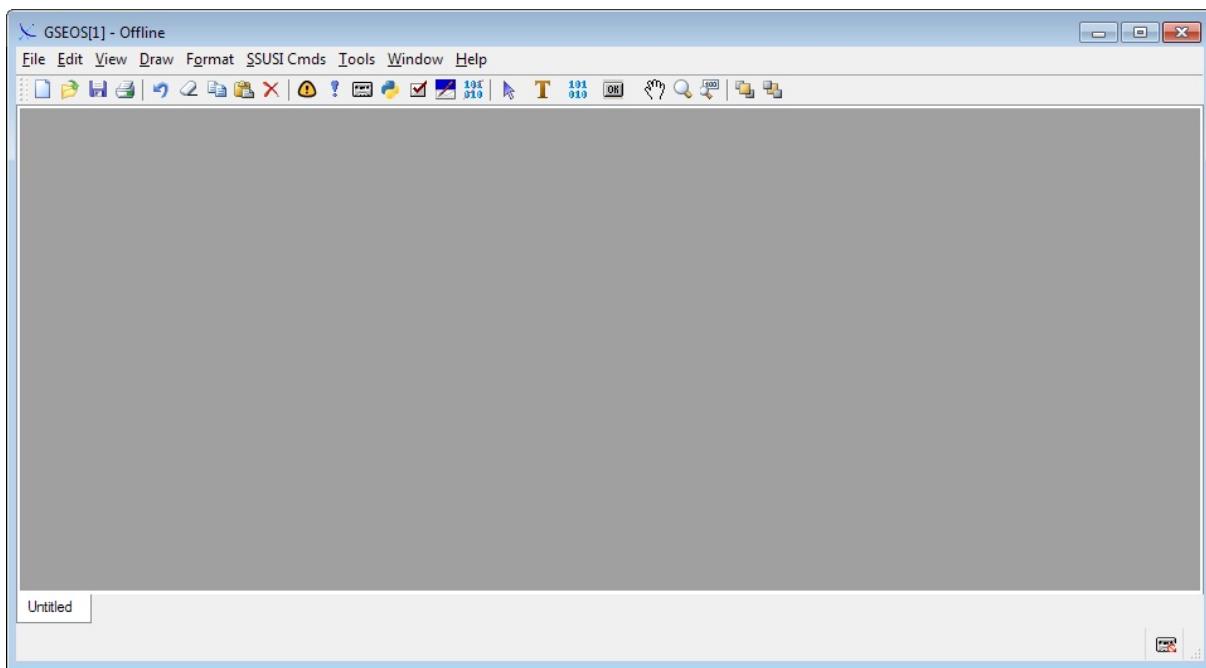
This file configures the system parameters like startup files, network settings, recorder path, project name, etc.

MyProject.blk

The block definition file(s) will hold your telemetry and other block definitions. You can split your block definitions over multiple files for easier management. The block definition files to be loaded are specified in the gseos.ini file. Changes to block definition files require a restart of GSEOS for these changes 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](#)^[150].

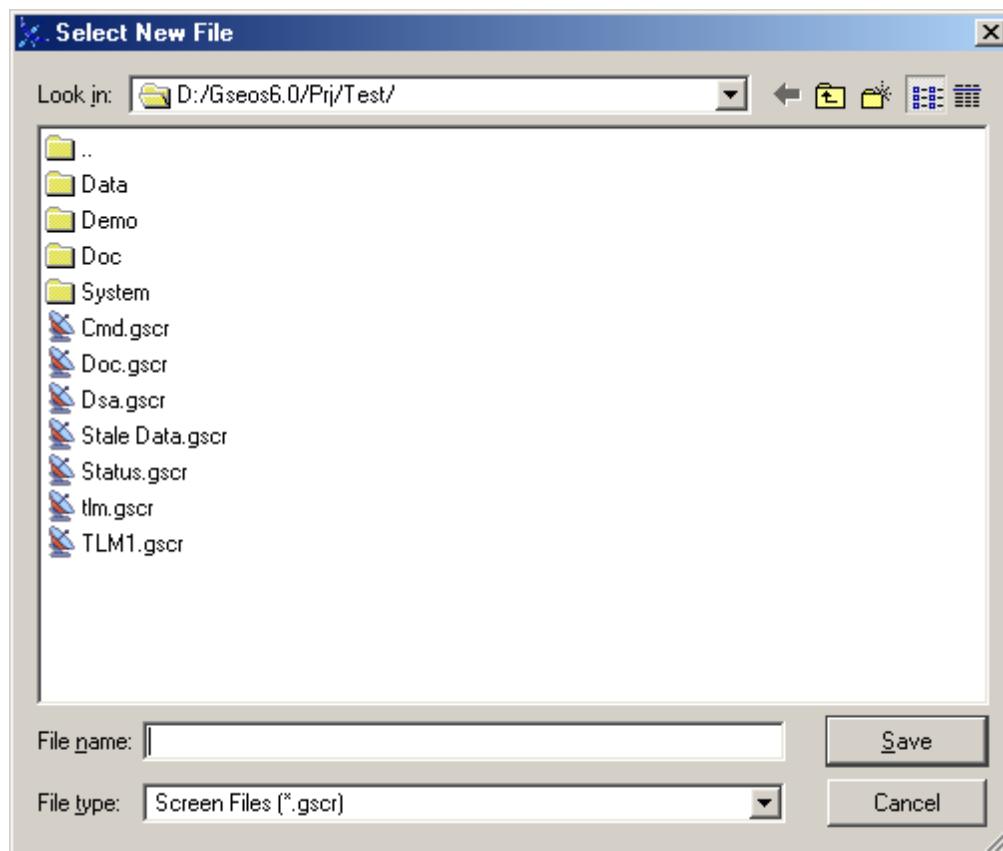
```
TLM {  
    (ApId      ,,, 16;  
     ,,, 5;  
    InstId    ,,, 4;  
    PacketId  ,,, 7;  
    Seq        ,,, 32;  
    Len        ,,, 32;  
    Data [40000] 0 ,,, 8;  
}
```

Once you configured your gseos.ini file and set up your block definition file you are ready to start GSEOS. The basic app will look like the image below.

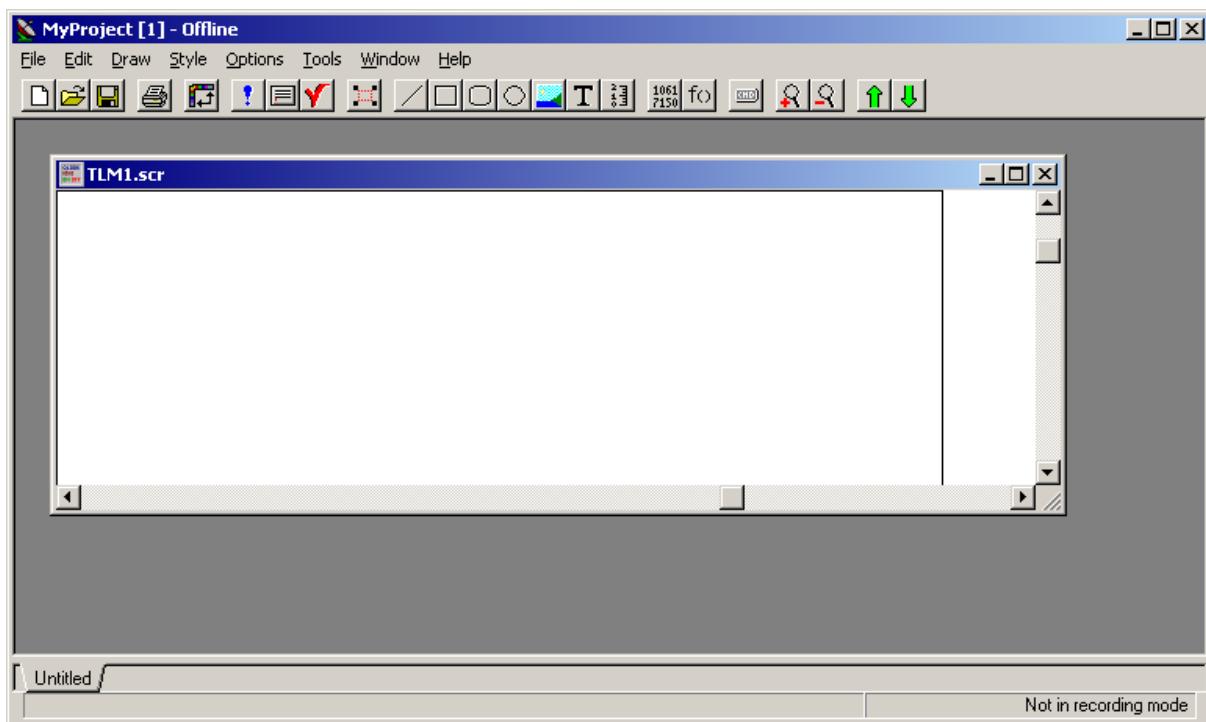


1.5.1 Add a screen window

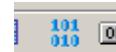
Now that the system is up and running lets display the TLM block that is defined in our block definition. To create a new screen you select File|New from the main menu or click the File| New toolbar button.



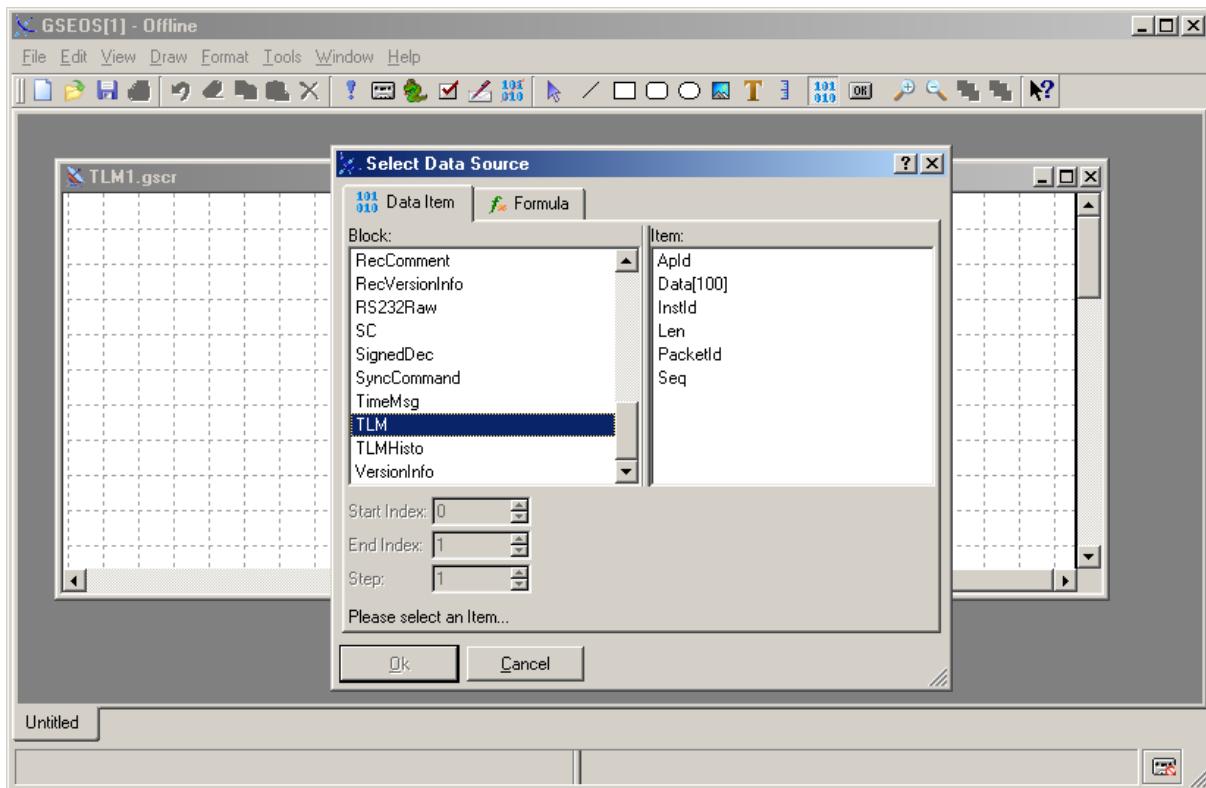
The above dialog opens and prompts you to enter a new file name. Note that in the combo box 'Save as type' you can select various different file types. In order to create a new screen file you have to select 'Screen (*.gscr)'. Once we specify the file name e.g.: TLM1.gscr an empty screen opens up in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



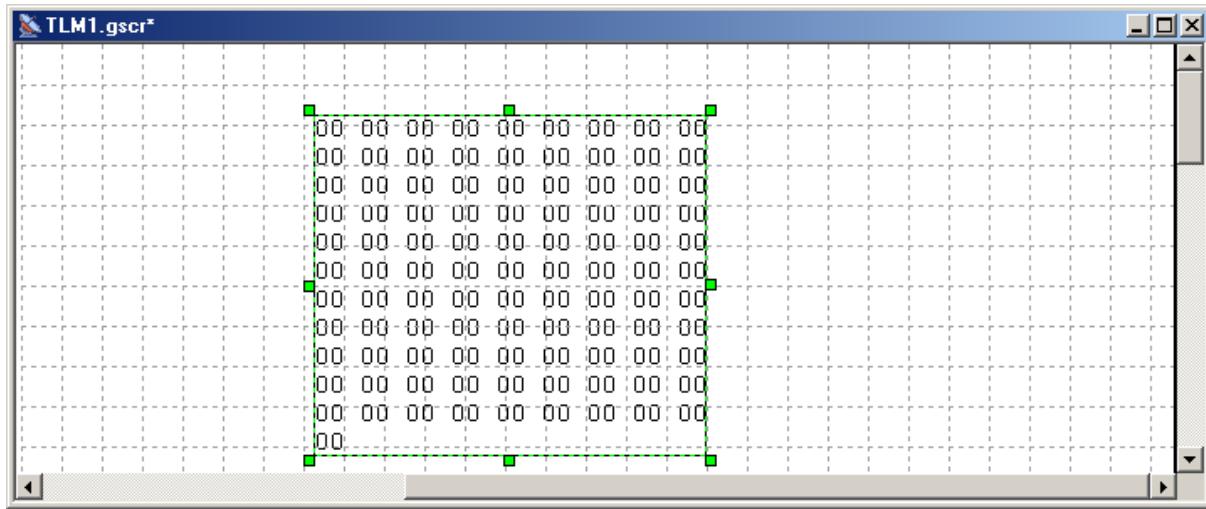
To place a data item on the screen we select the 'Data Item' tool. Hint: If you move the mouse cursor over the toolbar buttons their function is indicated in the status bar. The button shown below invokes the 'Data Item' tool (alternatively you can choose Draw|Data Item from the main menu).



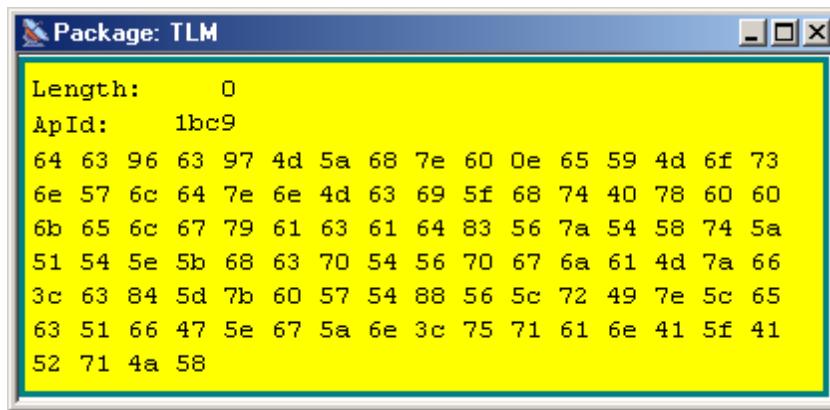
Once you select a tool the according button appears depressed to indicate the tool selected in addition the mouse cursor takes a tool specific shape. Now drag a rectangle on the screen by pressing the left mouse button and moving the mouse while holding the mouse button down. Once you release the mouse 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. By right-clicking on the item you will see the Format menu which allows you to change the format of the item. After placing multiple other items and some static elements like rectangles the TLM.scr file finally looks like this:



1.5.2 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:

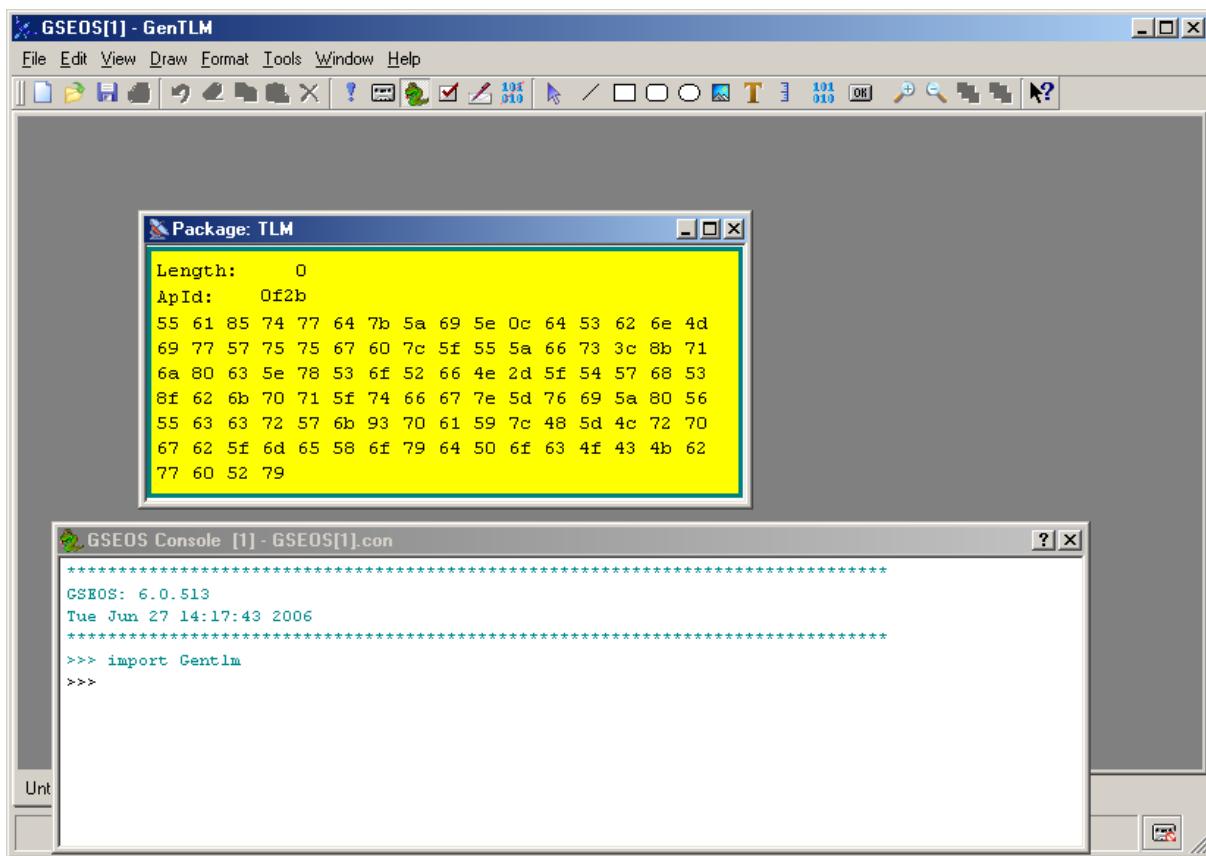
- Create the histogram block definition
- Setup the histogram decoder
- Setup the histogram display
- At first we have to define the destination block. The event values of the TLM block can be in the range [0..255]. We create a TLMHisto block that will take the histogram data and consists of one array 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 View|Console from the main menu or press F9. This will open the console window. The application will look like this:



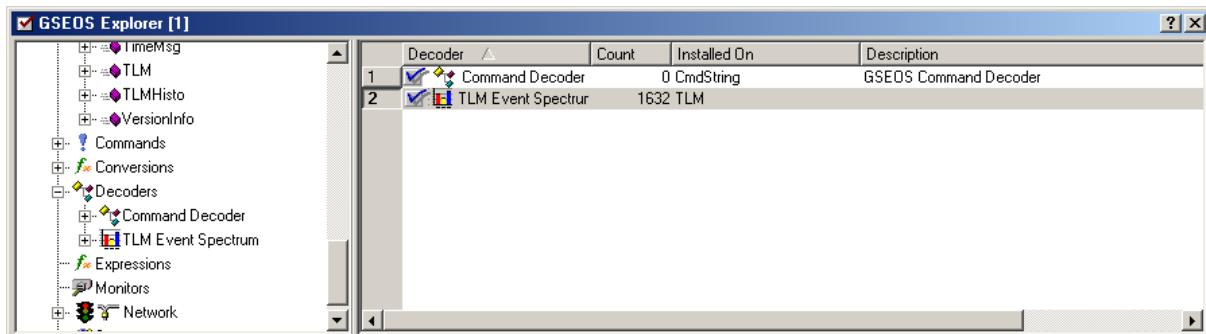
To create the histogram decoder we type into the console:

```

>>> import GeosHistogram
>>> oHisto=GeosHistogram.Thistogram('TLM Event Spectrum', 'TLM Data',
' TLMHistogram', 0, 256, 100)

```

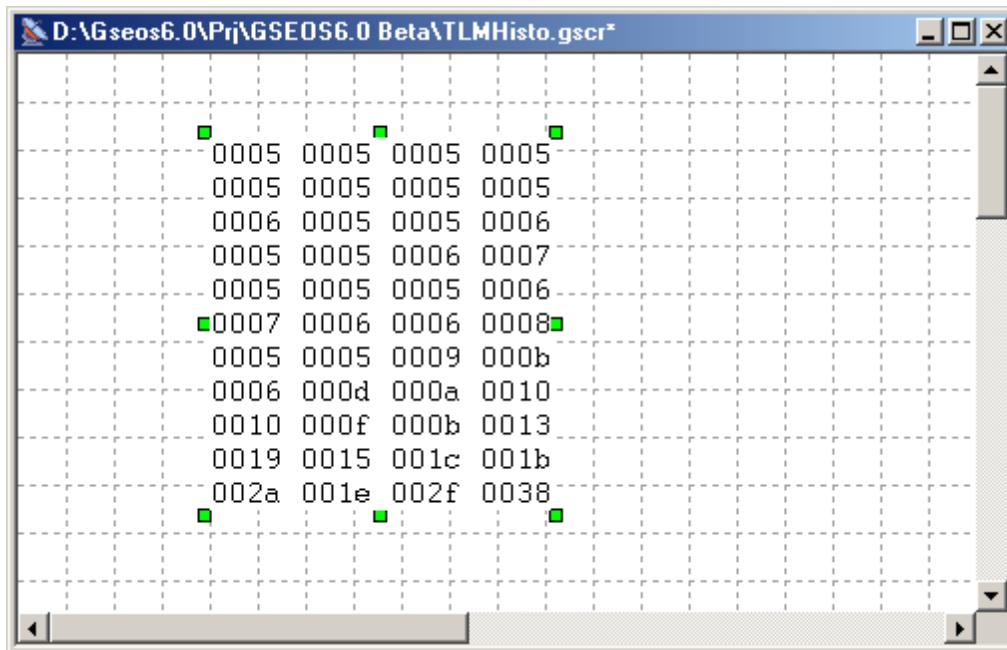
This creates the histogram decoder and starts it. We hold on to a reference with the oHisto variable. This allows us to call various functions on the histogram decoder later. Now lets check out the decoder in the GSEOS Explorer:



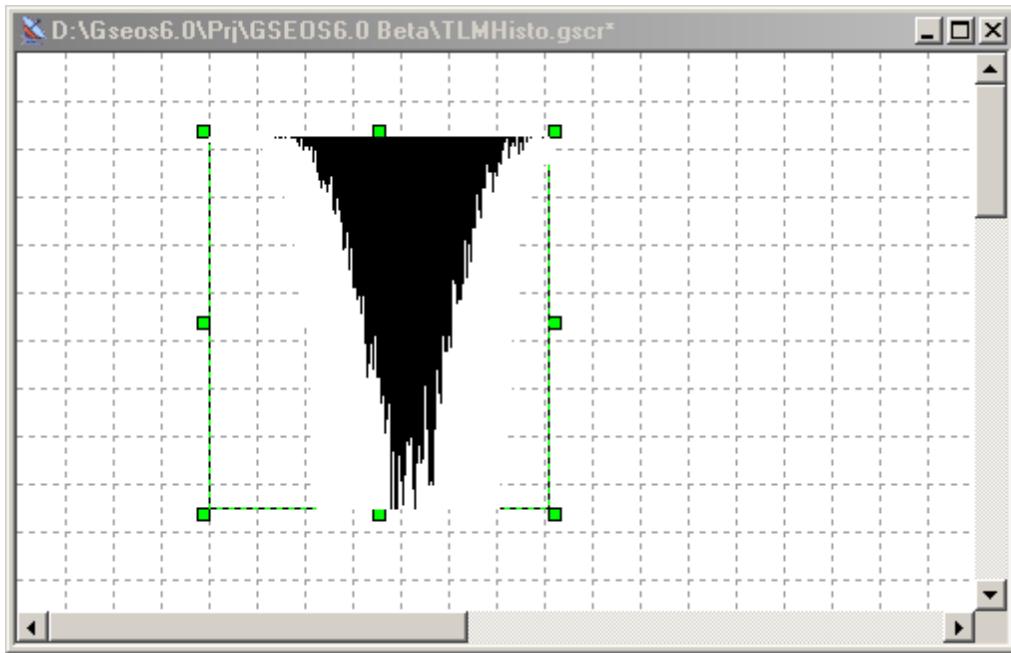
You can see the 'TLM Event Spectrum' histogram in the decoders node. The checkmark indicates that the decoder is running (as you can also tell from the count). The GSEOS Explorer is a useful tool to get a quick system overview and to manage your block

definitions, decoders, monitors, histograms, network connections, etc.

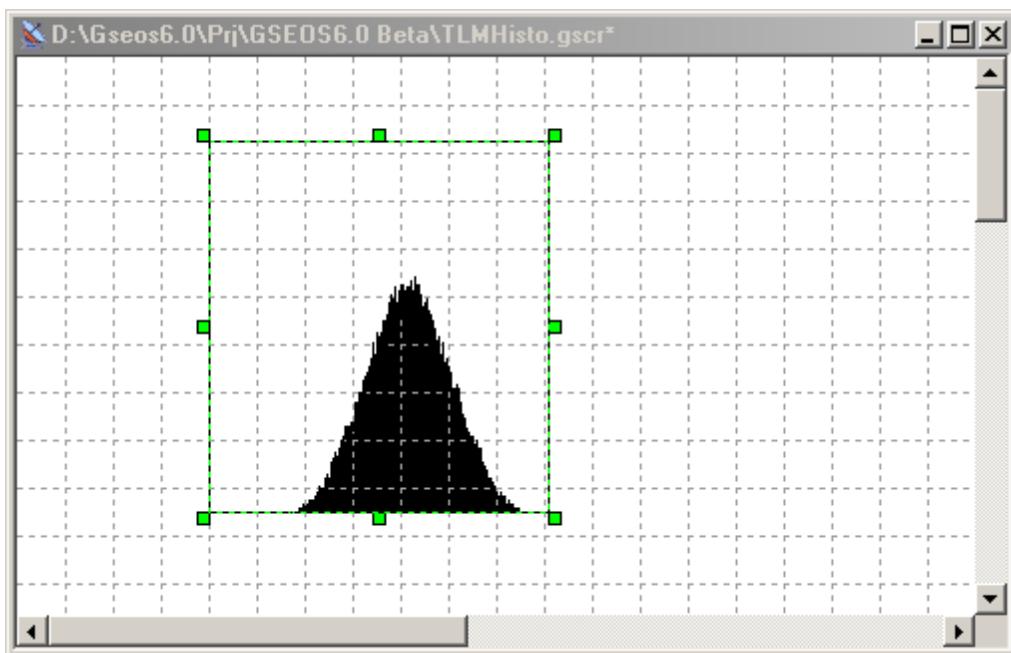
As the last step we have to create the spectrum display. We create a new screen that displays the TLMHisto.Histo item.



Lets change the display format 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 Data Item 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:



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 and logical 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 telemetry data from the connected hardware. GSEOS implements the following modules to provide this functionality:

- **[Commanding](#)**^[17].
Instrument and spacecraft commands can be sent from an easy to configure user interface. Command menus allow hierarchical command structures, shortcut command buttons can be placed on any screen.
- **[Data Monitoring](#)**^[285].
Data items can be checked for limit violations and actions like operator notification can be performed upon detection.
- **[Data Decoding](#)**^[246].
Telemetry data can be decoded according to the instrument protocol stack. This allows visibility at all protocol levels. Derived data points are 'first-class' data points and can be displayed, monitored, recorded, and also further decoded.
- **[Data Display](#)**^[38].
Data items can be displayed on the screen in numerical and graphical formats. 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 structuring of your data. The graphical editor makes it very simple to rapidly design data display screens.
- **[Data Archival](#)**^[145].
The recorder module allows you to store any telemetry data as well as commands to mass storages in real-time. Conventional hard disks and optical disks can be used as mass storage devices. The simple tape recorder like user interface lets you record and play back data instantly. Huge amounts of data can quickly be navigated by the user for custom time bases.
- **[Automatic test procedures](#)**.
GSEOS provides an easy interface to implement end-to-end testing. STOL (Spacecraft Testing and Operating Language) can be used to implement test scripts that issue commands and verify telemetry data.
- **[Networking](#)**^[128].
In payload integration or flight operation configurations data is often available through the MOC (Mission Operation Control) via a TCP/IP connection. GSEOS makes

it easy to tap into these data streams and reuse your existing displays and scripts throughout the lifecycle of the mission. It also allows you to connect multiple GSEOS stations together in arbitrary configurations and to distribute the data onto different workstations or to remote control your instrument.

- **Scripting.**

GSEOS uses the [Python](#) scripting language for system customization to your needs. Python is easy to learn and has a clear syntax that makes it easy to maintain the code.

2.1 Publish Subscriber Data Bus

Data Blocks

Within the DPU data items are assembled into Experiment Data Blocks (EDBs) according to experiment specific [block definitions](#)^[150]. The block definitions allow access to your data items from scripts and data display screens.

The backbone of GSEOS is a publish-subscribe data bus that moves these data blocks between the modules of the system. Tasks can generate as well as consume data. A task has to register its interest in a particular block with GSEOS. Every time this block arrives in the system it will be forwarded to this task. Each task can register for any number of blocks. The BIOS module is responsible of generating data blocks from instrument telemetry streams as well as outputting command data to the instrument.

2.2 Real-time control

After informing a task about the arrival of data, GSEOS accesses the whole data block or single data items which are defined in the block definition. The [Decoder module](#)^[246] also allows you to create new data blocks. At any given time the system uses one of the mutually exclusive data sources available: BIOS, Recorder, Net, or a custom data source. I.e. when data is played back from disk the [Recorder](#)^[145] is the data source and all data coming in from other data sources is discarded.

Data processing within GSEOS is inherently data driven. A configurable queue acts as a [buffer](#)^[178] between data sources and consumers.

3 Binary Streamers

Binary Streamers allow to write binary data directly to file. Like [logging](#)^[18] you can set up automatic rollover for binary streamer files. Binary streamers always dump the entire block. If the first item of the block is called 'Len' and is 32-bit wide you can enable 'variable length' blocks. That is the value of the Len item is interpreted as the number of valid bytes in the block following the Len field (not including the 4-byte Len field). In this case only the number of bytes as indicated by the Len field are written to disk. This allows for simple writing of variable length blocks. Unlike the [Recorder](#)^[145] binary streamers don't add any metadata to the file, only your binary data is streamed as is.

[Binary Streamer File Rollover](#)

For long running tests and for large amounts of data it is often desirable to use automatic rollover files as opposed to a single file. Automatic path and file rollover lets you configure roll-over conditions that automatically change the binary streamer file and/or directory based on certain conditions.

The effective streamer file is determined by two components: The path name and the file name. These components are configured with the PathTemplate and FileTemplate settings respectively. Both of these allow the use of date/time dependent formatting characters. When the path or file rolls over a new path or file name is determined based on the templates and the current date/time.

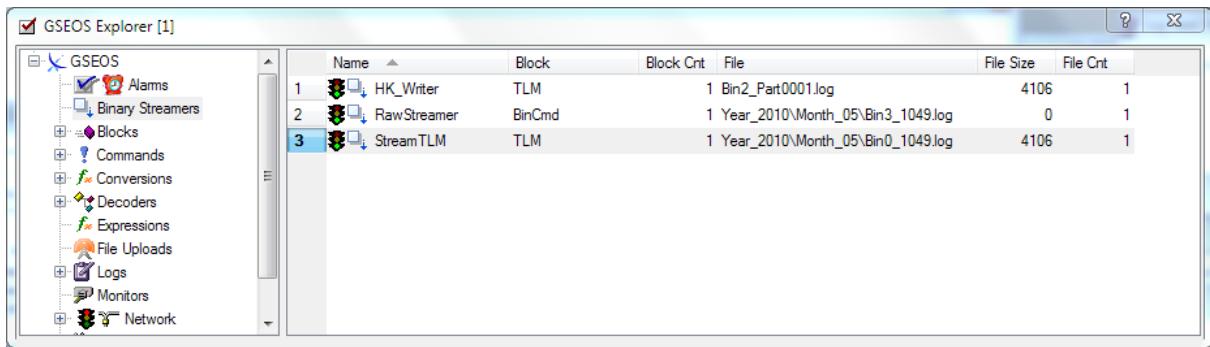
The roll-over can be triggered by your path or file template yielding a new name given the current date and time, a new session (start of the binary streamer) or reaching of a configured directory (number of files) or file (size of file) limit.

Typically a new directory or file will be created whenever the PathTemplate or FileTemplate configuration yields a new name. The PathRollOverSession and FileRollOverSession settings control how roll-over is handled when a new session is started. By default both PathRollOverSession and FileRollOverSession are "Disabled". In this setting we don't create a new directory or file unless the regular date/time based rollover mechanism yields a new directory or file. If this setting is Enabled either a new directory or a new file will be created regardless if the directory or file already exists (the path or file will be rolled over on session startup). If the template yields an existing directory or file the following text will be appended to the new directory or file: _PartNNNN, where NNNN is a counting number starting with _Part0000. If a directory or file already ends in a pattern that matches _PartNNNN we simply increment the counter. If the PathRollOverSession or FileRollOverSession is set to "Only" a new directory or file will not be created when the template would yield a new name. Instead only on session startup a new directory or file is created with the then current data and time information as configured with the template.

A roll-over can also be generated when a maximum number of files per directory is set with the PathRollOverMaxFiles settings. If such a limit is configured a new directory will be created when the configured number of files is reached. Similarly the FileRollOverMaxSize setting allows you to limit the log file size and roll-over to a new log file when this threshold is reached.

GSEOS Explorer

Binary streamers are configured in the [gseos.ini](#) file with the [\[BinaryStreamers\]](#) section. You can also use the [GSEOS Explorer](#) to add new binary streamers or change the properties of existing binary streamers. The changes are written to the gseos.ini file automatically. You can start/stop the binary streamers by right-clicking on the according streamer(s).



GseosBinaryStreamer module

To programmatically access the binary streamers you can use the [GseosBinaryStreamer module](#). It exports three functions you can use with binary streamers:

[GseosBinaryStreamer.ListBinaryStreamers](#)
[GseosBinaryStreamer.StartBinaryStreamer](#)
[GseosBinaryStreamer.StopBinaryStreamer](#)

4 Commanding

This chapter outlines the general command flow and configuration in GSEOS.

Defining your commands

You start out defining your commands by creating [command definition file](#)(¹⁵⁶)s. Once you have created the command definitions you have to load them into GSEOS. You can do this manually by opening your file with the [File/Open](#)(²⁰) menu. However, if you plan on automatically loading your command definition on system startup you can enter the file in the gseos.ini configuration file [\[Config\]](#)(¹⁷⁹) section.

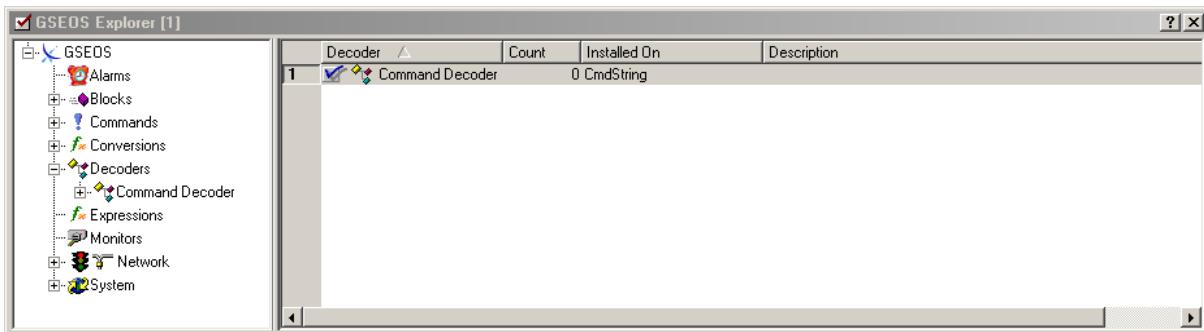
Accessing your commands

Once your commands are defined you can inspect them using the [GSEOS Explorer](#)(¹²⁴). To easily access your commands you can add them to a [command menu](#)(¹⁶¹) for hierarchical access or place the command on a [command button](#)(⁵⁵) with inclusion on a screen. To issue your commands you will use the [GseosCmd.ExecCmd\(\)](#)(²³⁸) function. This function takes your command as a string and queries for any missing arguments, compiles the command, and generates the **CmdString** block. If this is a critical command it will also pop up a dialog box asking the user to acknowledge the command before sending it. The CmdString block will be intercepted by the Command Decoder which will generate a **BinCmd** block containing the proper binary command data. This block in turn is processed by the BIOS which issues the binary command as required by your instrument.

Below is an example from a command menu file:

```
MenuItem Set &Time,           GseosCmd.ExecCmd ("EGSE_SET_TIME () ")
MenuItem Set &Subsecond Time,
GseosCmd.ExecCmd ("EGSE_SET_SUBSECOND_TIME () ")
```

If you disable the command decoder (shown below) commanding will effectively be disabled (unless you generate a BinCmd block by some other means like playing it back from the recorder).



Command logging

You can log both the command string and the binary command to a log file and/or the message window. The command logging is configured in the [gseos.ini](#)^[17] file with the section [\[Command\]](#)^[178]. You can use any log as described in the [chapter on logging](#)^[18].

The following example logs commands to the CmdLog.log file and also to the message window.

```
[Command]
Log=CmdLog.log
LogToMsg = Yes
```

5 Logging

GSEOS implements two logging mechanisms: Simple file based logging allows you to create log files on the fly. Automatic logs are configured in gseos.ini and facilitate automatic file and directory roll-over based on your configuration. Both logging types commit any Log() operation to disk immediately. Independent of the actual log file you can also open a log window that displays the current log content. The window behavior of single log windows is different from the automatic logs, see the chapter on [Log Windows](#)^[120] for more details.

Each log has a name. For simple file based logs this is the file name (if your log resides within the GSEOS distribution use relative file names), for automatic logs this is the log name as assigned in the [gseos.ini \[Logs\]](#)^[183] configuration. When writing to a log you use the [GseosLog.Log\(\)](#)^[280] function which takes the log name as the first argument.

Single File Logs

You can create single file logs by simply writing to a log file. If the file exists the log text is appended to the file. If the log doesn't exist the file is created. The file must end in .log or .html. If you use the .html extension the logs will be written in HTML format, otherwise plain text is used.

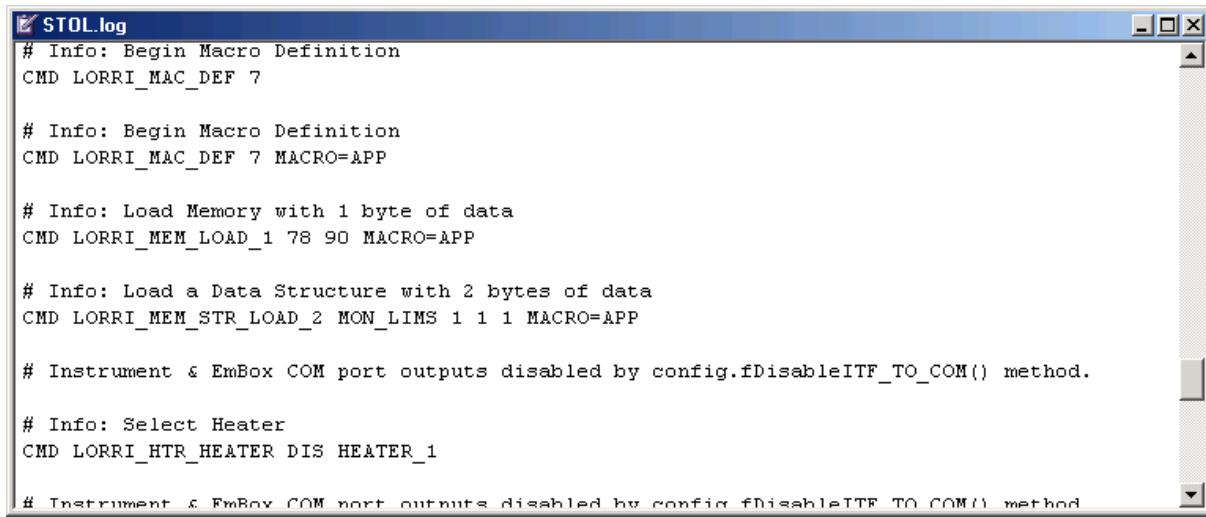
To open a log window you can choose the log file from the main menu: File/Open and select the file type: Log Files (*.log; *.html). This will open a new window displaying the log file contents. You can open multiple log windows for a single log file.

You can write content to a log file without a corresponding log window to be open. Log windows are regular MDI child windows like [screen windows](#)^[38]. They are located on a particular [desktop page](#)^[26] and their layout will be saved with the desktop settings. The caption bar starts with "Log: " and the log file the log window is associated with. This is important if you want to access the window with one of the Gseos.WindowXXX() functions.

You can specify font and color information for a log window. This information is saved with the [desktop file](#)^[26] and not in the log file itself. So if you close the log window and open it again the font and color information will be lost. However, it will be preserved in between GSEOS restarts since the desktop will apply the settings to the file as configured.

You can set color and attribute information when inserting text into a log file with the [GseosLog.Log\(\)](#)^[280] function. However, this format information is not saved and on reopening the file it will be lost. The reason for this is that we save the file in plain ASCII without any formatting information. So while you can mark up your logs while the system is running the output file will be a plain text file.

If you use an .html log file the color and font style settings will be preserved as the log is written as HTML code.



```
# Info: Begin Macro Definition
CMD LORRI_MAC_DEF 7

# Info: Begin Macro Definition
CMD LORRI_MAC_DEF 7 MACRO=APP

# Info: Load Memory with 1 byte of data
CMD LORRI_MEM_LOAD_1 78 90 MACRO=APP

# Info: Load a Data Structure with 2 bytes of data
CMD LORRI_MEM_STR_LOAD_2 MON_LIMS 1 1 1 MACRO=APP

# Info: Select Heater
CMD LORRI_HTR_HEATER DIS HEATER_1

# Instrument & EmBox COM port outputs disabled by config.fDisableITF_TO_COM() method.

# Info: Select Heater
CMD LORRI_HTR_HEATER DIS HEATER_1

# Instrument & EmBox COM port outputs disabled by config.fDisableITF_TO_COM() method.
```

Automatic Logs

For long running tests and for large amounts of log data it is often desirable to use automatic log files as opposed to simple single file logs. Automatic logs let you configure roll-over conditions that automatically change the log file and/or directory based on certain conditions. All automatic logs must be configured in the [\[Logs\] section of the gseos.ini](#)^[183] configuration file.

The effective log file is determined by two components: The path name and the file name. These components are configured with the PathTemplate and FileTemplate settings respectively. Both of these allow the use of date/time dependent formatting characters. When the path or file rolls over a new path or file name is determined based on the templates and the current date/time.

The roll-over can be triggered by your path or file template yielding a new name given the current date and time, a new session (start of GSEOS) or reaching of a configured directory (number of files) or file (size of file) limit.

Typically a new directory or file will be created whenever the PathTemplate or FileTemplate configuration yields a new name. The PathRollOverSession and FileRollOverSession settings control how roll-over is handled when a new session is started. By default both PathRollOverSession and FileRollOverSession are "Disabled". In this setting we don't create a new directory or file unless the regular date/time based rollover mechanism yields a new directory or file. If this setting is Enabled either a new directory or a new file will be created regardless if the directory or file already exists (the path or file will be rolled over on session startup). If the template yields an existing directory or file the following text will be appended to the new directory or file: _PartNNNN, where NNNN is a counting number starting with _Part0000. If a directory or file already ends in a pattern that matches _PartNNNN we simply increment the counter.

If the PathRollOverSession or FileRollOverSession is set to "Only" a new directory or file will not be created when the template would yield a new name. Instead only on session startup a new directory or file is created with the then current data and time information as configured with the template.

A roll-over can also be generated when a maximum number of files per directory is set with the PathRollOverMaxFiles settings. If such a limit is configured a new directory will be created when the configured number of files is reached. Similarly the FileRollOverMaxSize setting allows you to limit the log file size and roll-over to a new log file when this threshold is reached.

See below for a few examples:

```
#*
*****
*#
## * Use a fixed directory and roll over the log file each day.
* *#
#*
*****
*#
[Logs]
ErrorLog=Log

[ErrorLog]
PathTemplate = Logs/ErrorLogs
FileTemplate = ErrorLog_%Y%m%d

#*
*****
*#
## * Create directories for each year, each month within the year and
* *#
## * create a daily log file within the month directory.
* *#
## * Every time we start a new session we start with a new log file.
* *#
#*
```

```
*****
*#
[Logs]
ErrorLog=Log

[ErrorLog]
PathTemplate      = Logs/ErrorLogs/%Y/%B
FileTemplate      = Log_%d
FileRollOverSession = Enabled

#*
*****
*#
## * We create one log file for each session. The log file has the
* *#
## * start date and time of the session.
* *#
#*
*****
*#
[Logs]
ErrorLog=Log

[ErrorLog]
PathTemplate      = ErrorLogs
FileTemplate      = Log_%Y%m%d%H%M%S
FileRollOverSession = Only
```

Writing to a log file is similar to single file logs, just that instead of using a file name you use the log name which in the above case is: ErrorLog. Otherwise the arguments and behavior for inserting log text into automatic logs is the same as for single file logs.

You can write content to a log file without a corresponding log window to be open. As opposed to single file log windows there is exactly one window per automatic log. As the file or directory changes the title changes and also the contents of the log since it displays the contents of the new log file. You can access the automatic log windows from the View/Logs menu. Since the file associated with automatic logs changes you can't open the log window using the File/Open menu (although it will open a log window it will not adjust to a new path or file as the auto log window does).

Python scripting

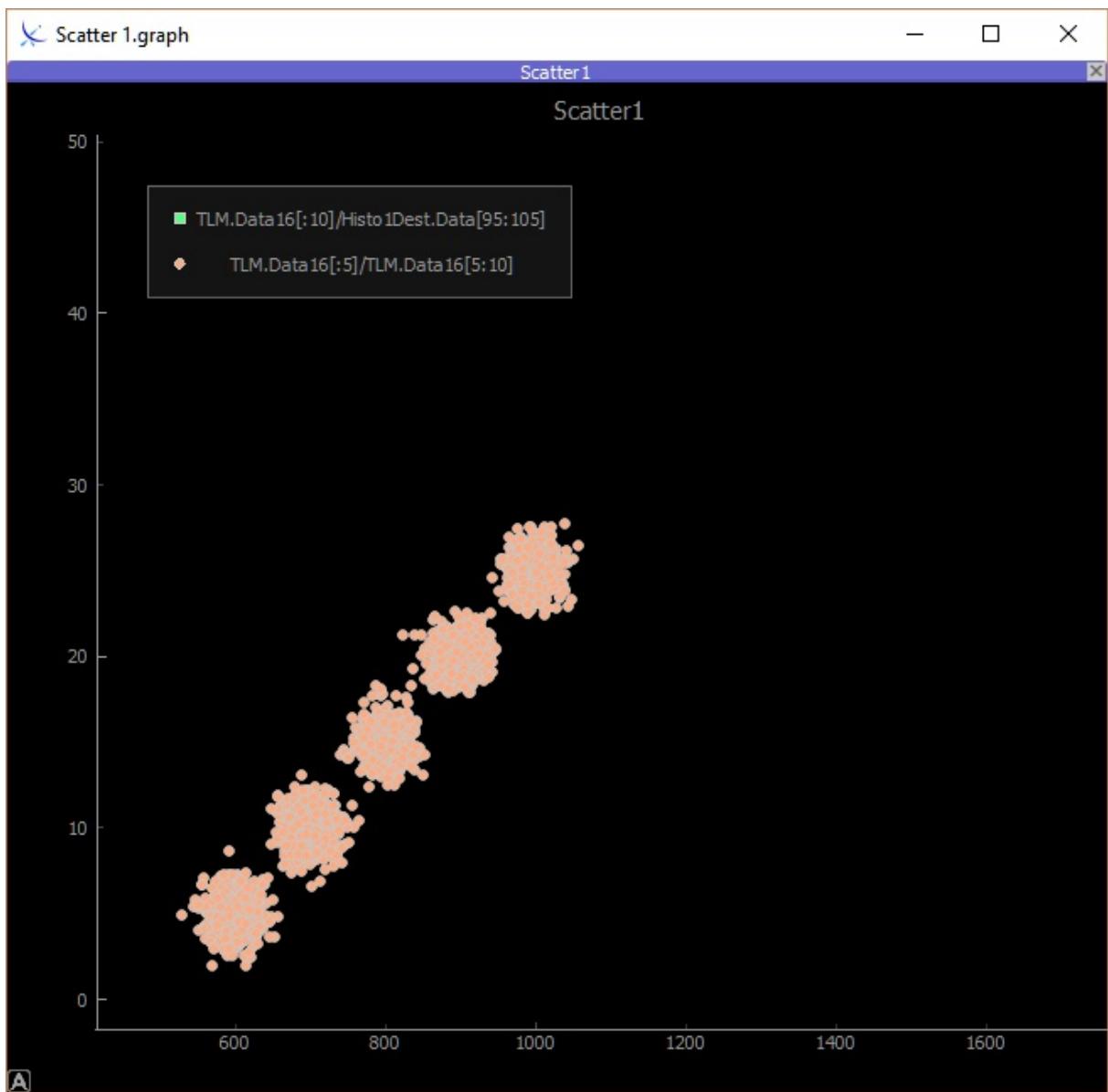
You can use the [GseosLog](#) module for programmatic access to the logging functions.

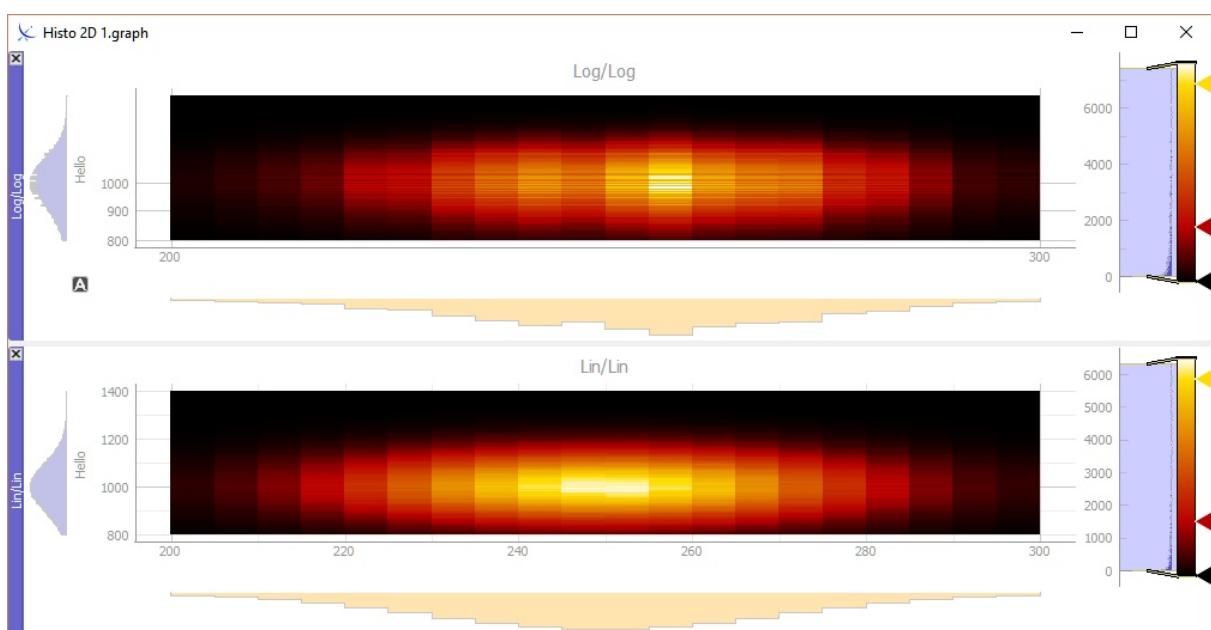
6 Graphs and Charting

While you can define stripcharts on regular [GSEOS screens](#) there is a specific GSEOS Graph module that allows you to quickly create real-time plots of your data. The Graph

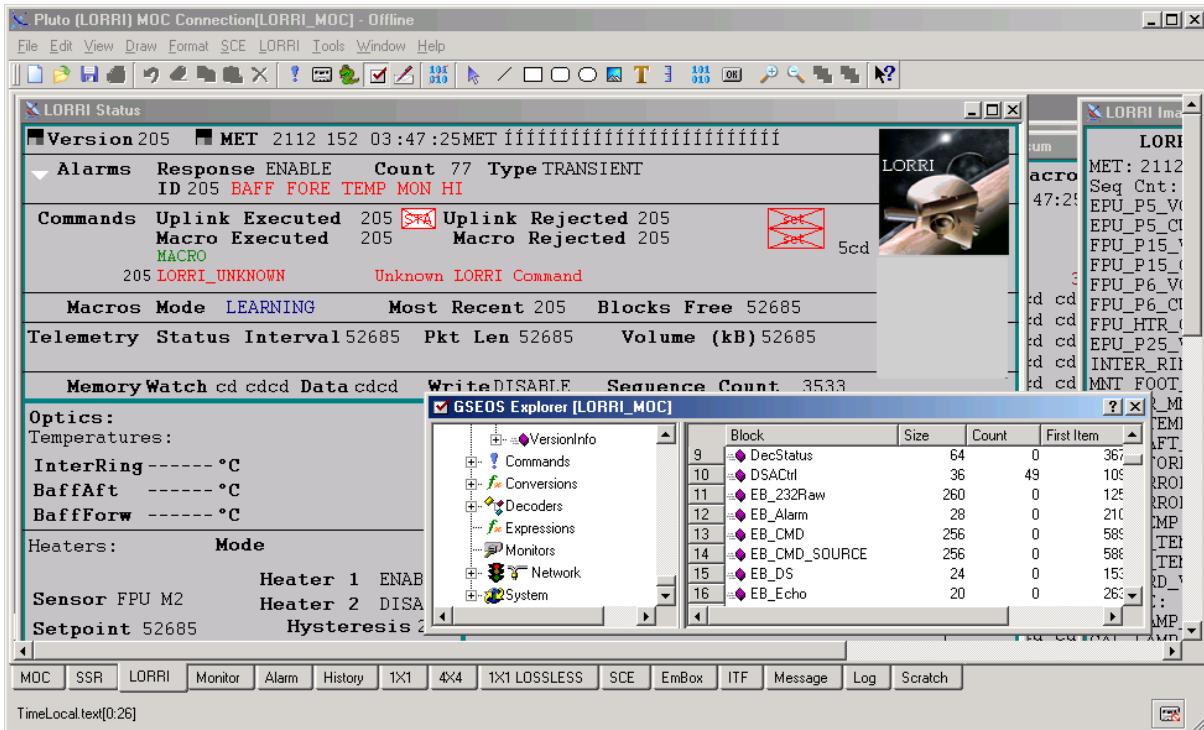
windows also allow you to interact with the plots and graphs you create and zoom and pan in the graph and quickly adjust the display settings. The following images shows a few sample plot windows. The section on [Graph Windows](#)⁸⁶ gives more details on setup and configuration.





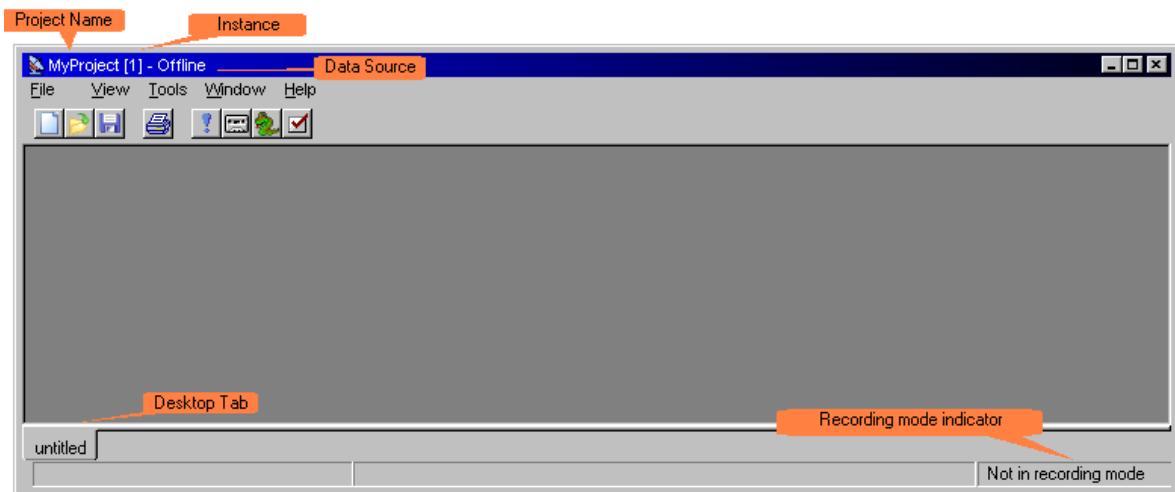


7 User Interface



7.1 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. The picture below 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](#)^[191].
- [1]: The current instance number/name. Multiple instances of GSEOS can be run concurrently. You can specify the instance number/name on the [command line](#)^[147], the \I switch will specify the instance number/name. This allows you to [configure different application settings](#)^[182] 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](#)^[235].

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.

7.2 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 (*.dt). On shutdown of GSEOS the system saves the current desktop into the file AutoDeskxx.dt where xx refers to the [instance](#)^[182] 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...](#)^[30] menu, or restore a previously saved one.

By default the loaded desktop is saved on system exit. You can override this behavior with the following setting in the gseos.ini [\[System\]](#)^[195] section:

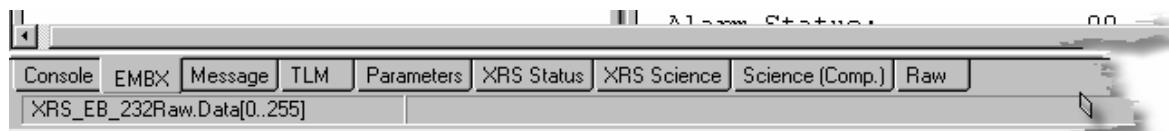
```
[System]
DesktopAutoSave=No
```

If no desktop has been loaded the AutoDsk*.dt automatic desktop file will always be saved, independent of the above setting.

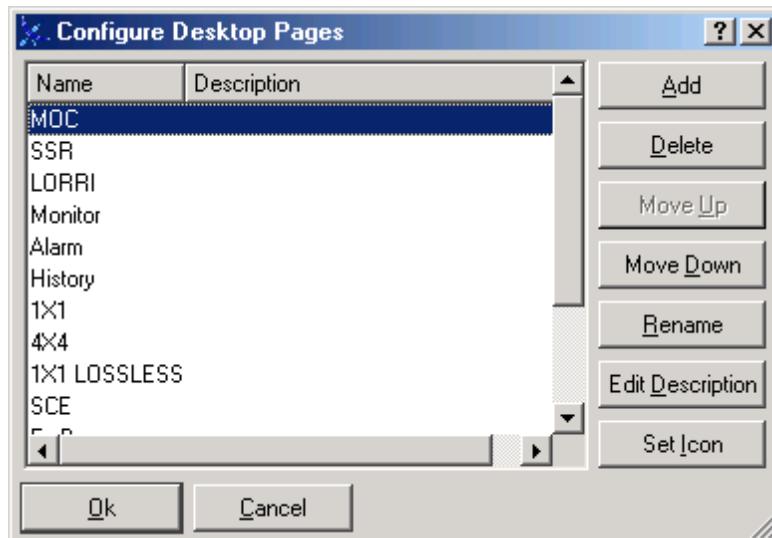
You can override this behavior by loading a desktop file in the [gseos.ini Config](#)^[179]/Load section. This will cause GSEOS to always appear as determined by the desktop file specified regardless in what state the system was terminated previously.

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 configure the desktop pages from the [File/Configure Tabs...](#)^[28] menu or by right-clicking on the desktop tab bar. This will open a dialog that allows you to edit the destkop pages.



To programmatically set the active page use the following command:

```
>>> Gseos.SetActiveDesktopPage(str PageName)
>>>
```

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

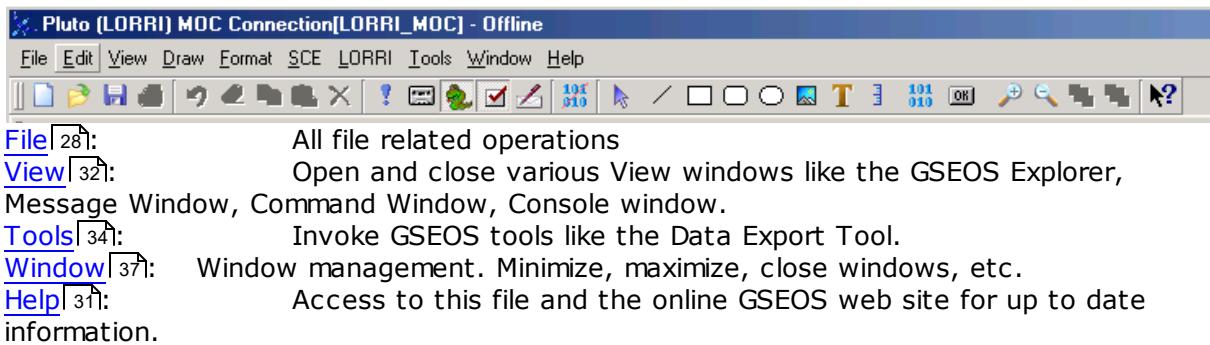
System Windows

Besides regular screen files and log files there are also some special windows: [Message Window](#)^[140], [GSEOS Explorer](#)^[121], [Console Window](#)^[135]. These windows can have on of three states. They can either be hidden, floating, or pinned to a desktop page. The tool bar button or the main menu toggles between the floating state. If the window is initially hidden it will be displayed as a floating window. The advantage of the floating window is that you have it in view at any time, independent of which desktop page you are currently on. However, it also means that this window takes up valuable desktop space. If it is pinned down you will only see it on the according desktop page and it is therefore not in the way while reviewing other desktop pages. If you want to move one of these windows to a different desktop page you have to perform the following steps:

- Set the window into the pinned state.
- Close the window using the close button in the upper right corner of the window.
- Navigate to the desktop page you want to place the window on.
- Display the window in the floating state using the tool bar button or the according menu.
- Toggle the window state to pinned using the tool bar button or the menu again.

7.3 Menus

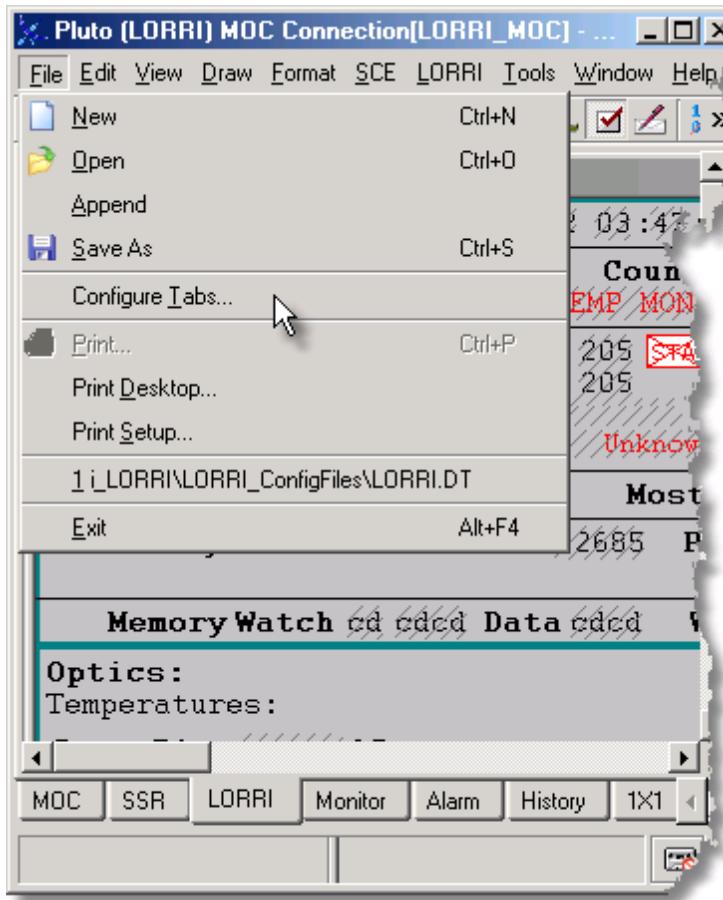
The GSEOS main menu lets you quickly access the most common functions:



Depending on what kind of child window is active in GSEOS specific menus will be enabled/disabled in the submenus. In addition to the standard menus you can define your own custom menus that will be placed before the Tools menu. In this case the SCE, and LORRI menus have been added.

7.3.1 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. You can create new files of type *.gscr (GSEOS Screen files), *.log (Log Files), and *.lst (Block Listing Files).

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 [\[System\]](#)¹³¹ 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. You can open files of the following types:

Screen Files (*.gscr)

- Screen Files (5.x) (*.scr)
- Desktop (*.dt)
- Log Files (*.log)
- Python Modules (*.py; *.pyd; *.pyc)
- Configuration Files (*.qlf; *.tr; *.alarm; *.cfg)
- Command Definition Files (*.cpd)
- Command Batch Files (*.cpb)
- Command Menu Files (*.cm)

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...](#)¹²⁸ menu.

Besides the GSEOS 6.x screen files of type *.gscr you can also load GSEOS screen files of versions 3.x, 4.x, and 5.x. Once you make modifications to a *.scr file it will be saved in *.gscr format and can not be read by the older versions of GSEOS any longer.

A Desktop file loads an entire screen configuration. This allows you to save the window and desktop page layout in a single file. You can also combine different desktop files by using the [File/Append...](#)¹²⁹ menu.

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 [Message Window](#)¹⁴⁰ and also pop up a message box with the according error message.

There are different kind of command files, [Command Definition Files \(*.cpd\)](#)¹⁵⁶ define new commands, [Command Batch Files \(*.cpb\)](#)¹⁵⁴ allow the execution of time tagged command batches, and [Command Menu Files \(*.cm\)](#)¹⁶¹ add command menus to the GSEOS main menu for easy access of your commands.

Other configuration files like Formula Files (*.qlf), Text Reference Files (*.tr), and Alarm Limit Files (*.alarm) are grouped together in the Configuration Files type.

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...^{\[30\]}](#) 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.

Configure Tabs...

In order to add additional desktop pages to the desktop or to change the order of desktop pages use this menu. Please refer to the [Desktop section^{\[26\]}](#) for more information on desktop management.

Print...

The print menu opens the print dialog that allows you to print the active window.

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 system 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.

Customizing the File Type list

You can also add your custom file extensions to the file type list. To find out how to register your own custom file types in the GSEOS file management check the [Gseos module^{\[202\]}](#) for more information.

7.3.2 The Edit Menu

The edit menu gives you access to the common edit functions Cut, Copy, Paste, Delete. Depending on the window type active (screen window, log window, etc.) several other menu entries might be enabled.



Undo

If the active file has been changed and an undo operation is available the Undo menu is enabled. You can step back through your changes until you get to the last point the file was saved or loaded. Changes to a screen file are indicated with an asterisk after the window title.

Select All

For screen files this selects all items on the screen. This might be useful to identify all items since sometimes items can be "hidden". For example a bitmap display that is all white on a white background.

Find, Find Next, Replace

These options are enabled for log files and the console window only.

To Top, To Bottom

These options are enabled for screen files only and move the selected item(s) to the top of the z-order stack or to the bottom respectively.

Spread Horizontal, Spread Vertical

The Spread Horizontal and Spread Vertical menus let you distribute the selected items evenly within the selection range (for the horizontal spread the selected items are distributed over the width of the selection, for the vertical spread the selected items are evenly distributed over the selection height).

7.3.3 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 license information.



The GSEOS version number is the major and minor version, in this case 7. The last number represents the release in this case 511.

The next line indicates the current Python version number.

7.3.4 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.

The Align..., and Grid... settings are only available when a display screen is active.



Command...

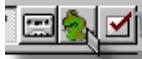
The Command menu entry opens/closes the modeless [Command dialog](#)¹³³. You can also use the F7 hotkey or the  toolbar button to open or close the Command dialog.

Recorder...

The Command menu entry opens/closes the modeless [Recorder dialog](#)¹⁴⁵. 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](#)¹³⁵. 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](#)¹²¹. 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.

Mark Stale

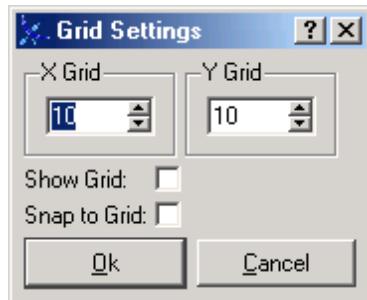
Marks all data items as stale.

Align...

Controls the alignment of screen items.

Grid...

Enables/disables the grid and sets grid properties.



Zoom

The zoom menu lets you control the zoom factor of the active window.

Options...

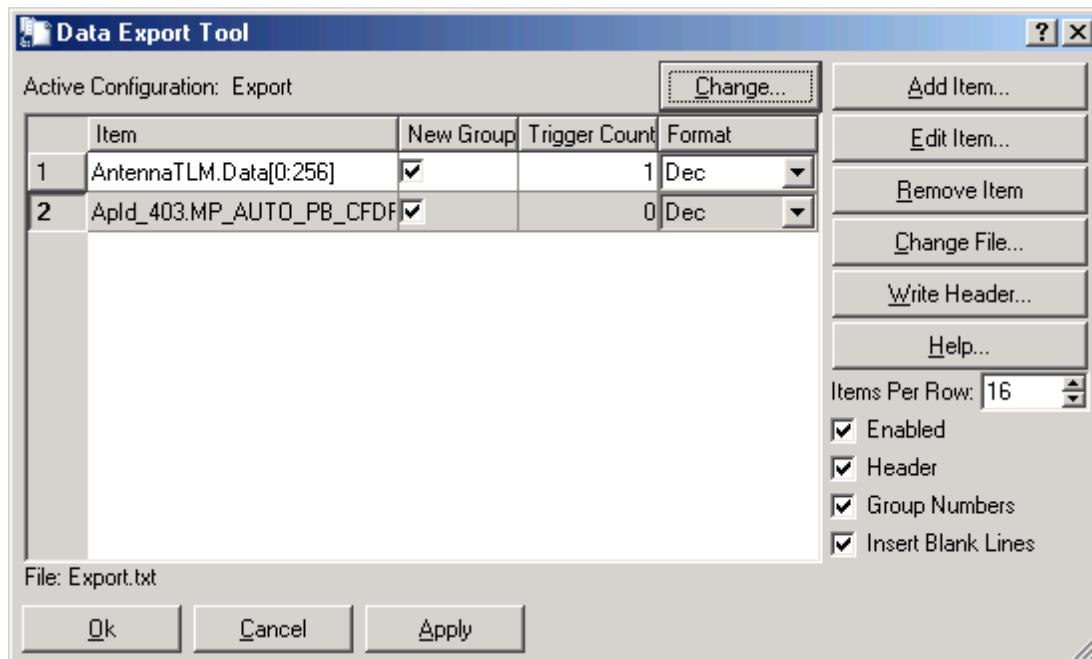
The Options menu entry opens the Options dialog which lets you edit general GSEOS system settings. For more information check in the chapter about the [Options dialog](#) [147].

7.3.5 The Tools Menu

The Tools menu allows you to access various GSEOS tools like the [Data Export Tool](#) [34] or configuration wizards. Also any plug-ins you have loaded may add an entry to the Tools menu.

7.3.5.1 The Data Export Tool

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.abyData[0:511]
```

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

The header displays the output line number assignment. If the 'Line Numbers' check box is checked the data it 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.

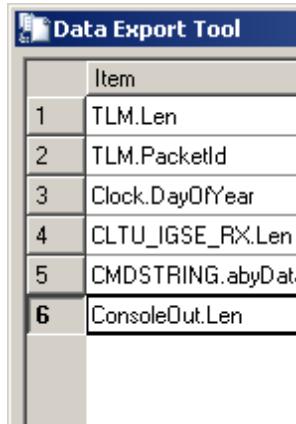
Format

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.



7.3.6 The Window Menu

The Window menu allows you to modify the display status of the child windows on the [desktop](#) page. 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:



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

Lays out the windows in a tiled 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.

Cascade

Lays out the windows in a cascaded 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

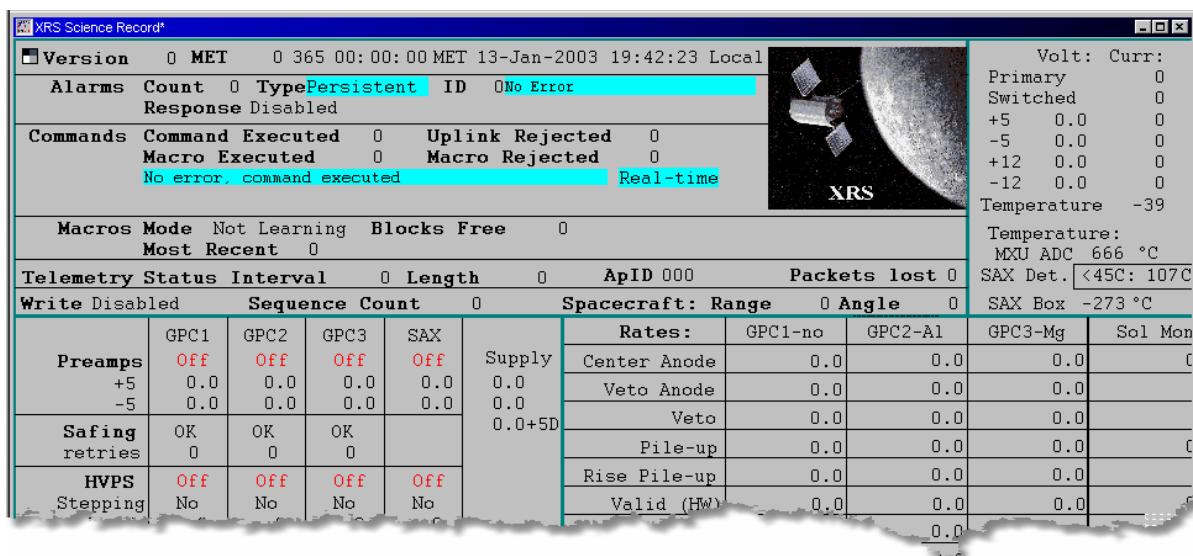
Restores all windows on the current desktop page to their normal size. The windows will take on their normal positions and size as configured.

Close All

Closes all windows on the current desktop page.

7.4 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 .gscr file. The following picture shows a screen window:



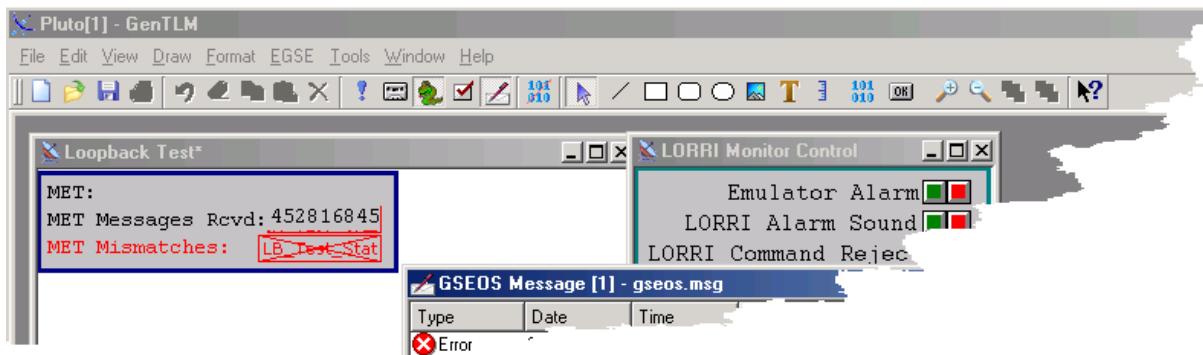
Screen windows are regular MDI child windows and are placed on a particular [desktop page](#)²⁶. You can arrange multiple screen windows on a page to group related data. When in edit mode the following menus will be activated:

- [Edit](#)³⁰: Copy and paste objects, change their Z-order, and zoom.
- [Draw](#)⁴⁸: Place various objects on the screen window.
- [Format](#)⁵⁹: Modify the display style of an object a selection of objects.

The [next chapter](#)³⁹ explains how to place graphical objects on the screen and modify the properties of display objects. The [Menus chapter](#)⁴⁷ details the various menu options. If you want to get a quick overview of how to create a screen file please refer to the [Quick Tour](#)⁶ tutorial

7.4.1 Placing objects

When a screen window is active GSEOS enables several screen specific menus and toolbar buttons for editing and manipulating screens.



Placing an object involves several steps:

- [Selecting a drawing tool](#)³⁹
- [Selecting a drawing region](#)⁴⁰
- [Selecting object properties](#)⁴¹
- [Adjust display style](#)⁴³

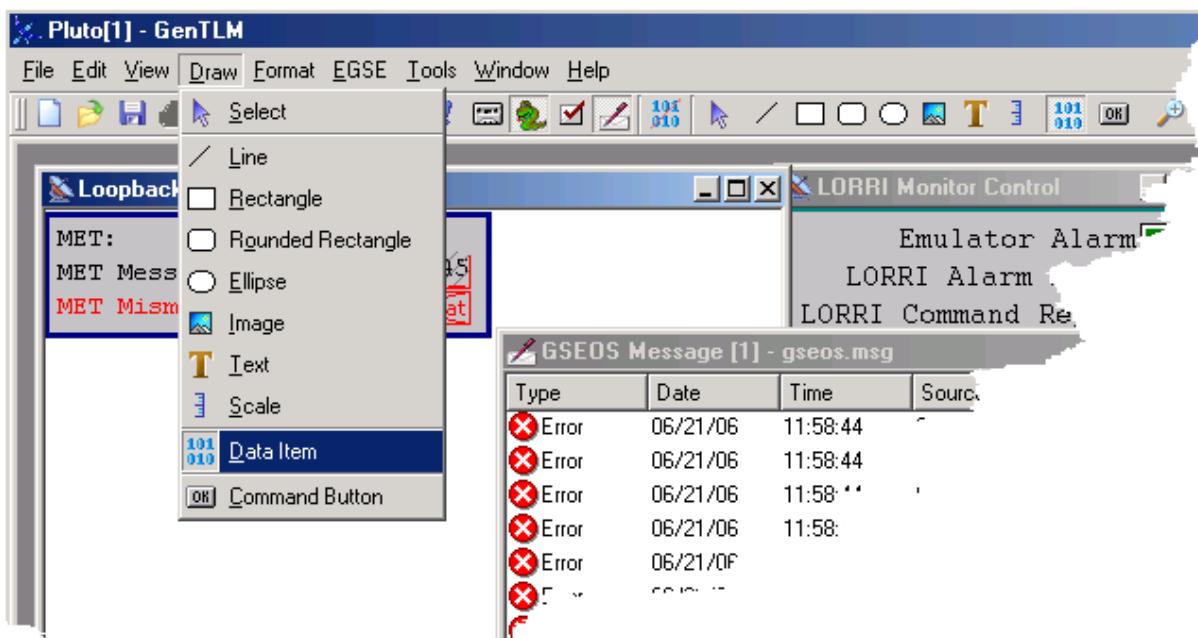
7.4.1.1 Selecting a Drawing Tool

You can either use the [Draw menu](#)⁴⁸ 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 ⁴⁹		Draws a line.
Rectangle ⁴⁹		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 .
Command Button		Draws a command button.

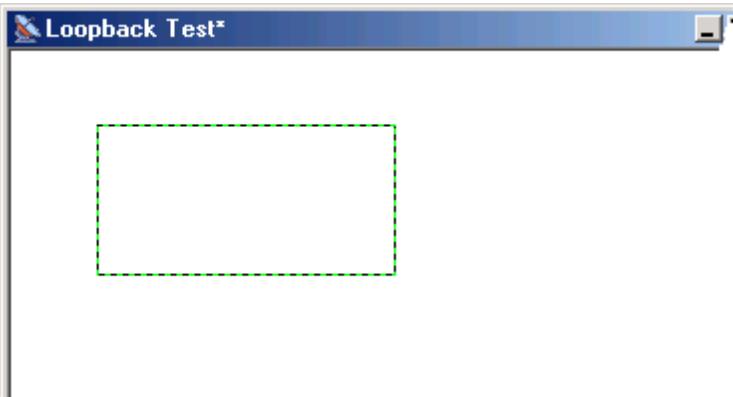
Once you've selected a drawing tool the according submenu in the Draw menu will be checked and the toolbar button will be depressed. The following picture shows the drawing tool 'Data Item' selected:



7.4.1.2 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 at. 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 indicating 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:

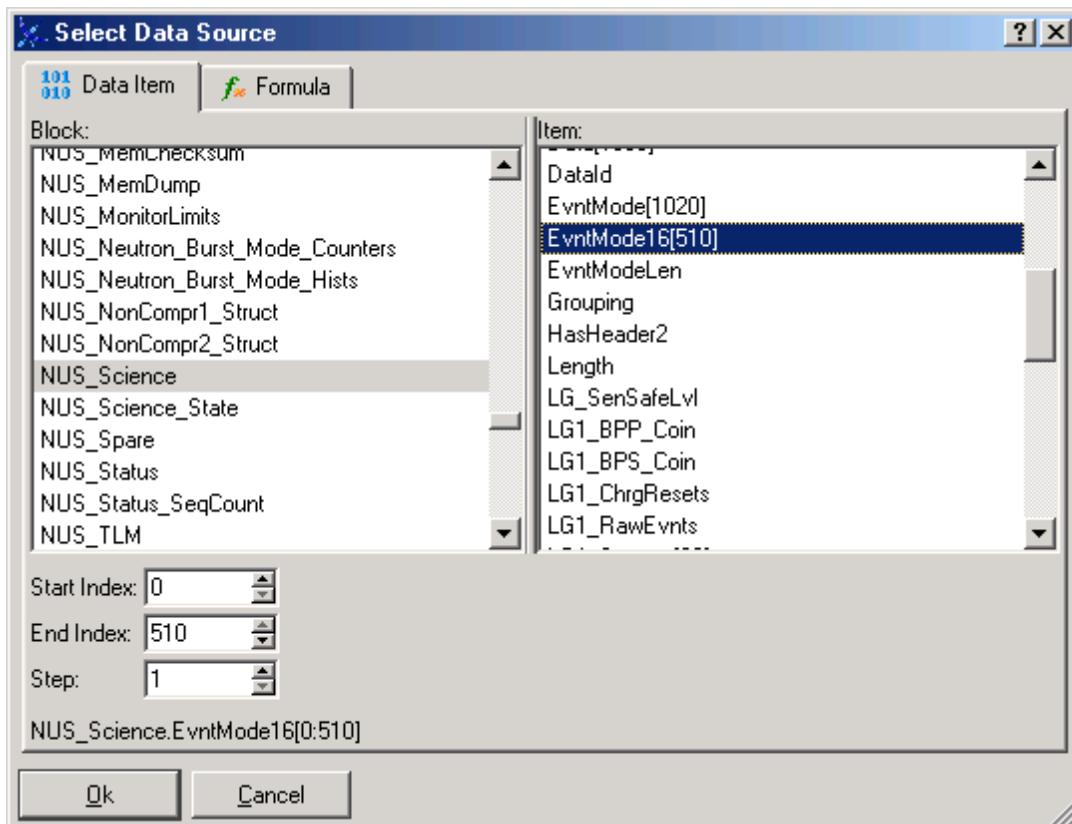


Once you release the mouse button the selection of the drawing region is complete and, depending on the drawing tool, a properties dialog will allow you to [specify properties](#)⁴¹ of the object you are about to place.

7.4.1.3 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 format](#)⁴³.

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](#)⁴⁸ in the [Menus chapter](#)⁴⁷. 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 Data Source dialog specific to the [drawing tool Data Item](#)⁵¹ will pop up:

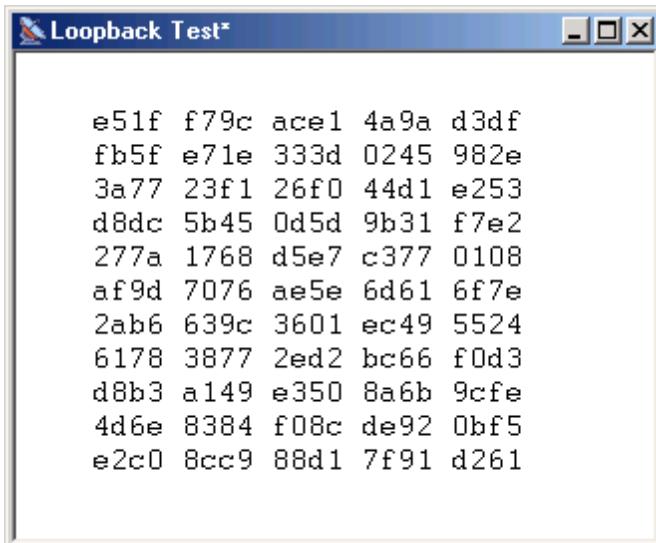


The Select Data Source dialog allows you to select the data item you want to display. You can choose between a plain data item or a formula can combine multiple data items to compute a new display value on the fly. For a simple data item you will see all the blocks defined in your various [block definition files](#) ¹⁵⁰ 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 Start Index, End Index, and Step fields are enabled and you can choose the part of the array you want to display.

Note:

Although you might select a very large array only the items that will fit into the selected drawing area will be displayed, when you resize your object later more or less items might be displayed, up to the limit you have specified here.

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 format with the [Data Format Menu](#) ⁵⁹.

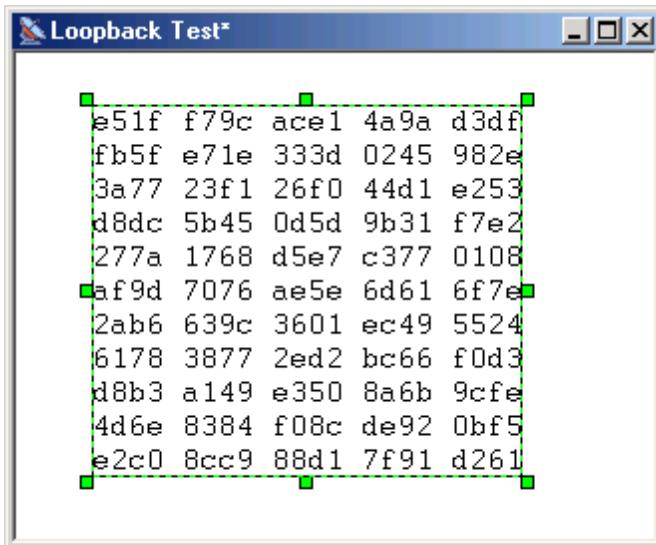


```
e51f f79c ace1 4a9a d3df
fb5f e71e 333d 0245 982e
3a77 23f1 26f0 44d1 e253
d8dc 5b45 0d5d 9b31 f7e2
277a 1768 d5e7 c377 0108
af9d 7076 ae5e 6d61 6f7e
2ab6 639c 3601 ec49 5524
6178 3877 2ed2 bc66 f0d3
d8b3 a149 e350 8a6b 9cf
4d6e 8384 f08c de92 0bf5
e2c0 8cc9 88d1 7f91 d261
```

The caption bar has an asterisk added to the title. This indicates that the screen has been changed. Once you save the screen file it will disappear. You can save your screen with the [File/Save As...!\[\]\(c5fc890ebce2e61be5f14af6b38725d1_img.jpg\)](#) menu.

7.4.1.4 Adjust Display Style

The newly created object is placed with default attributes. The [Format menu!\[\]\(f662e4abf0f142acedf80cdabac13adf_img.jpg\)](#) 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!\[\]\(1cdebd41c460a75b13440a36e964cfb7_img.jpg\)](#) explains the various object properties in detail.

To move an object you can simply drag it with the mouse, that is 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 moved on grid steps) basis.

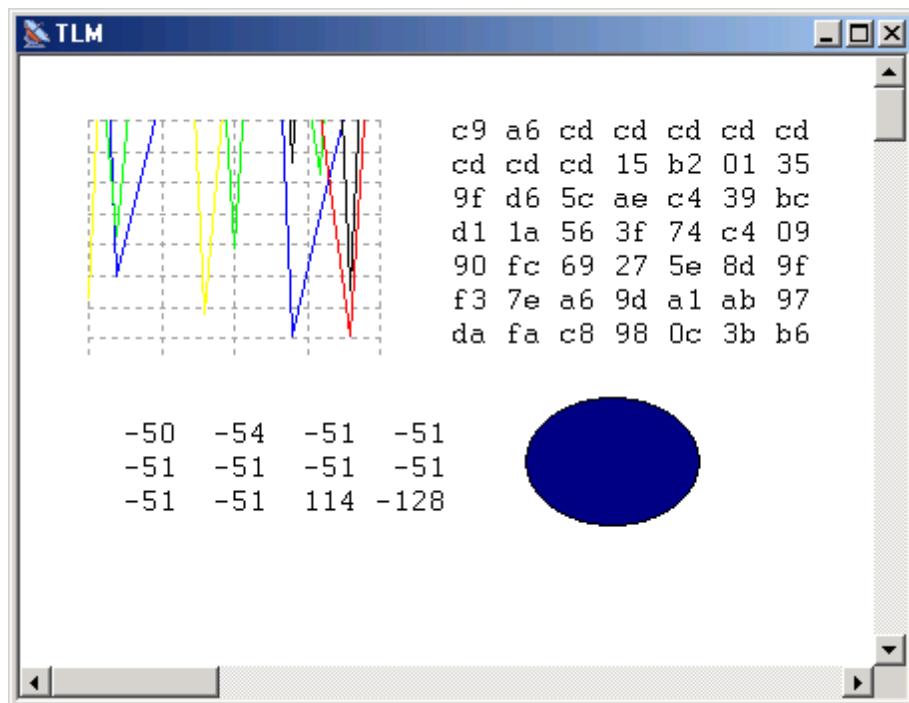
To size the object move the mouse on one of the choose pads and drag the mouse to the desired size of the object. Some objects only support discrete sizes, in this case the objects size will snap back to the closest allowed dimension.

By right-clicking on the object you will open the [Format menu](#) that will allow you to set the style of various attributes on the selected object.

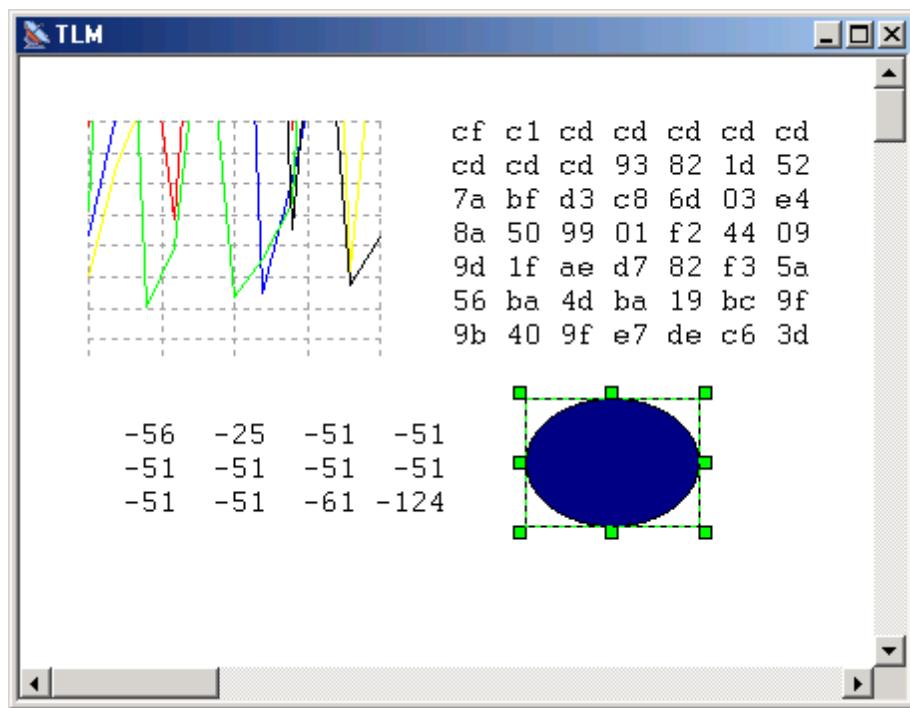
7.4.2 Selecting Objects

After you have placed several objects on a screen you can modify their attributes by selecting them either individually or as a group.

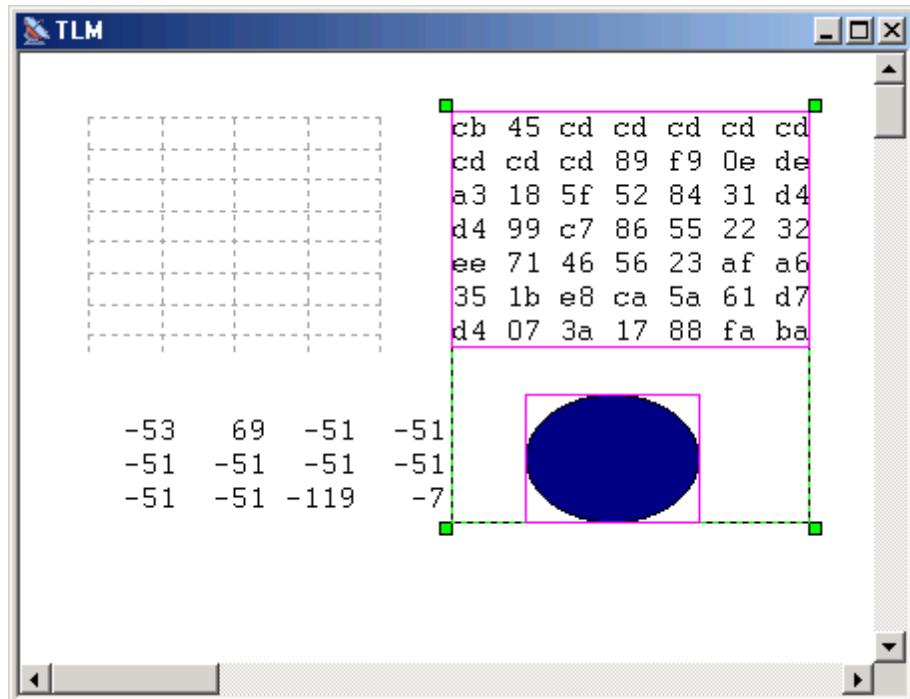
The image below shows a screen with several objects.



To enter select mode choose Draw>Select from the main menu or click on the Select toolbar button. If you click on one of the items it will be selected as indicated by the 'chooser' around the object.



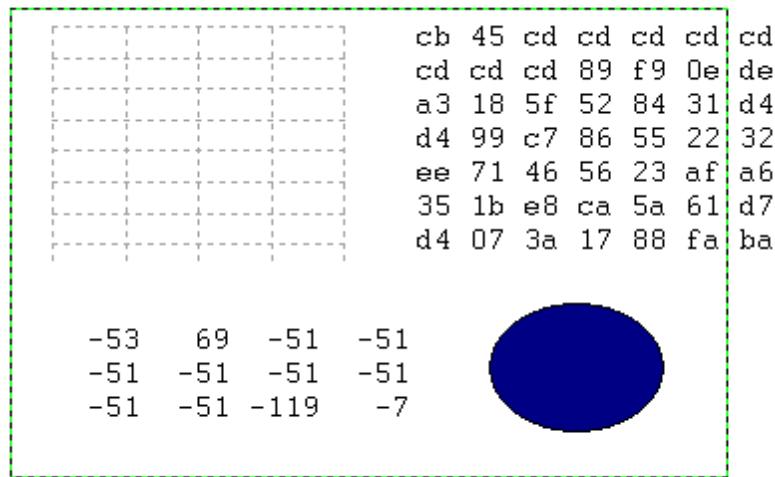
You can now either right click to get the Format menu to change the objects format or select Format from the main menu. To add other objects to the selection you can hold the **Shift key** down and click on the objects you want to add to your selection.



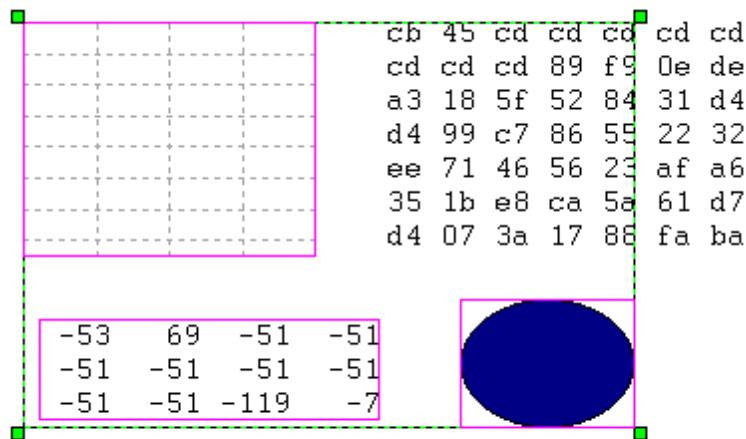
The selection rectangle is expanded and each object that is selected shows a frame around it. You can now apply any changes to your selected objects to the entire group. Depending on what kind of objects are in the group certain format options might be

enabled. Lets say you select data items only, the Format/Data Item... menu will be enabled, however, if you have a static object in the selection like in the sample above the Format/Data Item... menu will be disabled since the static object in the selection doesn't support Data Item formats. This applies similarly to other object combinations. If all objects in a selection have the same value for a certain property, say the foreground color is red, this value will be selected for this property. However, if at least one object in the selection has a different value from all other objects in the selection the value for this property is set to the default value.

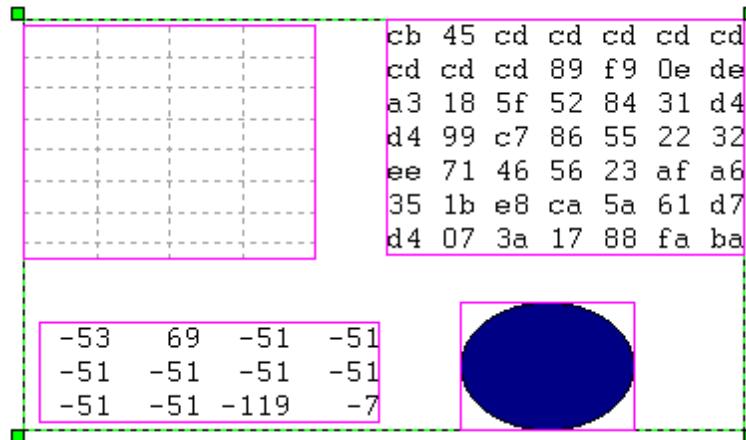
Another way of selecting one or multiple objects is to choose select mode and to click on the screen and drag open a selection rectangle. All objects that wholly fall within the rectangle will be selected:



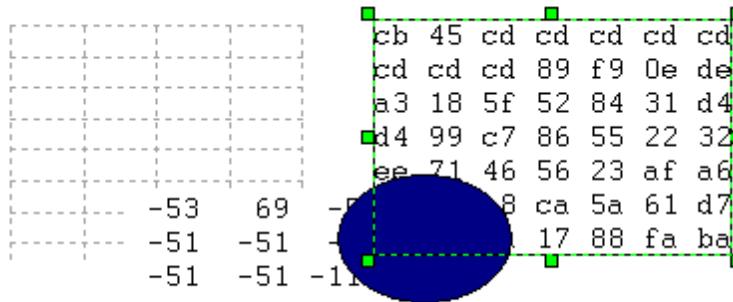
After you release the mouse the objects within the rectangle will be selected:



To select all objects on the screen you can choose **Edit/Select All** from the main menu:

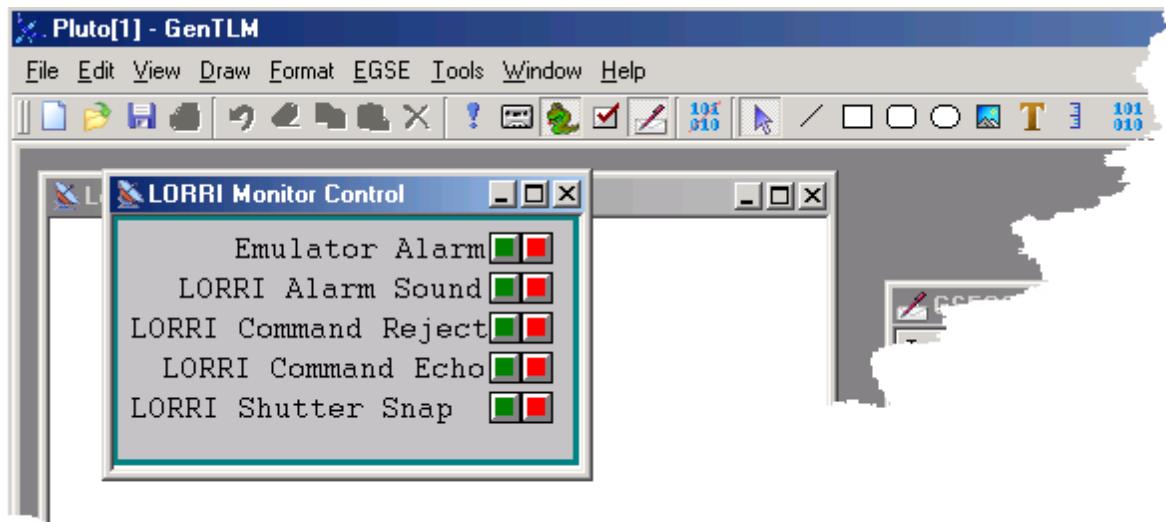


If items overlap on the screen you can iterate through all objects by placing the mouse on a point where all object intersect, holding the **Ctrl key** down and clicking on the objects. The same works if an object is completely obscured by another object.



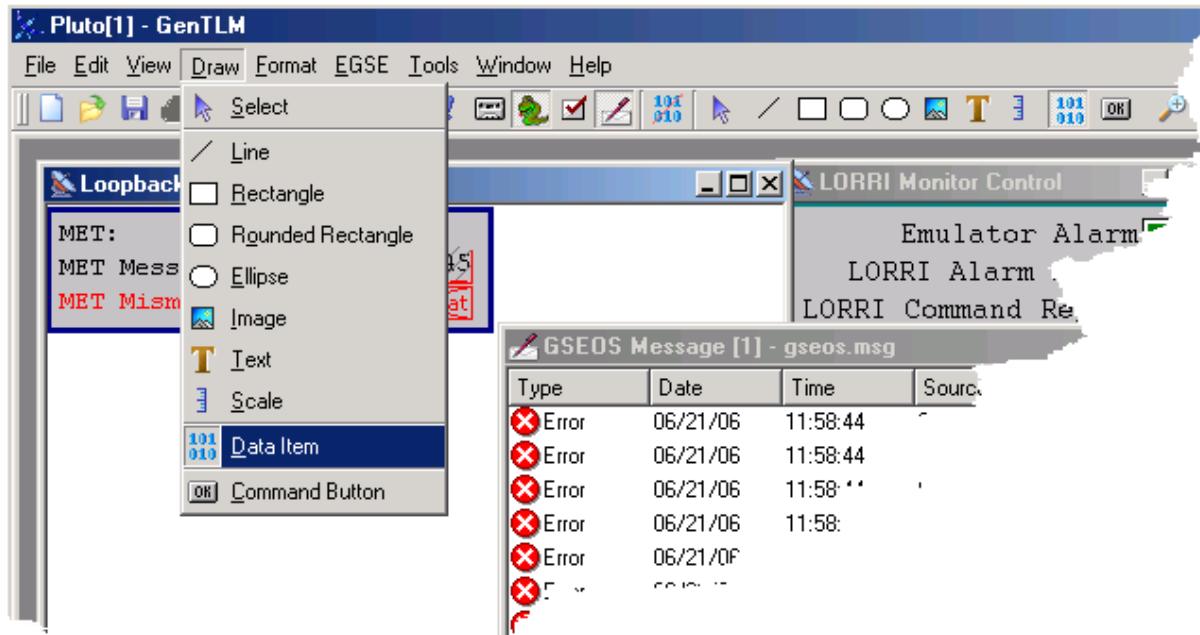
7.4.3 Menus

Screen windows support the [Draw](#)^[48], [Format](#)^[59] menus from the GSEOS main menu.



7.4.3.1 Draw

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](#) 49

[Rectangle](#) 49

[Rounded Rectangle](#) 49

[Ellipse](#) 49

[Image](#)  49
[Text](#)  50
[Scale](#)  50
[Data Item](#)  51
[Conversion Function](#)  57
[Command Button](#)  55

7.4.3.1.1 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](#)  62. The line width and [color attributes](#)  61 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.

7.4.3.1.2 Rectangle

The Rectangle drawing tool allows you to place rectangles on the screen. Rectangles like [Lines](#)  49 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](#)  62. 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 [Format menu](#)  59.

7.4.3.1.3 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](#)  49. The radius of the corners can not be adjusted.

7.4.3.1.4 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](#)  62. 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 [Format menu](#)  59.

7.4.3.1.5 Image

The Image tool places static bitmap images on the screen. The images can be in *.gif, *.jpg, *.bmp, or *.png 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 image file you wish to display. The image will be scaled to the dimensions of the drawing area you selected. There are no formats for the image object.

7.4.3.1.6 Text

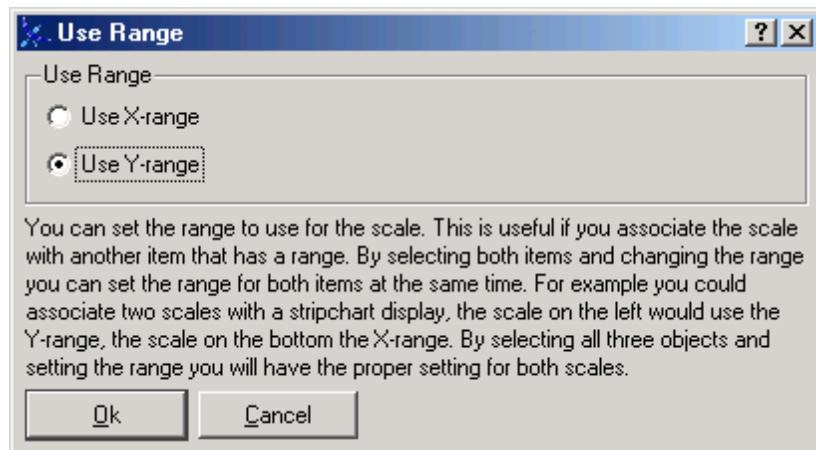
The Text tool allows you to place static text on the screen. Static text is useful in describing data items placed on the screen. The Text tool can also be activated from the toolbar with the  button. After you finish selecting your drawing area the specific dialog for the text tool pops up:



This dialog allows you to enter the text you want to display. The text object recognizes the [Font](#)^[60], [Color](#)^[61] and [Text](#)^[60] formats. You can change these attributes with the [Format menu](#)^[59]. To change the text simply double-click on the text object and the Text dialog will pop up and allow you to edit the text.

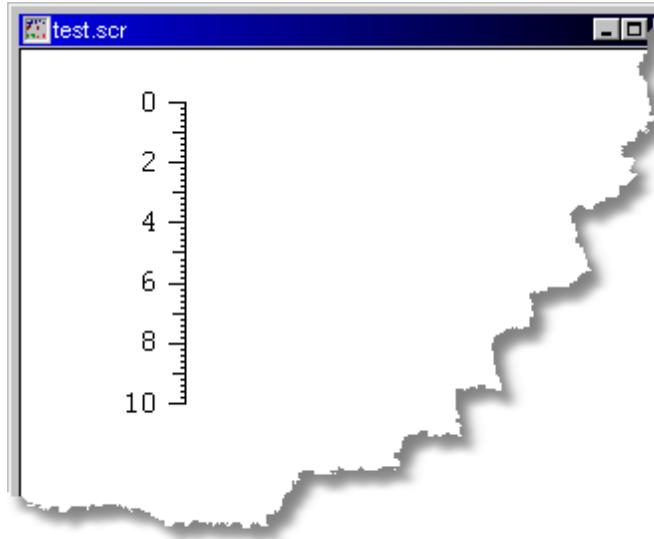
7.4.3.1.7 Scale

The Scale tool will place a scale on the screen. Scales are typically used in conjunction with other graphical objects like [Bargraphs](#)^[69] or [Stripchart](#)^[81] 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 to select the [range](#)^[62] 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):

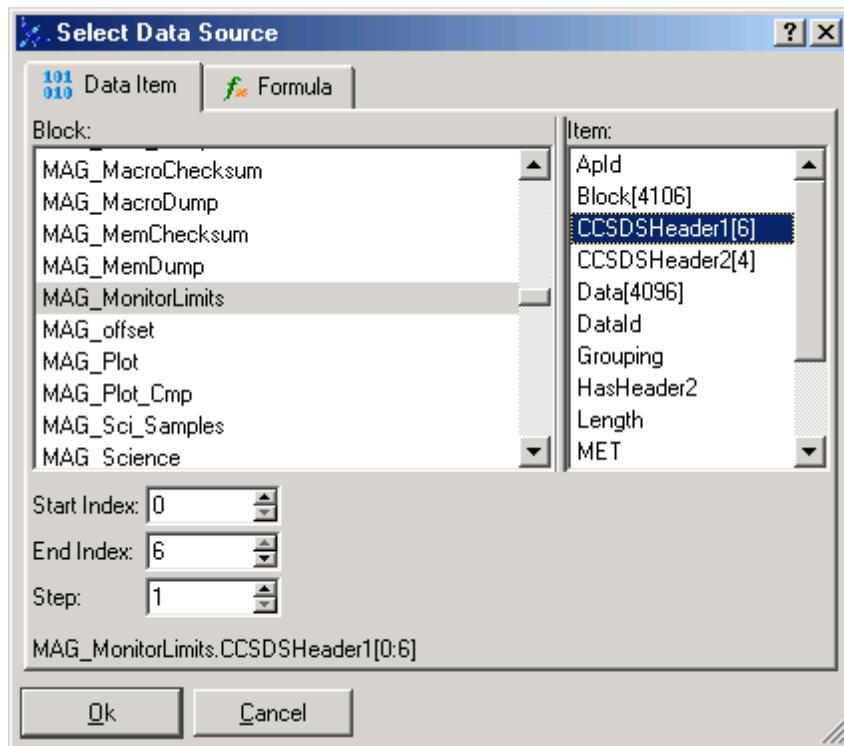


The formats you can set for a scale object include: [Color](#)^[61], [Text](#)^[60], [Range](#)^[62], and [Orientation](#)^[63].

7.4.3.1.8 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](#)^[150]. Data items can be displayed in a variety of formats, numeric as well as graphic formats. Please refer to the [Format/Data Item](#)^[65] 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:



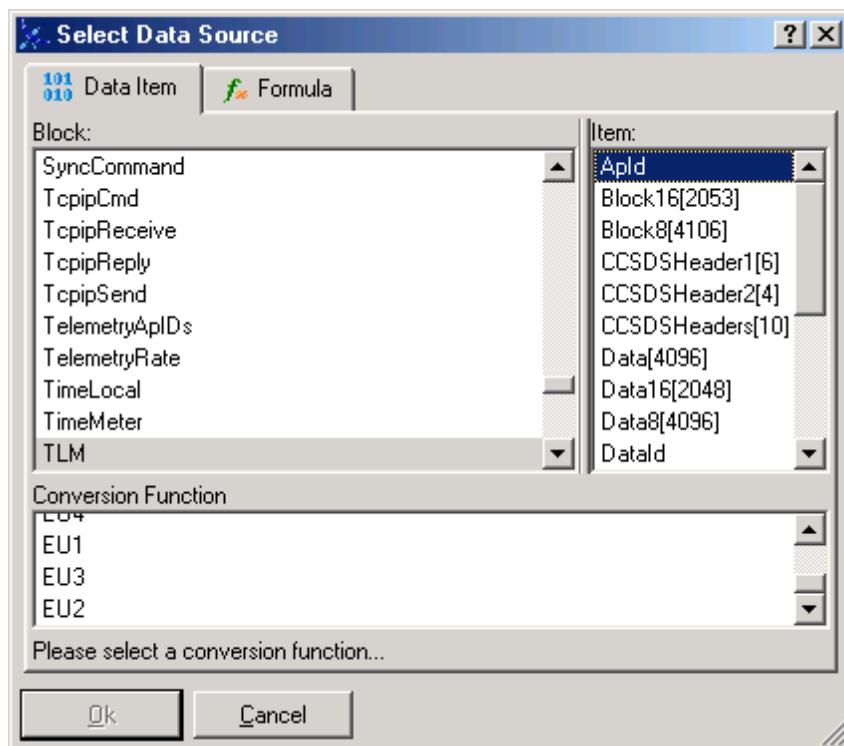
This dialog allows you to select the data item you want to display. You can either select a single item directly as defined in your block definition or you can select a formula that you have loaded. The tab on top of the dialog allows you to switch between the two. Let's first take a look at the single data items.

Single Data Items

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 **Start Index** edit box specifies the first index in the array to be displayed. Arrays are zero based. The **End Index** edit box specifies the last index (not including). 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 if the selection area can't hold all of the items you have selected it will only display the number of elements that fit into your selection area. This depends on the font and size you choose. When you move your cursor over the item on the screen the status bar will indicate which items are actually being displayed. Your selection area will be adjusted to fit tightly around an integral number of selected elements.

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 (see also the [Update Properties](#) of data items).

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 format](#) is a graphic mode ([Bargraph](#), [Stripchart](#), [Bitmap](#)) and the [range](#) is not set properly the display might appear empty.

Note

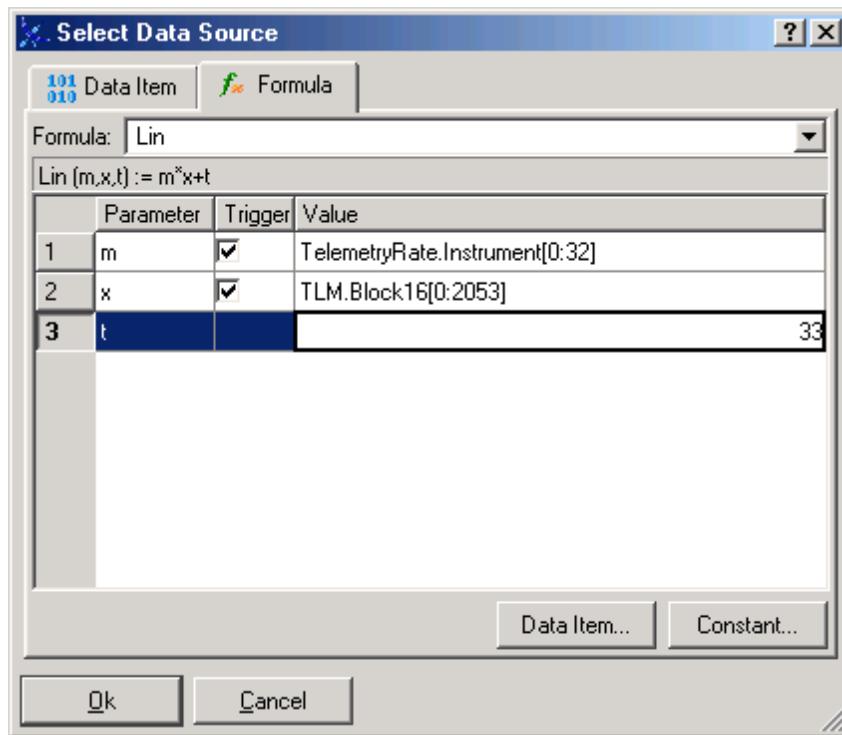
For fast updating items the font selection for numeric format display is critical to system performance. True Type fonts require a lot of system resources to render. The fastest fonts are fixed pitch fonts, variable pitch and True Type fonts are the slowest to display. This is mainly a problem for fast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

Expressions (Formulas)

Formulas let 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 an expression for display. 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.

Formula objects can be displayed in the same styles as a simple data item. Please refer to the [Format/Data Item](#) section for a detailed explanation of the various formats available. When you select Formula on the Data Source Dialog it will look similar to this:



The Formula sheet is structured into two parts. First you have to select a predefined Formula. Formulas 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 a Formula, the function is displayed with its 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.

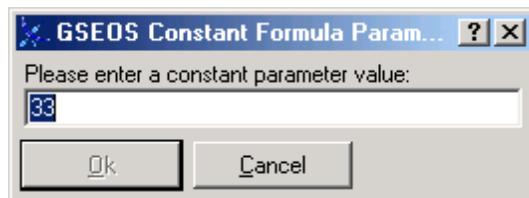
Triggers

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. 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, otherwise your data will never be updated.

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

When you click on the Data 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 of regular data items you have to use one of the conversion functions available. For more complex Expressions a specific decoder is the more appropriate solution.

Note

If your default [drawing format](#) is a graphic mode ([Bargraph](#), [Stripchart](#), [Bitmap](#)) display formats) 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](#).

7.4.3.1.9 Command Button

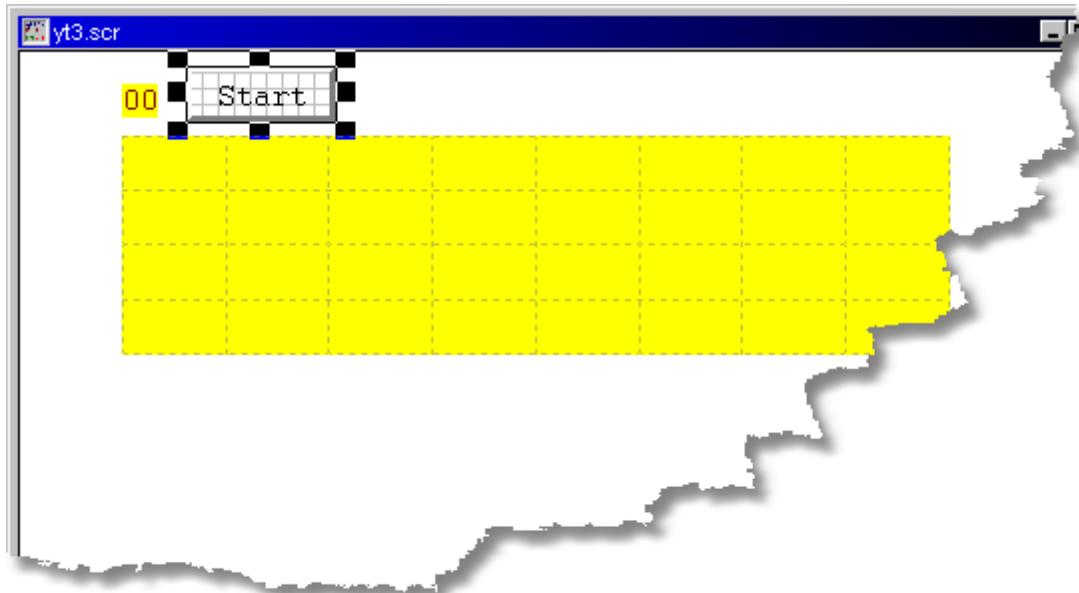
The Command Button tool allows you to place a Push-button on the screen. On activation this button will issue the command you configure. A simple text preprocessing module allows you to prompt for command parameters. Commands can be any valid command you define with your [command definition files \(*.cpd\)](#). 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 displayed 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.Command field and the CmdString.Len set properly. The CmdString in turn is processed by the GSEOS command processor and executes your command as defined in the command definition.

The following snapshot shows the resulting button:



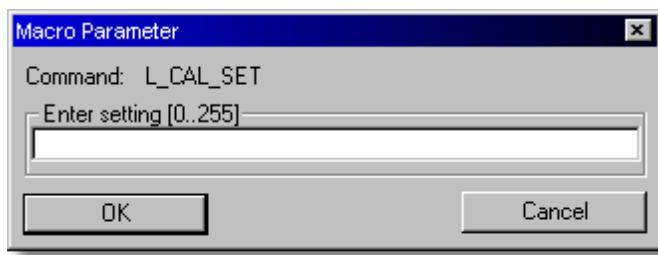
The Command String text allows simple text replacement:

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:

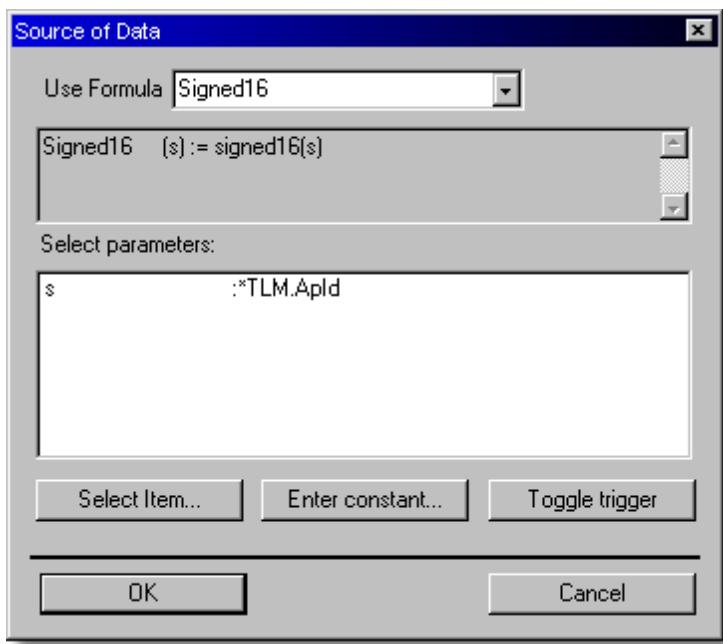
```
L_CAL_SET("E_RANGE1", 20)
```

When you move the mouse cursor over the button the Command String will be displayed in the GSEOS status bar. The command button object recognizes the following formats: [Text](#)⁶⁰, [Color](#)⁶¹.

7.4.3.1.10 Expression

The Expression tool is similar to the [Data Item tool](#)⁵¹ 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](#)¹⁶⁸ 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](#)⁶⁵ 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:



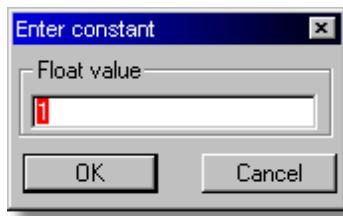
The Expression dialog is structured into two parts. First you have to select a predefined Expression. Expressions are defined in [Formula Definition files](#)^[168] and have to be loaded before they can be accessed. If you don't find your Expression function in the drop down list you can use the [GSEOS Explorer](#)^[127] to show you all loaded Expressions. Once you select an Expression, the function is displayed with its 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](#)^[54] 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:

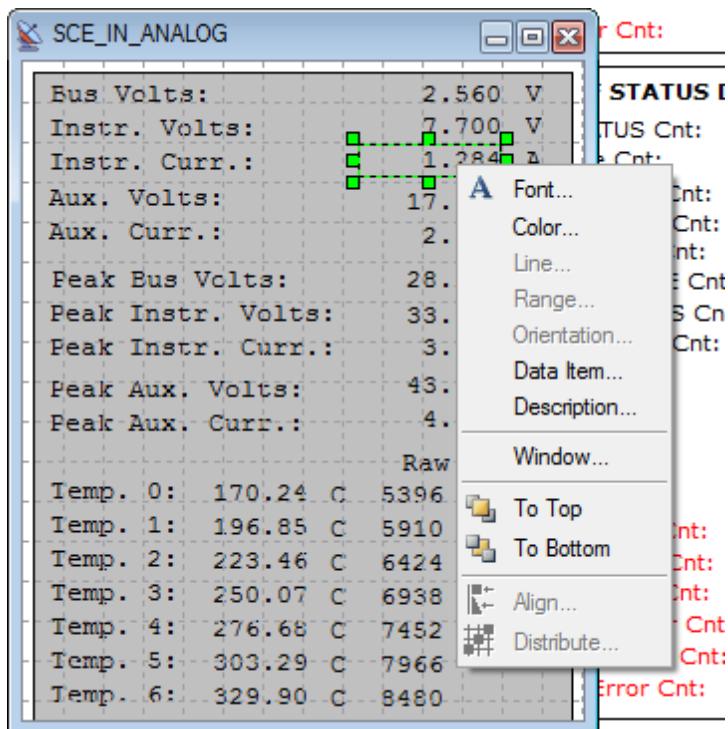


Any valid float number is acceptable as a constant.

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 style options that can be set for a [data item](#)⁵¹ are also applicable to Expressions.

7.4.3.2 Format

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:

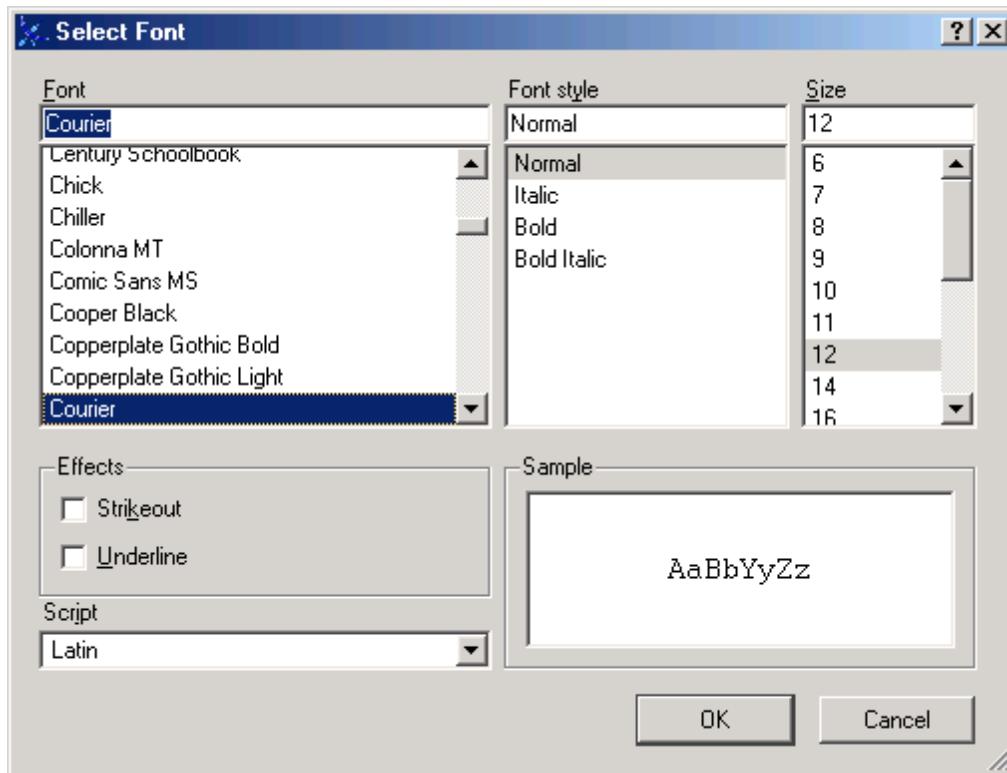
- [Font](#) [60]
- [Color](#) [61]
- [Line](#) [62]
- [Range](#) [62]
- [Orientation](#) [63]
- [Data Item](#) [65]
- [Description](#) [85]
- [Window](#) [85]

The entries on the bottom of the menu: To Top, To Bottom, Align..., and Distribute... let you arrange the items on the screen. To Top moves the select item(s) to the top of the z-order (keep in mind that data items will show when new data arrives even if they are obscured by items that are on top of them), To Bottom moves the selected items to the bottom of the z-order. The Align... and Distribute... menus let you align and distribute multiple items in relation to each other.

7.4.3.2.1 Font

The Font format sets the font of the object. Only numerical and textual objects like [Data Item](#) [51], [Text](#) [50], [Command Button](#) [55], 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. In general fixed pitch fonts are

faster to render than variable pitch fonts. Especially for fast updating items you might want to stick with fixed pitch fonts for better system performance.

The system default font is Courier 12pt which is available on all supported platforms (Windows, Linux, Mac OSX). If you do switch between platforms with a single project please make sure the font you choose is available on all platforms and renders similarly. You can set a custom default font, that is if you don't specify a font for a text object it will use that font instead of the system default of Courier 12pt.

You have to set the following entries in the [gseos.ini \[QLook\]](#) section:

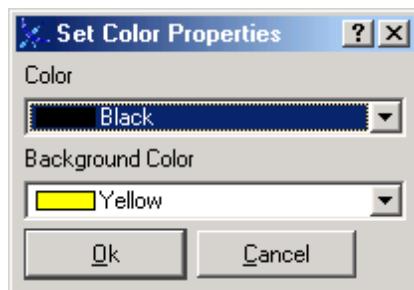
```
[QLook]
DefaultFontName = Courier New
DefaultFontSize  = 10
```

As mentioned earlier make sure you use a font that is available on all platforms you plan on using.

7.4.3.2.2 Color

The Color format modifies the foreground and background colors of the object. Almost all objects support foreground and background color, the only exception are [Data Items](#) that use the [Status Text](#), [Status Image](#), or [Bitmap](#) 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.

Stripchart 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 stripchart displays selected you will get the following dialog box:



The color for Graph 0 is equivalent to the foreground color. If you convert a stripchart display to a different type you will lose the settings for the colors of graphs 1 to 4.

7.4.3.2.3 Line

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

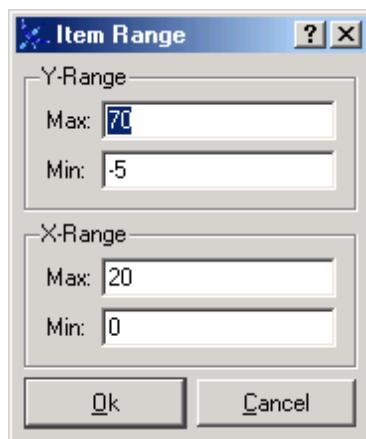
You can change the line width with the following dialog box:



7.4.3.2.4 Range

The Range format sets the range settings of the display object. Ranges are used for graphical objects like Bargraph, Stripchart, and Bitmap. For two dimensional items like Bitmap graphs (2D histograms typically), Stripchart both dimensions are used, for Bargraphs, and Scales only the y-dimension is used.

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



Bargraph

For [Bargraph](#) 69 displays the y-dimension determines 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.

Stripchart

For the [Stripchart](#) 81 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.

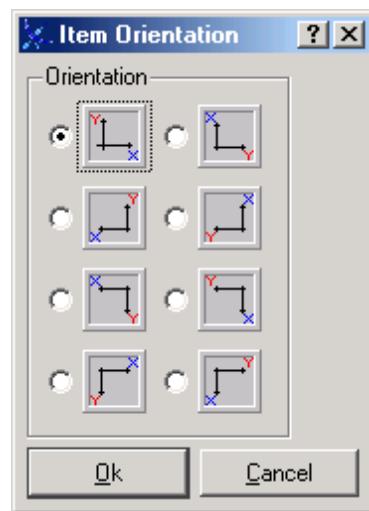
Bitmap

The [bitmap](#) 71 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.

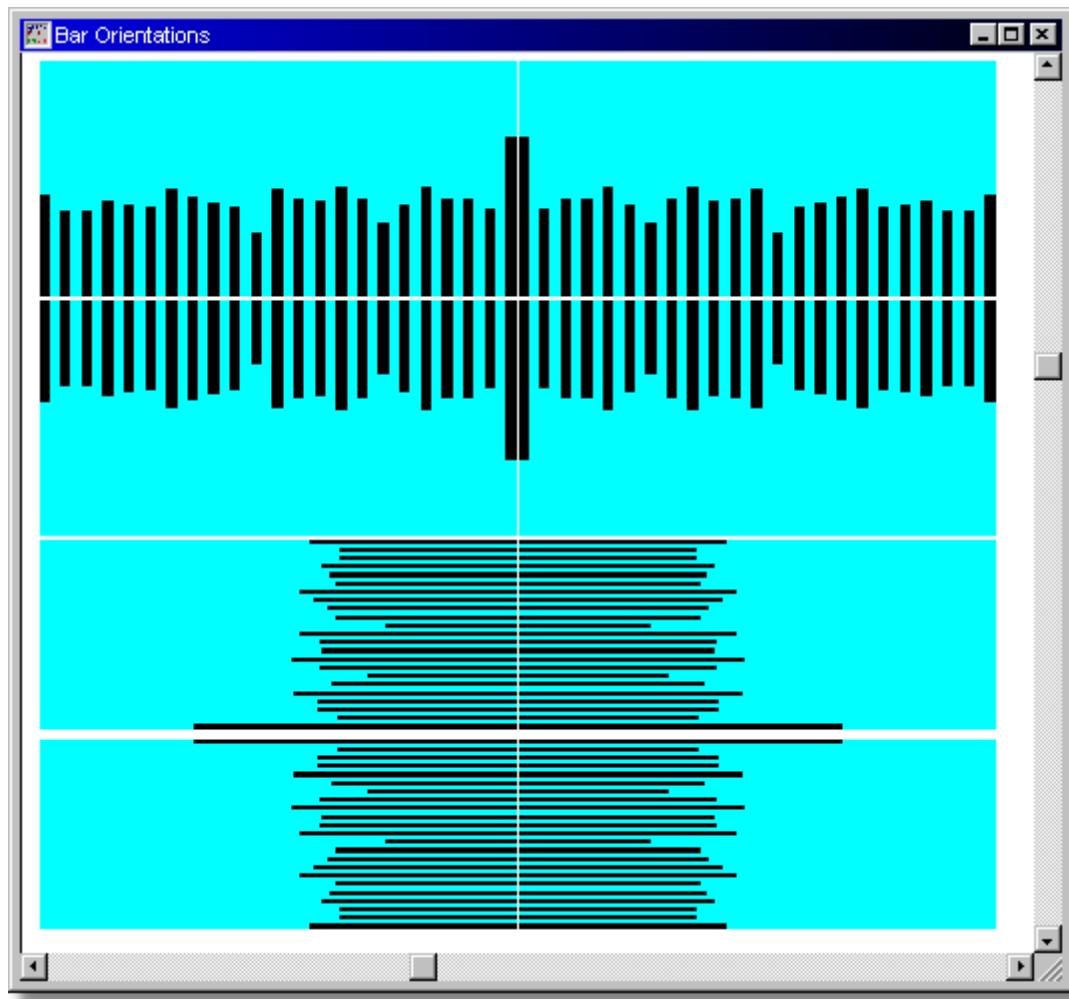
7.4.3.2.5 Orientation

The Orientation format modifies the way the data is oriented. This format applies mostly to graphical styles like Stripchart, Bitmap, RGB Bitmap, Bargraph, etc. There is a total of eight different orientations available.



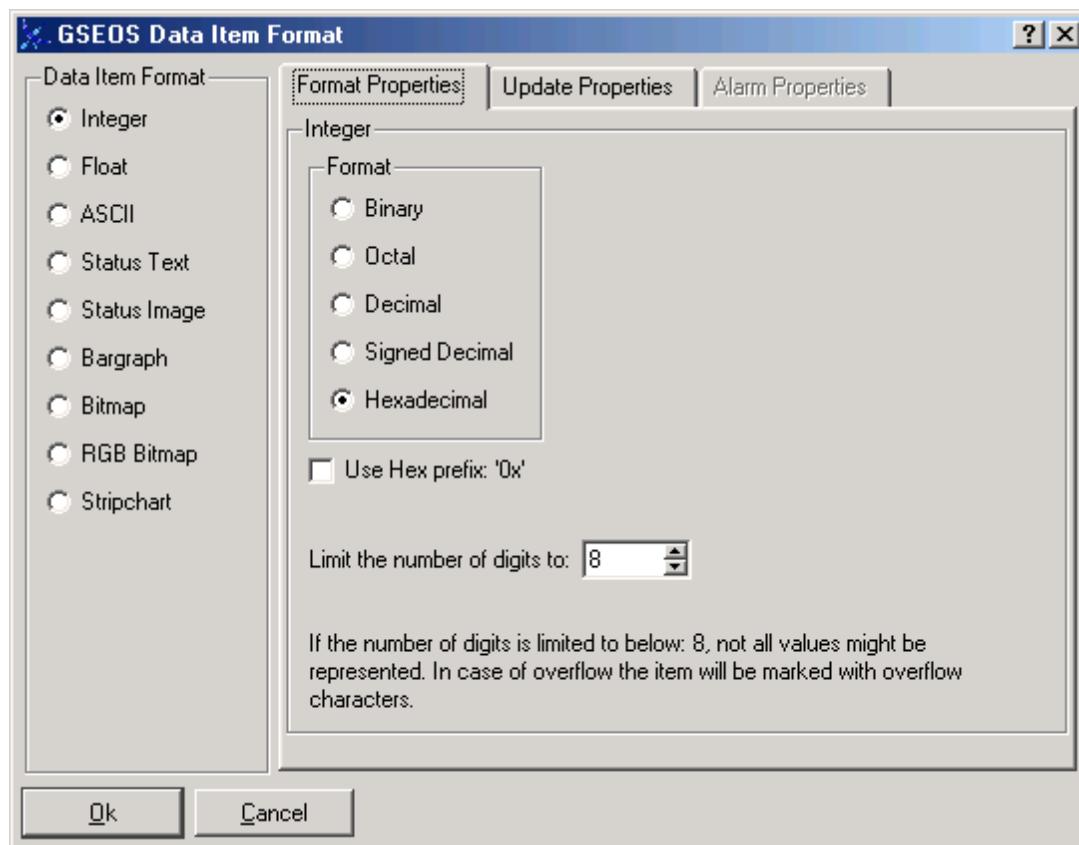
For graphical representations like Bargraphs, Stripcharts, and Bitmaps the origin is usually in the lower left corner which is the 1L (1st quadrant, left hand system) orientation.

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



7.4.3.2.6 Data Item

The Data Item format dialog box is invoked by selecting Data Item from the Format 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 representations like Stripchart, Bargraph, Bitmap, etc. Depending on the format you choose you can set different properties of the display. Besides the display format you can also set the update properties and alarm properties of the item if it applies. The next few chapters will explore the Data Item Format dialog in detail since it handles many settings. The following image shows the data item format dialog:



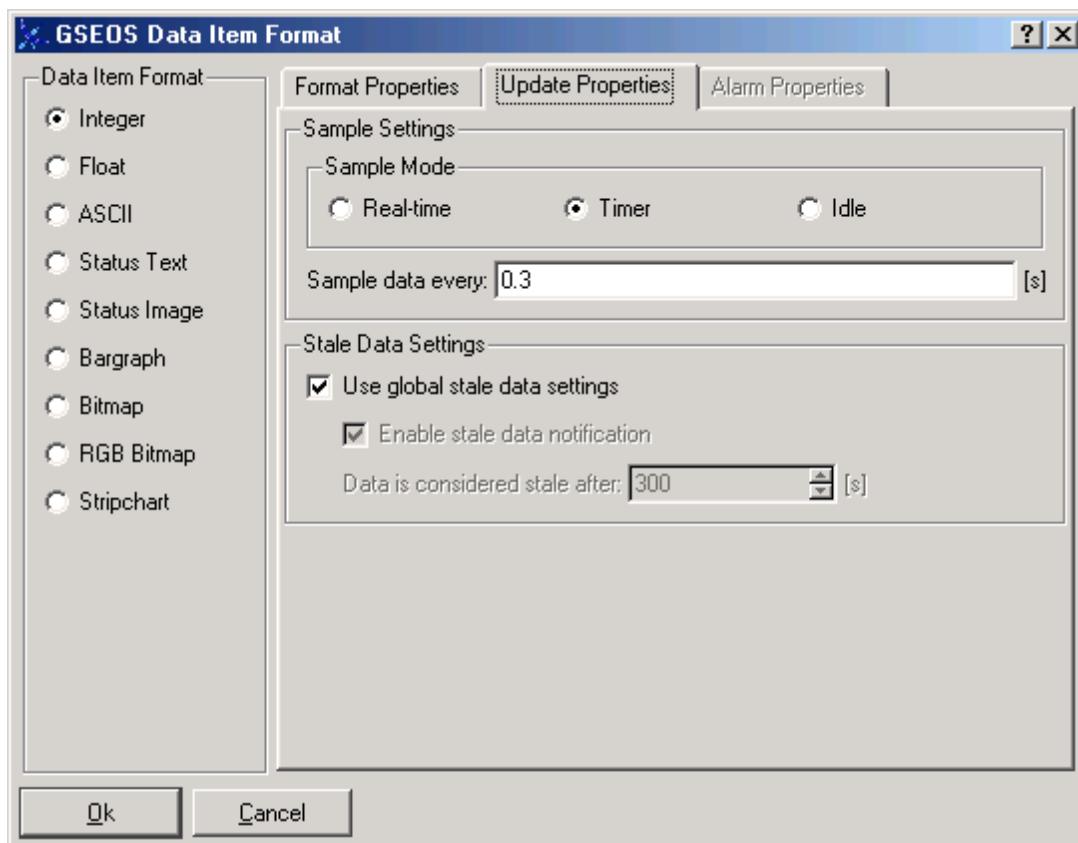
The dialog is separated into left hand pane and right hand pane. On the left hand pane you can select the format you want the item to be displayed in. The right hand pane allows you to modify the properties of that particular format. Besides the Format Properties you can also change the [Update Properties](#) and the [Alarm properties](#). In the dialog above the format Integer is selected and you can set the specific Integer type in the format properties tab. In our case it is set to Hexadecimal.

The numeric formats supported are [Integer](#), and [Float](#). The available text formats are [ASCII](#), and [Status Text](#). The graphic formats are [Bargraph](#), [Bitmap](#), [Status Image](#), [RGB Bitmap](#), and [Stripchart](#).

Some formats also allow you to select alarm properties, in this case the dialog enables the '[Alarm Properties](#)' tab that controls the alarm settings.

7.4.3.2.6.1 Update Properties

The update properties determine when and how often an item is updated.



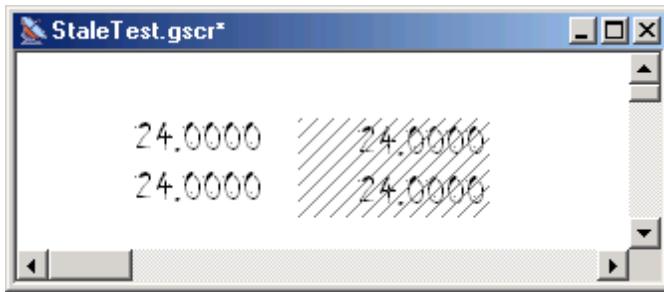
The update properties are grouped into two sections: Sample Settings, and Stale Data Settings. The Stripchart format has an additional setting for the update properties, please refer to the [Stripchart](#) chapter for more details.

Sample Settings

The sample mode determines how often data gets sampled. In Real-Time mode the data gets updated every time a new block arrives, this is the most accurate mode since all arriving data will be updated accordingly. You should not enable real-time mode for high frequency items for performance reasons. For once it is hard to visually distinguish quickly updating data and secondly the high update rate will use up CPU resources. Timer mode samples at the given rate and updates when it detects new data. This is the default setting. The shortest sample time is 100ms and it can be increased in 100ms steps. Idle mode updates the display whenever the system is idle and uses the least system resources.

Stale Data Settings

The stale data settings determine how stale data is handled for this item. The global settings can be modified with the [View/Options...](#) menu. If you uncheck the "Use global stale data settings" you can override the global settings with your preferences for this item. If you enable stale data notification the item will be marked stale when it does not arrive within the configured timeout. Enabling stale data notification on this local level overrides the global setting. When a data item is detected to be stale its display is changed to a crossed out style. The item on the right is the stale version of the one on the left.



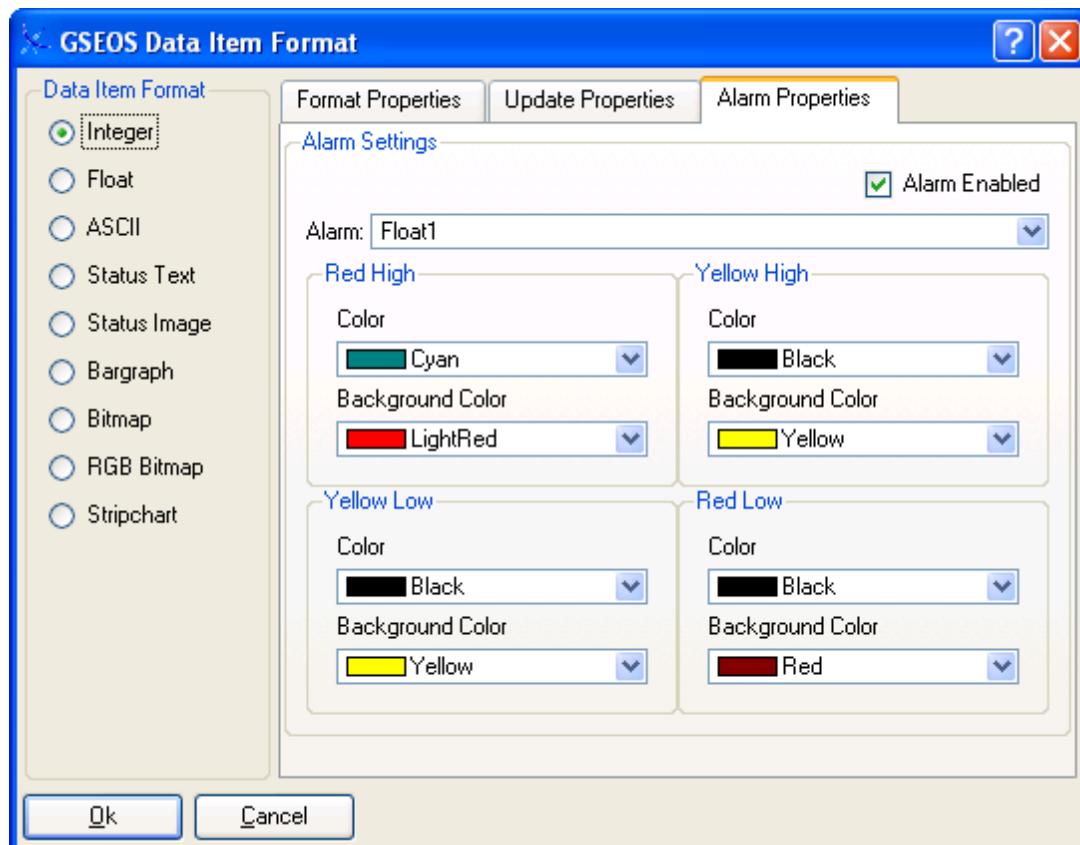
7.4.3.2.6.2 Alarm Properties

The following data item formats support alarmed display mode:

[Integer](#)⁷⁵

[Float](#)⁷⁴

If you select one of these formats the Alarm Properties tab will be enabled:



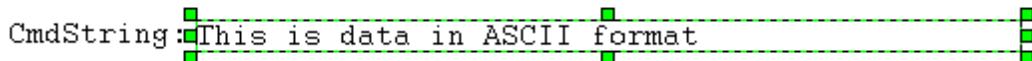
The settings on this tab let you specify the [alarm definition](#)¹⁴⁹ to use. Keep in mind that for your alarm definitions to show up in the select box you have to load them first. If you select no alarm the Enabled box will be unchecked and disabled. You can also manually uncheck the Enabled box even when you have selected a valid alarm. Setting alarm checking on an item adds considerable processing overhead and should be used

cautiously. We recommend to use a [timer based update](#). Especially alarm checking on fast updating items and large array items might slow down the system.

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

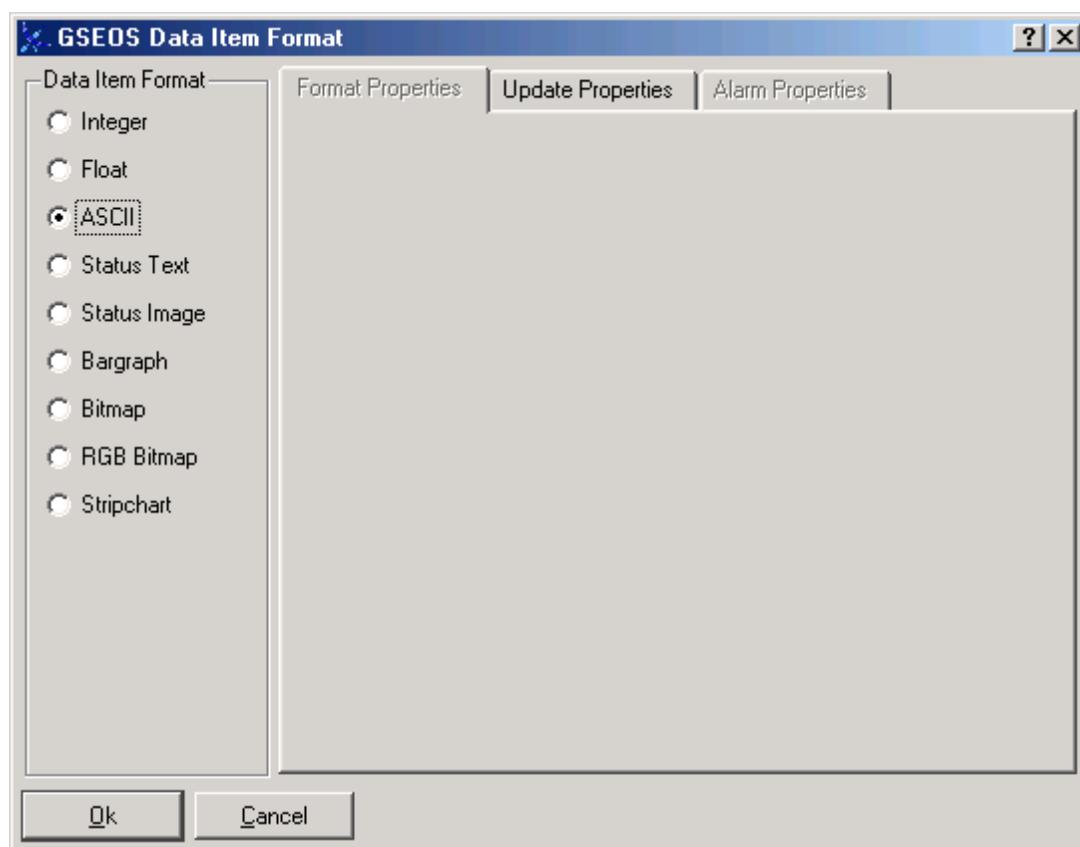
7.4.3.2.6.3 ASCII

The Data Item format ASCII 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.



CmdString : This is data in ASCII format

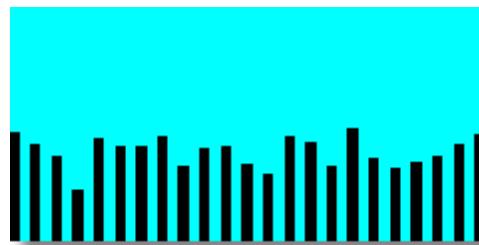
The Data Item Format dialog box for the ASCII format looks as follows:



The ASCII format has no custom Format properties.

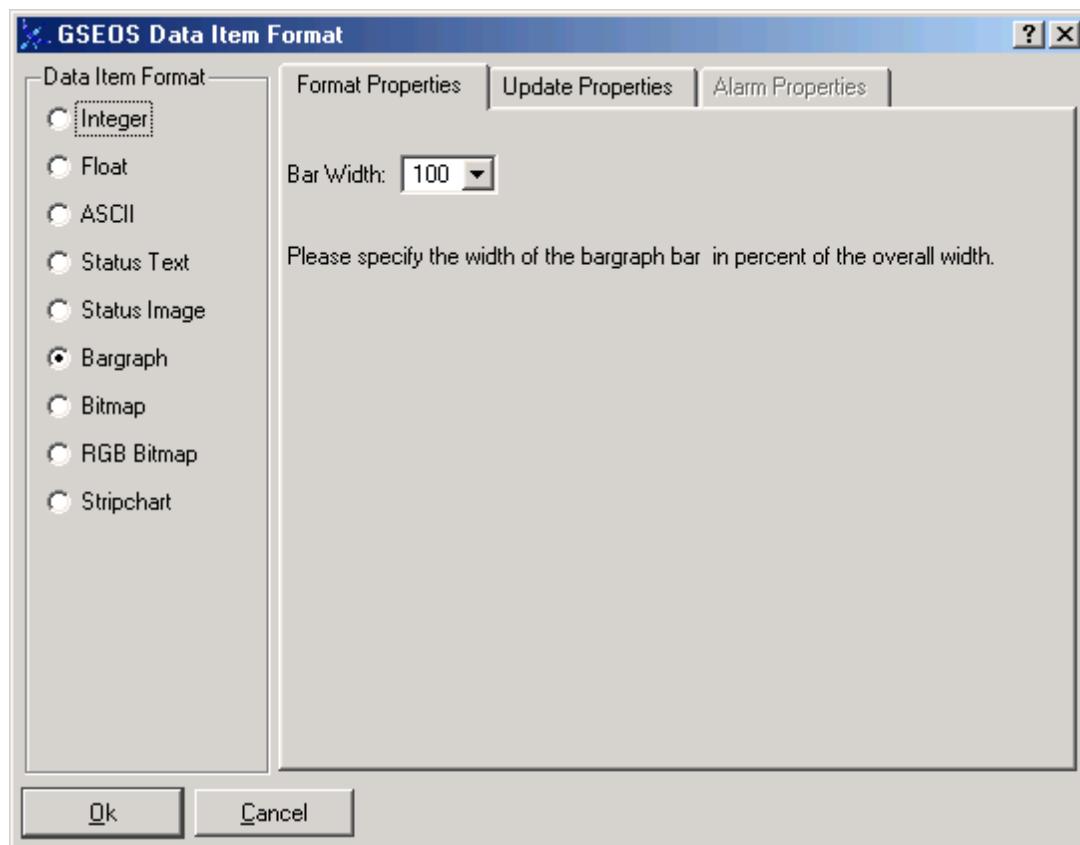
7.4.3.2.6.4 Bargraph

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



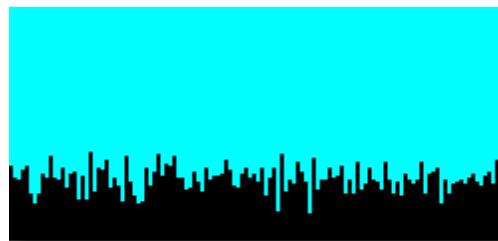
The above display shows an array item in Bargraph format. You can specify the width of the graphs in percent, this allows for a configurable gap between the bars. If you set the width to 100% there will be no gap. For scalar items the Width setting has no effect.

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



Note

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.

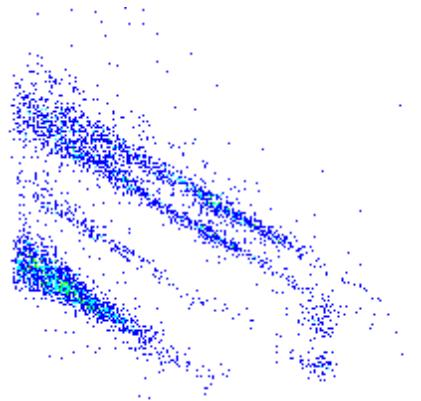


In the above picture an array item with 1000 elements is displayed. The screen can not hold that many values on the display area and bars get dropped. The status bar will display how many elements are being displayed.

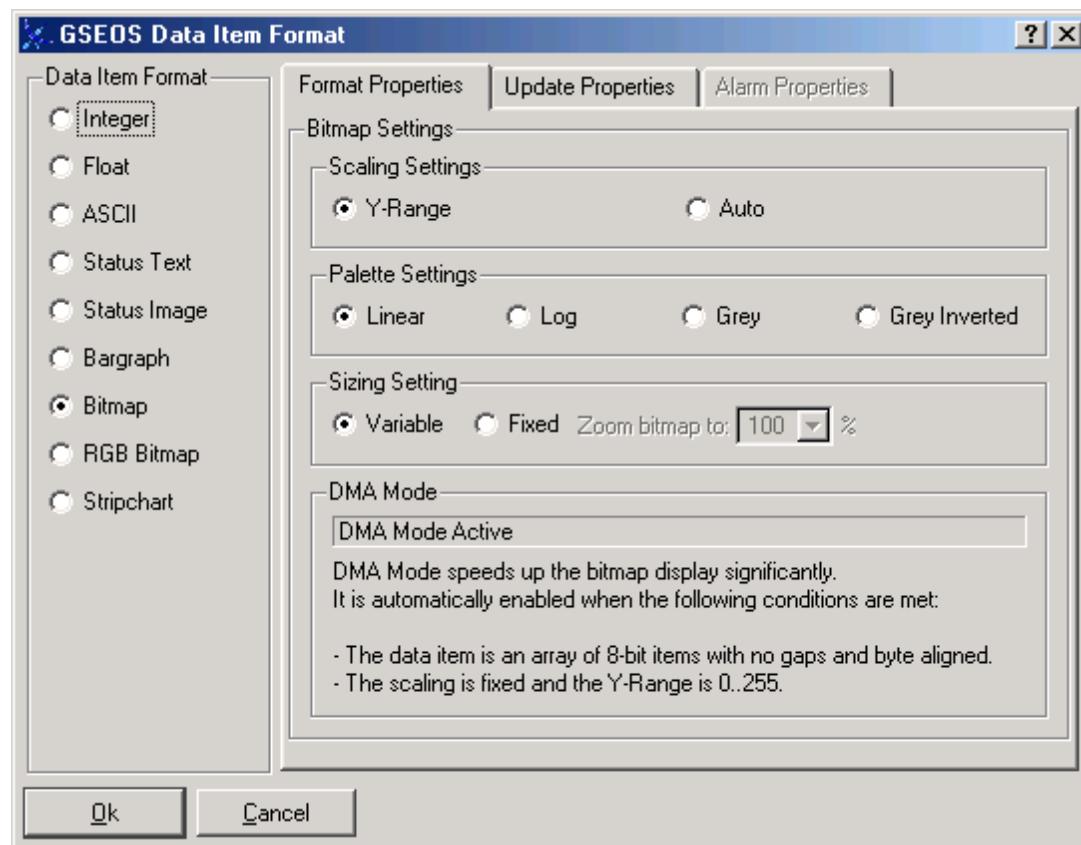
As with other graphical displays the proper setting of the [range](#)^[62] is important. In the case of the Bargraph display only the y-dimension of the range is used. It maps the values onto the graphical display. Say your y-range is 0..1000. If your data value is 333, the bar will extend to about 1/3 of the total display height. The orientation of the bar depends on the settings of the [Orientation](#)^[63] property.

7.4.3.2.6.5 Bitmap

The Data Item format 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](#)^[264] section for more information on histograms. The following picture shows a Bitmap display:



The Data Item format 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](#)^[62] 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](#)^[63] setting). The y-Range is used to map the data values onto colors as outlined in the following paragraph.

Scaling Settings

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.

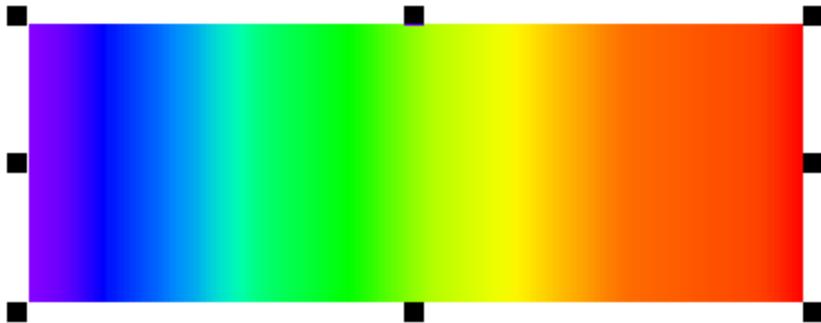
Sometimes it may be desirable to have a relative mapping, that is the largest value on the display should be displayed with the 'highest' color, all other values are mapped onto the interval 0 .. largest value. This Auto mode utilizes the entire color mapping. However, the mapping will change depending on the currently largest value on the display. This is an appropriate mode for histograms to display the relative distribution of data.

Palette Settings

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. Four palettes are available, Linear, Grey-scale, Inverted 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.

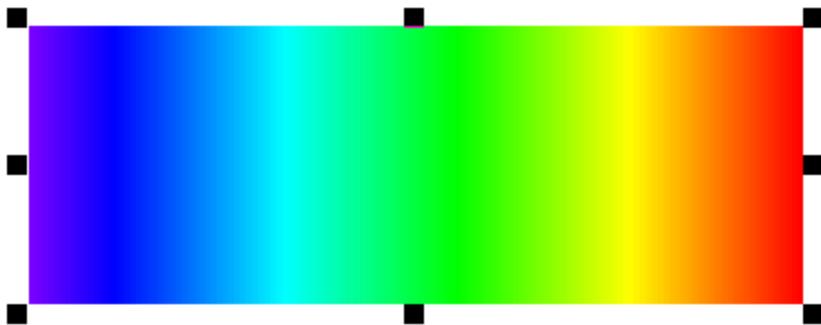
Linear palette:



Grey-scale palette:



Log palette:



Sizing Settings

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 may be appropriate for larger bitmaps. The variable mode allows for an arbitrary sizing but might lead to distortions. In Fixed Sizing mode a zoom factor can be selected that will be from 25% to 400% of the original image size.

DMA Mode

Bitmaps have a potentially large amount of pixels (array element items). So performance is critical. If your data block is layed out in a specific format GSEOS can use a very efficient

access mechanism (direct memory access) which will speed up the display by several orders of magnitude. It will be automatically enabled when the following conditions are met:

- The data item is an array of 8-bit items with no gap and byte aligned.
- The scaling is Fixed Scaling and the y-range is 0..255.

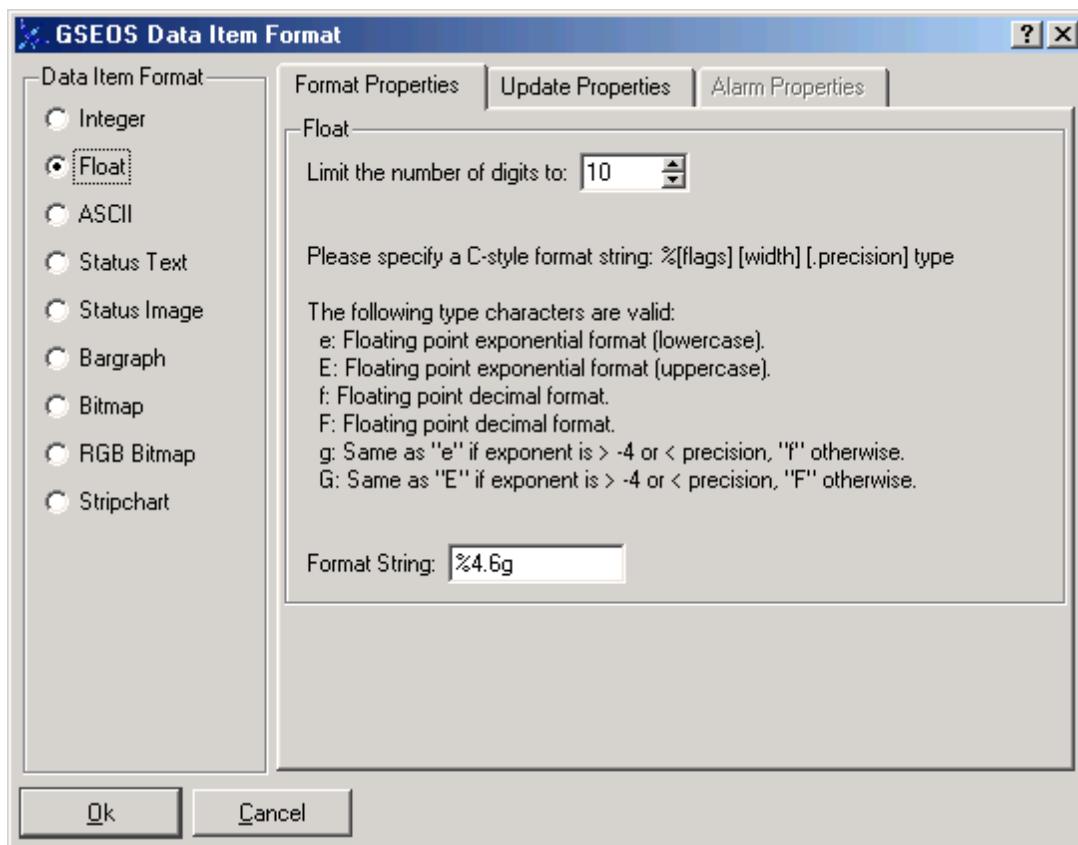
An indicator in the DMA Mode group notifies you if DMA mode is active or inactive.

7.4.3.2.6.6 Float

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

284.33	468.675
315.575	196.843
528.04	643.647
28.1205	37.494
637.398	781.125
162.474	268.707
518.667	381.189
106.233	428.056
459.301	390.563
734.257	221.839

Since GSEOS does not assign data types usually you want to use a formula as the data source. If your instrument generates IEEE float values you will need to use one of the conversion functions from GseosConversion to interpret the raw bit pattern as a floating point number. 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 format dialog box for the Float display looks as follows:



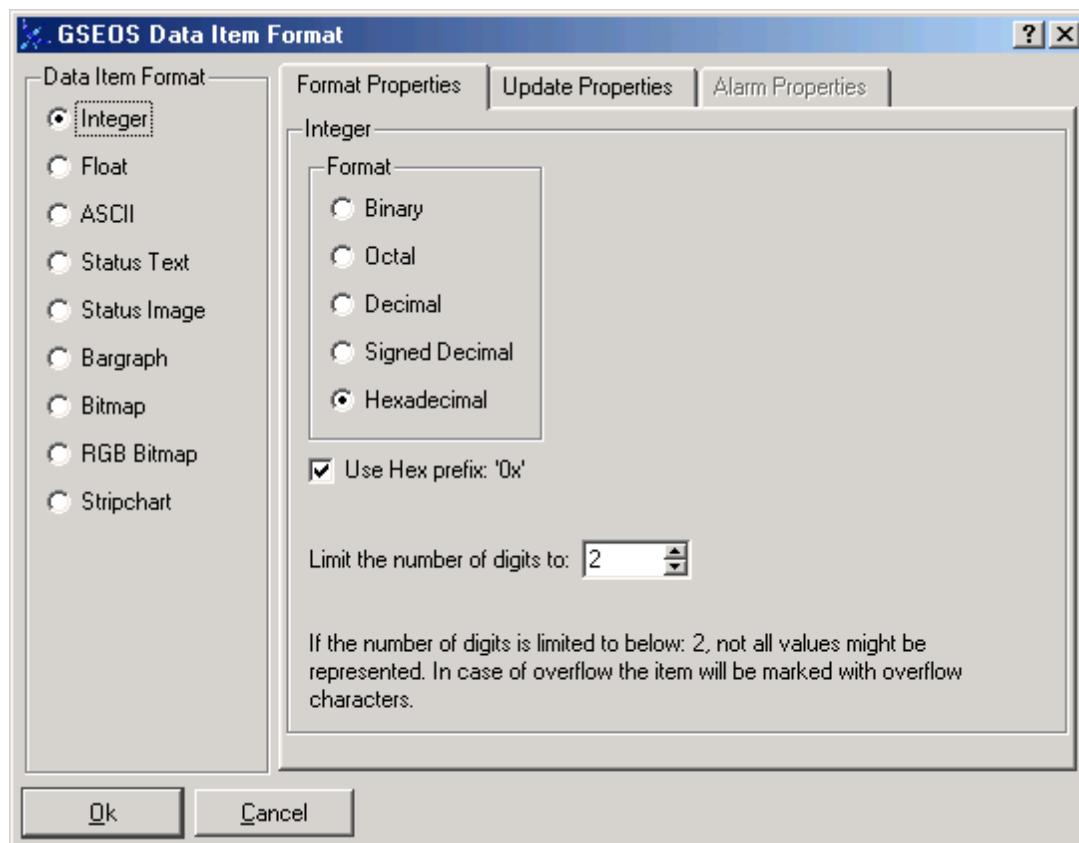
The limit of digits sets the number of digits to display per item. This does not include the leading negative sign. If the item can not be displayed within this limit the display will show as hash characters #####. The default value of the limit corresponds to the bit length of the item selected. The 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 dtor() or ltor() functions before using the float style.

7.4.3.2.6.7 Integer

The Data Item format Integer displays a data items in numerical integer format. Various options can be set for the integer properties. The picture below shows the format dialog for integer:



Besides selecting the radix of the integer you can optionally display a hex prefix if you display the integer to the base 16 (hexadecimal).

The limit of digits sets the number of digits to display per item. If the item cannot 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 and the radix selected.

The formats Binary, Octal, and Hexadecimal are left padded with zeros. Decimal and Signed Decimal formats are left padded with spaces.

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

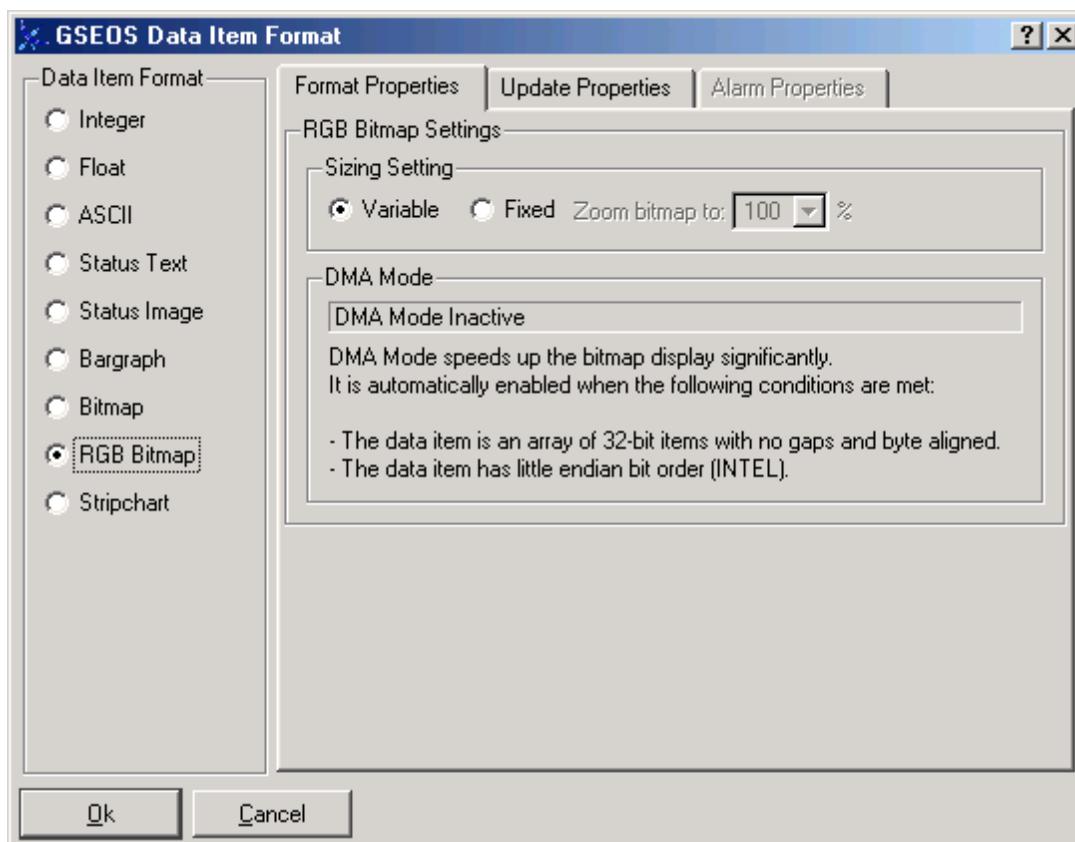
■	-88	85	99	84	97	105	109	■
	98	82	84	115	116	100	89	
■	97	84	81	117	111	109	60	■
	96	95	91	101	54	64	104	
	70	97	76	125	99	-107	98	
	92	88	108	112	85			
■				■			■	

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.

7.4.3.2.6.8 RGB Bitmap

The RGB Bitmap display is similar to the Bitmap display in that it displays the data as a bitmap. However, compared to the Bitmap display the RGB Bitmap doesn't need a color palette to look up a color for a data item value. The RGB Bitmap interprets the data as three channels of Red, Green, and Blue intensities with 8-bit each. For 32-bit items the MSB byte is not used but skipped. Only 32-bit items allow DMA mode, see below.

The Data Item format dialog box for RGB 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](#)^[62] 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](#)^[63] setting). The y-Range is used to map the data values onto colors as outlined in the following paragraph.

Sizing Settings

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 may be appropriate for larger bitmaps. The variable mode allows for an arbitrary sizing but might lead to distortions. In Fixed Sizing mode a zoom factor can be selected that will be from 25% to 400% of the original image size.

DMA Mode

RGB Bitmaps have a potentially large amount of pixels (array element items). So performance is critical. If your data block is layed out in a specific format GSEOS can use a very efficient access mechanism (direct memory access) which will speed up the display by several orders of magnitude. It will be automatically enabled when the following conditions are met:

- The data item is an array of 32-bit items with no gap and 32-bit DWORD aligned.
- The data item has little endian (INTEL) bit order.

An indicator in the DMA Mode group notifies you if DMA mode is active or inactive.

7.4.3.2.6.9 Status Image

The Data Item format Status Image displays a data item as an image looked up in a reference file.

This is similar to the [Status Text](#) format where a text string is displayed depending on the item value.

This display mode is suited for status data to indicate the status graphically. The lookup file has the extension .tr, the same as for text references.

The Data Item format dialog box for the Status Image display looks as follows:

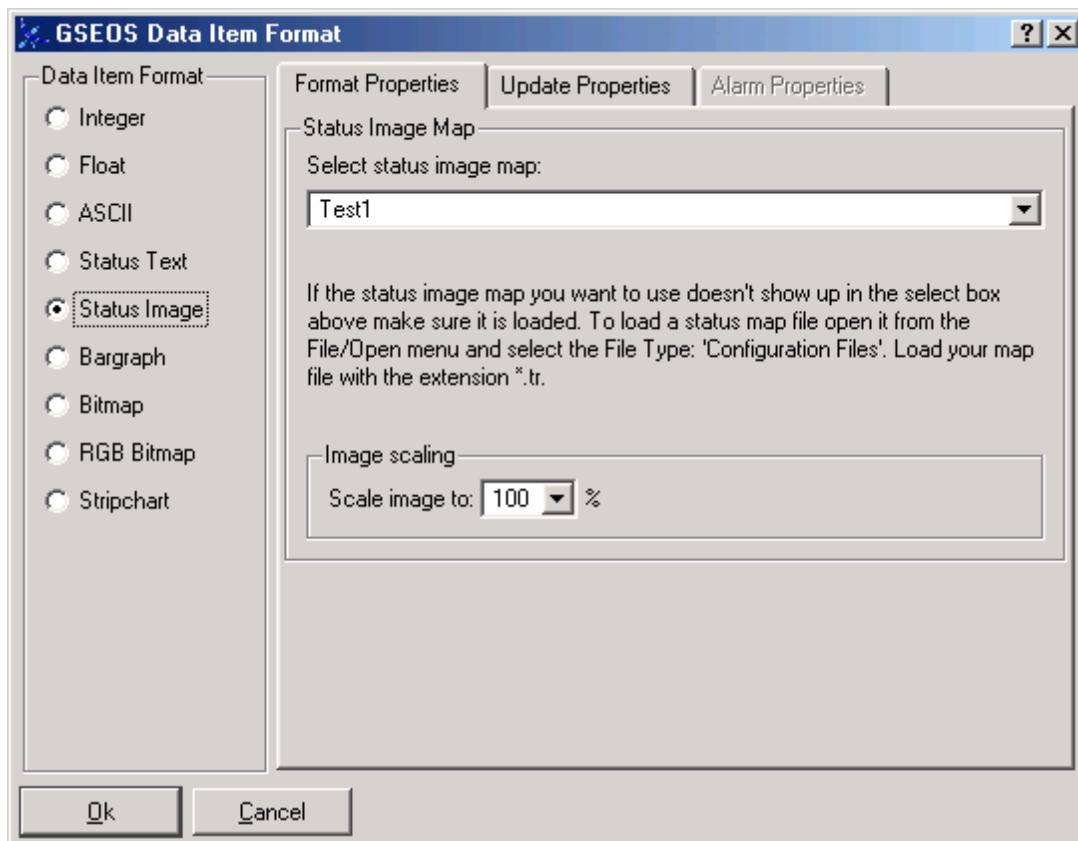


Image Scaling

If you use images of varying sizes they will be scaled to the size of the largest image in the collection.

You can specify a scaling factor for the images which will be applied in addition to scale the images to the largest image in the collection. Otherwise the status images can not be resized.

Examples:

```
Image: Name="Image1" File="Image1.jpg"  
Image: Name="Image2" File="Image2.jpg"  
Image: Name="Image3" File="Image3.jpg"
```

```
ImageMap: BootApp {  
    0, Image1;  
    1, Image2;  
    2, Image3;  
}
```

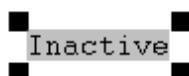
Note

Make sure to load the text reference files you want to use in your screens. You should add the text reference file you use for your screens to the gseos.ini [[Config](#)] [179] section with a Load entry.

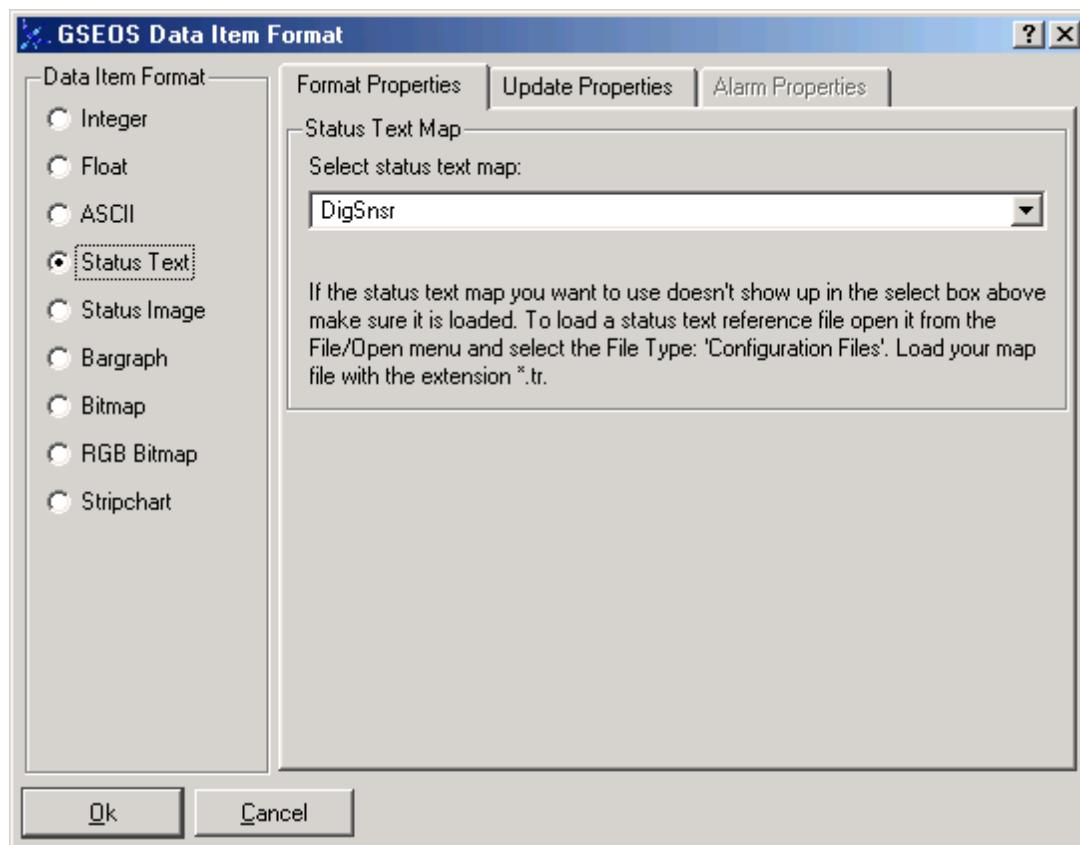
7.4.3.2.6.10 Status Text

The Data Item format Status Text displays a data item as text looked up in a reference file. This is similar to the [Status Image](#) [78] format where an image is displayed depending on the item value.

This display mode is suited for status data or table lookup for non linear data items. The lookup file has the extension .tr for text reference. The following example shows an item displayed in Status Text format.



The Data Item format dialog box for the Status 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", ForegroundColor on BackgroundColor
```

Examples:

```
ErrOk      { 0 ,          "Error", LightRed on White;
            1-65355 , "Ok",      Green on White }

Illegal     { 0 , "Legal",   Black on Yellow;
            1 , "Illegal", LightRed on Yellow }
```

```
EPU_Status { 0 , "Booting", Blue on White;
             1 , "Calc ...", Cyan on White;
             2 , "Running", Black on White }
```

For more details about text reference files refer to the chapter about [Text Reference Files \(*.tr\)](#)^[196].

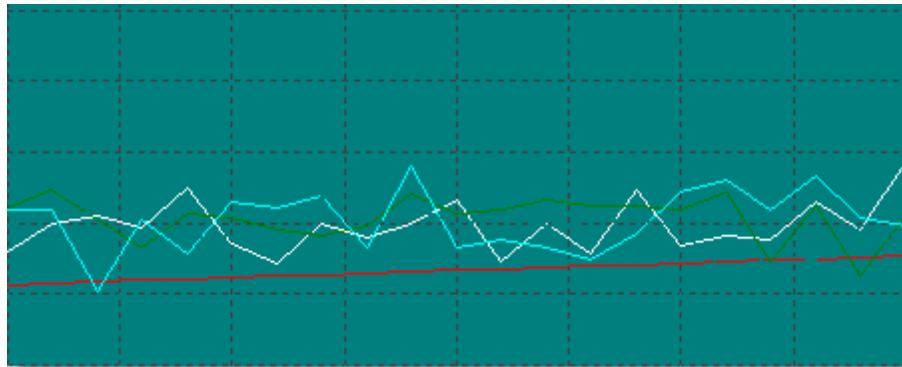
The reference name describes the name of the text reference and is selected in the dialog displayed above. The item definitions define which values get mapped to what text and in what color representation. You can either specify an individual value or a range that gets mapped to the display text. The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

Note

Make sure to load the text reference files you want to use in your screens. You should add the text reference file you use for your screens to the gseos.ini [\[Config\]](#)^[179] section with a Load entry.

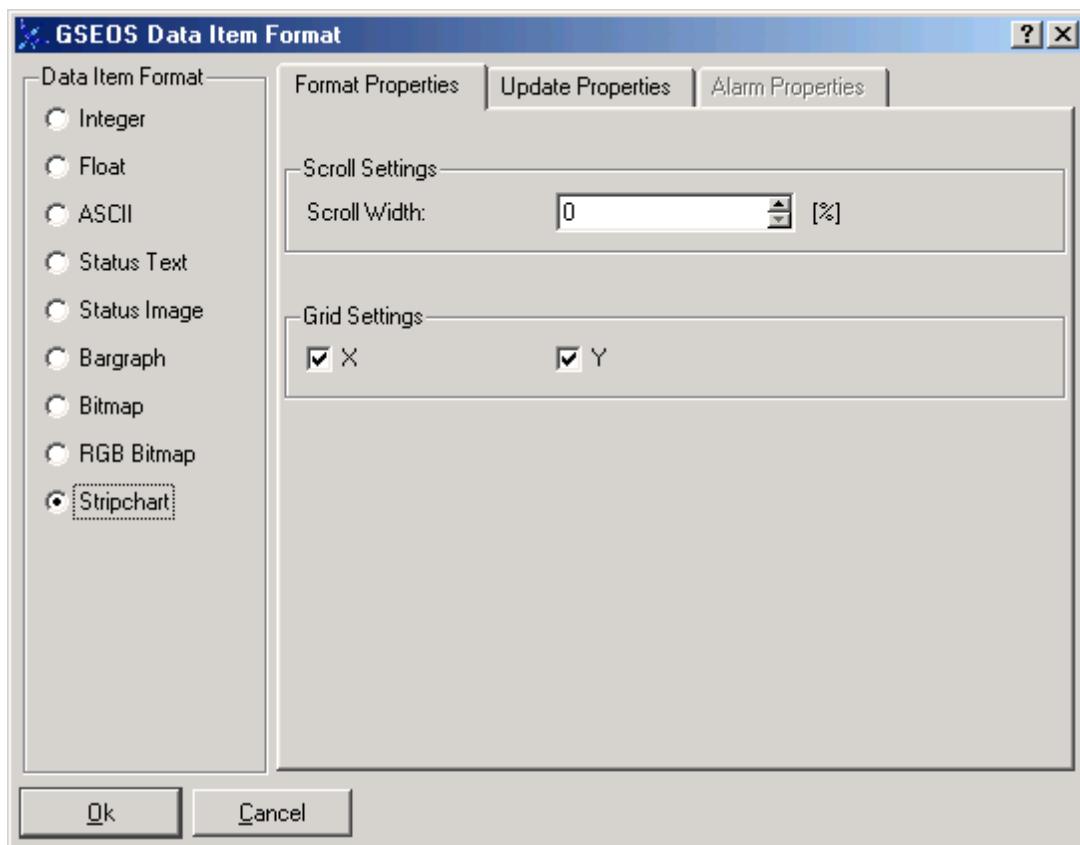
7.4.3.2.6.11 Stripchart

The Data Item format Stripchart displays a data item as a strip chart, that is the data value is displayed over time. The following picture illustrates a typical Stripchart display:



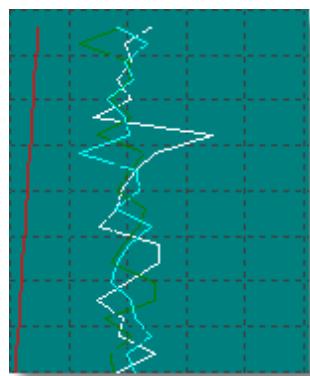
The Stripchart format can display up to five graphs. If you select an array item and choose to display five or more elements the first five elements will be charted in the color specified with the [color dialog](#)^[61].

The Data Item format dialog box for Stripchart displays looks as follows:

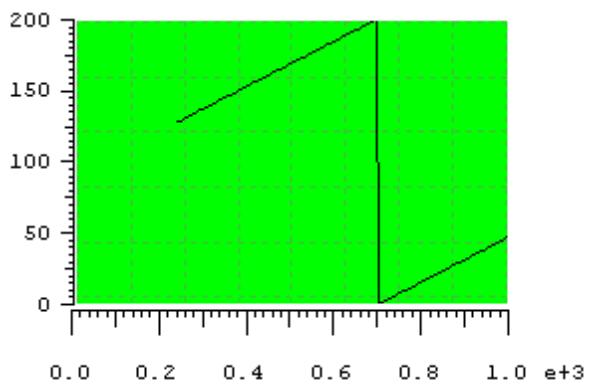


The scroll width determines how much of the x-range is displayed at least at any given time. Say you specify 50%, the display will scroll back 50% as soon as it reaches the end of the display. Scroll width 0% equals smooth scrolling but incurs the most overhead. If you have very fast updating data items 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. So the faster the update rate is the smaller you want to set the scroll width.

As with other graphical displays the [Orientation](#) dialog allows you to configure different orientations. The most appropriate for a Stripchart 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.

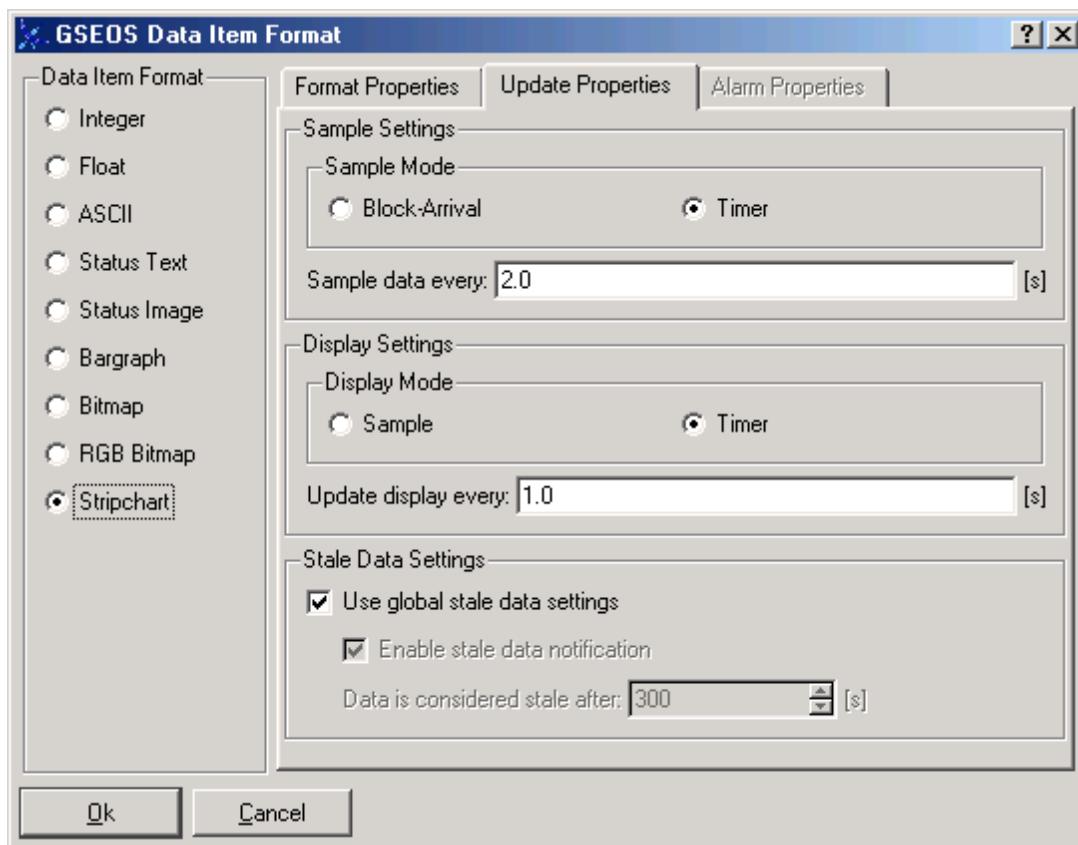


The check boxes for the x- and y-Grid let you turn on/off the respective grid. To make the display range visible a [scale](#) object can be attached to a Stripchart. This will result in a display similar to the one below.



Update Properties

The Stripchart display offers additional update properties. The following image shows the Stripchart Update Properties sheet.



A major difference of the Stripchart display compared to other display formats is that the Stripchart keeps state. So it is important how often we sample the data to get an accurate display. The sample mode offers two settings: Block-Arrival and Timer based.

There are basically two scenarios that we use a stripchart for: To monitor an item that is periodic and updates regularly and we know the time constant of the update period. In that case we would use the Block-Arrival mode. We can enter the time base in the edit field in the Sample Settings group.

This way you will get a correct time base. The x-dimension of the Range⁶² 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 'Sample data every' 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.

If you don't have a periodically updating item or you want to display an item over clock-time you would use the Timer based sampling and enter the sample time in the time base edit field. Say you set the sample time to 3s, this would result in sampling the data source every 3 seconds.

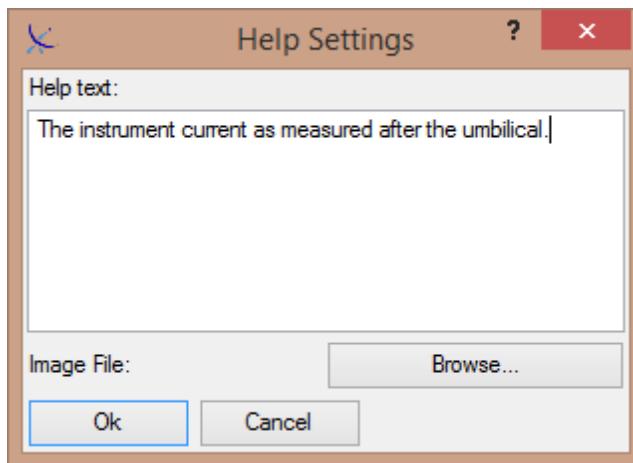
The settings above determine how often the data source gets sampled. This doesn't have to coincide with the update of the display. The Display Settings group allows you to configure the update rate. Since updates of the Stripchart can potentially be very expensive you can configure the sample timing and the update timing separately. Say you want to use the Block-Arrival sampling and your data item arrives with a 1KHz frequency

you might want to decide to use a timer based display update of 200ms. In that case you would choose Timer for the display update and 0.2 for the update period. You have the option of updating the stripchart with every sample if you choose Sample mode.

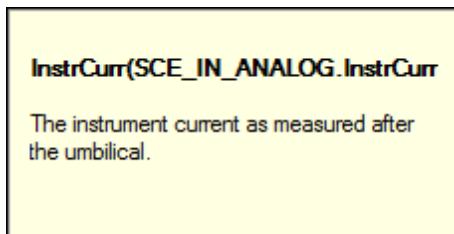
The stale settings are the same as for all other items.

7.4.3.2.7 Description

The Description menu lets you set a descriptive Help Text and/or Help Image. You can set descriptive help settings on any graphic item (not only data items but also say rectangles). The Help Settings dialog lets you specify a help text as well as an image. Both are optional.



When you enter Help Mode by clicking the Help Mode button on the toolbar and you click on the item the help text and/or image is displayed.



7.4.3.2.8 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.

The following image shows the Window format dialog:

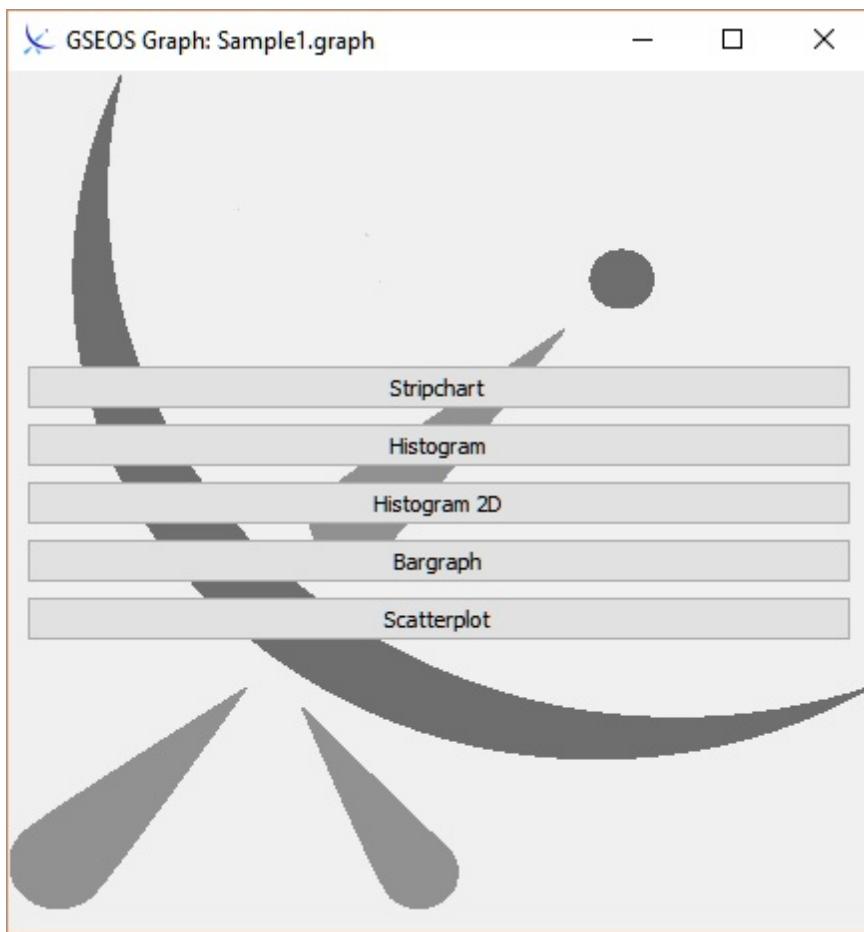


The window title is the text that gets displayed in the caption bar or the window (if any). If the title is blank the file name will be displayed. The Scrollbars check box allows you to turn scroll bars on or off. If you turn the scroll bars off while the screen is in a scrolled position you won't be able to display the top portion of the screen unless you turn the scroll bars back on and scroll the window accordingly.

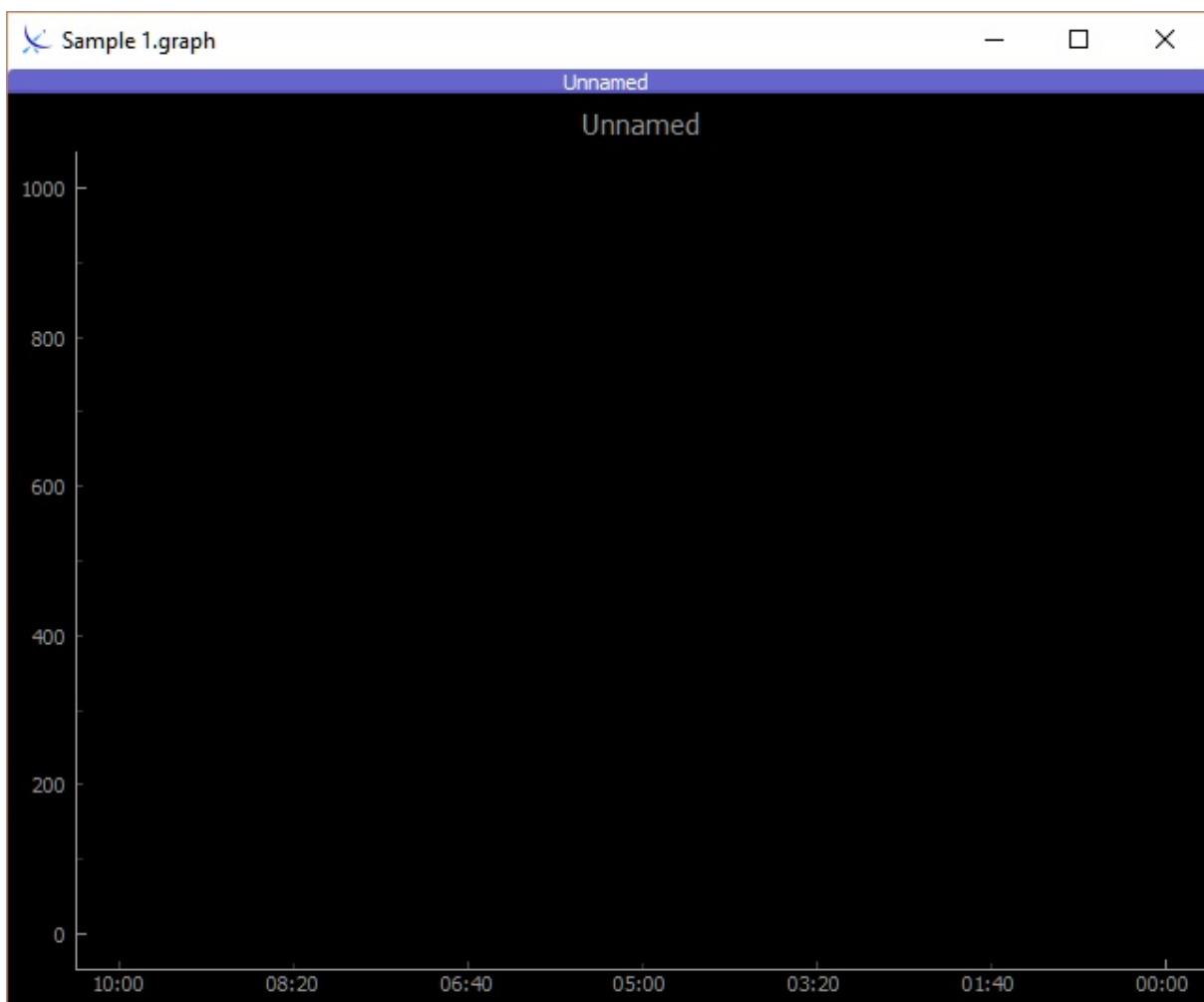
7.5 Graph Windows

The GseosGraph module is responsible for rendering your graphical displays like plots histograms, bargraphs, and scatterplots. It is run out of process to offload CPU cycles to a different core and also to make the system more modular. GSEOS communicates with the GseosGraph process over a TCP/IP connection on the local machine. The default port is 9999. You can [configure](#)^[182] a different port if that port is in use or not convenient.

A Graph window is represented by a graph file ending in *.graph. A Graph window can have one or more graphs. These are representations of graphical displays like plots. Each graph is displayed in a dock and you can arrange the graphs in the dock windows by resizing them and moving them around. You can even drag them on top of each other to generate a tabbed view. To create a new graph window choose File/New from the main menu. Then select the file type Graph Files (*.graph). This will open a new window with the chosen file name. Say we enter the file name: Sample 1.graph. The following Window will open up:

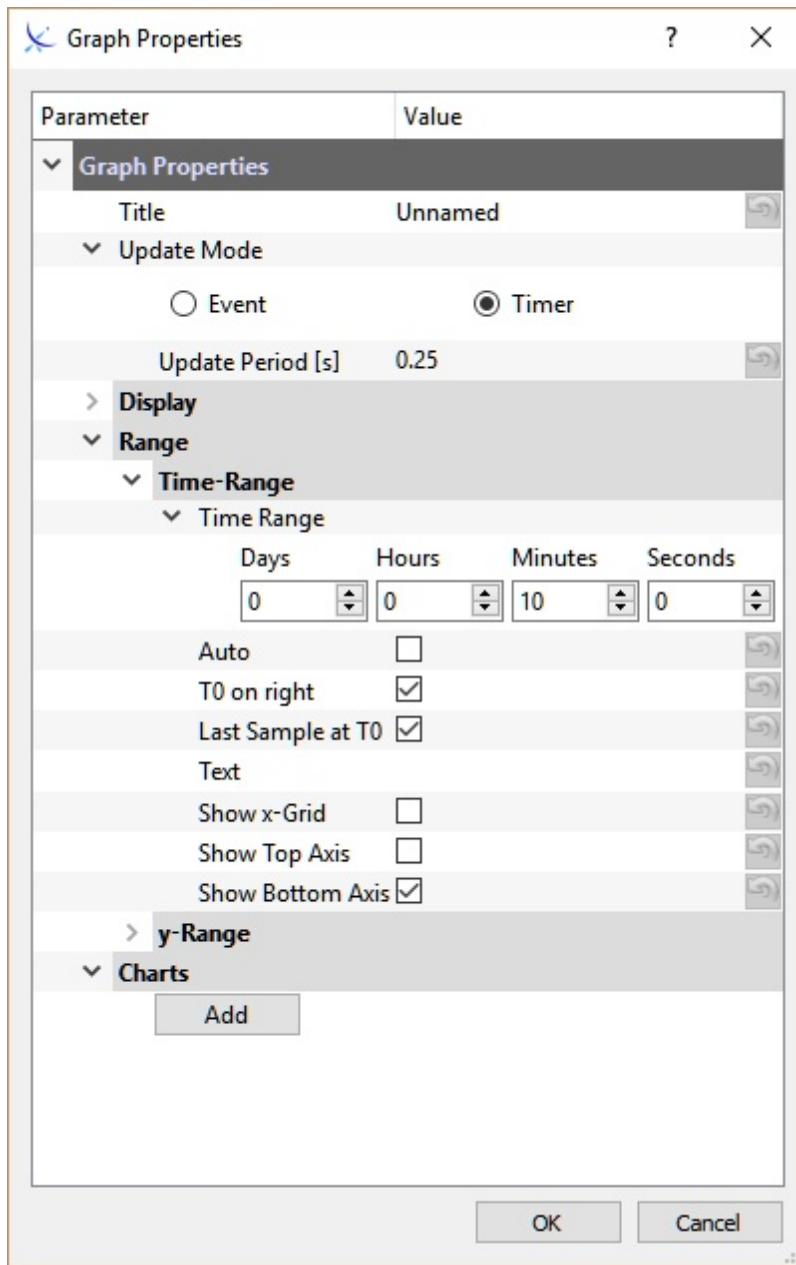


The Graph Select dialog displayed in the above image allows you to select the type of graph you want create. For this sample we choose Stripchart. This will open the following window:

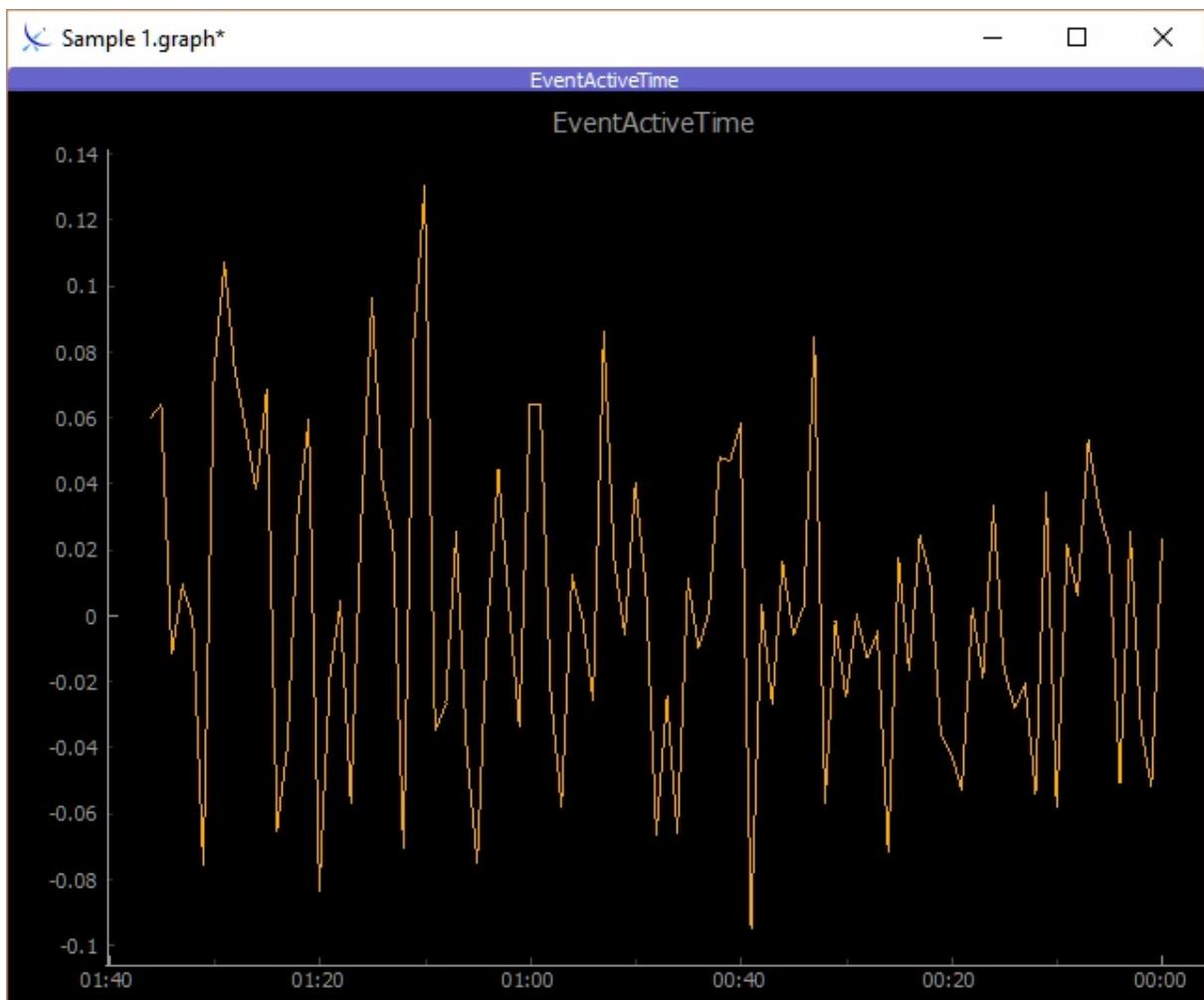


As you can see there is one dock window (the blue header on top) with one graph. A default name of Unnamed is assigned to the graph. We will change this shortly. Note how in Windows there will be a second tray item representing the Graph windows. All GSEOS Graph windows will be grouped under this tray item (if so configured in Windows).

Lets change the name of the graph to EventActiveTime, we will later place a chart for this item into the graph. To change any of the graph settings right click on the graph and choose Graph Properties:



We can now change the name of the graph with the Title setting. We will also add a chart to render one item in the graph. To do so we click the Add button under the Charts section. This lets us pick a data item. After we are done the item will be rendered as new samples arrive. Here is our first graph window after these settings:



The following chapters will go into more details of the various Graph/Plot/Chart settings.

7.5.1 Graph Concepts

The following sections address some basic concepts of the GseosGraph module and will make it easier to configure your graphs and charts within the graphs.

The chapter on [Layout](#) addresses how you can lay out your graphs and graph windows.

The following section gives you an overview of [display updates and data sampling](#).

Finally we talk about data [ranges](#) and the various ways to render your data.

There are five different types of graphs:

- [Stripcharts](#) [105]
- [1D Histograms](#) [110]
- [2D Histograms](#) [113]

[Bargraphs](#)¹¹⁶
[Scatterplots](#)¹¹⁷

7.5.1.1 Layout

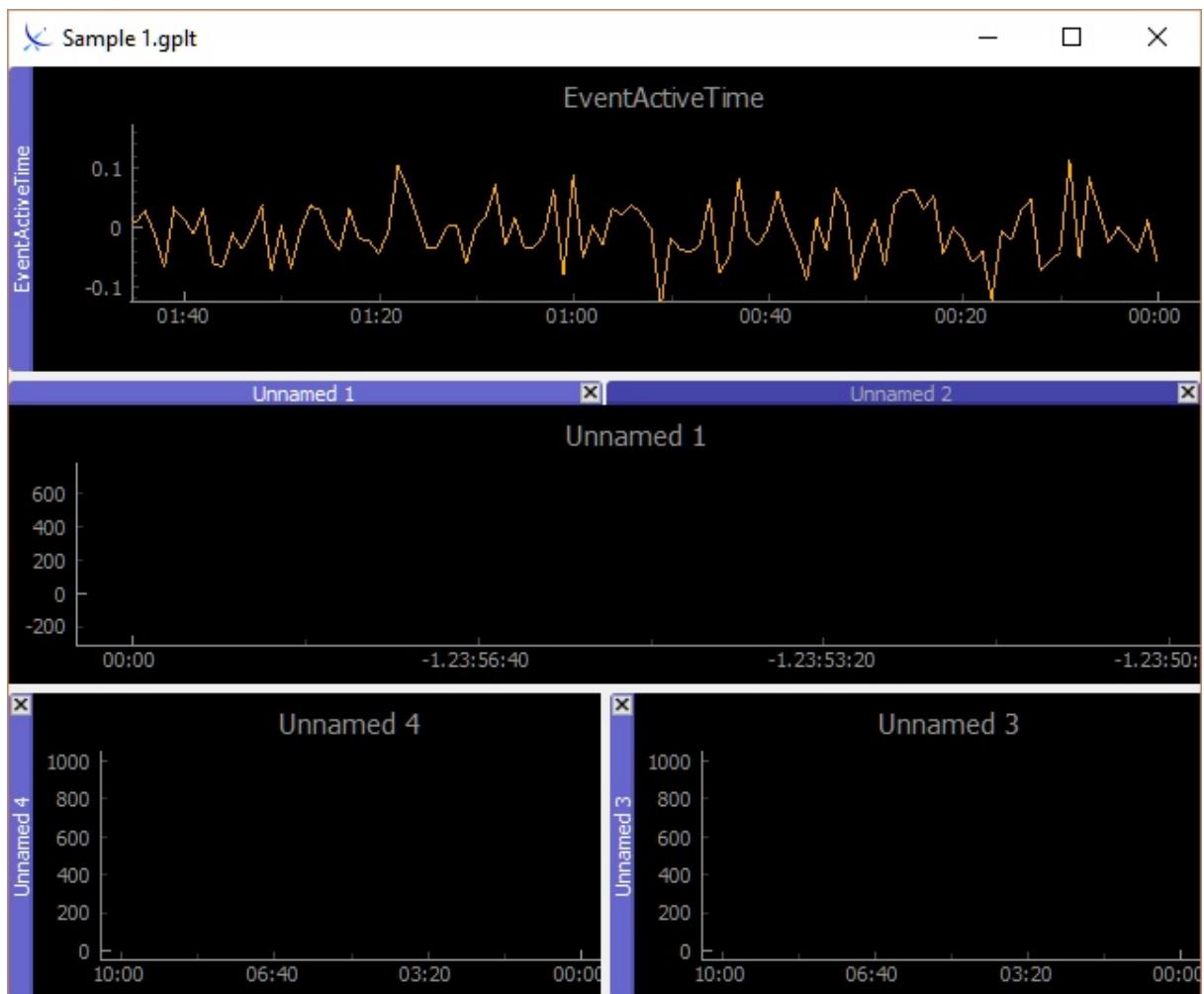
As mentioned earlier each graph is represented with a graph file. In the overview we created a new graph file named Sample1.graph. Since GSEOS communicates with the GseosGraph server the file name is used as a handle. Therefore a graph file can not be renamed. (If you need to rename a graph file you can do so with the file manager once you shut down GSEOS).

You can open an existing graph file with File/Open and create a new graph window with File/New and selecting the type **Graph Files (*.graph)**. Keep in mind that there can only be one window open for a specific graph file. So if you open an already open Graph window again the current window will be activated but no new Graph window created.

Each Graph window can contain one or more graphs. In the following sections we will demonstrate how to create and modify graphs of various types.

Each graph can contain one or more [charts](#)¹⁹⁹. Charts are the things that actually render your data items. This allows you to have multiple charts within one graph and multiple graphs within one Graph window. This can be useful if you want to compare data or correlate different items.

Each graph is contained in a dock. You can freely arrange the docks within the Graph window by moving them around with the mouse. If you drop them on top of each other you will create a tabbed dock. When you drag an existing graph by the dock area you will see a destination drop shade displayed. This allows you to arrange the graphs in various ways. The best is to simply try it. Here is a Graph window with numerous graphs:



You configure each graph with the [Graph Properties Dialog](#) [97] which you can open by right clicking on the according graph.

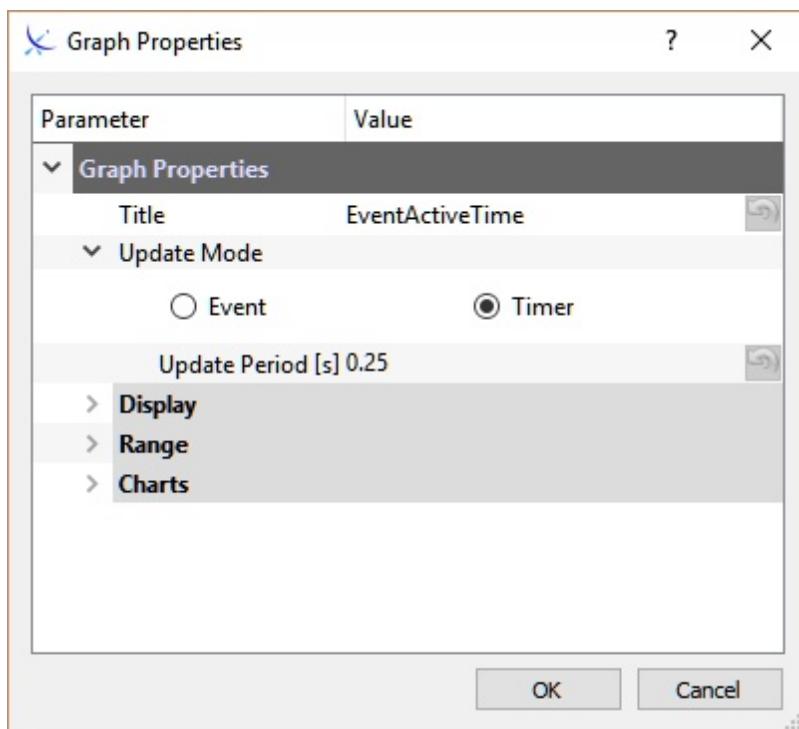
While you can resize and move the various graphs within a Graph window this does not mark a Graph window as 'changed'. This is to avoid asking you to save the file every time you simply resize one of the graphs. If you like you can right click on any graph within the Graph window and select **Save All**. This will save the changes to the underlying Graph file. However, even if you don't do so, the graph layout in the Graph window is saved in the desktop file. So when the Graph window is restored the next time you launch GSEOS your graph layout settings will be re-applied.

Each graph can have one or more charts. You add new charts with the [Graph Properties Dialog](#) [97]. Each graph type has a slightly different properties dialog depending on the display style, however, many options are similar or the same between the different styles. Each chart can then in turn be configured with the same dialog. The section on [Chart Properties](#) [98] gives the details on the configuration options.

7.5.1.2 Update and Sampling

The update settings determine when the chart is actually rendered. Since rendering a chart can be quite expensive in terms of CPU cycles it is important to configure your charts properly.

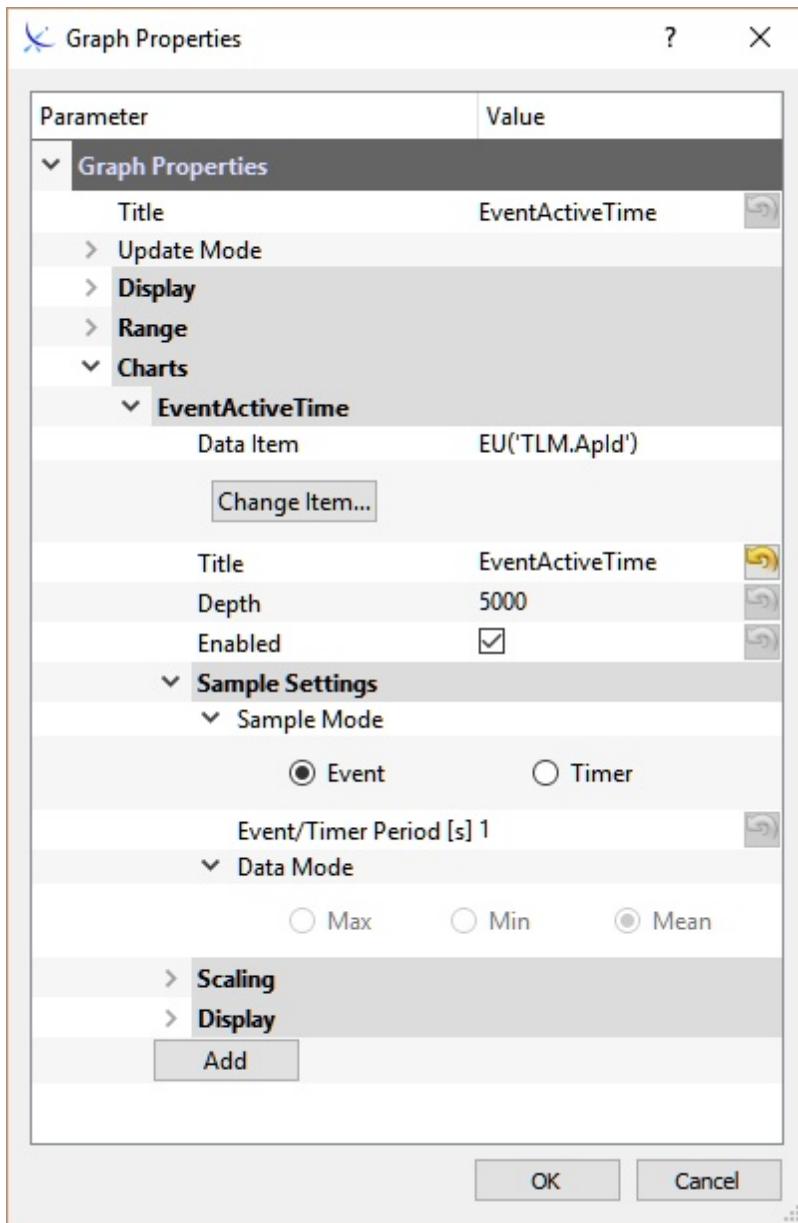
The update settings are applied on graph level, that means each graph renders all its charts given its update settings. The following image shows the properties dialog with the graph Update Mode:



You can choose between Event and Timer. In the case of Event each chart will be updated as soon as a new sample for the particular underlying data item arrives. While this gives the fastest updates it is also the most expensive option in terms of performance cost. This is especially important if you have very 'deep' charts that display thousands or tens of thousands of points. The other display option is Timer mode. In this mode the display is rendered whenever the timer expires. However, if no new data has arrived within the update period the data is not rendered again. Depending on the update period your display will be rendered more or less often, reducing the overhead with larger update times.

Note:

These settings only control when the display gets updated and are not related how the data is sampled. This can be controlled on chart level. Lets take a look at the chart configuration:



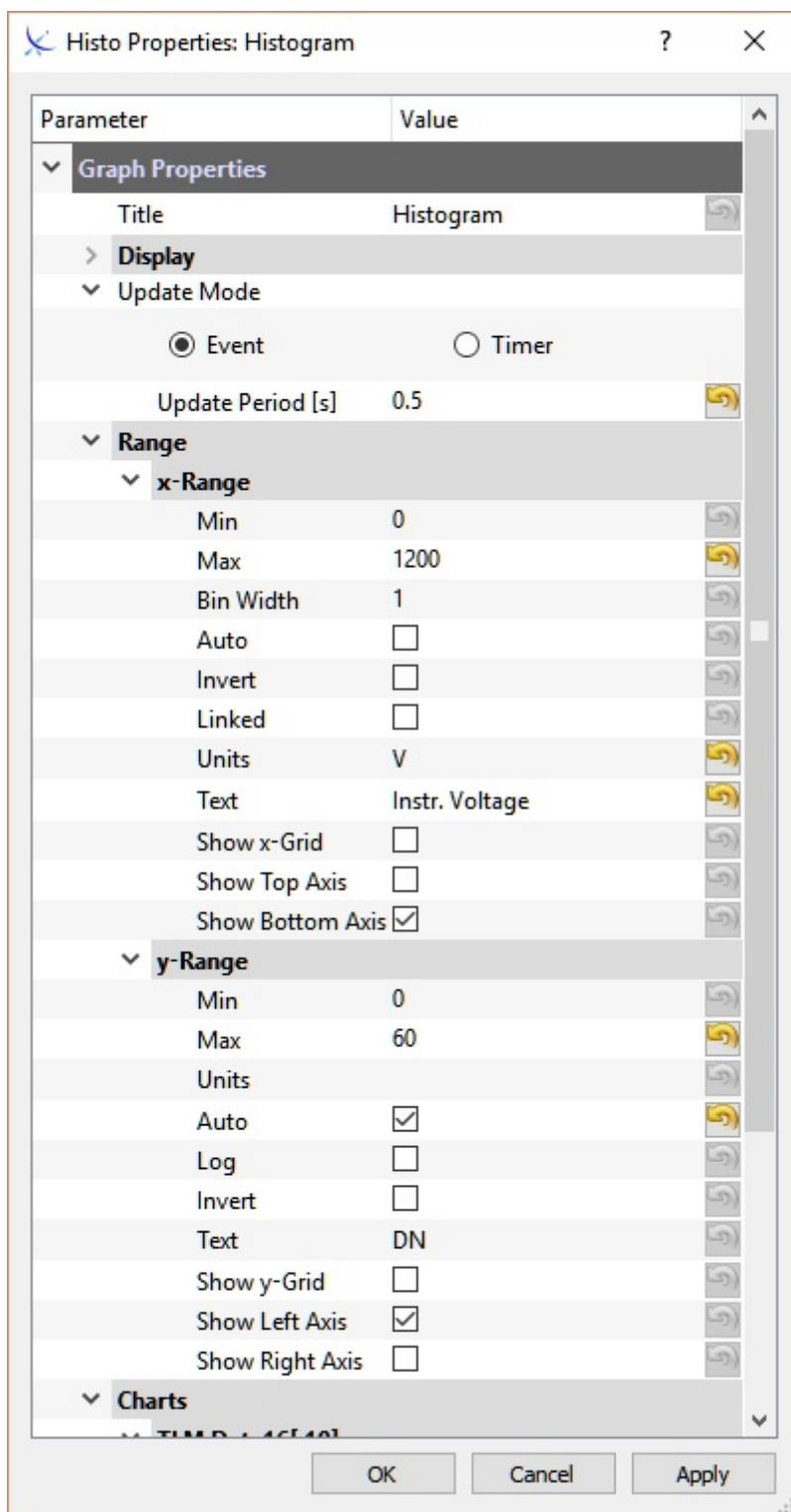
The above screenshot shows the Sample Settings for the EventActiveTime chart that we set up earlier. Each chart can be configured to use Event based or Timer based sampling. In the case of Event based sampling the chart is updated with each new data sample. This will gather all data points. In this case the Event period determines how much the x-axis (i.e. the time base for stripcharts or the x-dimension for other charts) advances. We assume that the data arrives with a specific periodicity and the Event Period should represent this interval. Say you want to chart a Housekeeping item that is updated and downlinked once a minute. You would set the Event Period to 60s so the time base is updated by 60 seconds every time this event arrives. While Event based sampling will collect all samples it has the drawback that in the case of very frequent items the display length can be limited. Say you have an update rate of 1KHZ there will be 1000 new points every second. After an hour you have 3.6 million points!

An alternative sample mode is Timer based. In this case we update the chart every time

the timer expires, which is determined by the Timer Period. Now, we have three different scenarios: It could be that zero, one, or more than one samples arrive during the timer timeout period. If no new samples arrive we simply push the last point into the chart, if one sample arrives we use that as our new point, however, if there are more than one sample during the timer period we have to somehow determine how to combine these into a single point. The Data Mode does accomplishes that. In Max mode we choose the largest value, in Min mode the smallest, and in Mean mode we average all samples. While in Timer mode we might lose data due to multiple points arriving within the Timer period we also can control how many points will accumulate over time so we can potentially chart a high frequency item for a longer period.

7.5.1.3 Range

The graph range is configured on graph level. The range is applied depending on the graph, some graphs have two axes (and therefore two ranges), 2D histograms also have a z-range. Since stripcharts are time based the x-range is represented as a time base and can be configured as such. See the following image for a sample:



x-Range

The x-Range lets you configure the range displayed on the x-axis. The charts are rendered into this range after any scaling (if configured) has been applied to the data.

If you invert the x-range it will render right to left while non-invert (normal) mode renders left to right.

You also have the option to 'link' several graphs. Any graph in your Graph Window that has the x-range Linked flag set is linked with the other graphs. Say you zoom in or pan in one graph the others will change the displayed x-range accordingly. To unlink a graph from other linked graphs simply uncheck the link flag of that graph.

y-Range

The y-Range typically represents the value of your data (for the 2D histogram the value of your data is encoded as a color). As for the x-range the effective value rendered is the one after the scaling has been applied.

While you can define a 'View window' here you can simply change this at runtime by zooming in or out of the graph. You can also choose Auto mode in which case the range will be automatically adjusted to show the entire chart(s).

You can also specify units with the Y-Range. We suggest to use the following units: m,s,g,W,J,V,A,F,T,Hz,Ohm,S,N,C,px,b,B

For these units we dimension specifies like u for micro, m for milli, k for kilo and so on will be added automatically. So don't specify say mA for your units but A. The [chart gain setting](#)^[99] lets you adjust your chart if you actually have mA as your engineering units.

The Invert setting lets you flip the Y-Range upside down.

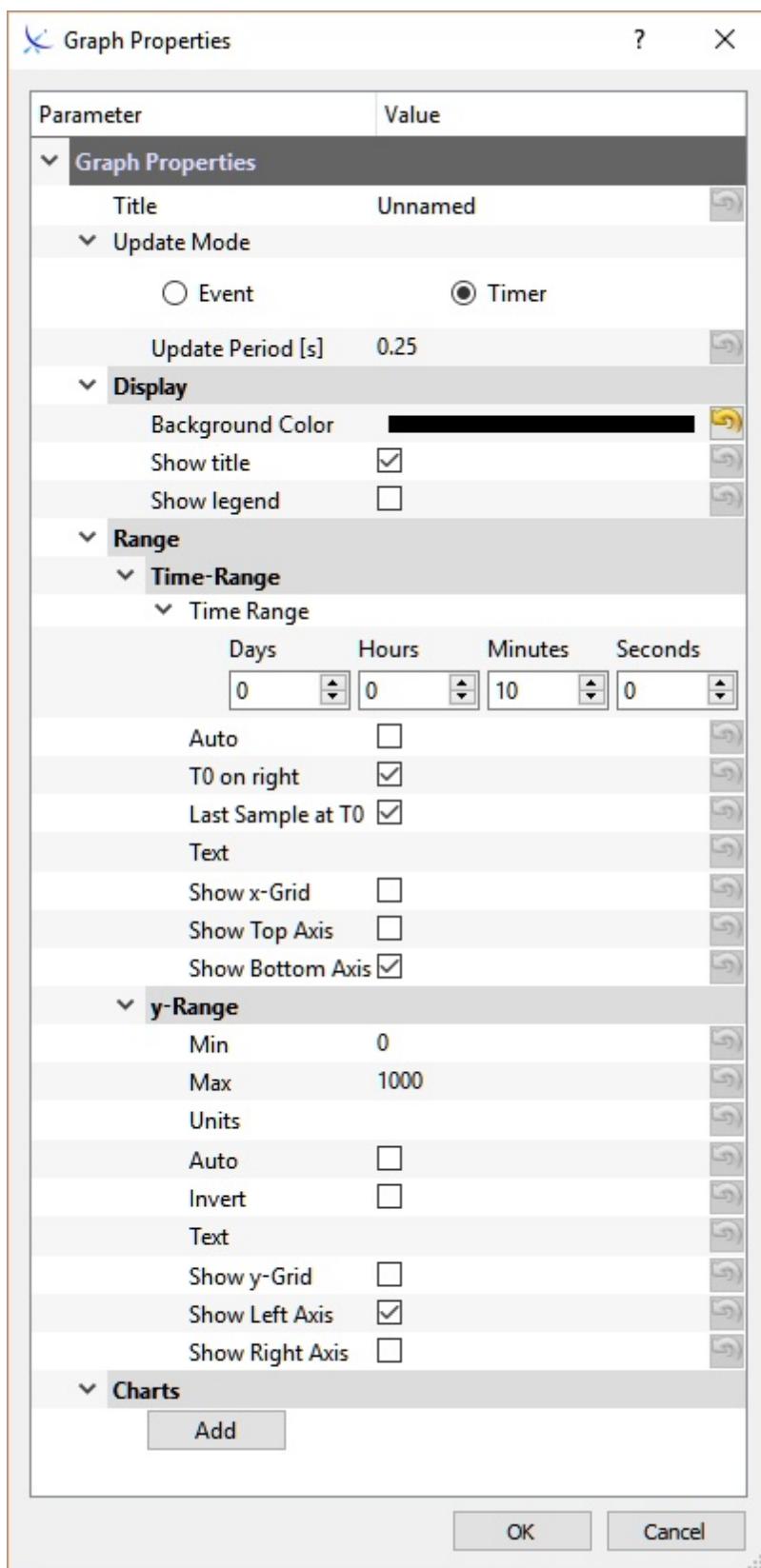
Log

Depending on the graph you can also set a log scale. Keep in mind that the log function is only defined for positive values. So if you have a range starting at 0 you will get an error message saying that your range is invalid. Please set your range accordingly if you plan on using a log scale.

7.5.2 Graph Properties

The Plot Graph Properties dialog allows you to set all plot and chart specific settings. The following will explain each of the available settings.

The following image shows all plot specific options:



The **Title** property lets you give the plot a name. Each plot needs to have a unique name. If not specified and you create a new plot it will get assigned the name: Unnamed[N] where N is a counting number. So the first thing to do after creating a new plot is to set a unique name (the name only needs to be unique within the Graph window).

The **Update Mode** [93] determines how the charts are updated. There are two modes: Timer and Event. In Event mode each time a new sample arrives from GSEOS the chart is updated, this is the fastest way to update but also the most expensive in terms of performance. In Timer mode the charts are updated whenever the timer expires. The timeout period can be set with the Update Period property. If there has no data arrived within the timeout period the plot is not refreshed.

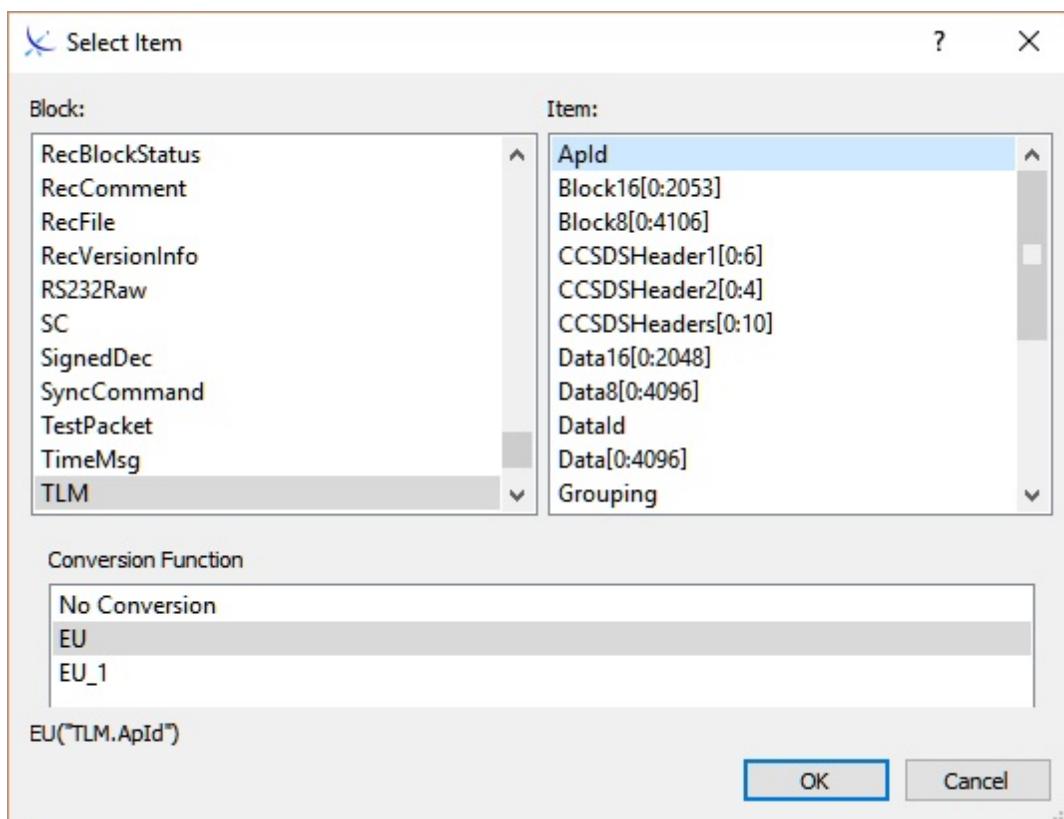
The **Display** setting for the plot allow you to set the background color, as well as turn the legend on/off and to display or hide the plot title.

The **Range** [95] settings allow you to configure the Time-Range as well as the Y-Range. The section on [Range](#) [95] gives you some more information regarding some of these settings. You can set a label text for both the x (Time) axis as well as the y-axis. You can also turn on/off a grid for either axis. Finally you can enable/disable either axis. The x-axis can be shown on top or bottom or both, the y-axis on left or right or both. In case of the y-axis, units will only be shown on the left axis. If the axis is scaled due to unit scaling (i.e. A -> mA) only the left axis will be scaled the right axis is not affected by unit scaling.

All the above setting affect the entire graph. The next chapter will discuss the individual chart settings in more detail. To add a chart you can click the Add button under the [Charts](#) [99] section.

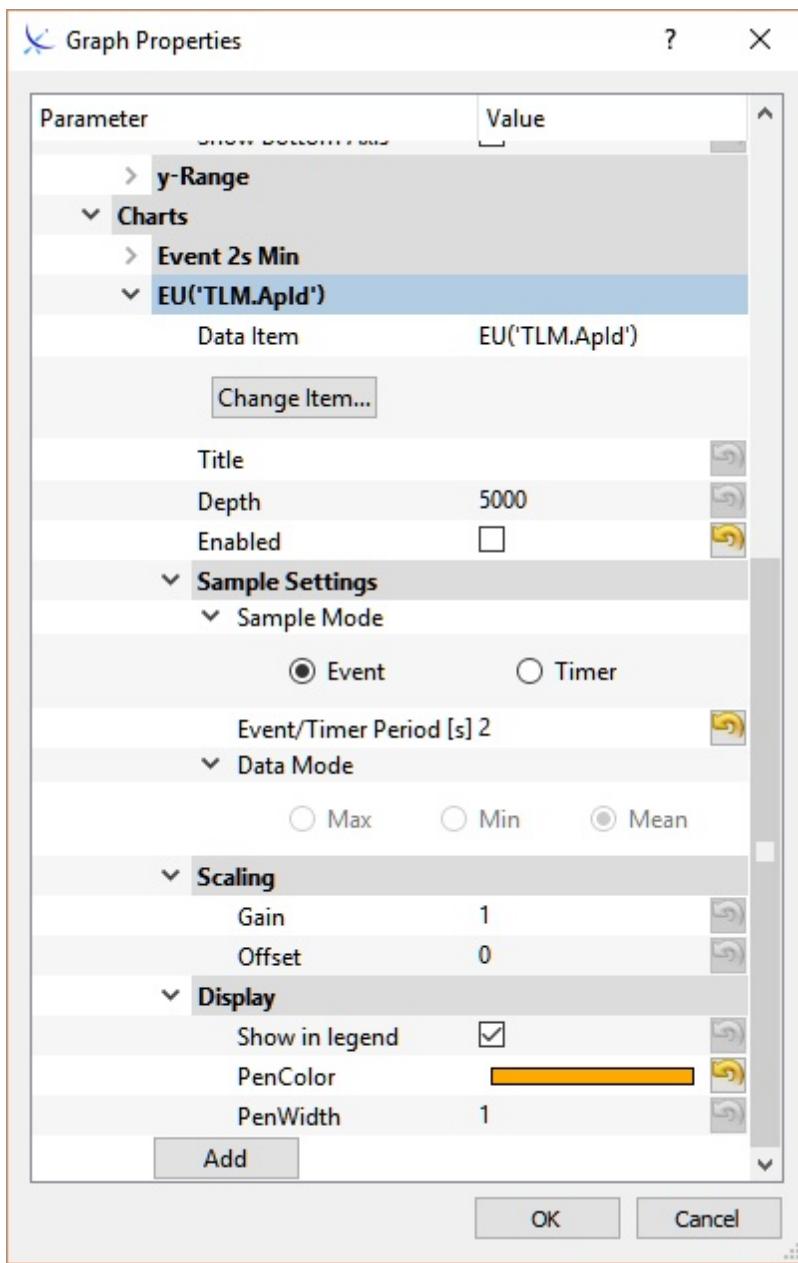
7.5.3 Chart Properties

Each plot can have one or more charts. You add new charts by clicking the Add button on the bottom of the Charts section in the Plot Properties dialog. This will pop up a data item selection dialog that lets you choose the data item you want to chart:



You can only select scalar items. However, you can pick a single value from an array item. If the item has conversion functions defined as in the above example you can choose the conversion function you want to apply to the item.

You can now configure the individual properties for the new chart. The properties dialog will have a new entry under the Chart section and look similar to the image below:



As you can see the name of the chart is the name of the data item we selected, in this case the item has a conversion function: EU('TLM.ApId'). The Data Item selected is listed as the first entry. If you made a mistake or want to change the underlying data item you can click on the Change Item button and the Data Item Selection dialog will allow you to change the item again.

Chart Order

If you have multiple charts in a plot you can arrange the order by dragging the caption of the chart you want to move and dropping it on top of the chart you want to place before. The charts are rendered from bottom to top. So the top chart will be rendered on top of the second last chart and so on. This allows you to set preferences in case some of the charts might overlap at times. You can also remove a chart by right clicking the chart caption and selecting Remove.

Chart Properties

The **Title** property lets you assign a title to the chart. This is used instead of the data item name and is the name that is shown in the legend if enabled. If you clear the title the data item name is used again.

The **Depth** field allows you to determine the 'depth' of the chart or how much history you will be able to see. The larger you set this the further back you can see and the more memory and performance you will be using up. The section on [Range](#) gives some more details for a good setting.

The **Enabled** setting allows you to turn off this chart. Please keep in mind that this not only removes it from the display but also stops gathering data.

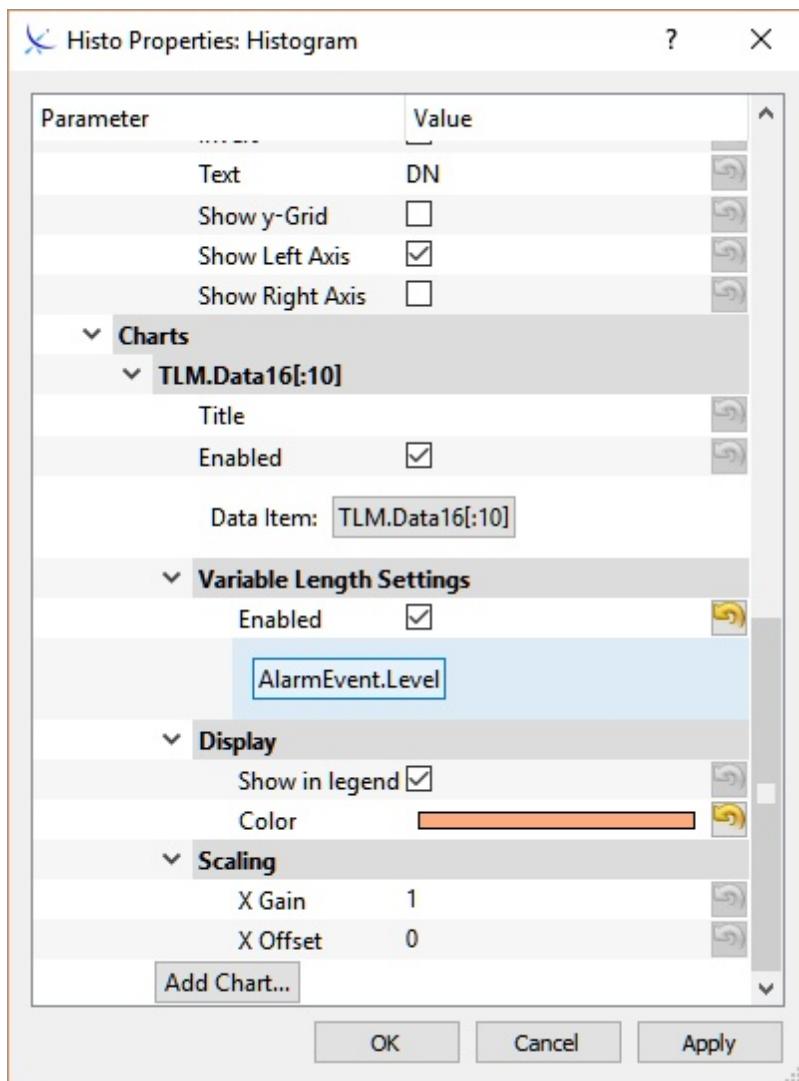
The **Sample Settings** let you configure how often data is sampled and how it is coalesced if in Timer mode. The section on [Update and Sampling](#) goes in more detail on this.

The **Scaling** section lets you set simple linear scaling options: Gain and Offset. This is useful if you want to use a SI unit like V or A but your data is actually in mA or mV. If you apply a gain of 1000 in that case you can set the units to V or A and will have proper unit scaling enabled. This can also be useful if you want to look in great detail at a value that is at a significant offset. Say you have a value of about 3000 but you are interested in movements between 3000.000001 and 3000.000002. You could apply an offset of -3000 and avoid having the scale read something like: 3000.000007, 3000.000008, 3000.000009, 3000.00001, 3000.000011 but 7, 8, 9, 10, 11 uV. Basically the scaling gets applied to the raw value (or conversion value equally) before it is being pushed into the chart.

The **Display** options allow you to select the chart color and pen width. You can also toggle the item in the legend on or off.

Variable Length Charts

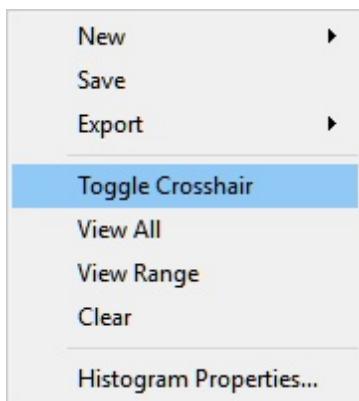
Some charts have a more dynamic nature. For example in a given histogram you might want to histogram an event array. However, the number of valid events might depend on another data item. You can configure this with the **Variable Length Settings** (if available for a graph). The image below shows a Chart properties dialog with variable length settings:



You can choose another data item that determines the number of elements valid in the given Chart data item. You also have the option to Enable/Disable variable length (while keeping the variable length item configured).

7.5.4 Context Menu

The context menu which you can open with a right click on any graph allows you to modify Graph Window or graph specific settings. Here a screenshot of the context menu of a histogram graph:



The New entry lets you add a new graph to the Graph Window. The Save option saves the current Graph Window to its underlying file. Export lets you export either the entire Graph Window or the specific graph to an image file.

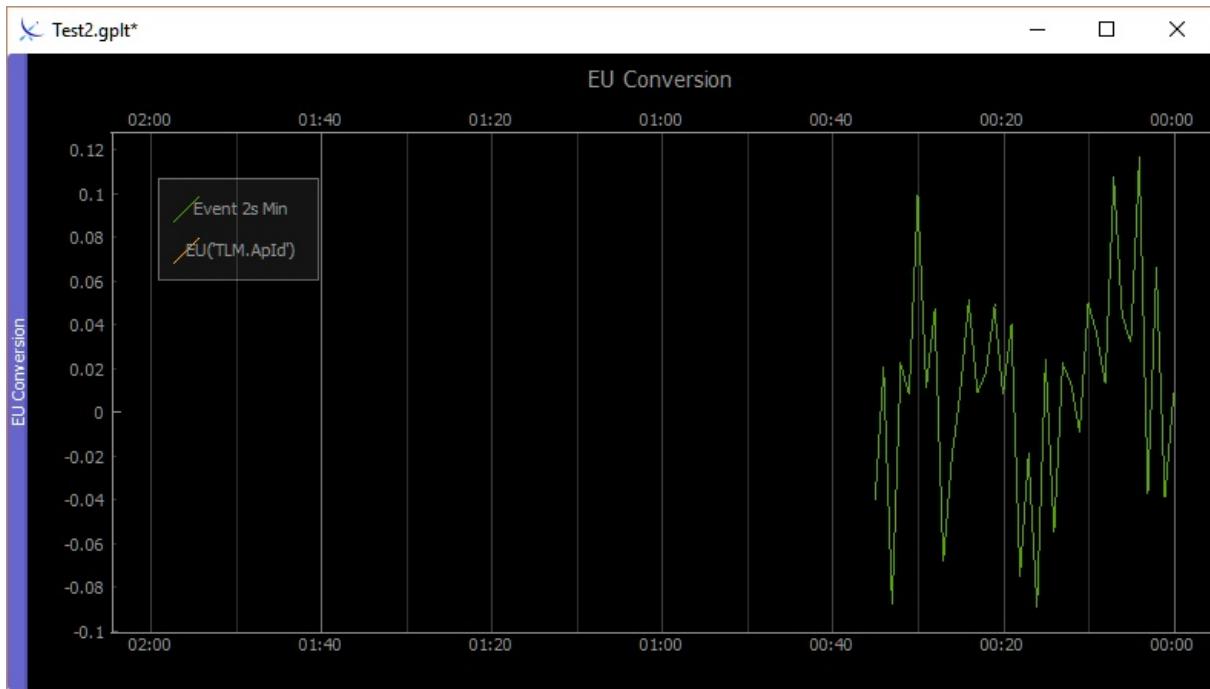
The Toggle Crosshair option lets you enter or exit crosshair mode. In this mode any screen updates are disabled and a crosshair with some coordinate information is displayed. This lets you examine the current graph without being disturbed by changing data. Once you leave crosshair mode the updates will resume.

The View All and View Range options let you switch to different view modes. In View All the range is adjusted so all your data is being displayed, the range won't update automatically unless your x-range or y-range has the Auto option set. The same is true for View Range, just that in this case your configured range is displayed.

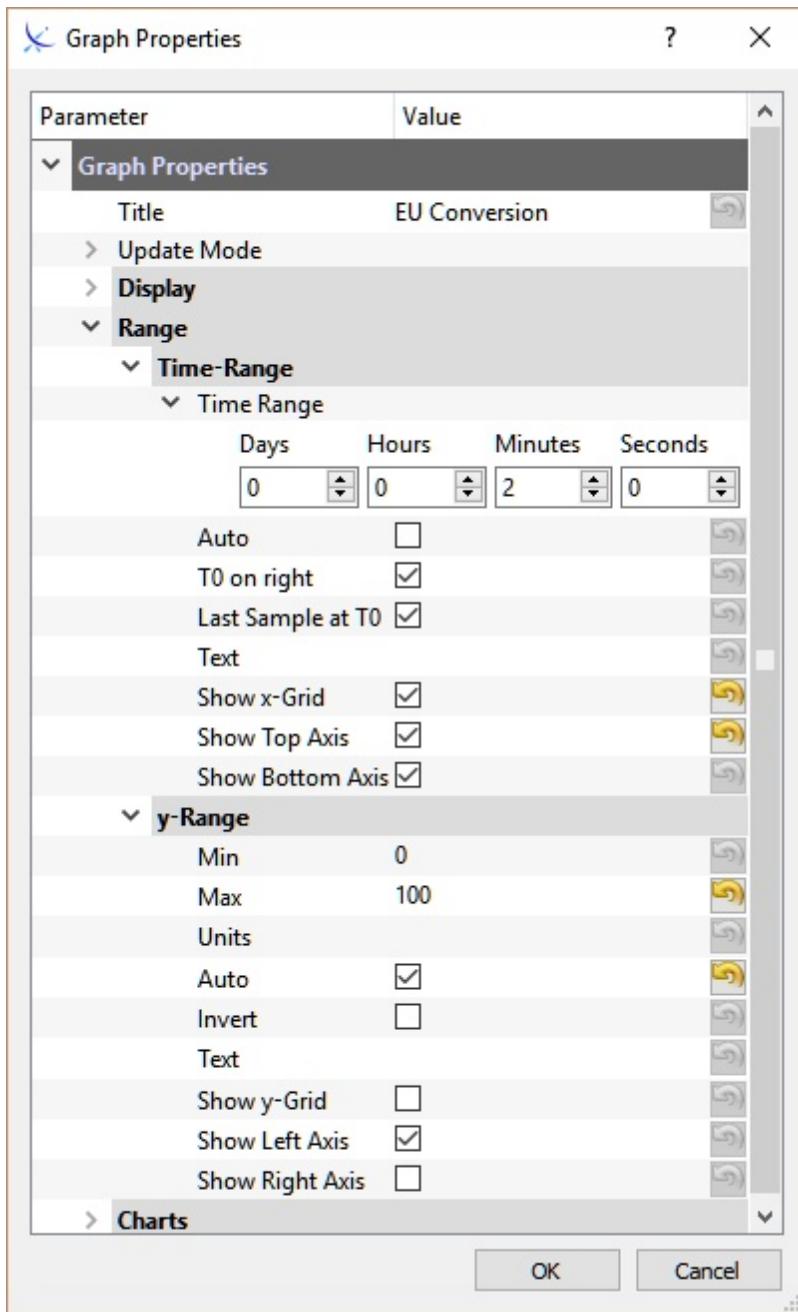
Keep in mind that you also can enter Auto mode by clicking on the A button in the lower left corner. This will turn on Auto mode in which the ranges are constantly adjusted to show all your data.

Some graphs like the histogram are 'integrating' displays, the Clear option lets you reset the data and start afresh. Finally, depending on the graph' you can open the [graph properties dialog](#) [97] which is specific to each graph.

7.5.5 Stripcharts



Stripcharts render the data points (charts) over time. So the x-range setting is actually a Time-Range setting. Below is the properties dialog for a Stripchart graph:

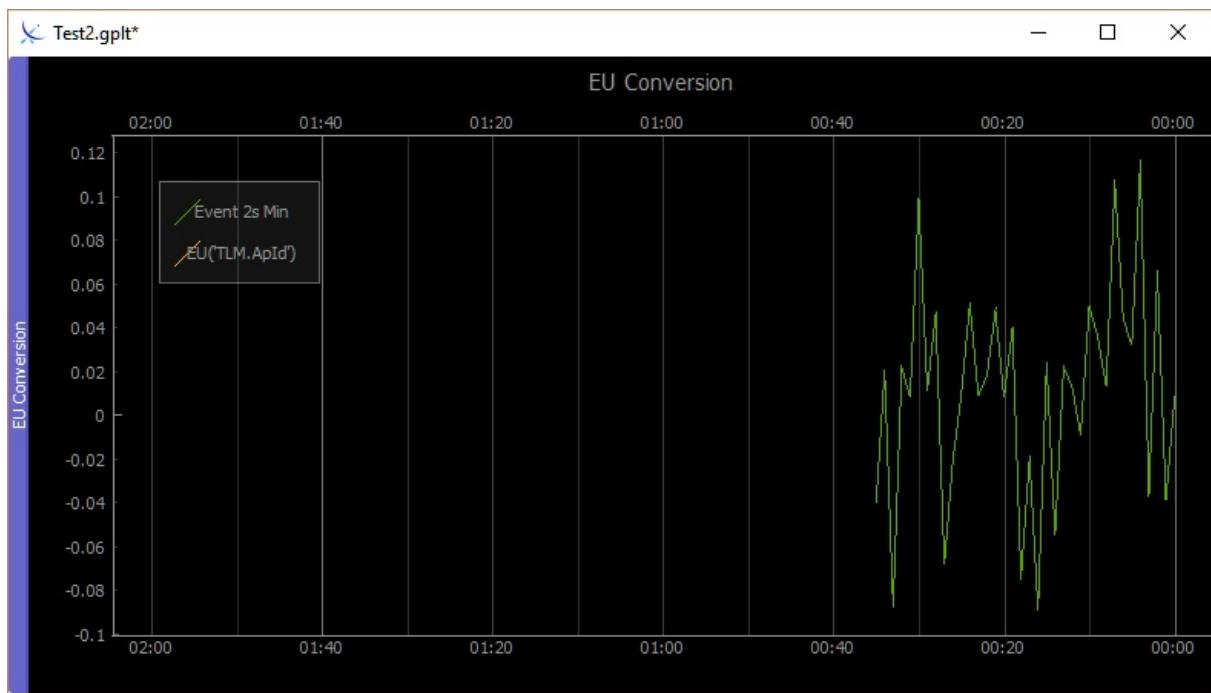


Time-Range

You can configure the Time range in Days, Hours, Minutes, and Seconds. While this determines the 'Time Window' you are looking at this 'Time Window' can be easily changed by zooming in or setting various different view modes. So this setting merely is an indication what your default time window should look like.

In Auto mode this will automatically be adjusted to the number of samples you have gathered and therefore will give you a complete view of your data.

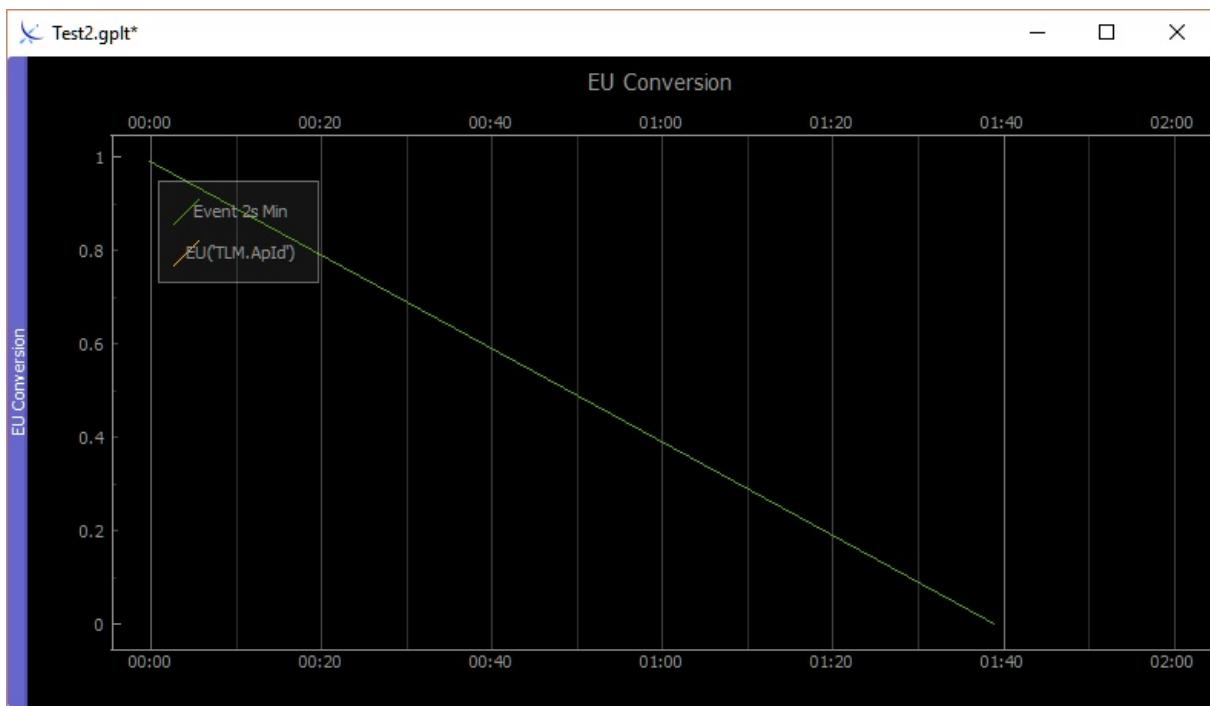
The next two settings: T0 on right, and Last Sample at T0 determine how your data is rendered. The default mode is that T0 (the current time) is on the right and older times move to the left. The following picture illustrates this:



The chart starts on the right at T0 (00:00) and older times are to the left. In the above chart there are about 35 seconds worth of data. The chart will move to the left with the most current point always at the right.

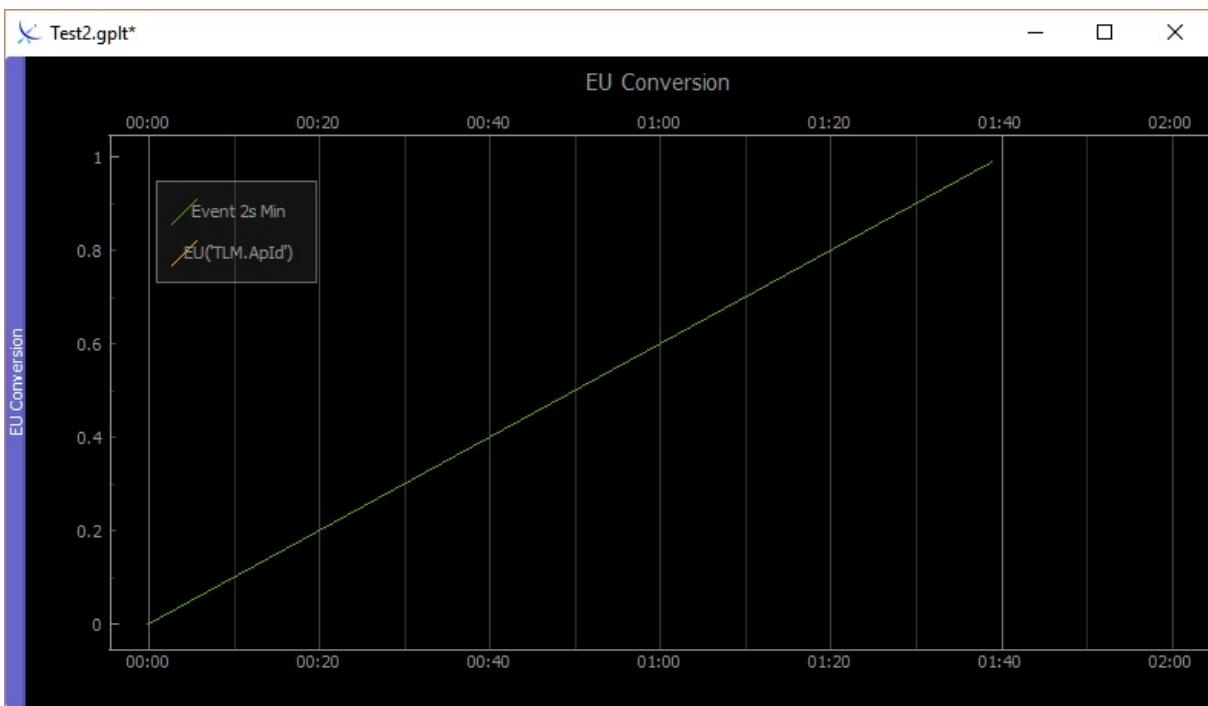
If you were to uncheck the T0 on right checkbox T0 would be on the left side and the chart (as well as time) would advance to the right. The latest sample will always be on the left.

There is a different mode that will render the oldest point at T0. Assume you place T0 to the left and plot a simple, the resulting chart would look something like this:



In this sample we plotted values from 0 through 1. The latest point (1.0) is on the left (T0). The oldest point at 1:40 (we plotted 100 values). So the chart advances to the right.

Now if you uncheck the 'Latest Sample at T0' the chart will render the oldest point at T0 (on the left in this case) and the newest point will move to the right. While this might at first seem more intuitive it does have a few disadvantages: When the chart moves on over time, it will eventually 'scroll out of your view'. This is something that you can avoid with auto update mode. The other drawback is that this mode is slightly less efficient. The following image show the same data when 'Latest Sample at T0' is unchecked:



Depth

In the case of a stripchart graph the Depth setting determines how much 'history' you can display. While you specify the number of points with this setting the time you can go back depends also on the sample settings. The larger you make this setting the more history you can see and the more memory and performance you will take up. This together with the [Sample Settings](#) is a tradeoff of how long of a history you want with memory and performance impact.

Say you set your chart to a depth of 100,000. That means that we allocate two arrays of 100,000 64-bit floats (one for the y-values and one for the x-values). For each new update we have to shift all items through the arrays and render the resulting data set. You can imagine that if we have to do this 100 times a second for a few dozen charts that this will be quite a performance impact. That was one of the reasons to run the plotting in its own process. With setting more conservative sampling and update settings you can mitigate this impact somewhat.

So you have to be careful how you set the Depth for your charts. While you can set the Time-Range to any period you want, ultimately you will be limited by the Depth you set in terms of how far back you can see.

Y-Range

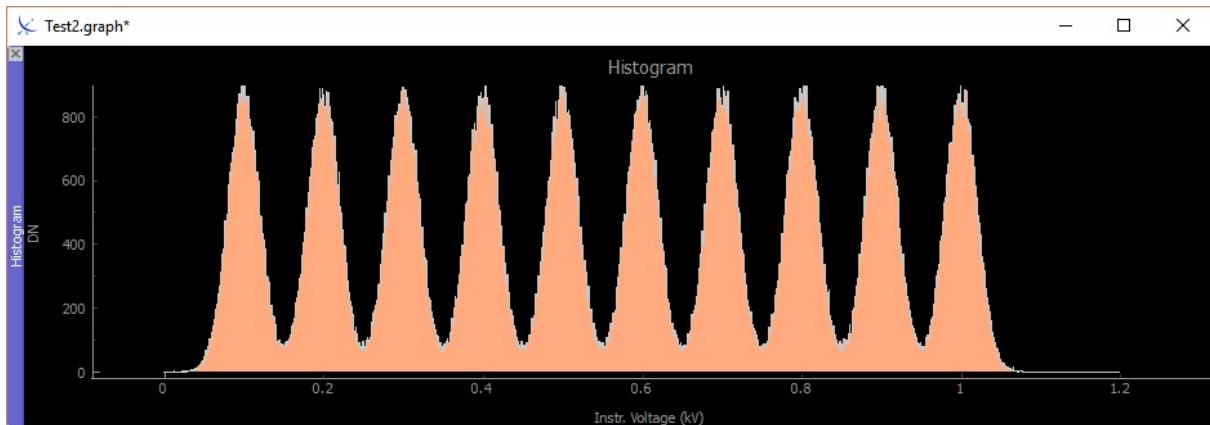
The Y-Range settings let you specify the range you want to see in respect to the y-values (your data values). The same applies as mentioned with the Time-Range. While you can define a 'View window' here you can simply change this at runtime by zooming in or out of the graph. You can also choose Auto mode in which case the range will be automatically adjusted to show the entire chart(s).

You can also specify units with the Y-Range. We suggest to use the following units: m,s,g,W,J,V,A,F,T,Hz,Ohm,S,N,C,px,b,B

For these units we dimension specifies like u for micro, m for milli, k for kilo and so on will

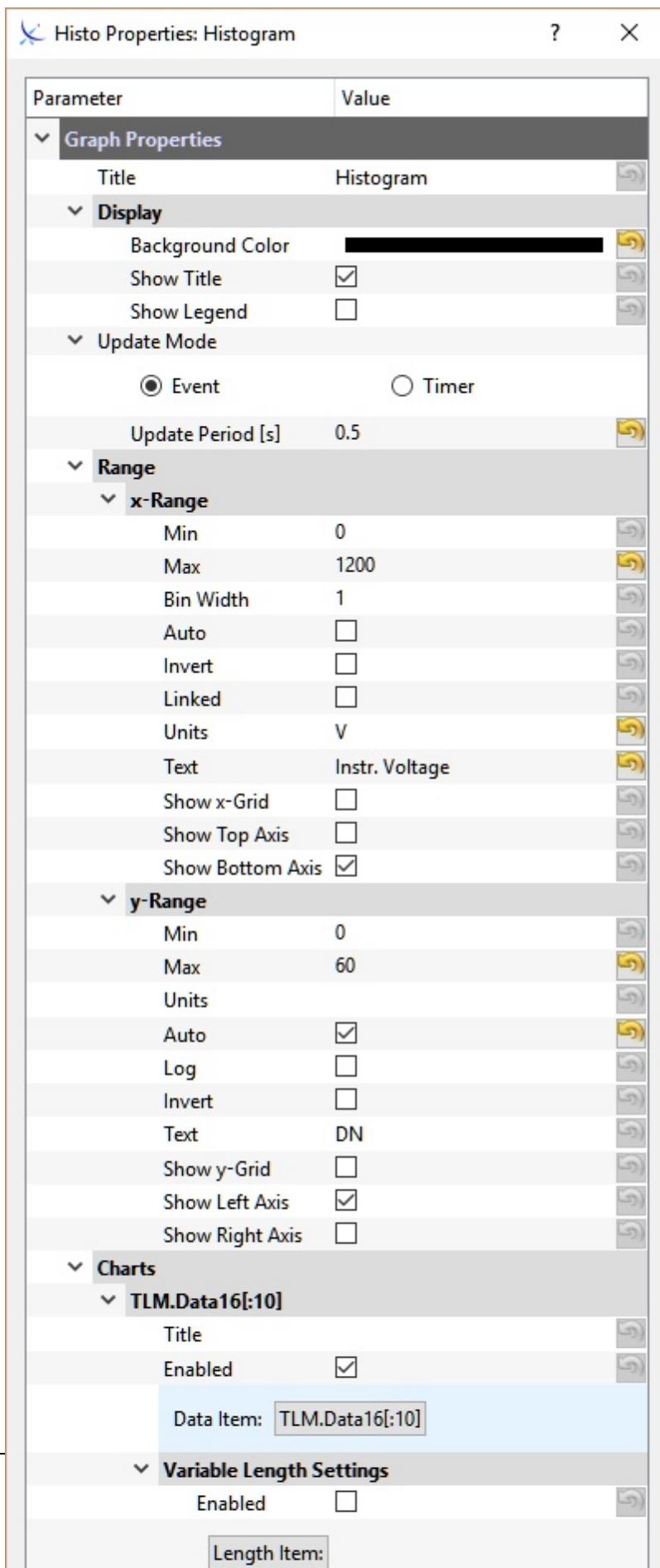
be added automatically. So don't specify say mA for your units but A. The [chart gain setting](#) lets you adjust your chart if you actually have mA as your engineering units.

7.5.6 1D Histogram



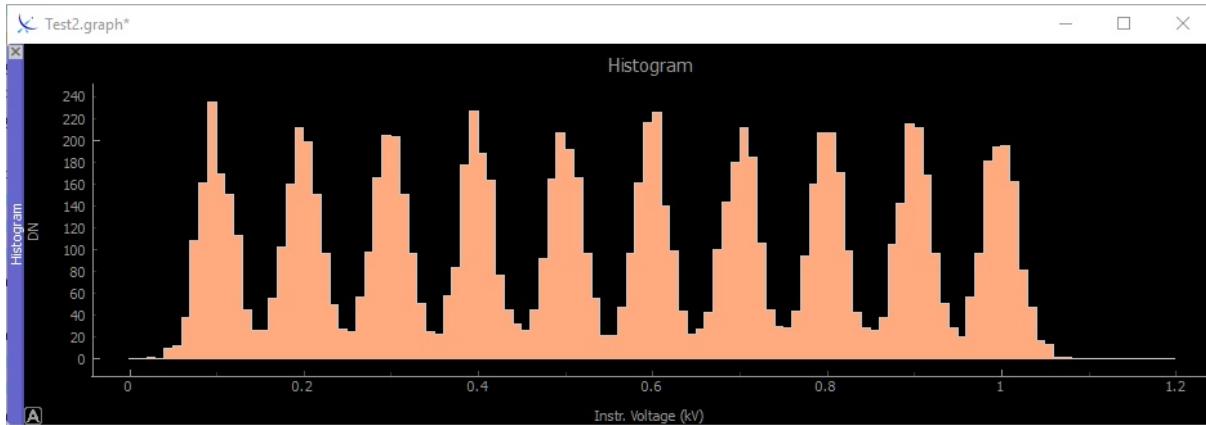
A 1D histogram lets you 'histogram' your data into bins. The 'histogrammed' data is then rendered in the graph. Note that this display integrates (or accumulates) data over time. So you can clear the histogram with the [context menu](#).

The x-axis represents the various 'bins' whereas the y-axis shows the accumulated number of 'hits' in that bin. Below a picture of the graph settings:



Many of these settings are common with the other [graph properties](#) and [chart properties](#). We'll point out the 1D histogram specific settings here.

The x-range has an additional setting of **Bin Width**. While this can be a fractional number we suggest to use integer values only. This setting determines your bin width and requires a clearing of the existing chart. While the display on top shows a histogram with a bin width of 1 the image below shows the same data with a bin width of 10.

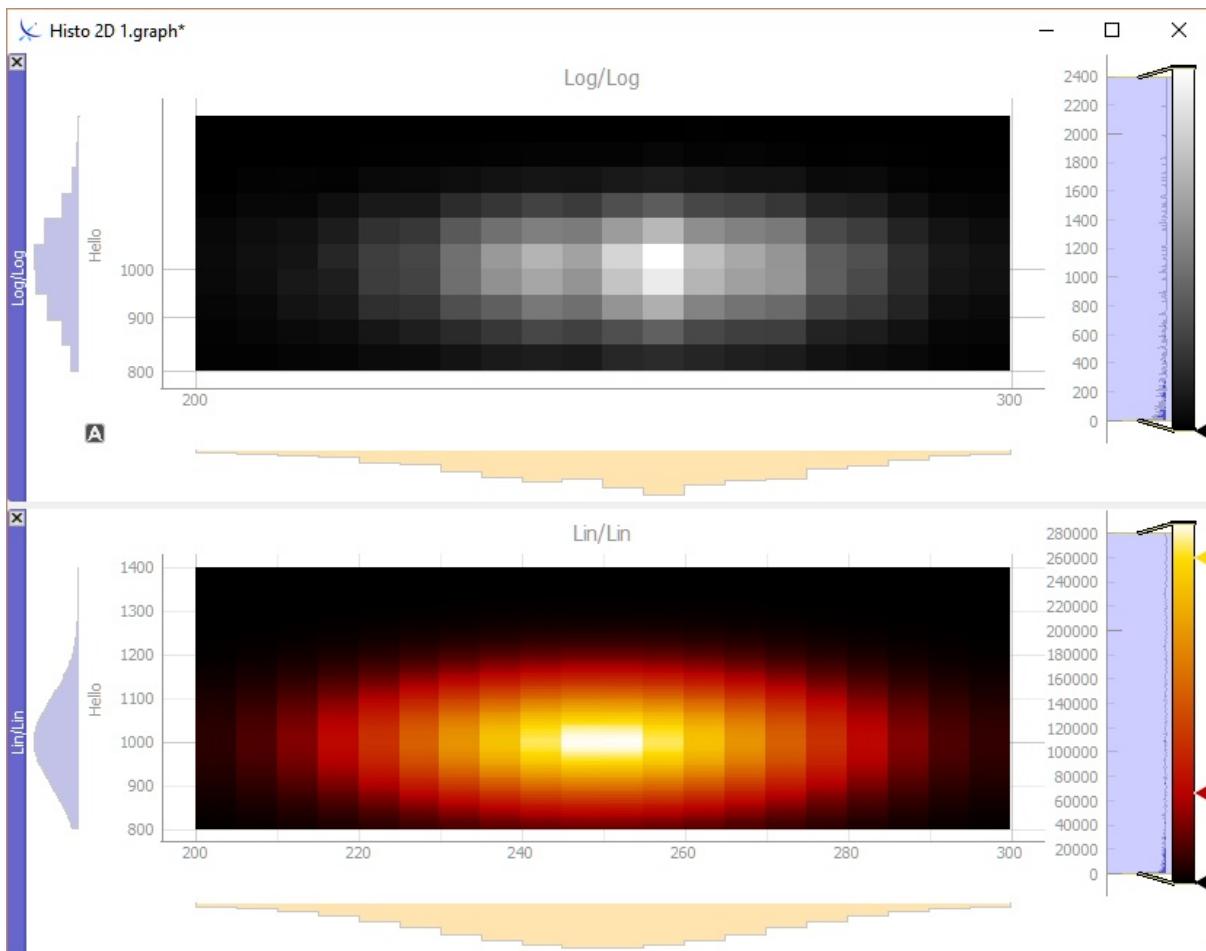


You can see the 'binning' in this display. Also, the values tend to accumulate faster since the bins are wider and you will get more hits than on smaller bins. One word of caution, if your bins are too small (fractional values) you might get display distortions since the binning might not work properly due to rounding issues.

You can also set Variable Length charts as explained in the [Chart Properties](#) chapter. This allows you to have dynamic data arrays.

The y-range also has a log option. Please note that the log function is not defined for values ≤ 0 . So you have to adjust your range accordingly. In order to avoid math exceptions when binning we 'pre-populate' all bins with 1 when in log mode.

7.5.7 2D Histogram



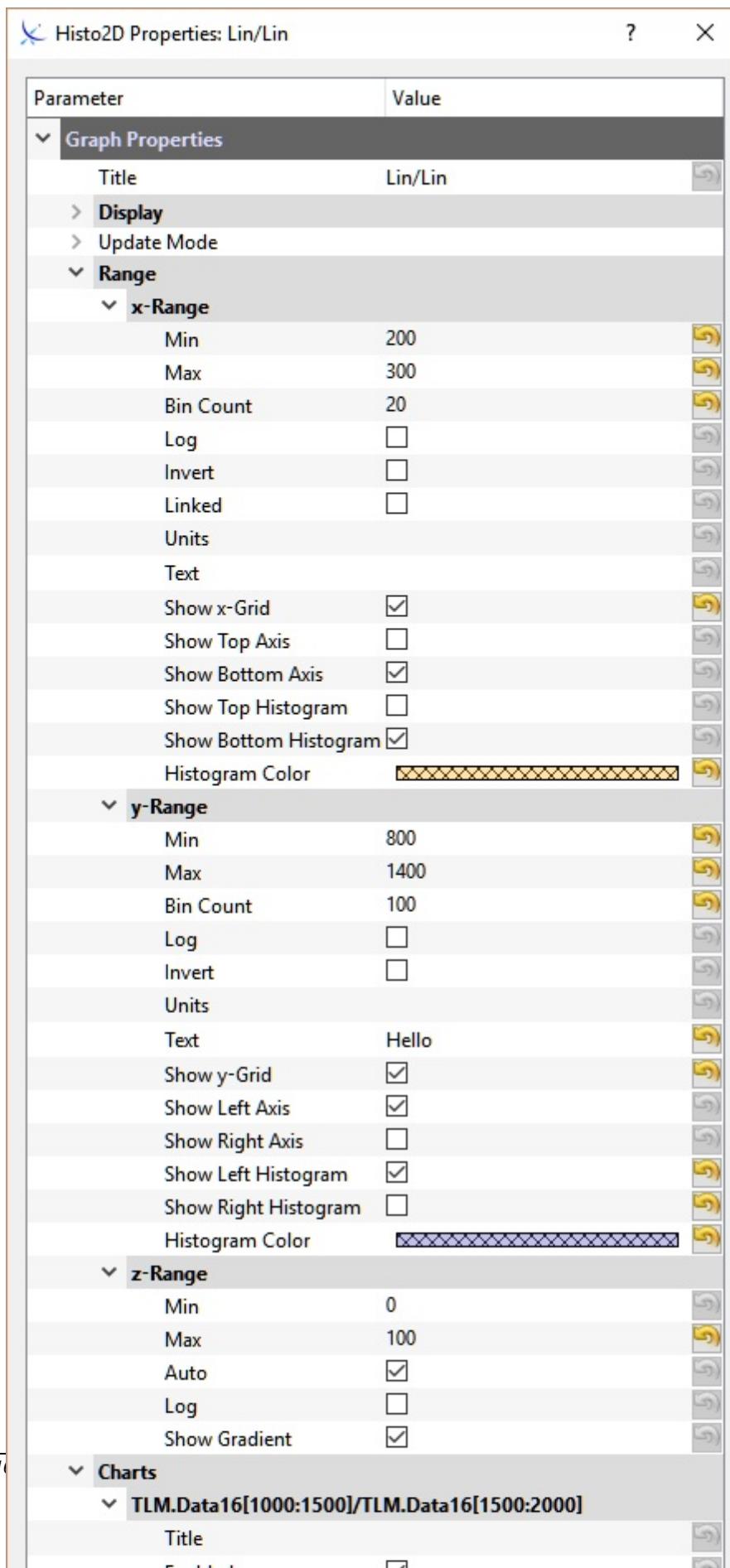
2D histograms allow you do display a distribution of two data items where the values are rendered as a color lookup table. There are a number of predefined color distributions that you can set. However, you can also create your own.

The 2D histogram algorithm is similar to the 1D case in that you can set bins. In the 2D case you will set bins for two data items the X and the Y item. Whenever your data arrives it will be 'binned' into the according X and Y bin depending on the data. For 2D histograms you specify the number of bins per axis. So the bin width is the range divided by the bin count:

$$\text{Bin Width} = (\text{Max-Min})/\text{Bin Count}$$

As in the 1D case the 2D histogram is integrating all values, so you might want to reset the counts from time to time. You can do so with the [context menu](#).

Below is the properties sheet for the 2D histogram:



In addition to the regular x- and y-ranges you also have a z-range that you can configure. The z-range is the counts in each bin. You can set a fixed range or use auto scaling. The mapping of the range is to the color space that you can define with the gradient editor on the right hand side of the graph. You can also show/hide the gradient editor. For the x- and y-axis you can also display a histogram of the distribution of the values over that axis as well as setting the axis histogram color.

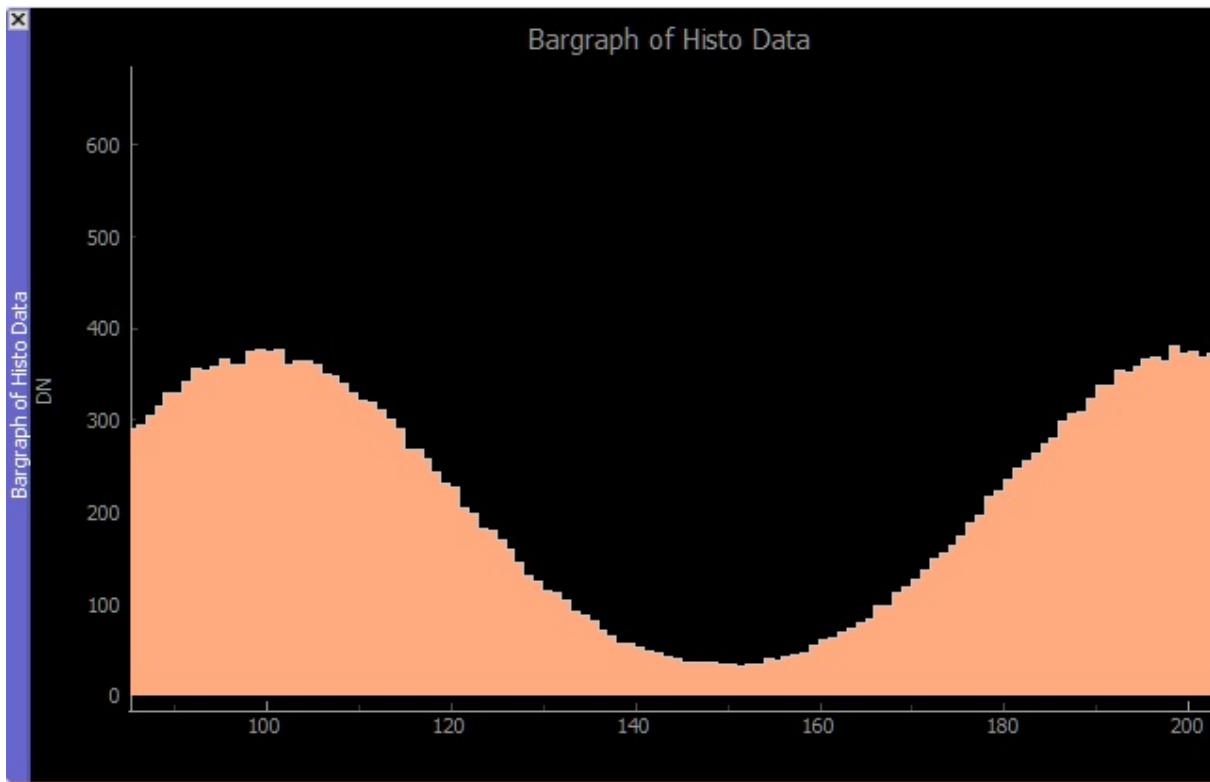
As with 1D histograms you can set a variable length item that determines the number of samples to evaluate. Also, since you specify two items as your data sources you have to specify a trigger item. When that item arrives the we evaluate the data. If both items are from the same block it doesn't matter which item you set as the trigger item. You can also set a scaling for both the X and Y data items. The scaling is applied before the binning is performed.

You can also set all ranges to Log. As with the other graphs keep in mind that the log range is only defined for positive values. Since a log range can span a large dimension it is very important to not set too many bins since you can get really small (fractional) bins which can lead to binning artifacts.

Gradient Editor

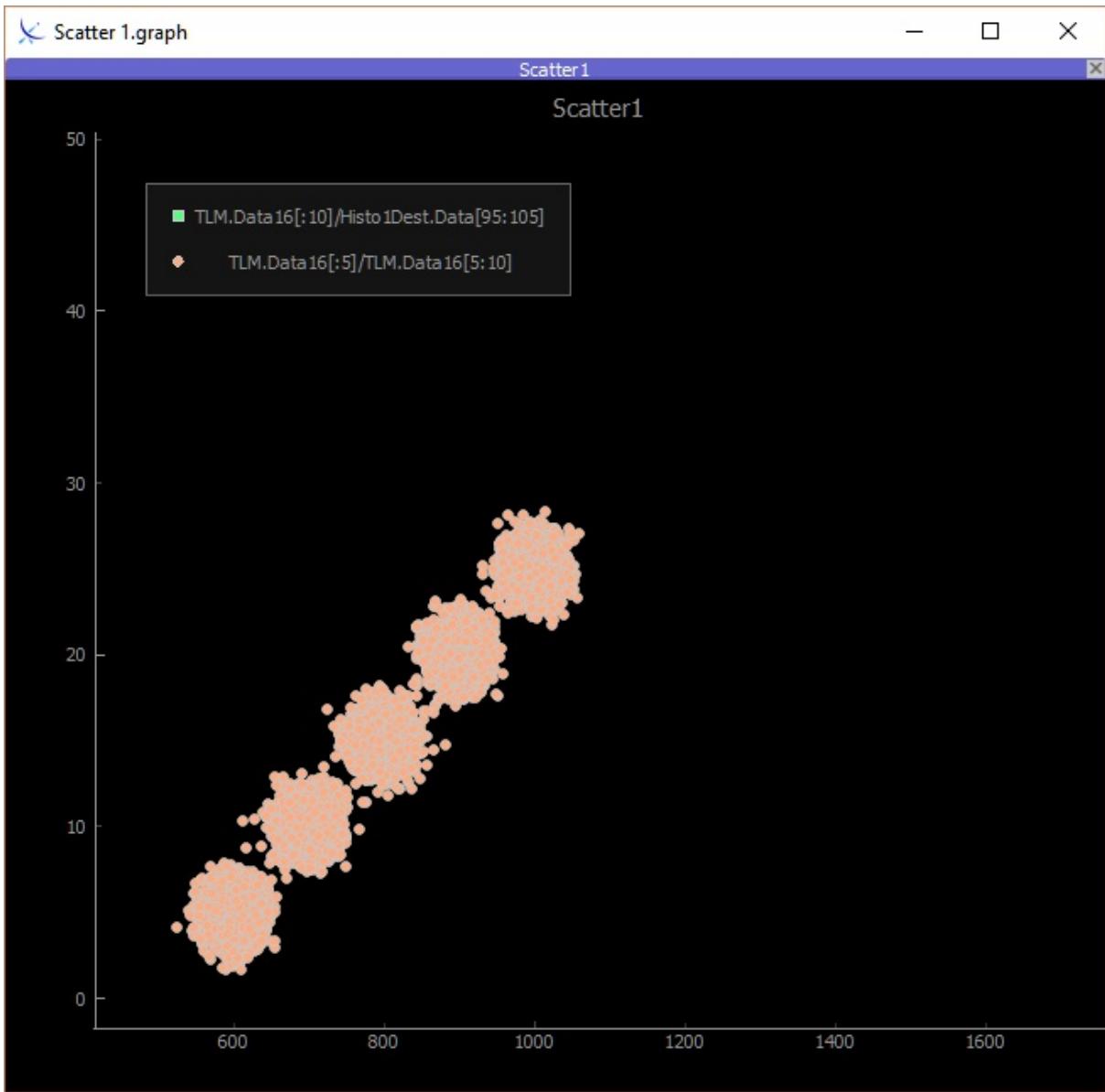
If you enable the 'Show Gradient' setting on the z-range you can change the color mapping. To do so right click on the color bar and select the mapping that fits the best for your data. You can also modify the mapping by dragging the ticks on the right of the color bar. Once you are done and save the Graph window the gradient mapping you have selected will be saved as well.

7.5.8 Bargraph



The Bargraph graph allows you to display your data as a bargraph. The x-range is the number of elements while the y-range is the according values. As for the other charts you can set scaling parameters for both the x- and y-ranges as well as a log scale for the y-range. You can overlay multiple bargraphs in the same graph. As opposed to the histogram graphs the bargraph display doesn't accumulate the data, it simply displays the data as a current view. Therefore this graph doesn't have any 'history'.

7.5.9 Scatterplot



The Scatterplot display allows you to display your data in a scatterplot fashion. It has similarities to the [2D Histogram](#) ¹¹³ display in that it takes two items for the X and Y axis. However, it does not histogram the data. However, there is an 'Integrate Data' setting. If you check this box that data will be accumulated until we have 'Depth' points. Once we reach the Depth with the number of points accumulated we remove the oldest point and add the new one. So you always have a few of the most recent 'Depth' points. If you un-check the 'Integrate Data' box only the current points will be rendered and when new data arrives the old one will be overwritten.

The following image shows the Scatterplot properties dialog:

Scatterplot Properties: Scatter1

Parameter	Value
-----------	-------

Graph Properties

Title Scatter1

Display**Update Mode** Event Timer

Update Period [s] 0.3

Range**x-Range**

Min 0

Max 1000

Auto Log Invert Linked Units Text Show x-Grid Show Top Axis Show Bottom Axis **y-Range**

Min 0

Max 1000

Units Auto Log Invert Text Show y-Grid Show Left Axis Show Right Axis **Charts****TLM.Data16[:10]/Histo1Dest.Data[95:105]**Title Enabled

Y Data Item: TLM.Data16[:10]

X Data Item: Histo1Dest.Data[95:105]

Trigger Item X Y

Depth 10000

Integrate Data

Both the x- and y-ranges allow a Log setting. Each chart has a Depth as outlined above, as well as symbol properties that you can configure with the Display section of the dialog. Similar to the 2D Histogram you can configure the data items and have to chose a trigger item.

7.5.10 Troubleshooting

Since the GseosGraph module is an external process there are few things that might cause trouble.

On Linux make sure the GseosGraph.so file is marked executable. This should be the case since we distribute the archive as a 7z archive which preserves Linux permissions. However, if the process can't be launched check that it has the proper access rights.

The port used between GSEOS and GseosGraph defaults to 9999. Usually there is no issue with firewall-ing localhost. However, you have to make sure the port can be reached. To help debugging any potential issues there are a few settings that might help in that regard.

In the [GseosGraph] section you can set the Verbose entry to Yes:

```
[GseosGraph]
Verbose=Yes
```

This will print diagnostic connection information to the GSEOS message window. The GseosGraph process is only kicked off when we actually need to open a Plot window. So you won't see anything until you try to open a Plot. Once that is the case GSEOS prints to the console window that the process was successfully started and then will attempt to connect to the server for about 10 seconds after which time it will time out. If there is a connection problem try to reconfigure the port with:

```
[GseosGraph]
Verbose=Yes
Port=7777
```

Use a port that is available and not protected (>1024 preferably).

Once the connection has been established you can track further problems by turning GseosGraph logging on. By default errors are logged to a file called GseosGraph.log. You can change the log file and the log level with the following options:

```
[GseosGraph]
Verbose=Yes
Port=7777
LogFile = MyPlotDbg.txt
LogLevel = Info
```

See the [\[GseosGraph\]](#)  section for more details.

7.6 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.

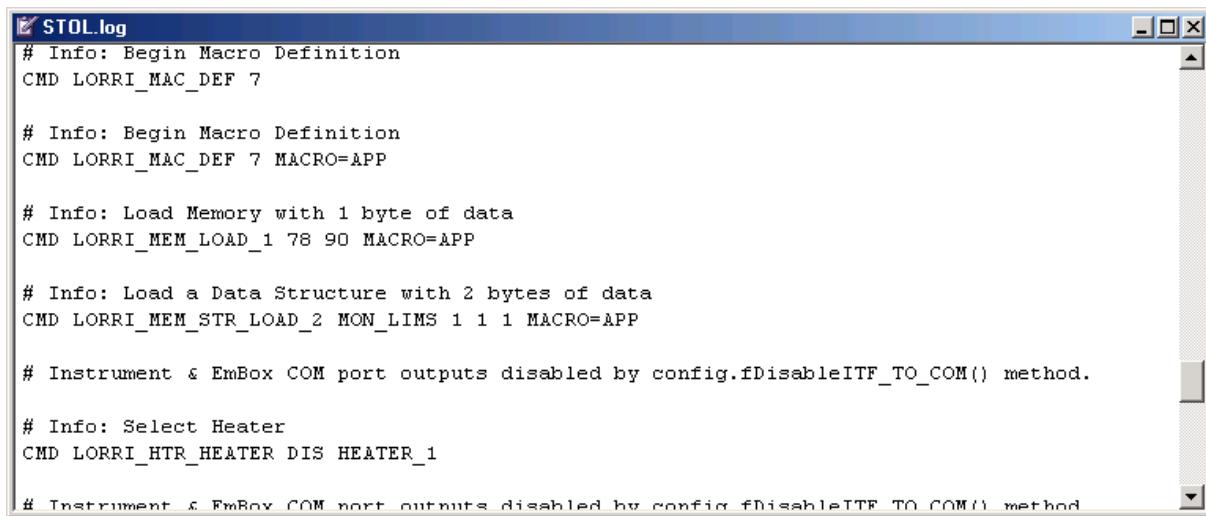
You can write content to a log file without the corresponding log window to be open. Log windows are regular MDI child windows like [screen windows](#)^[38]. They are located on a particular [desktop page](#)^[26] and their layout will be saved with the desktop settings. The caption bar indicates the log file the log window is associated with.

There can be only one window open per log file on the same desktop page. However, you can have several different log files open. You can use the [File menu](#)^[28] to open and save log windows.

As opposed to the [message window](#)^[140] you can copy text from a log window.

You can specify font and color information for a log window. This information is saved with the [desktop file](#)^[26] and not in the log file itself. So if you close the log window and open it again the font and color information will be lost. However, it will be preserved in between GSEOS restarts since the desktop will apply the settings to the file as configured.

You can set color and attribute information when inserting text into a log file with the [Gseos.Log\(\)](#)^[209] function. However, this format information is not saved and on reopening the file it will be lost. The reason for this is that we save the file in plain ASCII without any formatting information. So while you can mark up your logs while the system is running the output file will be a plain text file.



```
# Info: Begin Macro Definition
CMD LORRI_MAC_DEF 7

# Info: Begin Macro Definition
CMD LORRI_MAC_DEF 7 MACRO=APP

# Info: Load Memory with 1 byte of data
CMD LORRI_MEM_LOAD_1 78 90 MACRO=APP

# Info: Load a Data Structure with 2 bytes of data
CMD LORRI_MEM_STR_LOAD_2 MON_LIMS 1 1 1 MACRO=APP

# Instrument & EmBox COM port outputs disabled by config.fDisableITF_TO_COM() method.

# Info: Select Heater
CMD LORRI_HTR_HEATER_DIS HEATER_1

# Instrument & EmBox COM port outputs disabled by config.fDisableITF_TO_COM() method
```

The [Gseos module](#)^[200] exports three functions you can use with log windows:

[Gseos.Log](#)^[209]
[Gseos.LogSave](#)^[210]
[Gseos.LogReload](#)^[210]

7.7 GSEOS Explorer

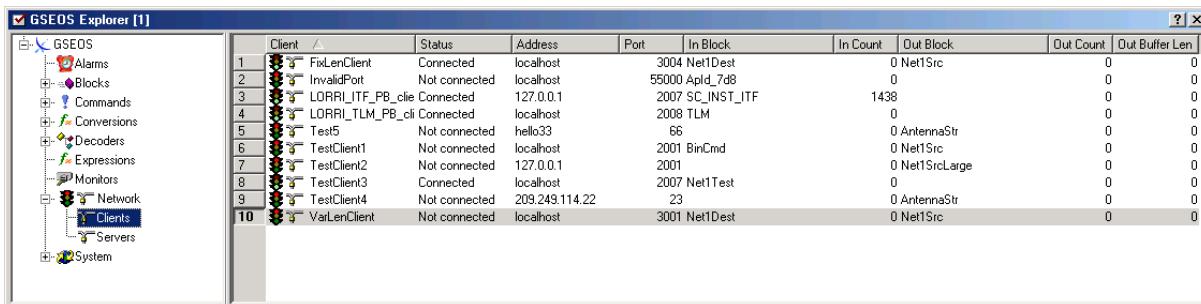
The GSEOS Explorer window allows you to control and monitor the GSEOS system status. You can monitor the currently installed Alarms, Blocks, Commands, Conversions, Decoders, Expressions, Monitors, Network connections, and System status. The System node gives access to system parameters and performance 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 GSEOS Explorer window between floating and pinned mode. If you want to pin the Explorer to a different desktop page check the instructions in the chapter on [Desktop Management](#)^[28].

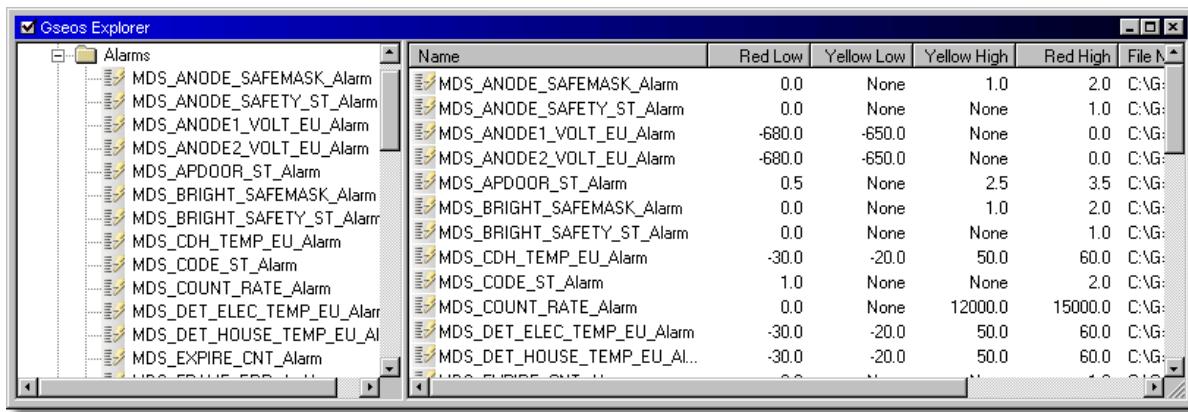
The Explorer window itself consists of two panes, similar to the Windows Explorer interface. On the left hand side you select the specific node you are interested in, on the right hand side you will see the detail information about the selected node. In the figure below the Net/Clients node is selected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



7.7.1 Alarms

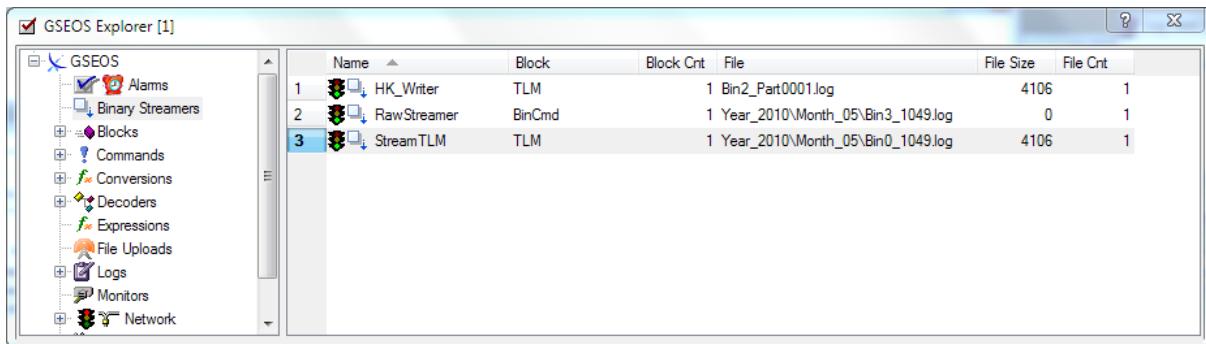
The Alarms node displays all the alarms currently loaded. To change or add new alarms load an [alarm file](#)^[149].

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

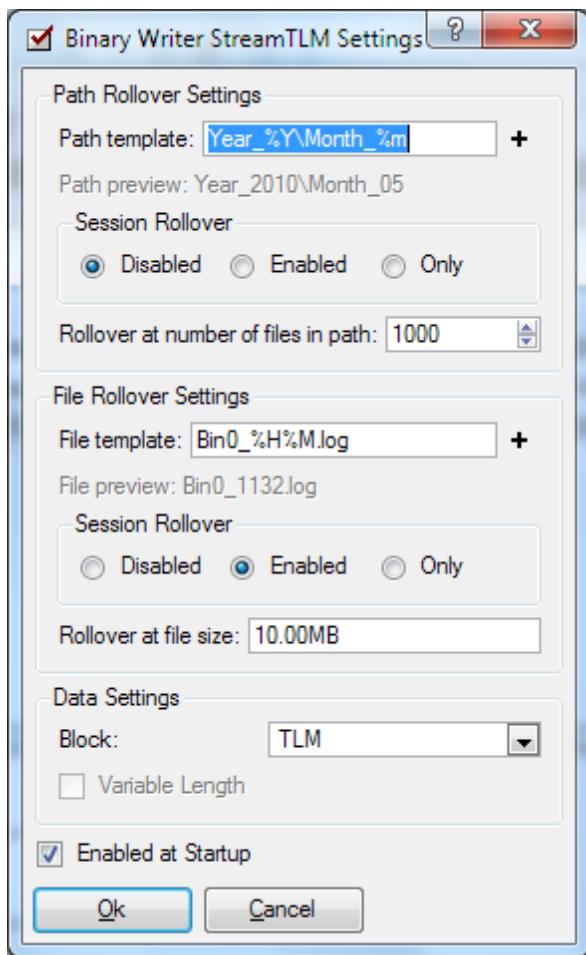


7.7.2 Binary Streamers

The Binary Streamers node displays all configured Binary Streamers. By right-clicking on any of the streamers you can start/stop the streamer or edit its properties. You can also add a new binary streamer by selecting the 'Add New...' menu entry.



The context menu also allows you to delete the selected binary streamer(s). All changes made to the binary streamer configuration is written to the gseos.ini file. To edit a binary streamer you use the Properties... entry on the context menu. The following dialog box shows the settings:



The settings that can be configured here are identical to the ones described in the gseos.ini [BinaryStreamer]^[174] section. If the streamer is running it will be stopped before applying your changes. If the 'Enabled at Startup' flag is set it will be re-started, otherwise you have to start the binary streamer manually. All changes are written to the gseos.ini file.

7.7.3 Blocks

The Blocks node displays all the blocks currently configured in the system. To change or add new blocks modify the [block definition files](#)^[150].

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.

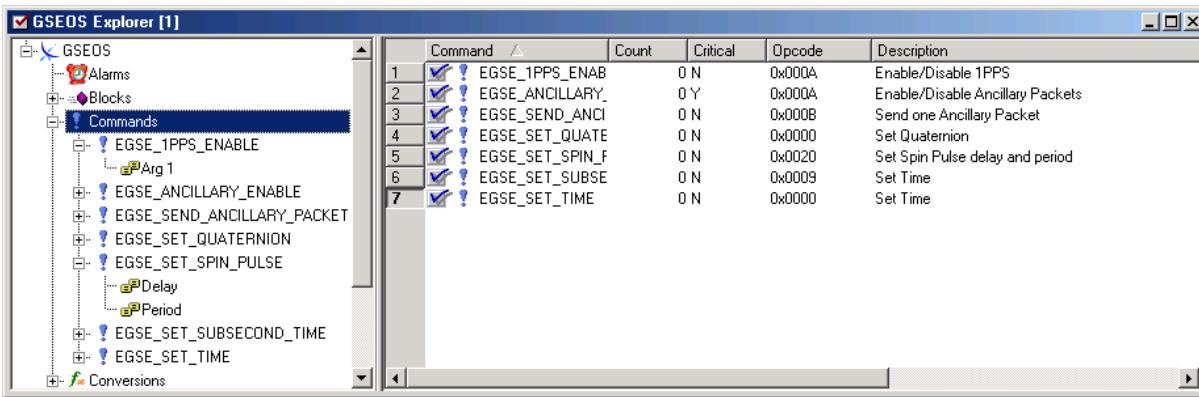
Block	Size	Count	First Item	Items	Tasks
1 AntennaStatus	104	0	18678	48	1
2 AntennaStr	256	0	18676	1	1
3 AntennaTLM	256	0	18677	1	1
4 Apld_400	74	0	0	222	1
5 Apld_403	202	0	222	67	1
6 Apld_406	224	0	2238	546	1
7 Apld_7d8	700	0	2784	203	1
8 Apld_8	316	0	20897	259	1
9 ArchiveRate	4	0	11581	1	1
10 bce_sync_blk	128	0	8531	32	1
11 BceCmdBlk	516	0	8610	2	1
12 BceTlmBlk	1275	0	8471	60	1
13 BinCmd	1032	0	18835	3	1

The Block node has three child nodes: Decoders, Items, Monitors. The Decoders folder lists all decoders that are installed on this block, that is all decoders that get processed upon arrival of this block. The Items node lists all the items defined for this block, and the Monitors node lists all the Monitors installed on this block, similar to the Decoders node. In the following example you can see that the decoder 'Command Decoder' is installed on the CmdString 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](#)^[126], 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](#)^[126], for more details on Monitors please refer to [GSEOS Explorer/Monitors](#)^[127].

Decoder	Count	Installed On	Description
1 Command Decoder	0	CmdString	

7.7.4 Commands

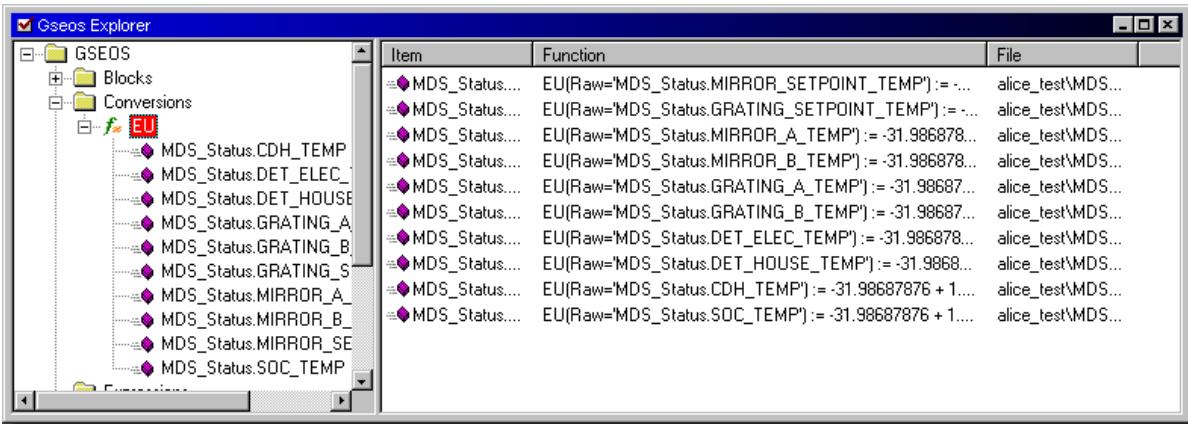
The Commands node displays all commands currently defined in the system. To change or add new commands you can load a [command definition file](#)^[156]. The picture below shows the GSEOS Explorer with a list of commands.



The Command nodes have child nodes listing their respective arguments. In the right hand pane you can pop up a context sensitive menu by right-clicking on the selected command(s). This menu allows you to enable/disable individual commands as well as delete them. You can also execute commands from this menu. If arguments need to be specified a command dialog will prompt you for the missing arguments.

7.7.5 Conversions

The Conversions node displays all the conversion functions currently loaded. To change or add new conversion functions modify the [formula file](#) [168] and load it. Conversion functions are special functions that get mapped to a telemetry point. In the Explorer view below you can see the conversion function EU which is defined for several telemetry points with potentially different algorithms/implementations. This allows to map one conversion function name to several mappings depending on telemetry point.



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.

7.7.6 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](#)²⁴⁶. 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.

GSEOS Explorer [LORRI_MOC]				
	Decoder	Count	Installed On	Description
1	1X1 Image Builder	0	LORRI_1X1_Line	1X1 Image Builder
2	1X1 Image Decoder	0	LORRI_1X1_RAW	1X1 Image Decoder
3	1X1 Image Update	218	DSACtrl	1X1 Image Display Update
4	1X1 Lossless Image Decoder	0	LORRI_1X1_LOSSLESS	1X1 Lossless Image Decoder
5	4X4 Image Builder	0	LORRI_4X4_Line	4X4 Image Builder
6	4X4 Image Decoder	0	LORRI_4X4_RAW	4X4 Image Decoder
7	4X4 Image Update	218	DSACtrl	4X4 Image Display Update
8	4X4 Lossless Image	0	LORRI_4X4_LOSSLESS	4X4 Lossless Image Decoder
9	Alarm History Decoc	0	LORRI_ALARM	Alarm History Decoder
10	Analog Max Min De	0	LORRI_STAT	Analog Max Min Decoder
11	Archive ApID Init	268	MOCStatusPB	
12	Copy CDH_A to CD	0	CDH_A	
13	Copy CDH_B to CD	0	CDH_B	
14	Copy PA_A1 to Px_	0	PA_A1	
15	Copy PA_A2 to Px_	0	PA_A2	
16	Copy PA_D1 to Px_	0	PA_D1	
17	Copy PA_D2 to Px_	0	PA_D2	
18	Copy PB_A1 to Px_	0	PB_A1	
19	Copy PB_A2 to Px_	0	PB_A2	
20	Copy PB_D1 to Px_	0	PB_D1	
21	Copy PB_D2 to Px_	0	PB_D2	
22	Copy ST_A to STAT1	0	ST_A	
23	Copy ST_B to STAT1	0	ST_B	
24	Copy STAT_A to S1	0	STAT_A	
25	Copy STAT_B to S1	0	STAT_B	
26	Copy T_A to TRIO	0	T_A	
27	Copy T_B to TRIO	0	T_B	
28	Decode LORRI SC	0	EB_Status	
29	Decode_LORRI_Im	0	LORRI_IMG_DES	Normalize LORRI Image histogram, .
30	Determine active IE	0	ST_A, STAT_B, STAT_A, S	
31	...	0	...	

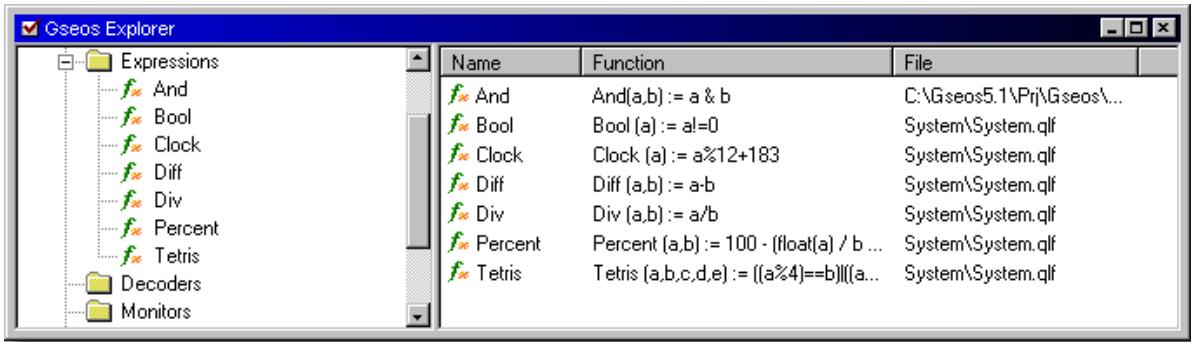
Upon expanding a particular decoder node in the left pane you will see the output blocks this particular decoder generates. From here on you have the same properties as the [Blocks node](#)¹²³.

Right clicking on any decoder node in the right hand pane will pop up a menu that allows you to 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](#)²⁶⁴ properties here.

You can select multiple decoders and invoke the operation on all the selected decoders at once.

7.7.7 Expressions

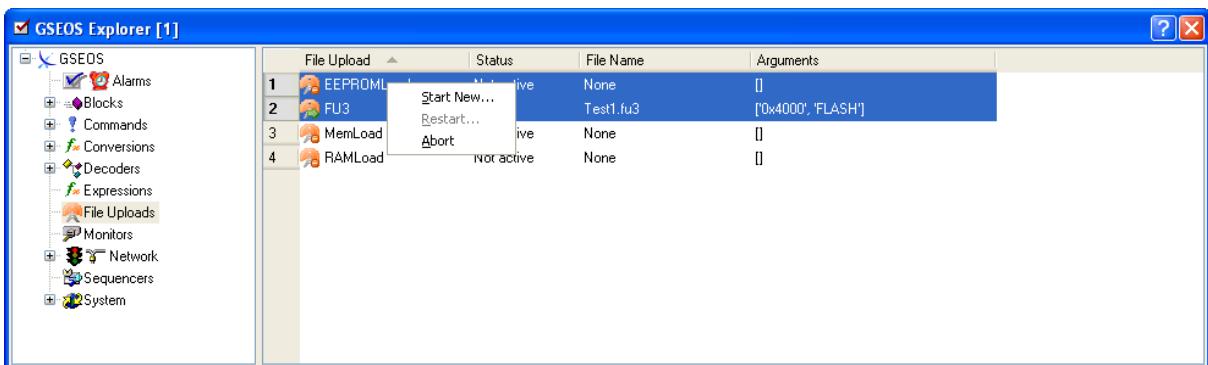
The Expressions node displays all the expressions currently loaded. To change or add new a new Expression modify the [formula file](#)¹⁶⁸ and load it.



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

7.7.8 File Uploads

The File Uploads node displays all configured File Uploads. It shows if any of the file uploads are active and the current/last file name together with the passed along parameters. For more information on how to configure a File Upload please check the [How to implement a File Upload.](#)³³² FAQ.

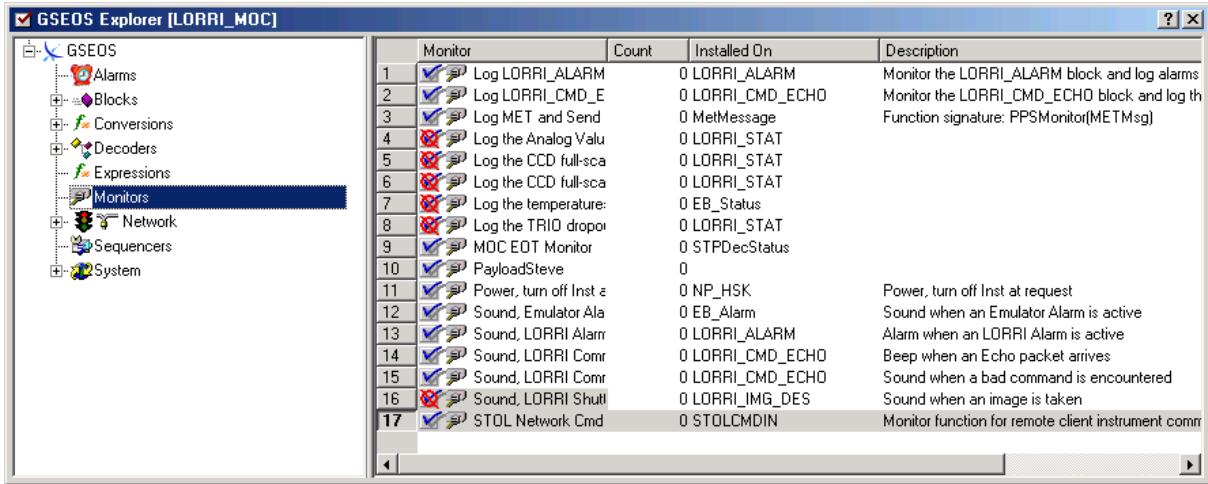


By selecting one or more File Uploads and right-clicking you can access the File Upload menu. This allows you to start a new file upload, restart a previous file upload or to abort an ongoing file upload.

7.7.9 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](#)²⁸⁵. 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 a menu that allows you to 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, this monitor will not be removed until you release this variable.

You can select multiple monitors and invoke the operation on all the selected monitors at once.

7.7.10 Network

The Network node lets you display and modify the status of your network connections. It is categorized into Servers and Clients. The network connections themselves are configured in the [gseos.ini](#)^[17] file in the [\[Net\]](#)^[18] section. The property pages of the servers and clients allow you to modify the server and client properties and to apply these changes while GSEOS is running.

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. Keep in mind that outgoing data is still active, even if the network is disabled. This is by design so that data generated by other data sources may be transmitted over the network. 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.

Clients

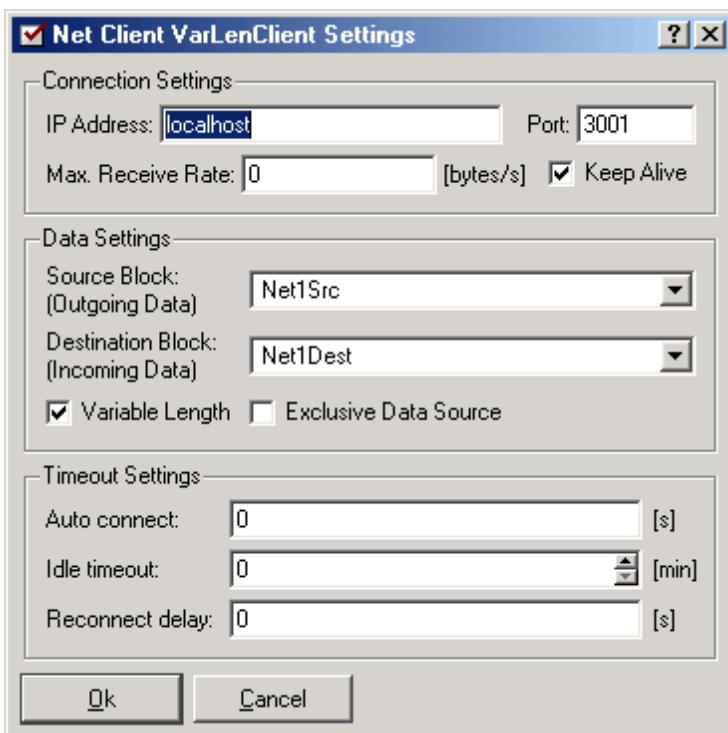
Client	Status	Address	Port	In Block	In Count	Out Block	Out Count	Out Buffer Len
1	Connected	localhost	3004	Net1Dest	0	Net1Src	0	0
2	Not connected	localhost	55000	ApId_7dB	0		0	0
3	Connected	127.0.0.1	2007	SC_INST_ITF	1438		0	0
4	Connected	localhost	2008	TLM	0		0	0
5	Not connected	hello33	66		0	AntennaStr	0	0
6	Not connected	localhost	2001	BinCmd	0	Net1Src	0	0
7	Not connected	127.0.0.1	2001		0	Net1SrcLarge	0	0
8	Connected	localhost	2007	Net1Test	0		0	0
9	Not connected	209.249.114.22	23		0	AntennaStr	0	0
10	Not connected	localhost	3001	Net1Dest	0	Net1Src	0	0

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.

The In Count counter will increment with incoming data, even if the network is not enabled and the data blocks are therefore not forwarded to GSEOS for processing. This lets you monitor the activity on the connection.

When GSEOS sends the outgoing (Source) block to the remote end it will buffer the data in an internal buffer. If the connection is slow or the receiver does not read accordingly this buffer will fill up over time. The Out Buffer Len counter lets you monitor the size of this buffer, in normal operations it should be zero. The count is given in bytes.

The context menu also allows you to add a new client connection, to delete the selected connection(s) or to edit the properties of an existing connection. All changes made to the client configuration are written to the gseos.ini file. To edit a client connection you will use the Properties... entry on the context menu. The following dialog box will allow you to change the settings:

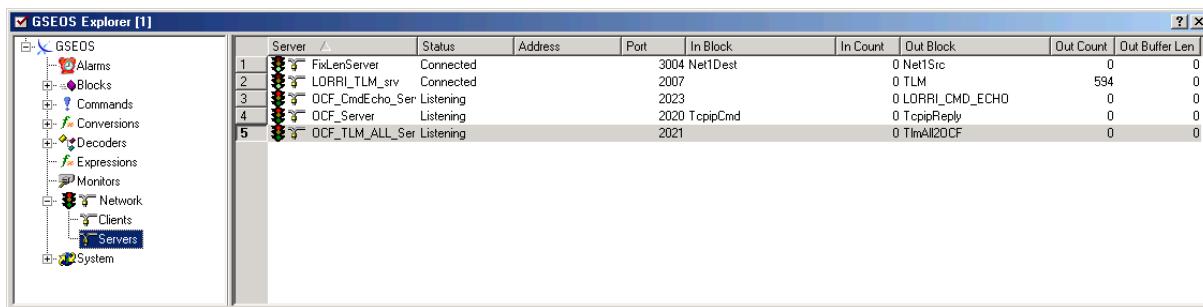


The settings that can be configured here are identical to the ones described in the gseos.ini [Net] section. The IP-Address field allows you to specify the remote server address. Although it will perform a name lookup we recommend against it due to unpredictable DNS server timeouts. If you configure a DNS name you will be asked if you want to switch to the according numeric IP representation.

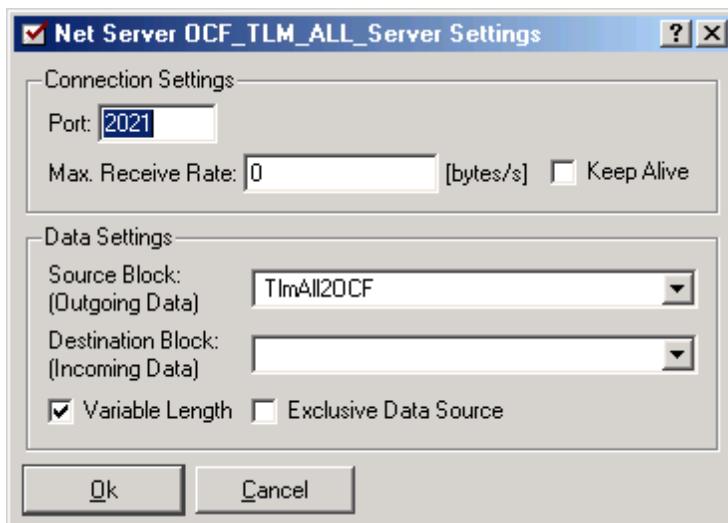
You have to specify at least one of the Source Block or Destination Block settings. If one is blank that means the connection will not support this direction of transfer. The Variable Length option is only enabled if the selected Source/Destination block(s) have the Len item with the proper attributes (first item in block and 32-bit in size).

If the connection is established and you exit the dialog with Ok the connection has to be terminated for the changes to take effect. All changes will be written to the gseos.ini file and therefore will be permanent.

Servers



Underneath the Servers node the configured network servers are displayed. The server status displayed is the same as in the client section. 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. The context menu also allows you to add a new server connection or to edit an existing one. The server settings dialog depicted below is a subset of the client settings as outlined above:



If the server is connected when you Ok the dialog you will be asked if you want to apply the settings and therefore disconnect the connection. If you choose 'Yes' the connection will be disconnected and the changes applied. The client will have to reconnect in order to re-establish the connection.

7.7.11 System

The System node give you system performance measures. The memory node lists all memory related performance counters.

Name	Value
1 Total	67108864
2 Free	65917760
3 Used	1191104
4 Used [%]	1
5 Free Block Count	4
6 Iterations	1
7 Alloc Calls	4531
8 Free Calls	4433
9 Diff Calls	98
10 System Load [%]	0
11 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. The System Load counter gives you a measure of the incoming block rate vs. the processed block rate, i.e. if the GSEOS block queue is growing or shrinking. Although temporary higher receive rates cause no problem, if this situation persists the buffer will eventually run out of space. You can use the Tasks view below to get more detailed information which task is taking up the most processing time.

Task	Count	Curr. Time	Total Time	Avg. Time	Out Block Cnt	Used Buffer	Lost Blocks	Ack	Terminated	Priority	Data Source
1 BDM	0	0.000	0.000	0.0	3978	0	0	0 Yes No	0 No	0	No
2 Net Task Exclusive	0	0.000	0.000	0.0	0	0	0	0 Yes No	0 Yes	0	Yes
3 Net Task Non Exclusive	0	0.000	0.000	0.0	0	0	0	0 Yes No	0 No	0	No
4 Python	4489	0.000	0.426	0.0	511	0	0	0 Yes No	100 No	100	No
5 Python Data Source	0	0.000	0.000	0.0	0	0	0	0 Yes No	0 Yes	0	Yes
6 Recorder	0	0.000	0.000	0.0	0	0	0	0 Yes No	0 No	0	No

Task

The name of the GSEOS task.

Count

The number of blocks that this task has processed.

Curr. Time

The time in ms this task took to process the last few blocks. 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 process a block. This is the quotient of total time over count.

Out Block Count

If this task also generates blocks the number of generated blocks is indicated by this counter.

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](#)¹³¹ 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 task 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.

Terminated

If a task has been terminated from processing this flag is set to Yes.

Priority

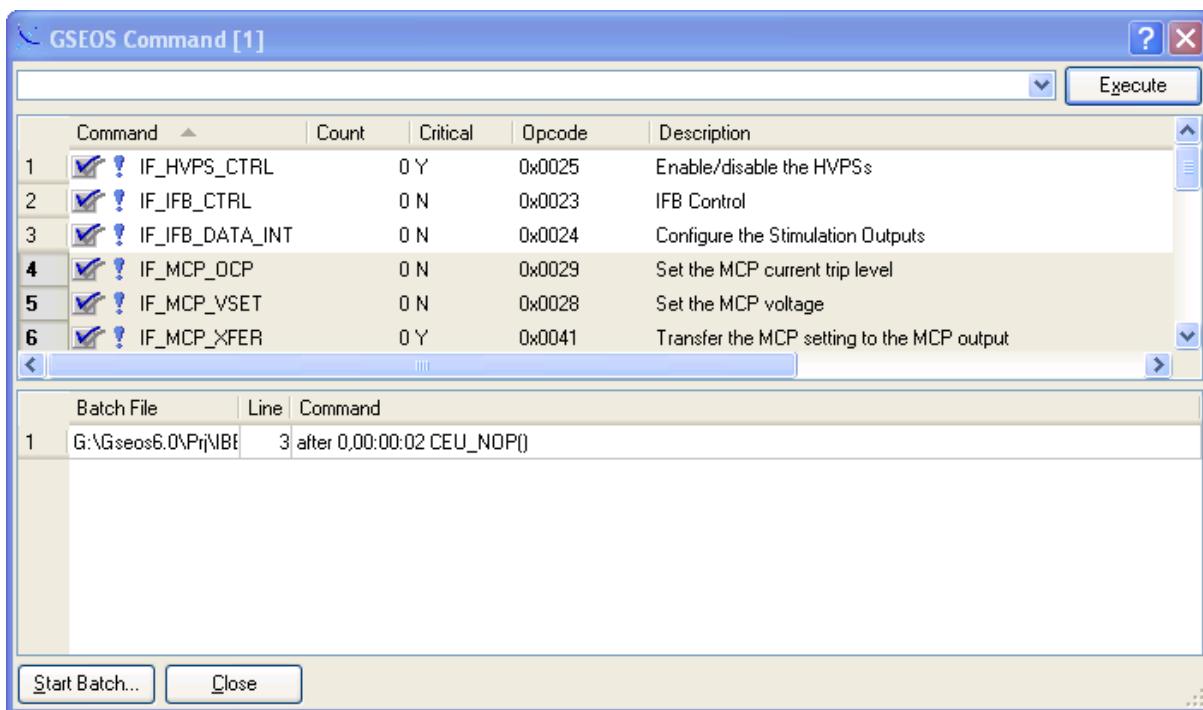
If this task is a decoder it will have a priority other than 0. Decoders are executed in the order of their priority. None decoder tasks have the lowest priority.

Data Source

If this task is a mutually exclusive data source this flag is set to Yes. Note that no data source should have a priority other than 0. Specifically, decoders should never be data sources.

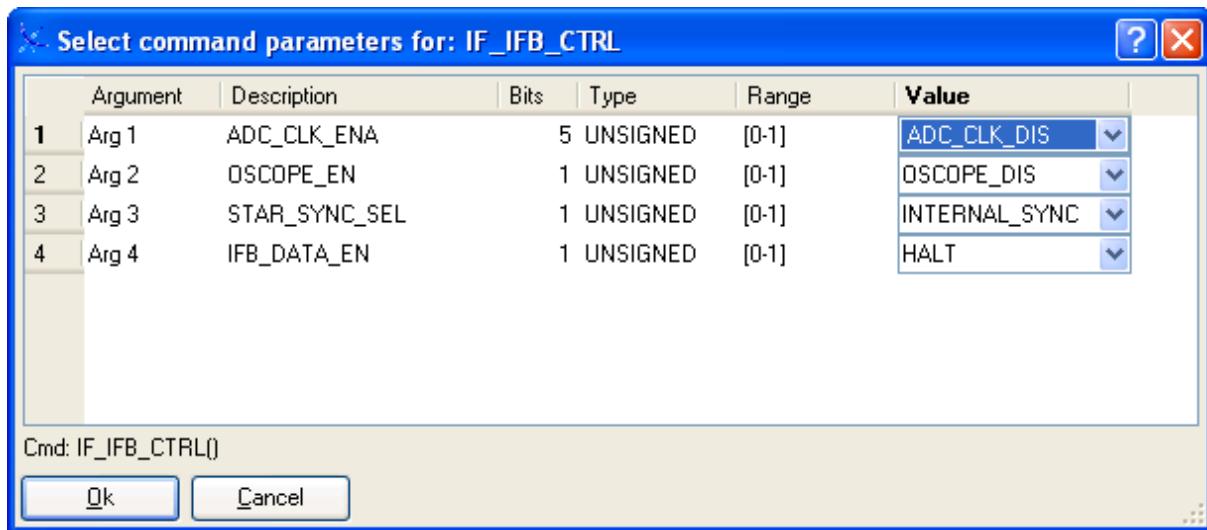
7.8 The Command Dialog

The Command dialog allows you to inspect defined commands and to issue them. It also shows the status of batch files and allows you to start batch files. An alternative to the Command dialog is the [GSEOS Explorer](#)¹²⁴, choose the Command node and you can access all defined commands from there.



You can open the command dialog from the [View menu](#)³², the F7 hotkey, or the ! toolbar button.

All [command definitions](#)¹⁵⁶ that were loaded can be accessed on the right hand list box. You can execute a command by either selecting the context menu with a right click or by double clicking on the command. If the command requires arguments a dialog will pop up and query for the missing arguments:



On the bottom of the command parameter dialog you can see the command as it you fill in the arguments. Once all arguments are specified you can click Ok and the command will be executed.

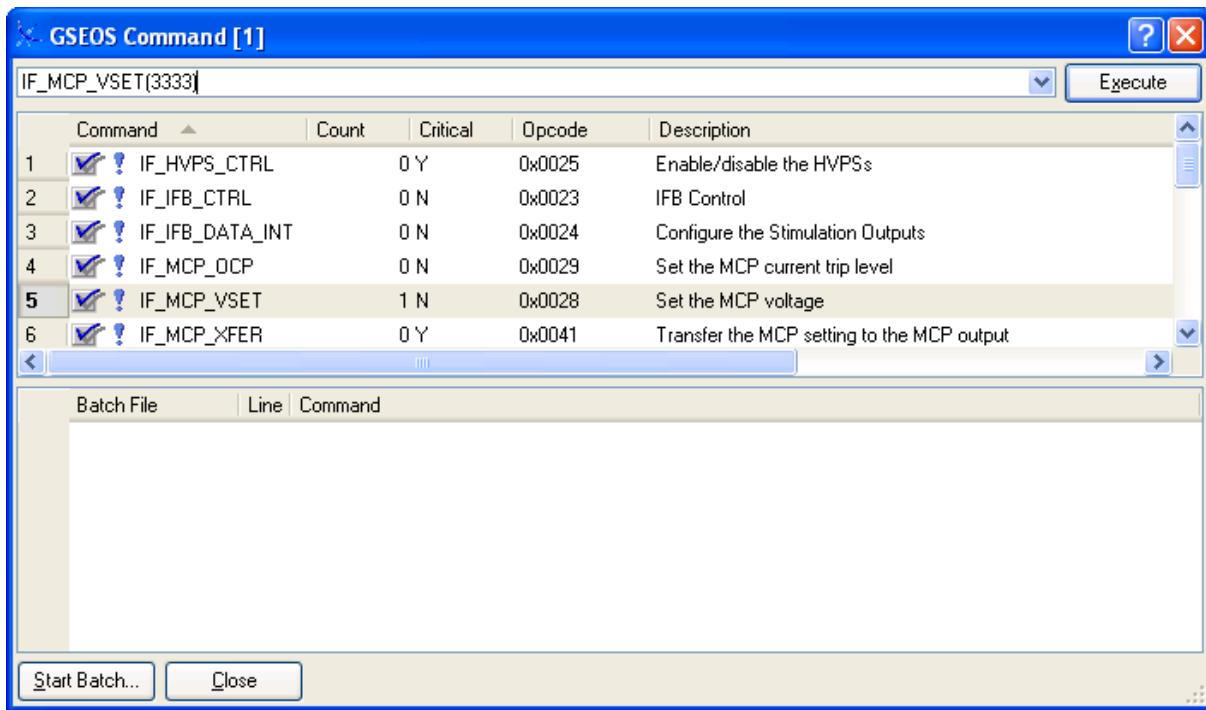
All commands executed will be added to the command history on the bottom of the command dialog. This allows you to quickly re-issue previously executed commands. You can either press return with a command selected in the command history drop down box or you can click the **Execute** button.

Note

Please note that you can control this behavior with the [gseos.ini](#) setting:

```
[Command]
CmdHistoryDialogOnly = Yes
```

you can turn off additions to the command history from all sources but the dialog itself. This is useful when you have scripts generating commands asynchronously.



Batch Files

To start a command batch click **Start Batch...** and select the batch file to execute. You can run multiple batch files concurrently. The batch file will show up in the list of batch files and the current line number and command are displayed. When the batch file completes it is removed from the list. See the chapter on [Batch Files](#)^[154] for more information.

To terminate an executing batch file you can right click on the batch file and select **Stop**.

7.9 The Console Window

The console window allows you to interact with the [built-in Python](#)^[198] 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. It can have one of three states: Floating, Pinned, Closed. Check in the chapter on [Desktop Management](#)^[261] for more information. The console window can be easily displayed and hidden with the [View menu](#)^[32], 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.

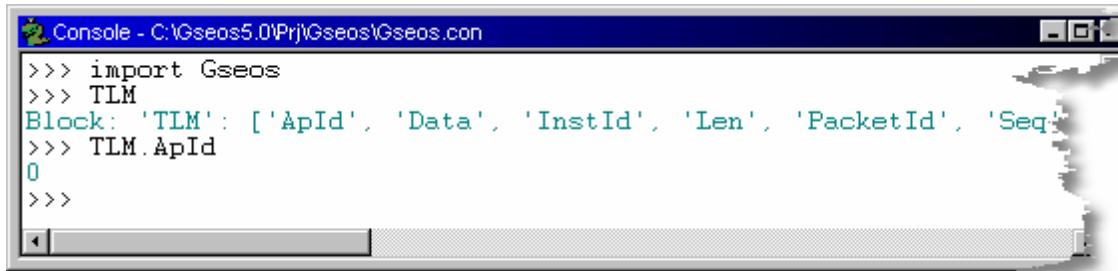
The Python icon on the tool bar allows you to open/close the console window:



The file name of the Console window can be specified in the [gseos.ini](#)^[171] configuration file. The entry FileName in the [\[Console\]](#)^[179] 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]^[19] 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:



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 buried 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 monitor 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](#)^[285] 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 GseosMonitor
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.PlaySound('boing.wav')

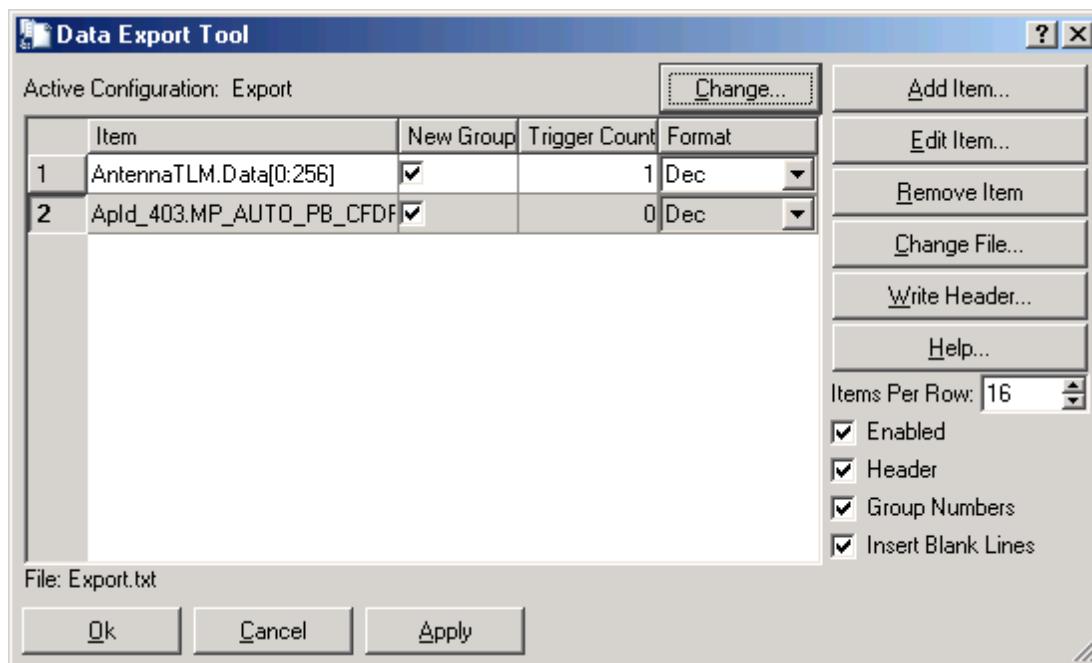
ConsoleOut.Monitors.append(GseosMonitor.TMonitor('Console Activity',
```

```
MonConsoleOut))
```

For more information about programmatic access to the console window check the [Gseos.ConsoleWrite\(\)](#)^[201] function.

7.10 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 picture shows the Data Export dialog.



You can display the Data Export dialog from the main menu: Tools\DataExchange. 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](#)^[168] 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 abyData[0:511]
##
## Line #4: ConsoleOut.Len
##
```

The header displays the output line number assignment. If the 'Line Numbers' check box is checked the data it 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

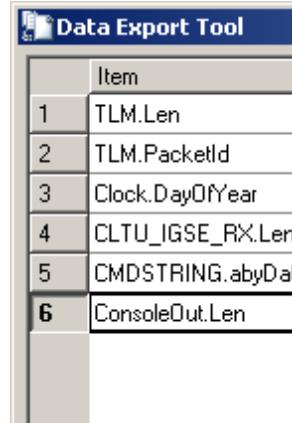
(the trigger count is 0) the line will not be printed.

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.



Programmatic Control

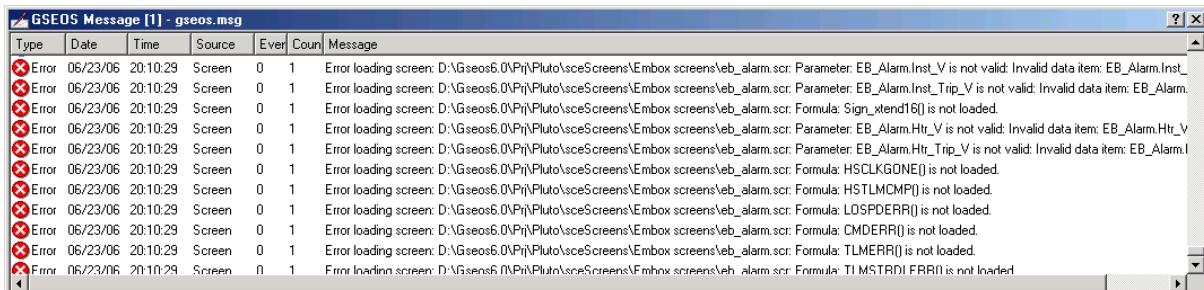
The data export can be enabled/disabled from Python. You can also set the active configuration. Check the chapter on [GeosExport](#)^[257] for more information.

7.11 The Message Window

The message window is where GSEOS posts system message, certain error messages and general purpose messages. You can also post messages to the message window.

However, [log windows](#)^[120] may be the more appropriate place to post user messages.

The following snapshot depicts the message window:



The message window can be floating, pinned to a desktop page or hidden. Refer to the [Desktop Management](#)^[26] section for more information. 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 file used is specified in the gseos.ini file in section [Message]. The entry FileName holds the current message file.

```
[Message]
FileName = gseos.msg
MaxFileSize = 2MB
```

You can also specify a maximum file size. By default the file grows without any limit. The message window organizes messages in rows. It logs the data and time, the source that generated the message, a message type (INFO, WARNING, or ERROR), an event ID, a count, and the message itself. If a message gets repeated the counter is incremented instead of adding the same message over and over. This is useful if a particular error condition generates the same error message on an ongoing basis. Instead of flooding the message window the counter for that message increments. However, this is only true if the new message is identical to the last, otherwise a new message will be added.

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

Below is the definition of the Message block as defined in System.blk:

```
Message {Type      ,,, 32;
          EventID   ,,, 32;
          Source[32] ,,, 8;
          Text[1024] ,,, 8}
```

You can intercept this block with a GSEOS [Monitor](#)²⁸⁵ or [Decoder](#)²⁴⁶ and perform the appropriate action.

Message Filtering

By default all messages are logged in the message window and written to the underlying message file. It is possible to filter out specific messages. Multiple filters can be specified. A message is not logged if the source string matches one of the filters and the message level is less than or equal the specified filter level. The following example suppresses all INFO messages from sources that match 'Cmd*', that is all strings that start with 'Cmd'. Warning and error messages for sources of 'Cmd*' are still logged to the message window and file. You can set the filters in the [\[Message\]](#)¹⁸⁶ section of the gseos.ini file.

```
[Message]
Filter = Cmd*, INFO
Filter = ASIST_TLM, ERROR, FANN sequence number out of order*
```

Besides the source and level you can optionally specify the message text and event ID. If you want to match multiple messages you can use the ? or * wildcards. If you specify message text and/or event ID they must match together with the source and level field in order to suppress the message.

Programmatic Access

The [Gseos module](#)²⁰⁰ offers the method: [Gseos.Message\(\)](#)²¹² which you can call to post

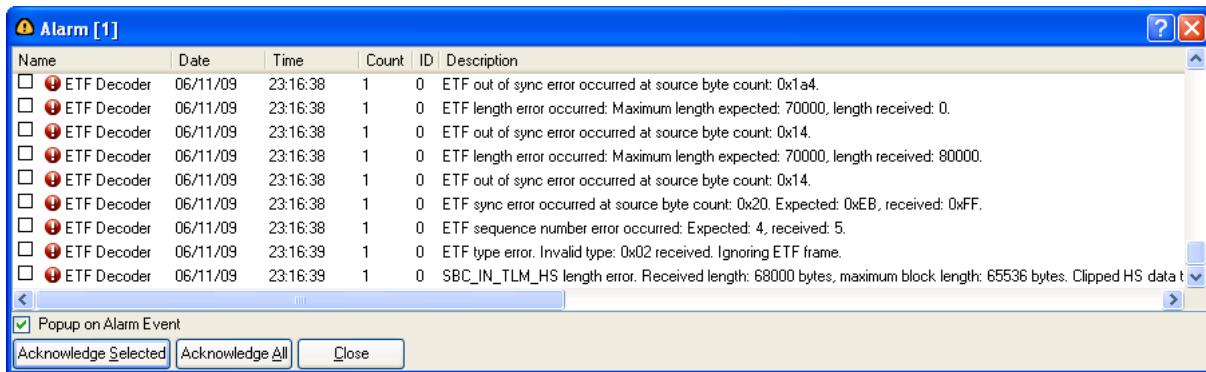
your own messages to the message window. This is useful for short, infrequent messages like exceptions or errors you want to report. For more extensive logging a [log window](#)¹²⁰ might be more appropriate.

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.

7.12 The Alarm Window

The alarm window is a way to notify the user of important alarm conditions. When an Alarm is generated the Alarm Notification Window pops up displaying the alarm message. The user can either selectively acknowledge individual alarms or clear all alarms. The notification will be opened again when new alarms occur.

The following snapshot depicts the message window:



If the Alarm Notification window is not open and the 'Popup on Alarm Event' checkbox is unchecked the Alarm Notification icon in the main Tool bar flashed to notify the user of the new alarm condition.

You trigger alarms by either generating the AlarmEvent block directly or by using the `Gseos.Alarm()` function which generates this block.

```
AlarmEvent {
    Level           ,,, 8;
    Name[64]        ,,, 8;
    Description[256],,, 8;
    ID              ,,, 32;
}
```

The Level item can be one of the following constants:

```
Gseos.ALARM_STATE_NONE
Gseos.ALARM_STATE_RED_LO
Gseos.ALARM_STATE_YEL_LO
Gseos.ALARM_STATE_YEL_HI
Gseos.ALARM_STATE_RED_HI
```

The user interface only displays yellow or red alarm state. The name should refer to the name of the alarm but can also indicate the source of the alarm. The description is the

actual alarm message that is displayed. The ID is optional and can be an arbitrary number.

The following code will generate an alarm by issuing the AlarmEvent block directly:

```
#  
# Generate AlarmEvent block.  
#  
import Gseos  
import GseosBlocks  
oBlkAlarmEvent = GseosBlocks.Blocks['AlarmEvent']  
  
oBlkAlarmEvent.Level      = Gseos.ALARM_STATE_RED_LO  
oBlkAlarmEvent.Name[:]    = "Thermistor 7"  
oBlkAlarmEvent.Description[:] = "Deployment latch temperature -45F, too  
low"  
oBlkAlarmEvent.ID         = 0  
oBlkAlarmEvent.SendBlock()
```

Alternatively to generating the AlarmEvent block yourself you can use the [Gseos.Alarm\(\)](#)
 function which generates the block for you. You can also configure an [Alarm Monitor](#)
 that can generate an AlarmEvent block.

You can acknowledge individual alarms or all alarms from the Alarm Notification Window. Once all alarms are cleared the Alarm Notification Window will close. Alarms can also be acknowledged by generating the AlarmAck block. This allows you to programmatically clear alarms. The definition of the AlarmAck block is listed below:

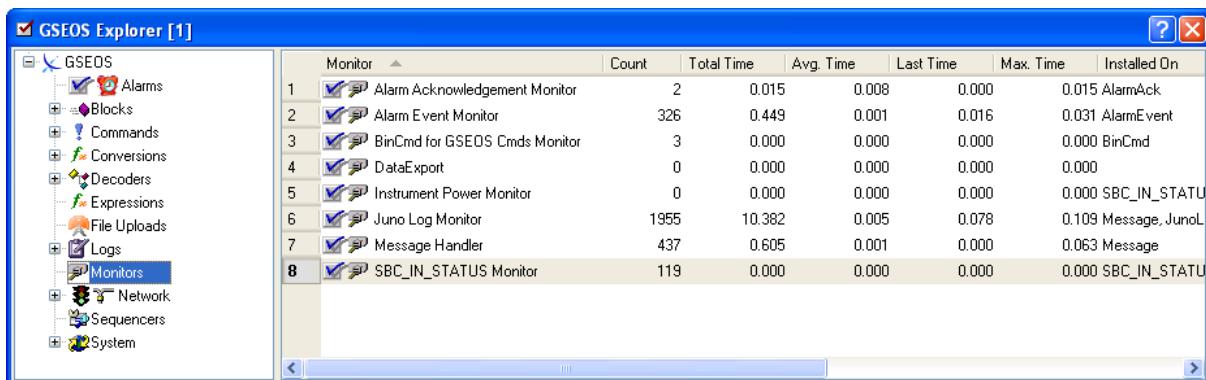
```
AlarmAck {  
    Name[64]      ,,, 8;  
    ID           ,,, 32;  
    AckAll       ,,, 8;  
}
```

You can generate this block to either acknowledge individual alarms or all alarms. To clear an individual alarm the **Name** **and** **ID** fields of the AlarmAck block must match the corresponding fields of an outstanding alarm. If the **AckAll** field is set to 1 all alarms are cleared. The following example clears the alarm we generated with the above event:

```
#  
# Clear previous alarm.  
#  
import Gseos  
import GseosBlocks  
oBlkAlarmAck = GseosBlocks.Blocks['AlarmAck']  
  
oBlkAlarmAck.Name[:] = "Thermistor 7"  
oBlkAlarmAck.ID     = 0  
oBlkAlarmAck.SendBlock()
```

The Alarm monitors are implemented a regular [GSEOS monitors](#)
 and can be found in the [GSEOS Explorer Window](#)
.

notification or the alarm acknowledgement you can do this by disabling the 'Alarm Event Monitor' or the 'Alarm Acknowledgement Monitor' respectively.



Distributed Alarm Notification

Sometimes it is useful to set up multiple workstations to monitor an instrument. In this case it might be useful to be able to clear an alarm from any of the stations and have it cleared on all attached stations as well. This can be easily configured. If you define a block called AlarmAckRemote with the same layout as the AlarmAck block the alarm acknowledgement monitor will use this block just like the AlarmAck block to clear alarms:

```
AlarmAckRemote {
    Name [ 64 ]           ,,, 8 ;
    ID                   ,,, 32 ;
    AckAll              ,,, 8 ;
}
```

Now in order to generate this block you need to set up the according [network connections](#) [326]. The idea is to route the AlarmAck block from Station A to the AlarmAckRemote block of Station B and the AlarmAck from Station B to the AlarmAckRemote block of Station A. Accordingly if you want more than two stations participate in the distributed alarm management.

7.13 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 RecExport utility lets you dump recorder file contents to ASCII or binary files without using GSEOS to process with your off-line tools. The recorder file format has changed from version 5.1 to version 5.2. Version 5.2 and higher will still read the old format but versions 5.1 and lower won't be able to read 5.2 and up recorder files.

The user interface for the Recorder is 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.

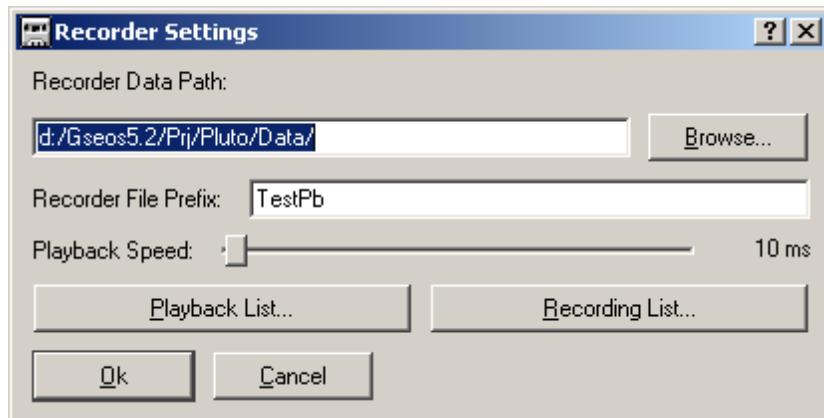
Settings	Recorder settings	This opens the Recorder Settings dialog.
More >>	Expand Recorder dialog	The More button expands the Recorder dialog and displays additional status information while recording or playing back.
<input type="checkbox"/> File	File mode	The file mode checkbox selects between automatic 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 ¹⁹³ file in the [Recorder] section with the entry FileSize. 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 switching is performed. This mode is more suitable for quick recording of short periods of time.



The navigation slider at the bottom of the recorder dialog lets you quickly navigate through your data. You can select any time base that you have configured and navigate through your data by that time base. For more details on how to set up time bases for your recorder data please refer to the [\[Timebase\]](#)¹⁹⁴ section of the gseos.ini file.

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 [\[Recorder\]](#)¹⁹³ section. The prefix is the string that will be prepended to all automatically generated recorder blocks.

The playback speed can be set to a value between 1ms and 1s and is the time between two blocks are being played back.

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](#)^[150]. 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 GesosRecorder module exports functions to perform most of the actions that can be accomplished with the Recorder dialog. Please refer to the [GseosRecorder](#)^[293] module for more details on how to script the recorder module.

7.14 The Options Dialog

The options dialog controls general GSEOS settings. Any modifications you make to the settings in the options dialog are written to the gseos.ini file to make them permanent. Some of the options only take effect after you restart the system. We will mention for each option for which this is the case.



The Stale Data Settings group controls when data is flagged as stale. You can enable/disable stale data notification on a global level. The global setting is overridden by the [individual settings](#)^[66] of a data item. In addition you can specify the global timeout period after which data is considered stale.

8 GSEOS Reference

This chapter describes the various configuration files and the GSEOS Python interface. Please refer to the subchapters for detailed information.

8.1 Command Line Options

Command line options let you configure GSEOS before starting up. This includes, setting a custom gseos.ini file, specifying the instance name, and setting a working directory.

See below of the options supported:

Option	Description
/D Path	Set Path as the GSEOS working directory.
/ini IniFile	The GSEOS initialization file (gseos.ini) to use.
/I Instance	Instance can be an instance number or a string.
/P Path	Add the path Path to the Python search path.

8.2 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 be changed in order to move the GSEOS directory to another location. The main directory that contains the gseos.exe executable is referred to as GSEOS root directory.

The GSEOS root directory contains all the GSEOS executable files like gseos.exe, 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 GSEOS. In case of multiple instrument support you might want to name the top folder after the mission or spacecraft, e.g.: New Horizons GSEOS. Each instrument should have it's own folder underneath the top level directory.

The Data directory is just by convention and receives the recorded files.

The Samples directory contains sample files.

The Doc directory contains this file which will be invoked from the GSEOS help. The System directory contains mostly executable system like dynamic library files.

A basic directory structure for a single instrument is shown below:



8.3 Configuration Files

GSEOS supports different configuration files for different tasks. The system configuration file is [gseos.ini](#)^[17]. Although you can specify a different file name by using the /ini switch from the command line. Check the [command line options](#)^[14] for more details.

GSEOS supports the following configuration files:

Configuration File Extensi Description

Type	on	
Alarm Limit Files ^[149]	*.alarm	Defines alarm limits that can be assigned to data items.
Block Definition Files ^[150]	*.blk	Defines your telemetry and data blocks.
Command Batch	*.cpb	Time tagged execution of commands.

Files	154	
Command Definition	*.cpd	Defines your commands.
Files	156	
Command Menu	*.cm	Hierarchical command interface in form of a menu.
Files	161	
Formula Definition	*.qlf	Defines conversion functions and expressions you can apply to your data items.
Files	168	
Status Definition	*.tr	Status text and image mappings for data items.
Files	196	

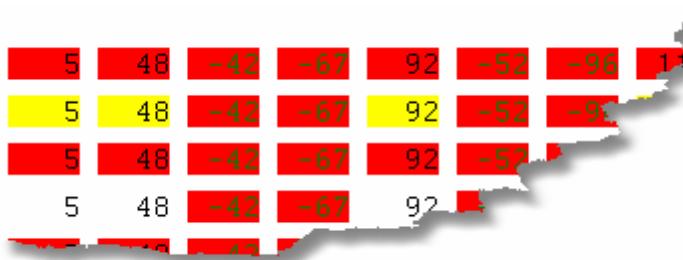
Loading of files

You can manually load most file types through the [File/Open](#) menu. However, most likely you want to load your configuration files automatically on system startup. You can enter any file you want to load at startup in the [\[Config\]/Load](#) section in the gseos.ini file. Block definition files (*.blk) have to be specified with the BlkFiles entry in the [Config] section.

8.3.1 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](#)] ¹⁷⁹ 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](#) ¹²¹. Once an alarm is loaded you can use it in your screen definition to apply it to a [data item](#) ⁶⁸.

8.3.2 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](#) ¹⁷⁹ 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](#) ¹⁹¹ 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) endianity or INTEL for Intel (little) endian. Motorola endianity means that the most significant bit resides on the highest address while the Intel endianity 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 endianity 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 endianity 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      Size  Gap      Endian  
                [Byte/Bit] [Bit] [Bit]  
-----  
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 has a resolution of 14 bit and Mass a resolution of 16 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] 16,  ,  , 14;
  Mass [10] 14, 1, 1, 16;
}

```

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 gap of 16 bits for the ToF elements to skip over the interlaced Mass elements and 14 bit for the Mass to skip over the according ToF elements.

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.

8.3.3 Command Batch Files (*.cpb)

Command batch files let you execute commands in a time driven manner. A simple ASCII file with *.cpb extension lists all the commands you want to execute prefixed with a time tag.

Comments can be inserted anywhere in the file. A comment starts with '#' character

and ends at the end of the line.

The format of each command line is:

after [ddd,] hh:mm:ss Command

Batch files can be started and stopped from the [GSEOS Command window](#)^[133]. You can also start a batch file programmatically with the [GseosCmd](#)^[236] module.

Although you can issue Python statements from a batch file a batch file operates line oriented, so you can not write a Python script in a batch file (although you can invoke a Python function).

Batch files run concurrently, that is you can start multiple batch files and they will run at the 'same' time. If you start a batch file from within a top level batch file (nested batch files) the top level batch file will not pause execution and wait for the nested batch file to complete but it will continue execution in parallel with the nested batch.

If you run nested batches (i.e. start a new batch file from within another batch file) you can specify the nested batch file to 'block', in this case the calling batch will wait for completion of the nested batch file before continuing. Note that you have to set the bBlock parameter for every batch file you want to block. For example if you have three nesting levels: Batch1 calls Batch2 which calls Batch3 and you call Batch3 from Batch2 using the following statement: GseosCmd.StartBatch('Batch3.cpb', True) Batch1 and Batch3 will execute concurrently and Batch3 will block Batch2 until it completes.

Example

The following code shows a sample batch file:

```
### ----- DPU processing-----
after 0,00:00:01 EPU_CNTRL(RAW_EVENTS)
after 0,00:00:01 EPU_CNTRL(CHEMS_ONLY)

### -----
### ----- CHEMS_ON (start)-----
### -----
### ----- Make sure HV is disconnected !!!-----
### -----
### -----
after 0,00:00:10 PWR_CNTRL(CHEMS_PROC, ON)
after 0,00:00:01 C_ANALOG(ALL, ON)
after 0,00:00:01 C_PHA_MODE(NORMAL)
after 0,00:00:01 C_EV_LOGIC(E_OR_T)
after 0,00:00:01 C_THRESHOLD(ALL, 36)
after 0,00:00:01 C_BIAS(ALL, ON)
after 0,00:00:01 RT_HV_ENABLE()
after 0,00:00:01 PWR_CNTRL(CHEMS_DPPS, ON)
after 0,00:00:01 C_HV_LIMIT(DPPS, 1)
after 0,00:00:01 C_HV_ENABL(DPPS, ON)
after 0,00:00:01 C_DPPS_FRZ(1)
```

8.3.4 Command Definition Files (*.cpd)

Command definition files define specifics about your commands. Commands are defined in an XML format as described below.

You can load as many command definition files as desired, new commands will simply be added to the already defined commands. The [GSEOS Explorer](#)^[124] lists all commands currently defined. All XML element and attribute names as described below are case sensitive. The command mnemonics and argument names you define are case sensitive as well.

If you specify a Default value for an argument you also have to specify a Keyword argument. If a Default value is specified you don't have to supply the value in the actual parameter list, in that case you won't be prompted for the argument.

When specifying absolute start bit positions for the individual arguments keep in mind that this is relative to the entire command including the opcode. Say your opcode is 32-bits wide the start bit position of the first argument would be 32. If you don't specify a start bit position the arguments are packed tightly. The byte order for multi-byte values is big endian (Motorola) which is MSB first.

You can define the entire command to use little endian (Intel) by specifying Endianity="Little" or Endianity="Intel". All multi-byte values (both command arguments as well as opcode) will be generated using little endian. You can also selectively specify the endianity for individual arguments therefore overwriting the default of big endian or if you defined the command to use little endian you can override individual command arguments to use big endian. The endian conversion is only applied to multi-byte arguments respectively. If you specify the endianity on an item that is not a multiple of 8-bits an error is generated.

You have to make sure that all bits in the command are defined. It is an error if there are bit gaps in your command.

Once you have defined your commands you can issue them using batch files, command menus, command buttons, scripts, etc. GSEOS parses the command string and generates the appropriate binary command representation (as defined with the command definition). It then publishes the BinCmd block which contains your binary command data. This block is then processed by the Bios and finally sent to the instrument using the instrument specific hardware.

Command Definition Syntax

All <Cmd> elements are child nodes of the <GSEOS> root node:

<Cmd>

The main command definition node.

Attributes:

Name	Format	Option	Default
		al	
Mnemonic	String	N	
Opcode	Int	N	
NumBits	Int	Y	16
Critical	Y, N	Y	N

Channel	Int	Y	0
Endianity	String	Y	Big
Description	String	Y	

Elements: <Arg>, <Const>, <Copy>, <Inv>, <Checksum>, <ZeroPad>, <CmdLen>

<Arg>

Defines an argument. Must be within a <Cmd> element.

Attributes:

Name	Format	Option Default	
		al	
Keyword	String	Y	
NumBits	Int	Y	16
StartBit	Int	Y	
Type	see below	Y	UNSIGNED
Endianity	String	Y	Big
Default	same as Type	Y	If a Default is specified the argument must be a Keyword argument and doesn't need to be specified when the command is issued.
ScaleFactor	Numeric	Y	1
Offset	Numeric	Y	0
DataRangeLo	Numeric	Y	
w			
DataRangeHi	Numeric	Y	
gh			
Description	String	Y	

Elements: <Enum>, <Poly>

<Enum>

Defines an enumerated argument mnemonic. Must be within an <Arg> node.

Attributes:

Name	Format	Option Default	
		al	
Name	String	N	
Value	same as Arg	N	16
Description	String	Y	

Elements: -

<Poly>

Defines an polynomial to convert the passed in argument accordingly. This element can only be specified if no ScaleFactor or Offset is set. Must be within an <Arg> node.

The resulting floating point value is converted according to the data type of the argument (i.e. if the argument data type is unsigned and the result of the polynomial conversion is -

4.356 the resulting value is -4. The polynomial conversion algorithm is:

Result := Coeff0 + Coeff1 * Value + Coeff2 * Value² + Coeff3 * Value³ + Coeff4 * Value⁴ + ...

Attributes:

Name	Format	Option Default	
		al	
Coeff0	Numeric	N	
Coeff1	Numeric	Y	0
Coeff2	Numeric	Y	0
Coeff3	Numeric	Y	0
Coeff4	Numeric	Y	0
Coeff5	Numeric	Y	0
Coeff6	Numeric	Y	0
Coeff7	Numeric	Y	0

Elements: -

<Const>

Defines a constant command value that is not specified when issuing the command. Must be within a <Cmd> node.

Attributes:

Name	Format	Option Default	
		al	
Value	same as Type	N	
NumBits	Int	Y	16
StartBit	Int	Y	
Type	see below	Y	
Endianity	String	Y	Big
Description	String	Y	

Elements: -

<Copy>

Copies bits from a previous bit position in the command. Must be within a <Cmd> node.

Attributes:

Name	Format	Option Default	
		al	
FromBit	Int	N	
NumBits	Int	Y	16
StartBit	Int	Y	

Elements: -

<Inv>

Inserts the inverted previous NumBits bits. Must be within a <Cmd> node.

Attributes:

Name	Format	Option Default
		al
NumBits	Int	Y 16

StartBit Int Y

Elements: -

<ZeroPad>

Pads the command to NumBits bits with zeroes. Must be within a <Cmd> node.

Attributes:

Name	Format	Option Default
		al
NumBits	Int	Y 16

StartBit Int Y

Elements: -

<Checksum>

Inserts a checksum of NumBits. The checksum is computed with the algorithm specified by Type, currently only XOR is implemented. The checksum is computed starting with FirstByte. Must be within a <Cmd> node.

Attributes:

Name	Format	Option Default
		al
NumBits	Int	Y 16

StartBit	Int	Y
Algorithm	String	N
FirstByte	Int	Y 0
Endianity	String	Big

Elements: -

<CmdLen>

Inserts the command length in WordSize units. Must be within a <Cmd> node.

Attributes:

Name	Format	Option Default
		al
NumBits	Int	Y 16

StartBit	Int	Y
WordSize	Int	Y 16
Endianity	String	Big

Elements: -

The allowed argument data types for commands are: UNSIGNED, SIGNED, FLOAT32_IEEE, FLOAT64_IEEE.

Example File

```
<!--
# **** IBEX EGSE Command Definitions. *
# ****
-->
<GSEOS Version="366">
    <Cmd Mnemonic="EGSE_SET_TIME" Opcode="0x0" NumBits="8" Channel="1"
Description="Set Time">
        <Arg Keyword="Time" NumBits="32" Description="Time"/>
    </Cmd>

    <Cmd Mnemonic="EGSE_SET_QUATERNION" Opcode="0x0" NumBits="5" Channel="1"
Description="Set Quaternion">
        <Arg NumBits="1" Description="Quaternion Type">
            <Enum Name="RAW" Value="0" Description="Raw Quaternion"/>
            <Enum Name="FILTERED" Value="1" Description="Filtered Quaternion"/>
        </Arg>
        <Arg NumBits="2" Description="Quaternion Number"/>
        <Arg NumBits="32" DataRangeLow ="20" DataRangeHigh="4000"
Description="Quaternion Value"/>
    </Cmd>

    <Cmd Mnemonic="EGSE_SET_SUBSECOND_TIME" Opcode="0x9" NumBits="8"
Channel="1" Description="Set Time">
        <Const Value="0" NumBits="16"/>
        <Arg Keyword="SubsecondTime" NumBits="16" Description="Subsecond Time"/>
    </Cmd>

    <Cmd Mnemonic="EGSE_ANCILLARY_ENABLE" Critical="Y" Opcode="0xA"
NumBits="8" Channel="1" Description="Enable/Disable Ancillary Packets">
        <Arg NumBits="32" DataRangeLow="0" DataRangeHigh="1"
Description="Enable/Disable">
            <Enum Name="ON" Value="1" Description="Enable Ancillary Packets"/>
            <Enum Name="OFF" Value="0" Description="Disable Ancillary Packets"/>
        </Arg>
    </Cmd>

    <Cmd Mnemonic="EGSE_SEND_ANCILLARY_PACKET" Opcode="0xB" NumBits="8"
Channel="1" Description="Send one Ancillary Packet">
        <Const Value="0" NumBits="32"/>
    </Cmd>

    <Cmd Mnemonic="EGSE_1PPS_ENABLE" Opcode="0xA" NumBits="8" Channel="1"
Description="Enable/Disable 1PPS">
        <Arg NumBits="32" DataRangeLow="0" DataRangeHigh="1"
```

```

Description="Enable/Disable">
    <Enum Name="ON" Value="1" Description="Enable 1PPS"/>
    <Enum Name="OFF" Value="0" Description="Disable 1PPS"/>
</Arg>
</Cmd>

<Cmd Mnemonic="EGSE_SET_SPIN_PULSE" Opcode="0x20" NumBits="8" Channel="1"
Description="Set Spin Pulse delay and period">
    <Arg Keyword="Delay" NumBits="16" Description="Start Delay in
Milliseconds from next 1PPS"/>
    <Arg Keyword="Period" NumBits="16" Description="Spin Period in
Milliseconds. If 0 turn Spin Pulse off after next 1PPS"/>
</Cmd>

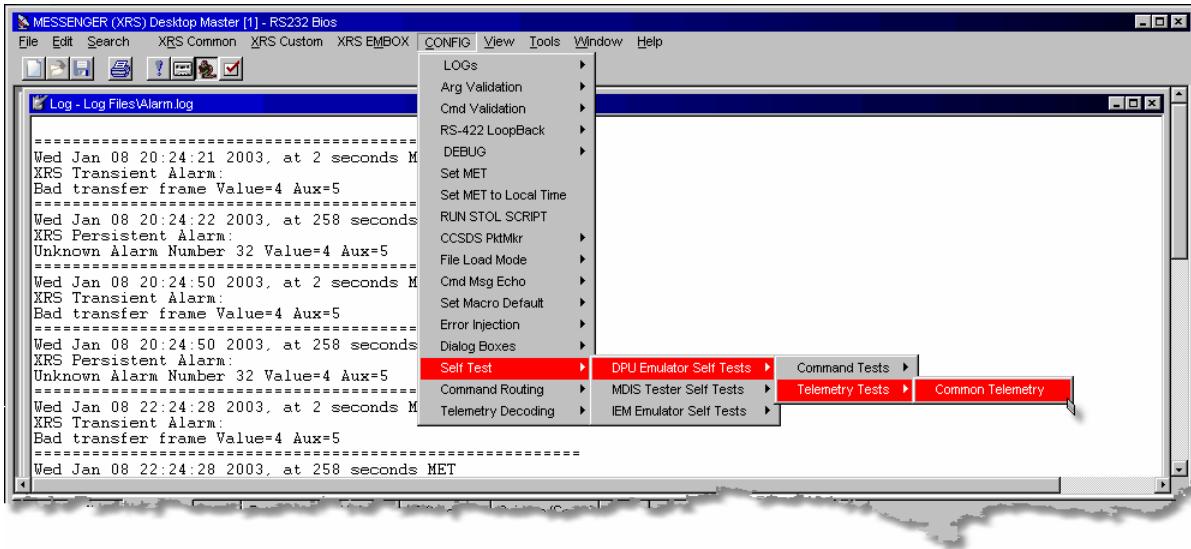
</GSEOS>

```

8.3.5 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 a flat ASCII files with the .cm extension. You can specify the command menu files to load in the Load entry of the [\[Config\]](#)¹⁷⁹ section of the [gseos.ini](#)¹⁷¹ 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 executed. 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. It is possible to issue arbitrary Python commands if the [ForwardToPython](#)¹⁷⁸ setting in gseos.ini is set. 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](#)⁵⁵ 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.

8.3.6 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](#)¹⁷¹ initialization file or at runtime from the [File/Open menu](#)²⁸¹.

The following subchapters explain the different configuration settings:

[Alarm Monitor configuration](#)¹⁶³

8.3.6.1 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 more actions. The trigger condition as well as the actions are defined in an 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 &lt; 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](#)¹⁶⁷ 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](#) 168 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. You only need to specify the name of the conversion function. Say you have defined the following conversion function:

EU('Analog.Temp1') you would list the <DataItem> node as follows:

```
<DataItem Name="Analog.Temp1" Conversion="EU" />
```

<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. 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.

- 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 note 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>
```

<AlarmWindow>

This element generates an AlarmEvent block which in turn activates the [Alarm Notification Window](#)^[142]. You can configure the items of the AlarmEvent block. The Name attribute defines the alarm name. The Level attribute maps to the according alarm level and the ID attribute defines the event ID. The **<Text>** node makes up the Alarm Event description. The alarm levels should be one of the following: "YellowLow", "YellowHigh", "RedLow", "RedHigh". The ID attribute is optional and defaults to 0.

```
<AlarmWindow Description="A red high alarm has occurred."
Level="RedHigh" ID="55"/>
```

<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. The **FileName** can either be a simple Log file name or the name of an [automatic log](#)^[143].

```
<LogFile FileName="MIMIAlarmLog.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 recommended to set a quota since an ill-behaved alarm monitor could possibly generate huge numbers of emails.

```
<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>
    This is the message body. In addition this text will be prefixed
    by
    some general information about the alarm.
  </Body>
</Email>
```

8.3.6.1.1 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
-->
<!-- =====-->
<!-->

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

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

      <Python Function = "Gseos.PlaySound('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>
```

8.3.7 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](#)⁵⁷ 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](#)¹⁷⁹ file. Once the formula is loaded it can be accessed from GSEOS when displaying an Expression or in the [item select dialog](#)⁵¹ when selecting a data item.

The loaded formulas are also displayed in the GSEOS Explorer underneath the nodes [Conversions](#)¹²⁵ and [Expressions](#)¹²⁷.

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 GseosConversion module. The two examples defined above can be accessed from Python in the following way:

```
GseosConversion.Linear(2.3, PHA.TOF1, 20)
GseosConversion.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](#)^[246] 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](#)^[57]. 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](#)^[28] function or at startup time in the gseos.ini [Config/Load](#)^[179] section. Once the functions are loaded they are available to be placed on display screens as [Expressions](#)^[57] or [Conversion Function items](#)^[57]. They can also be accessed from Python via the GseosConversion module. The module GseosConversion 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 (f1NadirCycle) := ltof(f1NadirCycle)
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)

```

8.3.8 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.

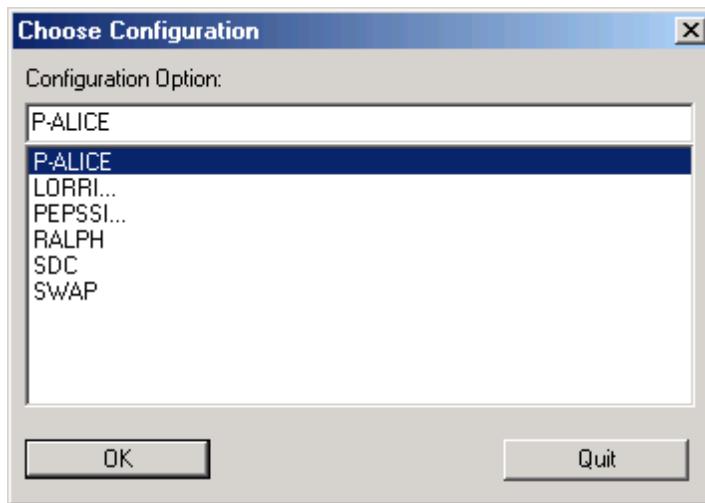
```
[MOC.Project]  
[MOC.Config]
```

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...
CmdLine = /I Master /ini i_LORRI/LORRI_ConfigFiles/gseos.ini

Option = PEPSSI...
CmdLine = /I Master /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.

[BinaryStreamers]	[174]
[Buffer]	[178]
[Command]	[178]
[Config]	[179]
[Console]	[179]
[FileUploads]	[180]
[Instance]	[182]
[License]	[183]
[Logs]	[183]
[Message]	[186]
[Net]	[187]
[Project]	[191]
[PyStartup]	[191]
[QLook]	[192]
[Recorder]	[193]
[System]	[195]

8.3.8.1 **BinaryStreamers**

The [BinaryStreamers] section defines all binary streamers. The keys you specify in this section are the names of the binary streamers you want to configure. The value must be: Writer.

```
[BinaryStreamers]
StreamTLM    = Writer
```

```
HK_Writer    = Writer
RawStreamer = Writer
```

The above example configures three binary streamers. You configure each streamer with a separate section where the section name is the name of the binary streamer. Below we configure the [StreamTLM]:

```
[StreamTLM]
PathTemplate=Year_%Y/Month_%m/
FileTemplate=Bin0_%H%M.log
PathRolloverSession = Disabled
PathRolloverMaxFiles = 1000
FileRolloverSession = Enabled
FileRolloverMaxSize = 10MB
Block             = TLM
Enabled           = Yes
```

The section above defines the StreamTLM binary streamer. Binary streamers use the same path and file-rollover logic as [log files](#)^[18].

A binary streamer log can have a path template and a file template. Both path and file can roll over independently depending on the date/time and/or session status or file limits. The templates are filled in with the actual date and time information when the directory/file is created. You can also specify reasons for rolling over a directory, like a maximum number of files within the directory or every session start. The file roll-over can be specified in a similar way and includes a maximum file size. If a file or directory rolls over and the current date and time settings would result in the same directory or file name a counter is appended to the directory or file name in the form of: _PartNNNN. So if the file Bin_26_102204.log already exists and a new log file has to be opened due to the roll-over conditions the new file is named: Bin_26_102204_Part0000.log. When the directory rolls over the current file is closed automatically and a new file is created in the new directory. Both the PathTempate and FileTemplate entries take strings that support the following formatting characters:

Format Character	Description
%a	Locale's abbreviated weekday name.
%A	Locale's full weekday name.
%b	Locale's abbreviated month name.
%B	Locale's full month name.
%d	Day of the month as a decimal number [01,31].
%H	Hour (24-hour clock) as a decimal number [00,23].
%I	Hour (12-hour clock) as a decimal number [01,12].
%j	Day of the year as a decimal number [001,366].
%m	Month as a decimal number [01,12].
%M	Minute as a decimal number [00,59].
%p	Locale's equivalent of either AM or PM.
%S	Second as a decimal number [00,61].
%U	Week number of the year (Sunday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday are considered to be in week 0.
%w	Weekday as a decimal number [0(Sunday),6].
%W	Week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year

%y
%Y
preceding the first Monday are considered to be in week 0.
Year without century as a decimal number [00,99].
Year with century as a decimal number. %Z Time zone name
(no characters if no time zone exists).

Entry	Description
PathTemplate	The path template. This can contain any of the above formatting characters and can be of any depth the file system allows. If you have a static directory it is not required to use any of the formatting characters. When a new directory needs to be created as determined by the roll-over conditions the new directory will be created based on this template filled in with the actual date and time information. The path template must not contain the file name, this is specified with the FileTemplate setting as described below.
FileTemplate	The file template. Similar to the path template this entry specifies the file template. Based on this template the actual file name is determined from the current date and time. If no date and time dependent file name is required this can be a fixed name (i.e. not containing any format characters). This template should not contain any directory information as this is specified with the PathTemplate entry.
PathRollOverSession	This entry determines how the directory is rolled over given a new session. A new session is defined as the binary streamer being started up. This entry can be one of three settings: "Enabled", "Disabled", "Only". The default is "Disabled". If the PathRollOverSession is set to "Enabled" the directory is rolled over with every new session (restart of the binary streamer). When the streamer is started up and the template path (as filled in with the current date and time information) exists we create a new path appending/incrementing a path counter (_PartNNNN). If the template path doesn't yet exist it is created. If the PathRollOverSession is set to "Disabled" we don't roll over the directory on a new session if the path already exists. If it doesn't exist a new path is created according to the current date and time information. If the PathRollOverSession is set to "Only" we create the directory like in the "Enabled" case, however, if the directory would roll over based on the changing date and time information we don't create a new directory but stay with the original one. This effectively only creates a new directory on session start.
PathRollOverMaxFiles	In addition to the above settings the directory can roll over when the maximum number of files is reached. By default there is no limit (other than the OS limitation) on the number of files in the directory. For PathRollOverSession settings of "Enabled" and "Disabled" a new directory based on the PathTemplate is created. If

	PathRollOverSession is set to "Only" and the maximum number of files is reached the current directory name is retained but a counter appended or incremented.
FileRollOverSession	<p>This setting is similar to the PathRollOverSession, except that it relates to the file name as determined by the FileTemplate setting. This entry determines how the file is rolled over given a new session. This entry can be one of three settings: "Enabled", "Disabled", "Only". The default is "Disabled".</p> <p>If the FileRollOverSession is set to "Enabled" we roll over the file with every new session. When the streamer is started up and the template file (as filled in with the current date and time information) exists we create a new file appending/incrementing a path counter (_PartNNNN). If the template file doesn't yet exist it is created.</p> <p>If the FileRollOverSession is set to "Disabled" we don't roll over the file on a new session if the file already exists. If it doesn't exist a new file is created according to the current date and time information.</p> <p>If the FileRollOverSession is set to "Only" we create the file like in the "Enabled" case, however, if the file would roll over based on the changing date and time information we don't create a new file but stay with the original one. This effectively only creates a new file on session start.</p>
FileRollOverMaxSize	<p>This setting is similar to the PathRollOverMaxFiles, except that it rolls over the file name and the limit is the size of the log file. This entry is in bytes. You can append KB, MB, or GB to indicate kilobytes, megabytes, or gigabytes. The value can be a floating point number (i.e. 10.2MB). The default (0) is no file size limit.</p> <p>If the maximum file size is reached according to this setting the file will roll over. For FileRollOverSession settings of "Enabled" and "Disabled" a new file based on the FileTemplate is created. If FileRollOverSession is set to "Only" and the maximum file size is reached the current file name is retained (i. e. with the date and time information of the time of the file creation) but a counter appended or incremented.</p>
Block	The block to stream.
VariableLen	Yes or No. If the block specified with the Block entry has an item Len as the first item and is 32-bit wide you can set the VariableLen entry to Yes. In this case only the number of bytes indicated by the Len field (not including the Len field itself) is written to disk.
Enabled	Yes or No. If set to Yes the binary streamer starts up automatically on system startup.

8.3.8.2 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](#)^[121].

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 4096 (4 MB).

Example

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

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

8.3.8.3 Command

The [Command] section allows you to configure the command module.

Entry	Description
Python	[Enable Disable]. Default is Enable. By default the only commands accepted are the commands defined with a command definition. For backward compatibility you can turn on Python forwarding. In this case if a command is not found in the command dictionary it will be forwarded to the Python interpreter. This allows you to issue regular Python statements from command sources like buttons, menus, batch files, etc. If the Python command raises an exception, this exception is passed on to the caller.
CriticalCmdDialog	[Enable Disable]. Default is Enable. Critical commands need to be acknowledged by the operator by confirming a dialog. For some automated tests this might not be a suitable setup. If you disable critical command notification critical commands will be executed without this confirmation. This is a potentially dangerous setting a popup dialog will remind you at every system start that critical command notification is disabled. It is also possible to suppress critical command notification on a command by command level. Check the GseosCmd ^[236] module for more details.
QueryForDefaultArgs	[Yes No]. Default is No. If your command contains default arguments they will be used if no argument is provided for this command argument. If you set this setting to Yes the parameter query dialog is popped up so you can modify the arguments.
Log	Log name (name of a log file or an automatic log). If set commands are logged to this log.

LogToMsg	[Yes No]. Default is No. If enabled commands are logged to the message window.
CmdHistoryDialogOnly	[Yes No]. Default is No. If enabled only commands issued from the command dialog are added to the command history.

8.3.8.4 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	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: *.cpd: Command definition file *.cpb: Command batch file *.cm: Command menu file *.dt: Desktop *.scr: Screen file *.log: Log file *.py?: Python module

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 (autodeskN.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. Loading the desktop file, if specified, is always deferred to the end of the load sequence.

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
```

8.3.8.5 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 200KB.
FileName	The name of the console window file. Defaults to ProjectName[Instance].con if not specified, where ProjectName is the name of the project as configured in the [Project] section. If the console window file is not writeable (read-only) the contents of the console window are not logged.

8.3.8.6 FileUploads

The [FileUploads] section allows you to configure your custom file or memory uploads. See the [FAQ on how to implement a file upload](#) for more details. The configuration of file uploads consists of two levels, the first one defined in the [FileUploads] section defines all file uploads. Subsequent sections with the same name as the file uploads defined in the [FileUploads] section further specify the details of each file upload. To access the file upload programmatically you can use the [GseosFileUpload](#) module. The first argument (the file upload name) to pass in any of the functions in the GseosFileUpload module is the name you specify in the [FileUploads] section (in our sample that would be: BinLoad, FU2, FU0).

```
[FileUploads]
BinLoad=FileUpload
FU2=FileUpload
FU0=FileUpload

[BinLoad]
Description=Binary File Upload
Extension=bin
Timer=200
FileUploadStartHandler=MyFileUpload.fOnStart
FileUploadTimerHandler=MyFileUpload.fOnTimer

Arg0 = StartAddr:
Arg1 = Memory: FLASH, RAM, EEPROM

[FU2]
Description=File Upload 2
Extension=fu2
Timer=200
FileUploadStartHandler=MyFileUpload.fStartHandler
FileUploadTimerHandler=MyFileUpload.fTimerHandler
Arg0 = First Arg: 99..299
Arg1 = Second Arg: One, Two, Three, Four, Five, Six, Seven, 10
Arg2 = Third Arg: One, Two, Three, Four, Five, Six, Seven, 10
Arg3 = Fourth Arg:
```

```

Arg4 = Last Arg: 0..77

[FU0]
Description=File Upload 0
Extension=fu0
Timer=0
FileUploadStartHandler=MyFileUpload.fStartHandler
FileUploadTimerHandler=MyFileUpload.fTimerHandler
Arg0 = First Arg: 99..299
Arg1 = Second Arg: One, Two, Three, Four, Five, Six, Seven, 10
Arg2 = Third Arg: One, Two, Three, Four, Five, Six, Seven, 10
Arg3 = Fourth Arg:
Arg4 = Last Arg: 0..77

```

The above example configures three file uploads. Each file upload handles a different type of file indicated by a specific file extension. The details of each file upload configuration are outlined below:

Entry	Assignment
Description	This should be a brief description of the file upload type. This will show up in the File/Open dialog File Type setting.
Extension	The file extension, this should be a three character extension that doesn't conflict with any of the other GSEOS common extensions. The '.' (dot) must not be included in the file extension.
Timer	The time out interval in ms. The FileUploadTimerHandler routine (see below) is called whenever this timer expires.
FileUploadStartHandler	This entry must specify your callback function that is called when the user starts a file upload. This function must take two arguments, the first one is the absolute file name of the file the user has selected, the second is a list of arguments the user has selected. Even if there is only one argument it will be delivered as a list with one argument. If the user specifies arguments from a select list they will be provided as strings, numeric arguments will be checked against the proper range and be forwarded as integer. You can specify a module and package name in typical Python namespace syntax. You have to make sure that this name is accessible (i.e. the module is loaded). If this function returns False the file upload is aborted, if True the upload timer is started and the FileUploadTimerHandler function called. If it returns a string the string is interpreted as an error string and the argument query dialog is popped up again displaying the error and let the user correct the argument entry.
FileUploadTimerHandler	This entry sets the timer callback. In this function you should service the file, read the next chunk of data and prepare and issue the command(s) as required. This function doesn't take any parameters, it is your responsibility to keep references to the open file, and any other data you need to process the file. Your function should return True if there is more data to be processed, it should return False when you are done with the file. Once this function returns False the file upload is finished and your routine won't be called until the user uploads another file.

Arg0...Arg4

You can configure up to five arguments the user can specify when choosing a file to upload. Each Argument takes the prompt followed by a colon and either a select list or a range. A select list is a comma delimited list. A range is indicated by two ellipses (n..m). If neither a list nor a range is specified the argument will simply prompt for any string. If given a range the user input will be verified against the range (both ends are inclusive), if a select list is provided the user can choose one of the settings specified. If a range is specified only decimal integer values can be entered by the user. If you require hex values you can simply use a string argument and interpret the value in your handler. The user entries will be passed as a list as the second argument to the FileUploadStartHandler callback function you supplied.

8.3.8.7 GseosGraph

The [GseosGraph] section allows you to control the handling of plot windows.

Entry

Port

Assignment

The GseosGraph process server port. The default is 9999. Can be overridden with this entry.

LogFile

This entry specifies a log file for any debug info the GseosGraph process might log. The default log file is GseosGraph.log.

LogLevel

This entry controls what to log. The available log levels are: Critical, Error (the default if not specified), Warn, Info.

Verbose

If set to Enabled GSEOS prints verbose info to the message window.

8.3.8.8 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

[Section]

Description

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]
;
Project      1st     2nd     3dr     4th     ...   instance
= 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
HSSInterrupt    = 11

[Bios2]
IOBaseAddress   = 0x140
HSSInterrupt    = 10

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

[Rec1]
DataPath         = ..\write
FileSize         = 1024

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

```

8.3.8.9 License

The [License] section specifies the licensee and the mission this version of GSEOS is licensed to.

Entry	Description
Licensee	The licensee name.
Mission	The mission this version of GSEOS is licensed for.

8.3.8.10 Logs

The [Logs] section defines all [automatic logs](#)^[18]. The keys you specify in this section are the names of the logs you want to configure. The value must be: Log.

[Logs]

```
STOLLog = Log
CmdLog  = Log
ErrLog  = Log
```

The above example configures three automatic logs. You configure each log with a separate section where the section name is the name of the log. Below we configure the [STOLLog]:

```
[STOLLog]
PathTemplate      = Logs/STOL/Year_%Y/Month_%m/
FileTemplate      = STOLLog_%02d_%H%M%S.log
PathRollOverSession = Enabled
PathRollOverMaxFiles = 1000
FileRollOverSession = Enabled
FileRollOverMaxSize = 10MB
```

The section above defines the STOLLog automatic log.

A specific log can have a path template and a file template. Both path and file can roll over independently depending on the date/time and/or session status or file limits. The templates are filled in with the actual date and time information when the directory/file is created. You can also specify reasons for rolling over a directory, like a maximum number of files within the directory or every session start. The file roll-over can be specified in a similar way and includes a maximum file size. If a file or directory rolls over and the current date and time settings would result in the same directory or file name a counter is appended to the directory or file name in the form of: _PartNNNN. So if the file STOLLog_26_102204.log already exists and a new log file has to be opened due to the roll-over conditions the new file is named: STOLLog_26_102204_Part0000.log. When the directory rolls over the current file is closed automatically and a new file is created in the new directory.

Both the PathTemplate and FileTemplate entries take strings that support the following formatting characters:

Format Character	Description
%a	Locale's abbreviated weekday name.
%A	Locale's full weekday name.
%b	Locale's abbreviated month name.
%B	Locale's full month name.
%d	Day of the month as a decimal number [01,31].
%H	Hour (24-hour clock) as a decimal number [00,23].
%I	Hour (12-hour clock) as a decimal number [01,12].
%j	Day of the year as a decimal number [001,366].
%m	Month as a decimal number [01,12].
%M	Minute as a decimal number [00,59].
%p	Locale's equivalent of either AM or PM.
%S	Second as a decimal number [00,61].
%U	Week number of the year (Sunday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday are considered to be in week 0.
%W	Weekday as a decimal number [0(Sunday),6].
%w	Week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday are considered to be in week 0.

%y	Year without century as a decimal number [00,99].
%Y	Year with century as a decimal number. %Z Time zone name (no characters if no time zone exists).

Entry	Description
PathTemplate	<p>The path template. This can contain any of the above formatting characters and can be of any depth the file system allows. If you have a static directory it is not required to use any of the formatting characters. When a new directory needs to be created as determined by the roll-over conditions the new directory will be created based on this template filled in with the actual date and time information. The path template must not contain the file name, this is specified with the FileTemplate setting as described below.</p>
FileTemplate	<p>The file template. Similar to the path template this entry specifies the file template. Based on this template the actual file name is determined from the current date and time. If no date and time dependent file name is required this can be a fixed name (i.e. not containing any format characters). The file name extension must be .log or .html. This template should not contain any directory information as this is specified with the PathTemplate entry.</p>
PathRollOverSession	<p>This entry determines how the directory is rolled over given a new session. A new session is defined as the system being started up. This entry can be one of three settings: "Enabled", "Disabled", "Only". The default is "Disabled".</p> <p>If the PathRollOverSession is set to "Enabled" we roll over the directory with every new session (restart of GSEOS). When GSEOS is started up and the template path (as filled in with the current date and time information) exists we create a new path appending/incrementing a path counter (_PartNNNN). If the template path doesn't yet exist it is created.</p> <p>If the PathRollOverSession is set to "Disabled" we don't roll over the directory on a new session if the path already exists. If it doesn't exist a new path is created according to the current date and time information.</p> <p>If the PathRollOverSession is set to "Only" we create the directory like in the "Enabled" case, however, if the directory would roll over based on the changing date and time information we don't create a new directory but stay with the original one. This effectively only creates a new directory on session start.</p>
PathRollOverMaxFiles	<p>In addition to the above settings the directory can roll over when the maximum number of files is reached. By default there is no limit (other than the OS limitation) on the number of files in the system. For PathRollOverSession settings of "Enabled" and "Disabled" a new directory based on the PathTemplate is created. If PathRollOverSession is set to "Only" and the maximum number of files</p>

is reached the current directory name is retained but a counter appended or incremented.

FileRollOverSession This setting is similar to the PathRollOverSession, except that it relates to the file name as determined by the FileTemplate setting. This entry determines how the file is rolled over given a new session. This entry can be one of three settings: "Enabled", "Disabled", "Only". The default is "Disabled".

If the FileRollOverSession is set to "Enabled" we roll over the file with every new session (restart of GSEOS). When GSEOS is started up and the template file (as filled in with the current date and time information) exists we create a new file appending/incrementing a path counter (_PartNNNN). If the template file doesn't yet exist it is created.

If the FileRollOverSession is set to "Disabled" we don't roll over the file on a new session if the file already exists. If it doesn't exist a new file is created according to the current date and time information.

If the FileRollOverSession is set to "Only" we create the file like in the "Enabled" case, however, if the file would roll over based on the changing date and time information we don't create a new file but stay with the original one. This effectively only creates a new file on session start.

FileRollOverMaxSize This setting is similar to the PathRollOverMaxFiles, except that it rolls over the file name and the limit is the size of the log file. This entry is in bytes. You can append KB, MB, or GB to indicate kilobytes, megabytes, or gigabytes. The value can be a floating point number (i.e. 10.2MB). The default (0) is no file size limit.

If the maximum file size is reached according to this setting the file will roll over. For FileRollOverSession settings of "Enabled" and "Disabled" a new file based on the FileTemplate is created. If FileRollOverSession is set to "Only" and the maximum file size is reached the current file name is retained (i. e. with the date and time information of the time of the file creation) but a counter appended or incremented.

8.3.8.11 Message

The [Message] section configures settings for the Message module. You can set the message file to use with the FileName entry. It is possible to restrict the message file size with the MaxFileSize entry. If not set or set to 0 the message file will not be limited.

Entry	Description
FileName	The file name of the message file. If not specified defaults to ProjectName[Instance].msg where ProjectName is the name of the project as configured in the [Project] section.
MaxFileSize	The maximum file size in kB. If not specified no file limit will be imposed

and the message file grows unrestricted. You can append either kB or MB to the limit to indicate the units. If not specified the units default to kB.

Filter	Allows to filter messages. There can be zero, one, or more Filter entries. Each entry takes a source and message level separated by comma. If a message matches the source string and the message level is equal or less than the specified level the message is suppressed. Valid levels are: INFO, WARN, ERROR. The source string can contain the ? and/or * wildcards. The ? character matches any character, the * character matches any number of characters. You can optionally specify a message text and event ID to match. If you specify one or both they must match in order to suppress the message. If you want to specify an event ID without specifying the message text simply leave the field for the message text blank but use the comma: Filter = Src, ERROR, , 55
--------	--

Example

```
[Message]
FileName      = Pluto.msg
MaxFileSize   = 8192    # Limit to 8 MB
Filter        = Debug, INFO
Filter        = Net, INFO, Client*
Filter        = User, ERROR, Spacewire Link*
Filter        = T_?, INFO, Sequence error,44
```

8.3.8.12 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. You can configure any number of network connections. Each network connection can be associated with at most two blocks. The source block is the outgoing block, the destination block is the captured incoming data. 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 (or both) 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.

There are two general network configuration options you can set in the [Net] section as well. These are the polling intervals for the servers and clients respectively. The value you specify is the polling interval in ms. The default is 10ms for both servers and clients. The smaller this number is the more frequently the network connections will be serviced. If the throughput is not sufficient you might consider setting this number smaller. The disadvantage of a smaller value is the system overhead.

```
[Net]
ServerPollInterval=50
ClientPollInterval=50
```

```

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](#)^[12]. 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 required 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. DNS names are supported but require a DNS lookup which might be slow or fail. Although server connections don't need this entry you can optionally supply the server IP address with this entry. This is useful in multi-hosted machines (computers with more than one NIC card), that way you can specify the network you want the server to bind to. If you don't specify this entry on a multi-hosted machine the OS will pick the network to bind to. If you don't have a multi-hosted computer you should not specify the IP-Address entry.
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	<p>['Yes' 'No']. 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.</p> <p>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.</p>
Exclusive	['Yes' 'No']. 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.
MaxReceiveRate	This setting determines the maximum receive rate in bytes/sec. If 0 (the default) no receive throttling is implemented. This setting might be useful for servers which generate bursty data like playback requests from MOC servers, etc. Keep in mind that the server needs to buffer the data accordingly and not all servers might implement this functionality which might lead to data loss.
KeepAlive	['Yes' 'No']. Sets the keep alive flag for this connection. TCP/IP will send a periodic keep alive signal on the connection.
IdleReconnectTimeout	Only for client connections. Some routers drop connections

after a long time of inactivity. The KeepAlive setting might help in this case. However, if the router still drops the connection you can use the idle reconnect option. After the idle timeout configured (in minutes) of inactivity the connection is dropped by GSEOS and a reconnect is attempted. If you have configured a delay (see below) the reconnect attempt will be delayed for the number of seconds configured. If 0 (the default) no idle timeout is configured.

IdleReconnectDelay	Only for client connections. If an idle reconnect is configured this setting sets the delay until reconnect after the connection has been dropped by GSEOS due to an idle timeout. If 0 (the default), no reconnect delay is configured.
--------------------	--

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 were 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
```

Note

Although it is possible to specify an IP-Address by name we advise against it. The name resolution requires a DNS lookup which is a blocking call and might take a long time, especially for hosts that can't be found or when no DNS server is available. In this case it will look like GSEOS is frozen until the DNS lookup times out.

8.3.8.13 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 ^[160] . (e.g. for the project 'image' the blk-file must be named 'image.blk'). If you specify block definition files in the Config ^[179] 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 an image file (*.bmp, *.gif, *.jpg, *.png) file 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
```

8.3.8.14 PyStartup

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

Entry	Description
LibDir	This entry sets a user path for 3rd party extension packages. You can specify one or more (or of course none) directories. They should be subdirectories of your GSEOS root directory and therefore relative paths. We suggest to specify at most one directory to avoid clutter. Your packages will be subdirectories in this folder. This path is scanned on startup for package and any import errors will be reported in the message window. Mind you that modules placed into this folder will not be picked up.

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 ^[329] .

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 *
```

8.3.8.15 QLook

The [QLook] section configures some screen file (Quick Look) properties. Most of these options can be configured through the user interface and will be written automatically to gseos.ini. The Size entry determines how large the screen canvas is. There are system dependent limitations and the largest screen size is 4096x4096. The default is 1024x1024. The other settings configure the screen grid as well as global stale settings. The font settings allow you to choose a default font, that is the font that is applied to all text objects without explicitly choosing a font. Please make sure the font is available on all platforms you want to use.

Entry	Description
Size	[Width, Height]. Default is 1024, 1024. The size of the screen canvas in pixels. The max is 4096, 4096. For performance reasons you should use the smallest size that meets your needs.
Grid	[Width, Height]. Default is 6, 6. This is the size of the screen grid.
SnapToGrid	[Yes No]. Default is Yes. If enabled snaps the object to the grid.
ShowGrid	[Yes No]. Default is Yes. If enabled shows the grid lines for easier object placement.
StaleTimeout	Default is 300. The global stale timeout in seconds.
StaleEnabled	[Yes No]. Default is Yes. Enables/disables global stale notification.
DefaultFontName	The name of the default font. The system default is Courier.
DefaultFontSize	The size of the default font. The system default is 12.

Example

```
[QLook]
Size = 2000, 3000
Grid = 10, 10
SnapToGrid = No
ShowGrid = No
```

```

StaleTimeout = 12
StaleEnabled = No
DefaultFontName = Courier New
DefaultFontSize = 10

```

8.3.8.16 Recorder

The [Recorder] section is used to control the recorder module. Another section that is used in conjunction with the recorder is [\[Timebase\]](#)¹⁹⁴.

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 ^[146] .
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 ^[146] .
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] ^[194] 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
```

8.3.8.16.1 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 [\[Recorder\]](#)^[193] section for other recorder specific configuration settings. The time base data is recorded as a Python long value and can be converted back to a readable format by the supplied format function.

Entry	Description
Name	The name is used as a label in the user interface. Please keep it relatively short since it will determine the size of the recorder dialog. 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 your 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. Both the evaluation of your time as well as the formatting into a readable format are executed within the GseosConversion module. That means you have all block definitions as well as your formulas and all other GseosConversion attributes (like the module time) available for your time conversion functions. If errors occur during the conversion they are logged to the message window.

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.localtime(Time))
```

8.3.8.17 System

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

Entry	Description
Configurable	[Yes No]. Default is Yes. If you specify 'Yes' the system allows the full flexibility and is interactively configurable by the user. This includes 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	[Enabled Disabled]. Sets the network as the default data source. By default the network is not enabled. You can also enable the network once the system is running using the GSEOS Explorer ¹²⁸ .
BlockListing	[Yes No]. This setting enables or disables the block listing capability. See also the Block Definition File ¹⁵⁰ 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.

ConfirmTermination	[Yes No]. 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	[Yes No]. The default is No. 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 ¹⁷⁹ 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.
ProcessTasksTime	The ProcessTasksTime specifies the number of ms the system will process GSEOS tasks before yielding the processor. The default is 20ms. The smaller this number the more responsive the application will be in high load situations at the expense of processing incoming data. If this number is larger the system will spend more time processing incoming data and be less responsive to user input.
Profiling	[Yes No] It turned on the GSEOS Explorer will display decoder and monitor profiling statistics. This is useful when debugging the performance of your decoder or monitor scripts. Turning profiling on with increase overhead slightly, so once your decoders/monitors are working well you might want to turn profiling off. By default this option is turned off.

Example

```
[System]
Net = Enabled
ConfirmTermination=Yes
```

8.3.9 Status Definition Files (*.tr)

Status definition files (formerly Text Reference Files) let you map numeric values to text or images for display. This is especially useful for status values. Besides the text representation a color code can be defined with the status value. If you display a data item represented as a [Status Text](#)¹⁷⁹ the colors will be displayed accordingly. Status Image items will be scaled to the largest image within a Status Image definition.

Status Text

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.

The ranges within a Status Definition must be unique, if there is overlap an error is reported. You can append Status Definition Files and therefore split your definitions over various files. However, if you define a Status Definition with the same name the two

definitions are merged. This way you can even split individual definitions over several files. Keep in mind that the unique range requirement still applies.

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

```
#  
# 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;  
           7-10, " Acquire-Histo ", " RED on WHITE}
```

Status Image

Status images are defined similarly to the Status Text items above. However, instead of

listing each image file we define symbolic image names that we will use in the actual definitions. See below for an example:

```

Image: Name="Green"      File="Green.jpg"
Image: Name="YellowLow"   File="YL.jpg"
Image: Name="YellowHigh"  File="YH.jpg"
Image: Name="RedLow"      File="RL.jpg"
Image: Name="RedHigh"     File="RH.jpg"

ImageMap: Alarms {
    0, Green;
    1, YellowLow;
    2, YellowHigh;
}

ImageMap: Errors {
    0-12, Green;
    13-15, RedHigh;
}

```

8.4 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](#)^[135]. 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](#)^[246], [Monitor](#)^[285], and general GSEOS functionality. 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](#)^[150] 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 SendBlock() command:

```
PHA.SendBlock()
```

Python structures it's modules in namespaces. The default namespace is `__main__`, this is the namespace all the block definitions are imported to. If you want to access blocks from your own scripts you will have to import these blocks from the `__main__` module. So your scripts should have something like this in the beginning:

```
from __main__ import *
```

The command handling is implemented in the [GseosCmd](#)²³⁶ namespace. To use the GseosCmd module in your own scripts you have to import it, e.g.:

```
import GseosCmd
GseosCmd.ExecCmd("DEV_POWER(ON)")
```

The following subchapters describe the various interface modules in detail.

8.4.1 Modules

Access to GSEOS functionality is grouped into modules. Please refer to the subchapters for details on the available modules. The `__main__` module exports all your block definitions so you can simply access them from the interpreter prompt in the [Console window](#)¹³⁵.

In general you can get information about the individual modules by entering:

```
print ModuleName.__doc__
```

Where ModuleName is the name of the module you would like information for. The same is true for functions within a module.

```
dir(ModuleName)
```

will give you an overview of the defined functions, you can then specify

```
print ModuleName.FunctionName.__doc__
```

to get information about the function in question.

GSEOS Naming Convention

Module names start with Gseos... (GseosCmd, GseosRecorder, ...). In general functions use capitalized words like (LogSave()). Classes start with the letter T (TDecoder, TMonitor, ...) and are in general named like the module they are contained in, e.g. GseosDecoder.TDecoder. This is not always the case, especially if there is more than one class defined in a module. All exception classes are defined in [GseosError](#)²⁵⁶. They have

the format of TExceptionNameError where ExceptionName is the particular exception name, ususally the same as the class that might generate the error. All exceptions are derived from TGseosError and you should be able to catch all GSEOS specific exceptions with an exception handler that catches TGseosError. Of course, other Python exceptions might be raised depending on your use of the language.

8.4.1.1 Gseos

The Gseos module implements several utility functions related to the GSEOS application. You should `import Gseos` whenever you need to access application specific services. The function [help\(\)](#)²⁰⁷ 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__`, where 'function' is the function you would like to get documentation for.

The [FileMenu\(\)](#)²⁰² function allows you to hook your own file types into the GSEOS file handling. The [Log\(\)](#)²⁰⁹ and [LogSave\(\)](#)²¹⁰ functions are used to append text to a GSEOS log and to save the log file respectively. The [MessageBox\(\)](#)²¹³ and [InputDialog\(\)](#)²⁰⁸ functions give you simple input dialog boxes. The function [PumpWaitingMessages\(\)](#)²¹⁴ has a special meaning: If you plan on performing time consuming computations in your scripts you will effectively block the GSEOS user interface. To keep the user interface responsive you should intersperse calls to `PumpWaitingMessages()` into your script. An alternative to this approach is using threads.

The Gseos module also provides functions to interact with the GSEOS User Interface like resizing and moving windows.

[FileOpen\(\)](#)²⁰⁴ allows you to open any GSEOS file. systemRecorder module exposes the Python interface to the GSEOS Recorder functionality. The window functions [WindowPrint\(\)](#)²¹⁹, [WindowMinimize\(\)](#)²¹⁸, [WindowMaximize\(\)](#)²¹⁸, [WindowRestore\(\)](#)²²¹, and [WindowClose\(\)](#)²¹⁸ allow you to control some of the window behavior. [StartApplication\(\)](#)²¹⁶ lets you invoke external programs.

8.4.1.1.1 Gseos.Alarm

Alarm(strName, strDescr, [wLevel], [dwID=0])

Issue an alarm event. This generates the AlarmEvent block which is displayed in the Alarm Notification Window.

Parameter	Description
strName	The alarm name.
strDescr	The alarm description.
wLevel	The alarm level, this should be one of the following constants: <code>Gseos.ALARM_STATE_RED_LO</code> , <code>Gseos.ALARM_STATE_YEL_LO</code> , <code>Gseos.ALARM_STATE_YEL_HI</code> , <code>Gseos.ALARM_STATE_RED_HI</code> . This parameter is optional and the default is <code>Gseos.ALARM_STATE_RED_HI</code> .
dwID	The alarm event ID. An arbitrary user defined value. This can be used to uniquely clear specific alarms. The default is 0.

Returns

None

8.4.1.1.2 Gseos.AddEventHandler

AddEventHandler(wEvent, fHandler)

Register an event handler function for one of the GSEOS events: 1s Timer, Shutdown. Your event handler function (which must take one parameter). To remove a previously installed event handler call [Gseos.RemoveEventHandler\(\)](#)^[214]. The event handler function fHandler is called with the event type as argument on each occurrence of the event.

The two events to register for are EVENT_1S_TIMER and EVENT_SHUTDOWN. The EVENT_1S_TIMER event is called once per second as a periodic timer event. The EVENT_SHUTDOWN event is called just before GSEOS shuts down. This can be used to register any clean-up actions.

Parameter	Description
wEvent	The event type, must be one of: Gseos.EVENT_1S_TIMER, Gseos.EVENT_SHUTDOWN.
fHandler	The handler function. This must be a callable object taking one parameter. The parameter passed is the event type the handler was registered for.

Returns

None

8.4.1.1.3 Gseos.ConsoleWrite

ConsoleWrite(szText, [iRed, iGreen, iBlue])

The ConsoleWrite() 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

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. The print statement will always print

in black, if you want to print in a different color you will have to use this function.

Example

The following examples prints some text in red to the console window.

```
Gseos.ConsoleWrite("Hello, World", 255, 0, 0)
```

8.4.1.1.4 Gseos.Exit

Exit()

Terminate the GSEOS application. If you have set a termination confirmation in the gseos.ini [\[System\]](#)¹⁹⁵ section, you will be prompted to terminate. Also, any unsaved documents will request confirmation to be saved.

Parameter	Description
-	-

Returns

None

8.4.1.1.5 Gseos.FileAppend

FileAppend(strFileName)

The FileAppend() function works like the [File/Append](#)²⁹¹ menu. However, it will not pop up a file selection dialog. Currently only desktop files allow the append operation. When appending a desktop file the tabs found in the new desktop will be added to the current desktop. If a file other than a .dt desktop file is specified the operation is not performed.

Parameter	Description
strFileName	The desktop file name to append.

Returns

None

8.4.1.1.6 Gseos.FileMenu

FileMenu(fCallback, strFilter, 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	Description
fCallback	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.
strFilter	The filter, the actual extensions should be in parentheses with a descriptive text before that: "Python Modules (*.py; *.pyd; *.pyc)". If multiple extensions are provided they must be separated by semicolon (;).
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 or the values together. Here the constants valid for wFlags: REG_FILENEW, REG_FILEOPEN, REG_FILEAPPEND, REG_FILES救人AS, REG_FILEPRINT.
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_FILES救人AS):
        fSaveMyResult(strFileName)

# Register with GSEOS
FileMenu(fOnImgFile, 'Image Files (*.img)', REG_FILEOPEN | REG_FILES救人AS)
```

8.4.1.1.7 Gseos.FileNew

FileNew(strFileName)

The FileNew() function works like the [File/New](#)^[28] menu. However, it will not pop up a file selection dialog. It will create a file/window of the type you specify in the file name. If the

extension is associated with a window like .gscr, or .log the window will be opened. If the extension is unknown no operation is not performed.

Parameter	Description
strFileName	The name of the file to create.

Returns
None

8.4.1.1.8 Gseos.FileOpen

FileOpen(strFileName)

The FileOpen() function works like the [File/Open](#)²⁹ menu. However, it will not pop up a file selection dialog. It will create a file/window of the type you specify with the file name. If the extension is associated with a window like .gscr, or .log the window will be opened. If the extension is unknown no operation is not performed.

Parameter	Description
strFileName	The name of the file to open.

Returns
None

8.4.1.1.9 Gseos.FileOpenDialog

FileOpenDialog([strDirectory], [strFileName], [strTitle])

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. Defaults to *.*
strTitle	Optional. The dialog box title, defaults to "GSEOS".

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

8.4.1.1.10 Gseos.FileSaveDialog

FileSaveDialog([strDirectory], [strFileName], [strTitle])

The FileSaveDialog() function opens a Windows file save 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. Defaults to *.*
strTitle	Optional. The dialog box title, defaults to "GSEOS".

Returns

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

8.4.1.1.11 Gseos.GetActiveDesktopPage

GetActiveDesktopPage()

Return the name of the currently active desktop page.

Parameter Description

-

Returns

strPageName The name of the currently active desktop page.

8.4.1.1.12 Gseos.GetIniFileName

GetIniFileName()

Return the file name of the initialization file used to start up GSEOS. This is either gseos.ini or the file specified on the [command line](#)^[147] with the \ini switch.

Parameter Description

-

Returns

strGseosIni The name of the gseos.ini file used to configure the system.

8.4.1.1.13 Gseos.GetInstrumentPath

GetInstrumentPath()

Get the instrument path, that is the path where the gseos.ini that was used to start GSEOS is located. This is typically the instrument directory. This will return a relative path if it is located underneath the project path. If a relative path is returned you can use the [Gseos.GetProjectPath\(\)](#)^[206] function to construct an absolute path.

Returns

The instrument path.

Example

Get the instrument path:

```
>>> Gseos.GetInstrumentPath()  
'xrs'
```

8.4.1.1.14 Gseos.GetInstance

GetInstance()

Gets the instance number or name 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](#)¹⁴⁷ you can specify an instance number or name. This allows for different configurations being launched for different instances. Check the [gseos.ini](#)¹⁸² file for more information on instance settings.

Returns

The instance string.

Example

```
Gseos.GetInstance()
```

8.4.1.1.15 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:\\gseos7\\prj\\\\messenger\\\\'
```

8.4.1.1.16 Gseos.GetSystemLoad

GetSystemLoad()

Calculates the current system load. The return value is the system utilization in percent. It can be > 100 if the system is temporarily overloaded.

Returns

The current system load in %, can be > 100.

8.4.1.1.17 Gseos.GetWindowPos

GetWindowPos(strTitle)

Get the window position of a GSEOS window. The window is located by the caption title on any desktop page. If the window can't be found raise the TWindowNotFoundError exception.

Parameter	Description
strTitle	The caption bar title. Not the entire title needs to be entered, any match on the first characters will suffice.
Returns	wX, wY The coordinates of the window.

Example

Get the coordinates of the GSEOS screen window TLM:

```
>>> Gseos.GetWindowPos('TL')
(148, 89)
```

See Also

[Gseos.GetSize\(\)](#)²⁰⁷, [Gseos.WindowResize\(\)](#)²²⁰, [Gseos.WindowMove\(\)](#)²¹⁹,
[Gseos.WindowRestore\(\)](#)²²¹

8.4.1.1.18 Gseos.GetSize

GetWindowSize(strTitle)

Get the window size of a GSEOS window. The window is located by the caption title on any desktop page. If the window can't be found raise the TWindowNotFoundError exception.

Parameter	Description
strTitle	The caption bar title. Not the entire title needs to be entered, any match on the first characters will suffice.
Returns	wWidth The width of the window. wHeight The height of the window.

Example

Get the size of the GSEOS screen window TLM:

```
>>> Gseos.GetSize('TL')
(456, 300)
```

See Also

[Gseos.GetWindowPos\(\)](#)²⁰⁶, [Gseos.WindowResize\(\)](#)²²⁰, [Gseos.WindowMove\(\)](#)²¹⁹,
[Gseos.WindowRestore\(\)](#)²²¹

8.4.1.1.19 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.

8.4.1.1.20 GseosInputDialog

InputDialog(strText, [strTitle="GSEOS"], [bModeless=False])

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. The dialog can be opened modeless or modal, if modal all user interface interaction with GSEOS is stopped until the dialog is confirmed, if modeless the GSEOS user interface remains responsive.

Parameter

Description

<code>strText</code>	The input dialog prompt text.
<code>strTitle</code>	Optional, The caption bar text. Default: GSEOS.
<code>bModeless</code>	True, if the dialog is to be shown as modeless and therefore allowing interaction with the main GSEOS application. The default is False, showing the dialog as modal. In either case, the <code>InputDialog()</code> function does not return until the dialog has completed.

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')
```

8.4.1.1.21 Gseos.lIdle

IsIdle()

Returns True when the GSEOS block processing is idle, that means there are currently no blocks waiting in the queue to be processed. This can be used for idle processing of background tasks.

Returns

<code>bIdle</code>	True, if the system is idle.
--------------------	------------------------------

8.4.1.1.22 Gseos.ListStatusTextMaps

LookupStatusTextMaps()

List all defined status text maps.

Parameter

Description

-

Returns

`ctStatusTextMaps` A list of the names of all defined status text maps (note that this does not include any status image maps).

8.4.1.1.23 Gseos.ListStatusTextMapItems

ListStatusTextMapItems(strTextMapping)

Lists all items of a status text defined in the status text mapping strTextMapping. See the chapter on [Status Definition files \(*.tr\)](#) for more details. The items are returned as a list of tuples with the following values: (fLow, fHigh, strText)

Parameter Description

`strTextMapping` The name of the text mapping as defined in a Status Definition File.

Returns

`ctItems` A list of tuples of all the items in the status text map. The values of the tuples are: fLow, fHigh, strText.

Exceptions

`KeyError` If the status mapping is not defined or loaded.

8.4.1.1.24 Gseos.Log

Log(strLogFile, strText, [wColorID=None], [byAttr=0])

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..

`wColorID` Optional, a color ID specifying the color of the text to insert. The module defines several color definitions:BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, LTGRAY, GRAY, LTBLUE, LTGREEN, LTCYAN, LTRED, LTMAGENTA, YELLOW, WHITE. If not specified (None) the configured foreground color will be used.

`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',  
byAttr=Gseos.BOLD)
```

8.4.1.1.25 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

strLogFile The file name of the log file to reload.

Returns

None

Example

Reload the log file from the previous example:

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

8.4.1.1.26 Gseos.LogSave

LogSave(strLogFile)

This function is maintained only for backward compatibility. Logs are saved instantly. This function is a NOOP.

Parameter

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:

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

8.4.1.1.27 Gseos.LookupStatusText

LookupStatusText(strTextMapping, fValue)

Looks up a status text defined in the status text mapping strTextMapping. See the chapter on [Status Definition files \(*.tr\)](#) for more details. If a match is found the status text is returned, otherwise an exception is raised. The color information associated with the status text can't be accessed.

Parameter Description

strTextMapping	The name of the text mapping as defined in a Status Definition File.
fValue	The value to map to a status text.

Returns

strStatus	The status text if the value matches a defined value or range.
-----------	--

Exceptions

KeyError	If the status mapping is not defined or loaded.
GseosError.TStatusMapNoMatchError	If the value doesn't match a defined value or range.

8.4.1.1.28 Gseos.MakePathAbsolute

MakePathAbsolute(strPath)

Make a relative path absolute. All relative paths are relative in relation to the project path ([GetProjectPath\(\)](#)) (where the GSEOS executable is located). If a path is already absolute it is returned as is. Returns a path relative to the GSEOS project path.

Parameter Description

strPath	The relative path.
---------	--------------------

Returns

The absolute path.

8.4.1.1.29 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:\\gseos7\\prj\\\\messenger\\\\xrs')  
'xrs'
```

8.4.1.1.30 Gseos.MakePathNormal

MakePathNormal(strPath)

Normalize a path name. On Linux we also convert backslashes into forward slashes. Use this function to make path names platform independent.

Parameter	Description
strPath	The path to normalize.

Returns

The normalized path.

8.4.1.1.31 Gseos.Message

Message(strMsg, [wType=MSG_INFO], [strSource='User'], [wEvent=0])

The Message function logs a message to the [GSEOS Message window](#) [140].

Parameter	Description
strMessage	The message to log.
wType	Optional, one of MSG_ERROR, MSG_INFO, MSG_WARN.
strSource	The message source, can be any string. Optional, defaults to 'User'.
wEvent	The event associated with the message. Optional, defaults to 0.

Returns

None

Comments

If multiple messages with the same content from the same source are written in succession, only a message counter is incremented. This is useful if a function causes repeatedly the same error and prevents the message log from filling up too quickly.

Example

Print a message into the message window.

```
Gseos.Message("Hello, World")
```

8.4.1.1.32 Gseos.MessageBox

MessageBox(strText, [strTitle="GSEOS"], [wButtons=MB_OK], [wIcon=MB_ICONINFORMATION], [bModeless=False])

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 with the wButtons parameter. An additional icon can be displayed and is specified with the wIcon parameter. This message box can be modal or modeless. As a modal window you can't access any other GSEOS user interface elements until you close the message box. As a modeless window the GSEOS user interface remains responsive.

Parameter	Description
strText	The message prompt text.
strTitle	Optional, The caption bar text. Default: GSEOS.
wButtons	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.
wIcon	Optional, specifies an icon to display. MB_ICONEXCLAMATION, MB_ICONINFORMATION, MB_ICONQUESTION, MB_ICONSTOP.
bModeless	True, if the dialog is to be shown as modeless and therefore allowing interaction with the main GSEOS application. The default is False, showing the dialog as modal. In either case, the MessageBox() function does not return until the dialog has completed.

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\nDo you want to
continue?',
                  'Confirm hazardous command', Gseos.MB_YESNO,
                  Gseos.MB_ICONEXCLAMATION)
```

8.4.1.1.33 Gseos.PrintDesktop

PrintDesktop()

Print the GSEOS desktop. The entire application window is printed. The printer dialog will pop up to let you specify print parameters before starting to print desktop. The desktop is printed with the current screen resolution.

You can specify an optional image output file. The file must have an extension of .bmp or .jpg. If such a file is specified the output will be directed to the file instead of the printer.

Parameter	Description
strFileName	Optional image output file name. Must have extension .bmp or .jpg. If specified prints to image file instead of printer.

Returns

None

8.4.1.1.34 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. If you do not yield while running a lengthy script the GSEOS user interface and block processing will not be processed and seem frozen. 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. Be careful with this function since it essentially establishes another message pump.

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()
    Gseos.PumpWaitingMessages()
```

8.4.1.1.35 Gseos.RemoveEventHandler

RemoveEventHandler(wEvent, fHandler)

Remove an event handler that was previously installed with [Gseos.AddEventHandler\(\)](#)²⁰¹.

The arguments passed to RemoveEventHandler() must be the same as were passed to the matching AddEventHandler() call.

Parameter	Description
wEvent	The event type, must be one of: Gseos.EVENT_1S_TIMER, Gseos.EVENT_SHUTDOWN.
fHandler	The handler function. This must be the same callable that was passed to the AddEventHandler() function.

Returns

None

8.4.1.1.36 Gseos.SetActiveDesktopPage

SetActiveDesktopPage(strPageName)

Sets the active [Desktop page](#)²⁰¹ to the one specified by strPageName.

Parameter	Description
strPageName	The name of the desktop page to activate.

Returns

None

If a page with the name strPageName does not exist the active desktop tab is not changed.

Example

Switches to the 'Test' page:

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

8.4.1.1.37 Gseos.SetStatusBarText

SetStatusBarText(strText)

Sets the text in the status bar of the GSEOS main window. The text is displayed in the right part of the status window, the left part is reserved for system messages. You can adjust the sizes of the message fields with the slider in the center of the status bar.



Parameter	Description
strText	The text to display in the status bar.

Returns

None

Example

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

8.4.1.1.38 Gseos.SetSplashScreenText

SetSplashScreenText(strText)

Sets the text in the splash screen that is displayed during system startup. To show the project version number use the function Gseos.SetSplashScreenVersion().

SetSplashScreenText() can be used by modules during system startup to display lengthy initialization progress. This function is only useful during the startup process.

Parameter	Description
strText	The text to display in the splash screen.

Returns

None

8.4.1.1.39 Gseos.SetSplashScreenVersion

SetSplashScreenVersion(strVersion)

Sets a version number in the splash screen that is displayed during system startup. To show custom text use the function [Gseos.SetSplashScreenText\(\)](#)^[215].

SetSplashScreenVersion() can be used by modules during system startup to display the instrument specific version number. This function is only useful during the startup process.

Parameter

strVersion

Description

The version number to display. This should be in the format:
Major.Minor.Release, e.g. 7.0.1027.

Returns

None

8.4.1.1.40 Gseos.ShellExecute

ShellExecute(strDocumentName)

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. Since Linux does not support application association with document names we always launch the default web browser and hope that the document gets rendered reasonably.

Parameter

strDocument

Description

The document to view.

Returns

None

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

Example

Open this file in HTML help:

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

8.4.1.1.41 Gseos.StartApplication

StartApplication(strPath, [ctArgs=[]])

Launches an application. This function does not wait for the application to complete but returns immediately. It returns the process identifier. The environment variable PATH is not referenced to resolve the application name strPath. You can pass arguments to the application in the ctArgs tuple.

Parameter	Description
strPath	The application path. The PATH environment variable is not referenced to resolve the path.
ctArgs	Tuple with application arguments, optional, the default is no arguments.

Returns

wProcessID The process identifier.

If an error occurs, like the file can not be found, an exception will be raised.

Example

Open notepad:

```
import Gseos
Gseos.StartApplication("c:\\windows\\notepad.exe")
```

8.4.1.1.42 Gseos.Version

Version

Print the GSEOS version. This is the version number of the GSEOS kernel, e.g. 7.0.xxxx. Where xxxx is the release number.

8.4.1.1.43 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.

8.4.1.1.44 Gseos.Window Close

WindowClose(strTitle)

Close a GSEOS window. If the window has been modified you will be prompted to save the window before it will be closed. The window is located by the caption title on any desktop page. If the window can't be found raise the TWindowNotFoundError exception.

Parameter	Description
strTitle	The caption bar title. Not the entire title needs to be entered, any match on the first characters will suffice.

Returns

None

See Also

[Gseos.WindowMinimize\(\)](#)^[218], [Gseos.WindowMaximize\(\)](#)^[218], [Gseos.WindowMove\(\)](#)^[219],
[Gseos.WindowResize\(\)](#)^[220]

8.4.1.1.45 Gseos.Window Maximize

WindowMaximize(strTitle)

Maximize a GSEOS window. The window is located by the caption title on any desktop page. If the window can't be found raise the TWindowNotFoundError exception.

Parameter	Description
strTitle	The caption bar title. Not the entire title needs to be entered, any match on the first characters will suffice.

Returns

None

See Also

[Gseos.WindowMinimize\(\)](#)^[218], [Gseos.WindowClose\(\)](#)^[218], [Gseos.WindowMove\(\)](#)^[219],
[Gseos.WindowResize\(\)](#)^[220]

8.4.1.1.46 Gseos.Window Minimize

WindowMinimize(strTitle)

Minimize a GSEOS window. The window is located by the caption title on any desktop page. If the window can't be found raise the TWindowNotFoundError exception.

Parameter	Description
strTitle	The caption bar title. Not the entire title needs to be entered, any match on the first characters will suffice.

Returns

None

See Also

[Gseos.WindowMaximize\(\)](#)²¹⁸, [Gseos.WindowClose\(\)](#)²¹⁸, [Gseos.WindowMove\(\)](#)²¹⁹,
[Gseos.WindowResize\(\)](#)²²⁰

8.4.1.1.47 Gseos.Window Move

WindowMove(strTitle)

Move a GSEOS window to a new position. The window is located by the caption title on any desktop page. If the window can't be found raise the TWindowNotFoundError exception.

Parameter

strTitle

Description

The caption bar title. Not the entire title needs to be entered, any match on the first characters will suffice.

wX

The new x-coordinate of the window.

wY

The new y-coordinate of the window.

Returns

None

Example

Move the window to the top left of the GSEOS desktop.:

```
>>> Gseos.WindowMove ('TL', 0, 0)
```

See Also

[Gseos.GetWindowPos\(\)](#)²⁰⁶, [Gseos.GetWindowSize\(\)](#)²⁰⁷, [Gseos.WindowResize\(\)](#)²²⁰,
[Gseos.WindowRestore\(\)](#)²²¹

8.4.1.1.48 Gseos.Window Print

WindowPrint(strTitle, strFileName=None)

Print a GSEOS window. The printer dialog will pop up to let you specify print parameters before starting to print the contents of the window. The window is located by the caption title on any desktop page. If the window can't be found raises the TWindowNotFoundError exception.

You can specify an optional image output file. The file must have an extension of .bmp or .jpg. If such a file is specified the output will be directed to the file instead of the printer. Keep in mind that this function prints the actual window, so if you have data that is outside of the visible portion of the window (i.e. scroll bars) only the visible portion is

printed.

Screen windows that are not visible can't be printed and raise a TGseosError exception. If a screen window is not on the active desktop or is minimized it is not visible and can't be printed.

If you want to print screen windows whether they are open and/or visible or not you can open the file, print it, and close it. The example below shows how to implement this.

Parameter	Description
strTitle	The caption bar title. Not the entire title needs to be entered, any match on the first characters will suffice.
strFileName	Optional image output file name. Must have extension .bmp or .jpg. If specified prints to image file instead of printer.

Returns

None

Example

Print a screen window. To ensure the window is open and visible we open it first using the [FileOpen\(\)](#) function, then print it and close it afterwards.

```
>>> def fPrintScreen(strScreenFileName, strScreenTitle, strOutFile):
...     Gseos.FileOpen(strScreenFileName)
...     Gseos.WindowPrint(strScreenTitle, strOutFile)
...     Gseos.WindowClose(strScreenTitle)
```

See Also

[Gseos.GetWindowPos\(\)](#), [Gseos.GetSize\(\)](#), [Gseos.WindowMove\(\)](#),
[Gseos.WindowRestore\(\)](#), [Gseos.PrintDesktop\(\)](#)

8.4.1.1.49 Gseos.Window Resize

WindowResize(strTitle)

Resize a GSEOS window. The window is located by the caption title on any desktop page. If the window can't be found raise the TWindowNotFoundError exception.

Parameter	Description
strTitle	The caption bar title. Not the entire title needs to be entered, any match on the first characters will suffice.
wWidth	The new width of the window.
wHeight	The new height of the window.

Returns

None

Example

Resize the TLM window:

```
>>> Gseos.WindowReize('TL', 200, 100)
```

See Also

[Gseos.GetWindowPos\(\)](#)²⁰⁶, [Gseos.GetWindowSize\(\)](#)²⁰⁷, [Gseos.WindowMove\(\)](#)²¹⁹,
[Gseos.WindowRestore\(\)](#)²²¹

8.4.1.1.50 Gseos.PlaySound

PlaySound(szWaveFile)

The PlaySound function plays a wave file.

Parameter**Description**

szWaveFile The wave file to play.

Returns

None

Exceptions

RuntimeError The wave file could not be played.

Example

Play a wave file.

```
Gseos.PlaySound("boing.wav")
```

8.4.1.1.51 Gseos.Window Restore

WindowRestore(strTitle)

Restore a GSEOS window from minimized or maximized state. If the window is in its normal state it will be activated. The window is located by the caption title on any desktop page. If the window can't be found raise the TWindowNotFoundError exception.

Parameter**Description**

strTitle The caption bar title. Not the entire title needs to be entered, any match on the first characters will suffice.

Returns

None

See Also

[Gseos.WindowMinimize\(\)](#)²¹⁸, [Gseos.WindowMaximize\(\)](#)²¹⁸, [Gseos.WindowClose\(\)](#)²¹⁸,
[Gseos.WindowMove\(\)](#)²¹⁹, [Gseos.WindowResize\(\)](#)²²⁰

8.4.1.2 GseosBinaryStreamer

The GseosBinaryStreamer module gives you access to the configured [Binary Streamers](#)^[15]. You can list all configured writers with [ListBinaryWriters\(\)](#)^[222], start a binary writer with [StartBinaryWriter\(\)](#)^[222], and stop it with [StopBinaryWriter\(\)](#)^[222]. It is also possible to change the Path and File templates using the [SetBinaryWriterPathTemplate\(\)](#)^[223] and [SetBinaryWriterFileTemplate\(\)](#)^[223] functions respectively. Please make sure the Writer is stopped before changing the Path or File template.

8.4.1.2.1 GseosBinaryStreamer.ListBinaryWriters

ListBinaryWriters()

Lists all binary writers currently installed in the system.

Parameter	Description
-	-

Returns

A list of names of binary writers. The name of the streamer (writer) can be used to start/stop the writer with [StartBinaryWriter\(\)](#)^[222] and [StopBinaryWriter\(\)](#)^[222] respectively.

8.4.1.2.2 GseosBinaryStreamer.StartBinaryWriter

StartBinaryWriter()

Start the binary writer strStreamer if it is not already running.

Parameter	Description
strStreamer	The name of the streamer to start.

Returns

8.4.1.2.3 GseosBinaryStreamer.StopBinaryWriter

StopBinaryWriter()

Stop the binary writer strStreamer if it is currently running.

Parameter	Description
strStreamer	The name of the streamer to stop.

Returns

-

8.4.1.2.4 GseosBinaryStreamer.SetBinaryWriterPathTemplate

SetBinaryWriterPathTemplate()

Set a new Path template for the binary writer strStreamer. Please make sure the streamer is stopped before changing the path template, otherwise the TBinaryStreamerError exception is raised.

Parameter Description

strStreamer	The name of the streamer to stop.
strPathTemplate	The new path template to set.

Returns

-

8.4.1.2.5 GseosBinaryStreamer.SetBinaryWriterFileTemplate

SetBinaryWriterFileTemplate()

Set a new File template for the binary writer strStreamer. Please make sure the streamer is stopped before changing the file template, otherwise the TBinaryStreamerError exception is raised.

Parameter Description

strStreamer	The name of the streamer to stop.
strFileTemplate	The new file template to set.

Returns

-

8.4.1.3 GseosBlocks

The GseosBlocks allows easy access to the GSEOS block definitions. All blocks are exported into the `__main__` namespace. One way of accessing blocks from a script is to import the `__main__` module. However, this will pull in all names from the `__main__` module if imported with `from __main__ import *` and we recommend against this technique. The other option would be to prefix every block with `__main__` or to specifically import the blocks needed in your script like `from __main__ import TLM`. A better alternative to using the block definitions is to use the GseosBlocks module which makes the blocks available via the **Blocks** dictionary. I.e.:

```
import GseosBlocks

TLM = GseosBlocks.Blocks['TLM']
```

Once you have the block object you can access the individual items like simple block attributes:

```
byTemp1 = TLM.Data[2]
ctTemps = TLM.Data[10:20]
wApID   = TLM.ApID
```

etc.

To assign data items you can simply write to scalar items like: TLM.ApID = 55. To write to array items you must use the slicing syntax:

```
TLM.Data[:] = range(10)
```

You will get an exception if you try to assign directly to an arry item like TLM.Data = range(10).

Keep in mind that writing a data item does not result in the block being posted to 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 [SendBlock\(\)](#)²³¹ function on the block once you have written all the data items you want to set for that block. GSEOS blocks are shared resources, if you want to write to a block from multiple threads you have to synchronize access to the block. The [GseosBlocks.TThreadSafeBlock](#)²³⁵ class allows you to access GSEOS blocks in a thread safe manner. To get metadata information about item definitions you can use the [TItemDescriptor](#)²³³ class.

TBDMBlock

Every block you define in your block definition files is instantiated as a TBDMBlock object and when you access a block it will be through the TBDMBlock class. You should never instantiate any objects of this class yourself. All block items can be accessed like regular attributes. Keep in mind that when writing data to a block item it will not be published until you invoke the [SendBlock\(\)](#)²³¹ function on the block. The blocks define the following attributes you might access:

strName: The name of the block.
 Items: A list with all the names of all items of this block
 bEnableSelect: 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. multipe 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].bEnableSelect = False
```

The TBDMBlock class implements a few other convenience functions like [GetBlockCountStamp\(\)](#)²²⁸, [IsScalarItem\(\)](#)²²⁸, [ReadItem\(\)](#)²²⁹, [ReadBlockAsString\(\)](#)²²⁸, and [WriteItem\(\)](#)²³⁰.

High performance block access

Accessing data items using the Python slicing syntax is fine and efficient for relatively small items (a few hundred elements) or for decoders that are not called frequently a few Hz. However, if you have a need for high performance decoders operating on large amounts of memory like histogram decoders or video frames using the Python slicing syntax is prohibitive. Using the slicing syntax requires a function call for each element since it needs to be fetched from its individual bit position as defined in the block definition file. Oftentimes large arrays are aligned on Byte or Word boundaries with no bit gaps etc. So the convenience of using the GSEOS bit level definitions is not required. In these cases the access to data can be sped up by as much as two orders of magnitude, depending on the amount of data that needs to be read or written. The functions [ReadString\(\)](#)²³⁰ and [WriteString\(\)](#)²³² of block array items offer high speed access to GSEOS data. These functions retrieve data from a GSEOS array item as a Python string. There are some requirements on the item definition like no gap bits, a multiple of 8-bit, and byte aligned. However, once these requirements are met you can read and write large amounts of data as a Python string. Ideally you don't want to convert the data bytes into an array since that will undo much of the performance gain in using these functions in the first place. If you need even more performance you can feed the output of the ReadString() function into a C extension module that performs the computations in C and generates an output string that you write back to the output block using the WriteString() function. For histogram decoders consider using the built-in efficient [histogram decoders](#)²⁶⁴.

You can also move Python strings into and out of entire blocks. Obviously, this circumvents the individual item definitions of a block but there are cases where this is helpful. The functions [ReadBlockAsString\(\)](#)²²⁸ and [WriteBlockFromString\(\)](#)²³¹ perform these functions respectively.

Multi-threading and GSEOS block access

Blocks are shared resources in GSEOS. If you plan on using multiple threads of execution and to write to blocks and to send blocks from these threads you have to deal with thread synchronization.

[GSEOS Sequencers](#)²⁹⁸ use threading, so if you generate blocks from your sequencer you have to deal with thread issues.

If you write to a block from multiple threads you can use Python's lock mechanism to ensure your writes are consistent. However, if the same block is generated from the main

thread without honoring this lock there will still be contention.

The solution is to use the TThreadSafeBlock class. You would create a new instance of this class for each thread and each block you want to modify in that thread.

The following example illustrates this:

```
# Get thread safe block access from multiple threads.
#
import time
import thread
import GseosBlocks

oThread1TLM = GseosBlocks.TThreadSafeBlock(TLM)
oThread2TLM = GseosBlocks.TThreadSafeBlock(TLM)

def f(oBlock, wValue):
    while 1:
        for i in range(len(oBlock.Data)):
            oBlock.Data[i] = wValue

        oBlock.SendBlock()
        time.sleep(0.1)

thread.start_new(f, (oThread1TLM, 55))
thread.start_new(f, (oThread2TLM, 77))
```

To create a new TThreadSafeBlock object you pass the block object you want to create thread safe access to into the constructor. In the above case this is the TLM block. You have to create a TThreadSafeBlock object for each thread you create and each block you want to generate in that thread. You can access the oThread1TLM and oThread2TLM blocks just like regular block objects. That is you can write items, read items, and send off the block once you are done with it.

There is no locking necessary when using TThreadSafeBlocks and all access to the underlying block is properly synchronized.

As opposed to a regular block object you can not install [Decoders](#)²⁴⁶ or [Monitors](#)²⁸⁵ on a TThreadSafeBlock, please use the regular block to handle decoders and monitors.

TItemDescriptor

Every block you define in your block definition files is instantiated as a TBDMBlock object and when you access a block it will be through the TBDMBlock class. You should never instantiate any objects of this class yourself. All block items can be accessed like regular attributes. Keep in mind that when writing data to a block item it will not be published until you invoke the [SendBlock\(\)](#)²³¹ function on the block. The blocks define the following attributes you might access:

strName: The name of the block.

Items: A list with all the names of all items of this block

bEnableSelect: 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 off 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].bEnableSelect = False
```

The TBDBMBlock class implements a few other convenience functions like [GetBlockCountStamp\(\)](#)²²⁸, [IsScalarItem\(\)](#)²²⁸, [ReadItem\(\)](#)²²⁹, [ReadBlockAsString\(\)](#)²²⁸, and [WriteItem\(\)](#)²³².

8.4.1.3.1 TBDBMBlock

The TBDBMBlock class is an internal class. You should never instantiate an object of this class. However, all blocks defined in the block definition files are implemented as instances of this class. So when you access GSEOS blocks you will be interacting with objects of this class. To retrieve an object of this class you can either directly access the block in the __main__ namespace like __main__.TLM.ApID (given you have defined a block named TLM with an item of ApID), or you can use the [GseosBlocks](#)²²³ module to access blocks.

The preferred way to get access to a GSEOS Block from a Python script is to access the GseosBlocks.Blocks list:

```
oBlkTLM = GseosBlocks.Blocks["TLM"]
```

If you want to generate blocks (either from a Decoder or as a data source) you will assign all items as necessary and then call the [SendBlock\(\)](#)²³¹ function of the block.

For example to send the binary command 0xFA, 0xC1, 0x08 on channel 0 you would do the following:

```
import GseosBlocks
oBlkBinCmd = GseosBlocks.Blocks['BinCmd']

oBlkBinCmd.Channel = 0
oBlkBinCmd.CmdData[:] = [0xFA, 0xC1, 0x08]
oBlkBinCmd.SendBlock()
```

Mind you that the block needs to be forwarded and processed by GSEOS before it can be received by the subscribers. So if you were to read the block without giving GSEOS a chance to process you will read the previous values.

8.4.1.3.1.1 TBDBlock.GetBlockCountStamp

GetBlockCountStamp()

Returns the current count stamp of the block. Each sample of a block has a unique counter (32-bit) that gets incremented with each new sample.

Parameter	Description
-	

Returns

dwCountStamp The count stamp of the block.

8.4.1.3.1.2 TBDBlock.IsScalarItem

IsScalarItem()

Returns True if the item is a scalar item, False if it is an array item or the item doesn't exist.

Parameter	Description
strItemName	The item name in question.

Returns

bScalar True if the item is a scalar.

Example

```
for strItem in TLM.Items:  
    bScalar = TLM.IsScalarItem(strItem)  
    print 'Item: %s is %s item.' % (strItem, ['an array', 'a scalar'][bScalar])
```

8.4.1.3.1.3 TBDBlock.ReadBlockAsString

ReadBlockAsString()

Sometimes it is useful to access an entire block directly. This is especially important for large blocks like image or video data. Single item access of large blocks is slow compared to accessing the entire block as once. The ReadBlockAsString() function returns the contents of the block as a string of 8-bit bytes. It is up to the user to interpret this data properly (i.e. handle bit shifts, etc. if they occur in the data). The ReadBlockAsString() function simply returns the entire block content as a string.

Parameter Description

-

Returns

strData The entire block as a byte string.

Example

```

def fMonImageBlocks(self, oImageBlock):
    #
    # - Get the image data to write to disk. Skip over the length field.
    #
    strData = oImageBlock.ReadBlockAsString()
    strData = strData[4:4+oImageBlock.Len]

    try:
        #
        # - Write the image file.
        #
        oFile = open(os.path.join(self.strImagePath,
self.strImageFileName), 'wb')
        oFile.write(strData)
        oFile.close()

    except Exception, oX:
        Gseos.Message(str(oX), GseosCmd.MSG_ERROR, "Image Writer")

```

8.4.1.3.1.4 TBDBlock.ReadItem

ReadItem(strItem, [dwIndex])

Read the value of an item of this block. If the item is an array item you can also specify an index. Only single values are returned (no slices). If the dwIndex parameter is not specified it defaults to zero. If the item strItem doesn't exist as a block item an AttributeError is raised.

Parameter Description

strItem The item name of the item to read.

dwIndex Optional. The index position of an array item. Defaults to 0. If the index is out of range an IndexError is raised.

Returns

dwValue The current value of the item read. Note that no conversion functions

are applied, so only the raw data value is returned. If you need to apply conversion functions access the data as outlined in the chapter on [Formula Definition Files](#)¹⁶⁸.

8.4.1.3.1.5 TBDBlock.ReadString()

ReadString()

Together with the [WriteString\(\)](#)²³² function the ReadString() function allows high speed access to GSEOS data items. This function is not a member of the TBDBlock class but a member of array items. If you don't need the bit level access GSEOS items offer or you can't afford the speed penalty that comes with this bit level access you can use the ReadString() function to handle array item data efficiently as a Python string. This is especially important for large blocks/items like image or video data. Slicing (TLM.Data[:20000]) access of large arrays is slow compared to accessing the entire field as a string. For once is the returned Python list polymorphic and incurs the according overhead, it also requires multiple function calls for each individual element since it needs to be fetched through the GSEOS bit level item definition. Although this operation is highly optimized (assembly code) it still can be significant for large arrays. The ReadString() function returns the contents of the item as a string of 8-bit bytes. It is up to the user to interpret this data properly (i.e. handle bit shifts, etc. if they occur in the data). The ReadString() function checks a few prerequisites to make sure the item can be accessed in a simple byte level manner. The item needs to be byte aligned, must not have any bit gaps and but be a multiple of 8-bits wide.

Parameter **Description**

iStartIdx	The start index, defaults to 0.
iEndIdx	The end index (non-inclusive), defaults to the entire array.

Returns

strData The item data as a byte string.

Example

```
def fDump(self, oImageBlock):
    #
    # - We write the entire Image item to file.
    - #
    #
    strData      = oImageBlock.Image.ReadString()
    file('Image.bin', 'wb').write(strData)
```

8.4.1.3.1.6 TBDMBlock.SendBlock

SendBlock([bCopy=False], [bDataSource=False], [bDecoder=True])

The SendBlock() member function is a method of all BDM blocks. You use this function to publish blocks of that particular type.

Parameter	Description
bCopy	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 GseosBlocks.EnableDataSource() ²³⁵ . You usually specify this parameter only if you write a data source in a Python module.
bDecoder	Optional, defaults to True if not specified. By default all blocks are generated as 'decoded blocks', that means they are inserted in the block queue in the proper location depending on the decoder task. This allows downstream decoders to operate on the blocks generated by upstream decoders. If you generate blocks that are not decoded from other blocks you should set this parameter to True. If you generate the blocks from a mutually exclusive data source like a device you should also set the bDataSource parameters to True. If you say generate command blocks (CmdString or BinCmd) that are not the product of a decoder you should set the bDecoder parameter to False. If the bDecoder parameter is set to False the block is inserted at the head of the queue. See the GseosDecoder ²⁴⁶ chapter for more information on this.

Returns

None

Example

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

8.4.1.3.1.7 TBDMBlock.WriteBlockFromString()

WriteBlockFromString()

Together with the [ReadBlockAsString\(\)](#)²²⁸ function the WriteBlockFromString() function allows high speed access to GSEOS data blocks (as opposed to item access). The passed in string gets written to the destination block, ignoring the block item definitions. If the string you write is shorter than the block definition the block is padded with zeros. If the string is larger the data is clipped.

Parameter	Description
strData	The data to write to the block. If the data is larger than the block it will clip.

Returns

-

8.4.1.3.1.8 TBDBlock.WriteItem

WriteItem()

Write an item of a block. It usually is simpler to just write to the block attribute instead of using this function. However, if you want to use metadata to access block items this function might be useful. It writes a value to a specific item of a block (possibly using an optional index if the item is an array item).

If the item strItem is not a valid block item an AttributeError is raised. If the index of the item is out of bounds an IndexError is raised.

Parameter

strItem

Description

The item name of the item to read.

dwValue

The value to write.

dwIndex

Optional. The index position of an array item. Defaults to 0. If the index is out of range an IndexError is raised.

Returns

-

8.4.1.3.1.9 TBDBlock.WriteString()

WriteString()

Together with the [ReadString\(\)](#) function the WriteString() function allows high speed access to GSEOS data items. This function is not a member of the TBDBlock class but a member of array items. Like the ReadString() function the WriteString() function allows you fast access to GSEOS data items. The same restrictions apply.

Parameter

iStartIdx

Description

The start index, defaults to 0.

strData

The data to write to the item. If the data is larger than the remaining item it will clip.

Returns

-

8.4.1.3.2 TItemDescriptor

Sometimes it is useful to have metadata information at runtime about the data items defined in your [block definition files](#)^[150].

You construct an object of class TItemDescriptor() passing in the fully qualified item information. This can include a conversion function if you have one defined. Alternatively you can pass in the block, item, and conversion function name separately. If the item is not valid the bValid attribute of the object is set to False.

**TItemDescriptor(strBlockOrFullName, strItemName=None,
strConversionName=None, dwStart=0, dwEnd=None, dwStep=1)**

The TItemDescriptor constructor creates a new TItemDescriptor object. You can either specify the Block name, Item name, and (if applicable) conversion function name as well as start index, end index, and step width if the item is an array item. Alternatively you can use a fully qualified item name as the first parameter and the constructor will parse the name appropriately. A few sample of fully qualified item names:

```
TLM.ApID
TLM.Data[5]
TLM.Data[:200]
PHA.Rates[50:250:10]
EU('HK.Temp1')
```

If you specify a fully qualified item name as first parameter it has to be properly formatted. If the format is not valid a TItemDescriptorError exception is raised. Even if the name is formatted properly the item you specify might not be valid. This might be due to the block you specify not existing or the item not existing in the block you specify or the conversion not being valid or not loaded. In any of these cases the bValid attribute is False. If the bValid attribute is False the methods of the object should not be called and might raise exceptions if called.

Parameter	Description
strBlockOrFullName	You can either specify the fully qualified item name including conversion function (optional), block name, item name, and index positions if desired (array items). Alternatively you can pass in the block name and specify the other parts of the full item name with the following parameters.
strItemName	The (block relative) item name. You should only specify this parameter if you pass in the block name as the first argument.
strConversionName	The optional conversion function name. You should only specify this parameter if you pass in the block name as the first argument. If you specify a conversion function and the function has not been loaded yet the bValid attribute of the object is set to False indicating that the item descriptor is not valid.
dwStart	An optional start index. For array items only. The default start index is 0.
dwEnd	An optional end index. For array items only. The default is None which uses the end index of the item as defined in the block definition.
dwStep	The step width, the default is 1. For array items only.

Returns

The new TItemDescriptor object.

Comments

You should always check the bValid attribute before accessing the item descriptor. You can compare TItemDescriptor objects. Two objects are considered equal if they refer to the same item and have the same array dimensions. The object exports the following attributes:

Attribute Description

bValid	True if the item is valid, False if not.
strBlockName	The block name of the item.
strItemName	The (relative) name of the item.
strConversionName	The name of the conversion function if specified. None if this item descriptor doesn't use a conversion function.
dwStart	The start index.
dwEnd	The end index.
dwStep	The step width.
dwStartByte	The byte offset position of the item relative to the block beginning.
dwStartBit	The start bit offset within the start byte.
dwBitSize	The bit size of the item.
bScalar	True if the item is a scalar, False otherwise. The item is an array item and you specify start and end indices such that there is only one element the item is still considered an array item and the bScalar attribute is False.
dwDimension	The dimension of the item as defined in the block definition file. This value does not reflect the start and/or end index positions specified in the constructor. It will always indicate the number of elements as defined in the block definition. For scalar items this value is 0.
oBlock	The TBDMBlock ²²⁷ object of the block this item belongs to.

Exceptions

TItemDescriptorError The item name format is not valid.

Example

The following example get the bit width of the TLM.ApID item.

```

import GseosBlocks
import GseosError

try:
    oItem = GseosBlocks.TItemDescriptor('TLM.ApID')

except GseosError.TItemDescriptorError, ox:
    print 'Error: %s' % ox

if oItem.bValid:
    print 'Item: %s.%s has %d bits' % (oItem.strBlockName, oItem.strItemName,
oItem.dwBitSize)

else:
    print 'Item: %s.%s is not valid' % (oItem.strBlockName,
oItem.strItemName)

```

8.4.1.3.3 GseosBlocks.EnableDataSource

EnableDataSource(strName, bEnable)

This function enables/disables the Python data source. If enabled all blocks sent with the bDataSource parameter set will be mutually exclusive to other data sources. If the Python data source is not enabled these blocks will be discarded. 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 [SendBlock\(\)](#)²³¹ function.

Parameter	Description
strName	The name of your data source. This name will be displayed in the caption bar if the data source is active.
bEnable	True to enable the data source, False to disable it.
Returns	
None	

8.4.1.3.4 GseosBlocks.IsDataSourceEnabled

IsDataSourceEnabled()

Get the current Python data source status. If the Python data source is enabled this function returns True.

Parameter	Description
-	-
Returns	
bEnabled	True, if the Python data source is enabled.

8.4.1.3.5 GseosBlocks.TThreadSafeBlock

TThreadSafeBlock(oBlock)

The TThreadSafeBlock constructor creates a new thread safe copy of a GSEOS block. You have to create one instance per thread, per block you want to generate from that thread. Block access is identical to the regular block, you can read and write items and send the block with [SendBlock\(\)](#)²³¹. Write access to the underlying GSEOS block is automatically synchronized.

Parameter	Description
oBlock	The block object you want to create a thread safe block for. All block objects can be accessed by name with the GseosBlocks.Blocks[]

dictionary.

Returns

The thread safe block instance.

Comments

You can't install Decoders or Monitors on a thread safe block, use the original block to do this.

Exceptions

TBDMBlockError The passed in object is not a TBDMBlock object.

Example

The following example creates a thread safe instance of the TLM block. You can use the oTLMsafe block to access from a new thread of execution. If you have multiple threads accessing the TLM block you will have to create an instance for each thread.

```
import GseosBlocks

oTLM      = GseosBlocks.Blocks['TLM']
oTLMsafe = GseosBlocks.TThreadSafeBlock(oTLM)
```

8.4.1.4 GseosCmd

The GseosCmd module implements the GSEOS command interface. The commands available are defined in a [command definition file \(*.cpd\)](#)^[156]. You have to load the command definition file before you can issue the commands. You issue commands with the [ExecCmd\(\)](#)^[238] function, if your commands define descriptions you can access the command description with [GetCmdDoc\(\)](#)^[240]. Range checking is controlled with the function [EnableRangeCheck\(\)](#)^[237]. [IsRangeCheckEnabled\(\)](#)^[240] allows you to query if range checking is enabled or disabled.

ExecCmd() parses your command and validates it against the command database. If it is a critical command it will prompt with a dialog to confirm the command unless it confirmation has been globally disabled in the gseos.ini file or for this command specifically with the bCriticalCmdDialog flag. If not all parameters are specified it will prompt for the remaining arguments. The output of ExecCmd() is the system block CmdString. This block in turn will be processed by the command processor and generate the BinCmd block depending on your command definition. This block in turn is processed by the BIOS module and will command your instrument hardware as necessary.

The gseos.ini setting:

```
[Command]
Python=Enabled
```

controls if the command is handed to the Python interpreter. If a command mnemonic is not recognized it will be passed on to the Python interpreter. If the Python interpreter raises an exception this exception will be re-raised. This is the default behavior. To prevent commands being passed to the Python interpreter you can set the above setting to Disabled.

You can assign commands to hierarchical [command menus](#)¹⁶¹ for easy access or place them on [command buttons](#)⁵⁵ on your screens.

8.4.1.4.1 GseosCmd.ListCmdMnemonics

ListCmdMnemonics()

Return a tuple of all defined command mnemonics (at the time of the function call).

Parameter	Description
------------------	--------------------

-

Returns

ctCmdMnemonics Tuple of currently defined command mnemonics.

8.4.1.4.2 GseosCmd.GetCmdOpcode

GetCmdOpcode(strCmd)

Return the command opcode for the command strCmd.

Parameter	Description
------------------	--------------------

strCmd The command to retrieve the opcode for.

Returns

dwOpcode The command opcode.

8.4.1.4.3 GseosCmd.EnableRangeCheck

EnableRangeCheck(bEnable)

By default command arguments are checked against their valid ranges. However, sometimes it is desirable to issue erroneous commands with parameters out of range to verify the response of the instrument. To test this scenario you can disable range checking with this function. To check if range checking is currently turned on or off you can use [IsRangeCheckEnabled\(\)](#)²⁴⁰.

Parameter	Description
------------------	--------------------

bEnable True to turn on range checking, False to turn it off.

Returns

None

8.4.1.4.4 GseosCmd.ExecCmd

```
ExecCmd(strCmd, [wQueryForArgs=GseosCmd.QUERY_MISSING],  
[bCriticalCmdDialog=True], [bSendCmdString=True],  
[bQueryForDefaultArgs=False])
```

Executes a command that has been loaded into our command dictionary. If the command is valid and all arguments are within limits the CmdString block with the command text will be generated. If the optional parameter wQueryForArgs is set to QUERY_MISSING (default) or QUERY_ALWAYS a dialog will query for the (missing) arguments if any and complete the command as necessary. If set to QUERY_NONE an error is issued if the command is not complete.

There are several ways to specify missing arguments. If no arguments are given at all like 'CMD()' all arguments will be requested. If there are arguments supplied and others that need to be filled in place a comma for every argument that is missing: CMD(Arg1,,Arg3). The above command would query for Arg2 and any arguments higher than Arg3 if there are any.

We currently assume that there are no string arguments and therefore no special characters (especially a comma) can occur within a command argument.

Keyword arguments must follow non-keyword (positional) arguments. Keyword arguments must specify the keyword in the actual argument, e.g. Cmd1(1, 2, 3, Arg4=4, Arg5=5). An optional keyword argument, e.g. Arg4 does not need to be specified: Cmd1(1, 2, 3, Arg5=5).

You can specify prompts by inserting the escape sequence: '\$'MyPrompt', where MyPrompt is the text the user will see as the prompt. The entire escape sequence including the '\$' and terminating ' will be replaced by the user input.

If this is a critical command a dialog box will be displayed to confirm execution of the critical command. The confirmation dialog can be suppressed by either passing in False for the bCriticalCmdDialog parameter or by having notification globally disabled in [gseos.ini](#) [Command]/DisableCriticalCmdNotification=Yes.

If the command is in error a TCmdError exception will be raised.

The ExecCmd function generates the CmdString block if the command is validated successfully. The command processor will then intercept the CmdString block and convert the command string into a binary command by issuing the BinCmd system block. This block in turn will be processed by the BIOS to command your instrument as appropriate.

The gseos.ini setting:

```
[Command]  
Python=Enabled
```

controls whether the command is handed to the Python interpreter. This is enabled by default.

If a command mnemonic is not recognized it will be passed on to the Python interpreter. If the Python interpreter raises an exception this exception will be re-raised. To prevent commands being passed to the Python interpreter you can set the above setting to Disabled.

Parameter	Description
-----------	-------------

strCmd	The command to execute.
wQueryForArgs	Query for arguments. GseosCmd.QUERY_NONE: Don't query. GseosCmd.QUERY_MISSING: Query if arguments are missing (default). GseosCmd.QUERY_ALWAYS: Always query.
bCriticalCmdDialog	If this is a critical command and you don't want the critical command warning pop up set this parameter to False. The default is True.
bSendCmdString	Sends the CmdString to execute the command. If False the command parameters will be validated and the command string returned without issuing the CmdString block which will effectively not issue the command. The default is True.
bQueryForDefaultArgs	This argument overrides the setting as configured with the [Command] ¹⁷⁸ /QueryForDefaultArgs gseos.ini entry. If True the command dialog will pop up if there are any default arguments that are not explicitly specified. If False the dialog won't pop up even if so configured with the gseos.ini setting.

Returns

strCmd The completed command string.

8.4.1.4.5 GseosCmd.GetCmdArgDoc

GetCmdArgDoc(strCmd, strArgName)

Get the documentation string of the argument named strArgName of command strCmd. If the argument can't be found a TCmdError exception is raised.

Parameter

strCmd	The command mnemonic.
strArgName	The argument name.

Returns

strDoc The documentation string of the argument.

8.4.1.4.6 GseosCmd.GetCmdArgName

GetCmdArgName(strCmd, wArgIdx)

Get the name of the argument at position wArgIdx of command strCmd. If the argument doesn't have a keyword name a generic name of 'Arg X' is returned. If the wArgIdx is out of range a TCmdError exception is raised.

Parameter

strCmd	The command mnemonic.
wArgIdx	The argument position. 0 is the first argument.

Returns

strName	The name of the argument.
---------	---------------------------

8.4.1.4.7 GseosCmd.GetCmdDoc

GetCmdDoc(strCmd)

Return the command documentation string if one was specified.

Parameter	Description
strCmd	The command to retrieve the documentation string for.
Returns	
strDoc	The command documentation string.

8.4.1.4.8 GseosCmd.IsRangeCheckEnabled

IsRangeCheckEnabled()

Return the current range check status. To turn range checking on or off use [EnableRangeCheck\(\)](#)²³⁷.

Parameter	Description
-	
Returns	
bEnabled	True if range checking is enabled, False otherwise.

8.4.1.4.9 GseosCmd.SendBinCmd

SendBinCmd(byChannel, ctData)

Send the binary data passed in ctData on command channel byChannel. This is a low level command that circumvents your [command definitions](#)¹⁵⁶ and lets you issue any binary command sequence desired. The ctData argument can be any sequence, even nested, containing strings and numbers. All sequences are flattened before sending. Each element is interpreted as a byte and has to be in the range -127..255, otherwise an exception will be raised.

Parameter	Description
byChannel	The command channel to issue the command on.
ctData	Arbitrary byte sequence. 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 endianity). Since only bytes are accepted the issue of endianity does not arise. The total number of terminal nodes (bytes) in ctData can not be larger than the BinCmd.CmdData field (1024 bytes).

Returns

None

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.

Example

The following examples show different sequences passed into GseosCmd.SendBinCmd().

```

#
# Just a simple integer
#
GseosCmd.SendBinCmd(22, 3)

#
# A tuple
#
GseosCmd.SendBinCmd(22, (1,2,3))

#
# A string
#
GseosCmd.SendBinCmd(0, "Hello, World")

#
# A nested sequence
#
GseosCmd.SendBinCmd(1, ("This goes out first", 2, [3, 4, 5, "Another
string", (6, 7, 8)]))

```

8.4.1.4.10 GseosCmd.StartBatch

StartBatch(szFileName)

The StartBatch() function starts the execution of a GSEOS batch file. See [batch files](#)¹⁵⁴ for a description of the GSEOS batch file format. Multiple files can be run at the same time. To check on the running batch files open the [command dialog](#)¹⁵³.

Parameter

szFileName

Description

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

bBlock Optional, defaults to False. If this parameter is True the calling batch will suspend execution until this batch file (the batch about to be started) completes. This only has effect for nested batch files, top level batch files are not affected.

Returns

None

Example

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

8.4.1.4.11 GseosCmd.StopBatch

StopBatch(szFileName)

The StopBatch() function stops execution of a running GSEOS batch file. See [batch files](#) [154] for a description of the GSEOS batch file format. Multiple files can be run at the same time. If there is no batch file with the name szFileName running the stop request is ignored.

Parameter

szFileName

Description

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

Returns

None

Example

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

8.4.1.5 GseosConversion

The GseosConversion module exports all conversions defined in GSEOS. It also exports all conversions and expressions defined with [formula \(*.qlf\) files](#) [168] as well as the basic math functions.

This allows you to access the conversion functions and expressions programmatically from Python.

E.g.: Say you have the defined and loaded the following expression:

```
MET_Hour(MET) := hour(MET+1640995200)
```

You can then access this expression from your Python script:

```
wHour = MET_Hour(dwMET)
```

To get a list of all the functions defined in this module (which of course depends on the formula files loaded) use:

```
import GseosConversion
dir(GseosConversion)
```

The [block definitions](#)¹⁵⁰ in GSEOS are not typed. This means all items are interpreted as raw bit storage with no sign bit or floating point interpretation. You may run into the problem that your items are not raw binary data but signed values or floating point data. The GseosConversion module offers functions to convert between various data representations:

dtoll() ²⁴³ :	Performs a double to long-long (64-bit) conversion.
ftol() ²⁴⁴ :	Performs a float to long conversion.
lltod() ²⁴⁴ :	Performs a long-long (64-bit) to double conversion.
ltof() ²⁴⁵ :	Performs a long to float conversion.
signed() ²⁴⁵ ,	
signed8() ²⁴⁵ :	Performs an unsigned to signed conversion.
signed16() ²⁴⁵ ,	
signed32() ²⁴⁵	

It also exports conversion functions for time related functions. The native GSEOS EPOCH is Jan/01/1958 and the time functions are based on that EPOCH. However, times before Jan/01/1970 can't be processed. The following time functions are available:

year(), month(), day(), hour(), minute(), sec(), yday()

All functions take the time in seconds as a parameter and return the according part of the time.

8.4.1.5.1 GseosConversion.dtoll

dtoll(dValue)

Performs a double to long-long conversion. The binary representation of the IEEE 64-bit double floating point value is returned as a 64-bit long. This function does not coerce to a long but interprets the double floating point value as a 64-bit signed long.

Parameter	Description
dValue	The IEEE 64-bit double floating point value.

Returns

The binary representation of the passed in floating point value as a signed long-long.

Example

The following example converts a floating point value to a binary representation that can be assigned to two 32-bit data items:

```
>>> GseosConversion.dtoll(73.4000000001)
4634865482759517462L
```

```
dTemp = 73.4000000001
```

```
HK.TempHi = GseosConversion.dtoll(dTemp) >> 32
HK.TempLo = GseosConversion.dtoll(dTemp) & (2**32-1)
```

8.4.1.5.2 GseosConversion.ftol

ftol(fValue)

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

fValue 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:

```
f1Temp = 72.34
```

```
HK.Temp = GseosConversion.ftol(f1Temp)
```

8.4.1.5.3 GseosConversion.lltod

lltod(lValue)

Performs a long-long (64-bit) to double conversion. The binary representation of the 64-bit long value is returned as a floating point double (64-bit) value. This function does not coerce to a double but interprets the 64-bit long parameter bit pattern as a IEEE floating point double.

Parameter Description

lValue The 64-bit long value to convert.

Returns

The corresponding IEEE floating point double representation.

Example

The following example converts a long long value (which is the binary representation of an IEEE double) into a double:

```
dTemp = 72.340000001
llTemp = GseosConversion.lltod(lTemp)
```

```
>>> print llTemp
4634790891890752471L
```

```
>>> print GseosConversion.lltod(l1Temp)
72.340000001
```

8.4.1.5.4 GseosConversion.ltof

ltof(lValue)

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

lValue	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:

```
f1Temp = 72.34
lTemp = GseosConversion.ftol(f1Temp)

print GseosConversion.ltof(lTemp)

>>> 72.34
```

8.4.1.5.5 GseosConversion.signed

signed(lValue, 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).

Shortcuts to commonly used functions are available with: **signed8()**, **signed16()**, and **signed32()** where the according sign bit is indicated in the function name.

Parameter Description

lValue	Unsigned long value to convert into a signed value.
wSignBit	The position of the sign bit, bits are counted starting with 0.

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.SendBlock()

print HK.Temp

>>> 510

print GseosConversion.signed(HK.Temp, 8)

>>> -2
```

8.4.1.6 GseosDecoder

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](#)^[121].

To construct a decoder you have to instantiate a decoder object of the decoder class. The [constructor](#)^[247] 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](#)^[121].

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.

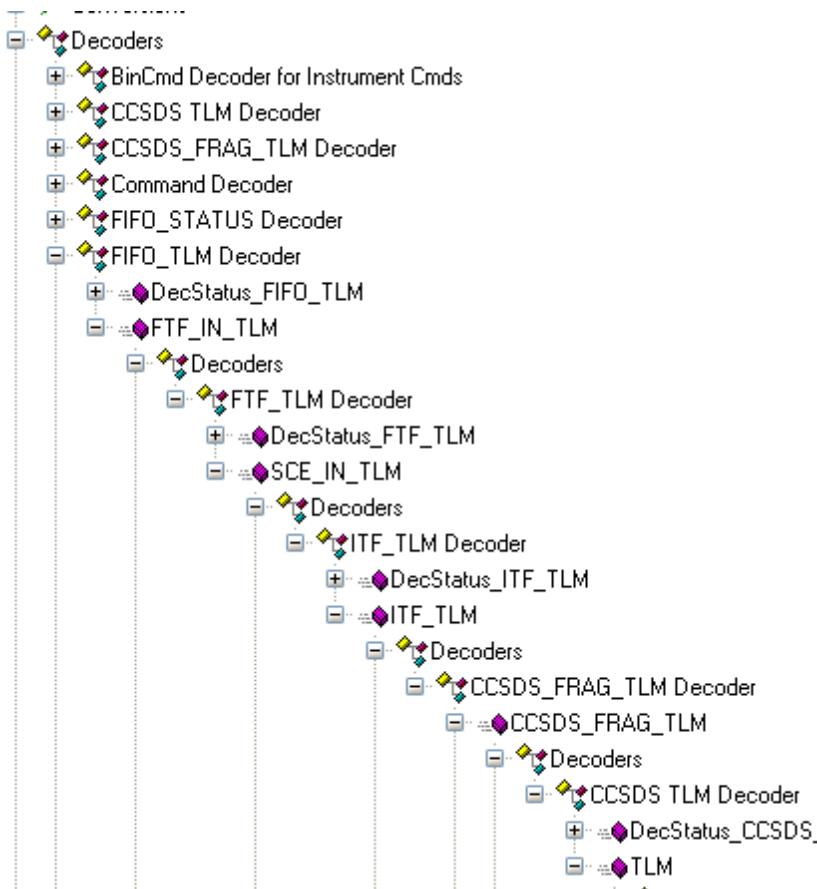
If the decoder raises an exception it will be disabled. You should catch all exceptions you don't want to terminate the decoder.

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

Often a decoder also generates status information to indicate the decoding process and/or any errors that might occur during the decoding process. Since this is such a common task you can use the [TDecoderStatus](#)^[249] class to simplify this process.

Decoder Hierarchy

Using blocks and decoders allows you to build a hierarchy of blocks and decoders. This results in high visibility of your data on all decoder levels. Usually a decoder implements a particular protocol level, this way you can verify your data at every level simplifying development of the corresponding protocol levels in your flight code. See the image below for such a hierarchy.



For more information about decoders please refer to the examples:

- [Simple dispatch decoder](#)
- [Variable length spin decoder](#)

8.4.1.6.1 GseosDecoder.TDecoder.bEnable

Property: bEnable

Enable or disable the decoder.

Example

Disable the decoder from a script:

```
oMyDec.bEnable = 0
```

8.4.1.6.2 GseosDecoder.TDecoder

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

8.4.1.6.3 GseosDecoder.TDecoder.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
```

8.4.1.6.4 GseosDecoder.TDecoder.dw Cnt

Property: dwCnt

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

8.4.1.6.5 GseosDecoder.TDecoderStatus

TDecoderStatus(strDecStatusBlock, TError)

The TDecoderStatus class makes it easy to report status information of a decoder. It will automatically export any attributes you assign to this class and that are prefixed with 'dw' or 'fl' to a specific block. The dw prefix is used to indicate DWORD which refers to an unsigned integer value, the fl prefix indicates float (this is only a naming convention). You pass in the name of the decoder status block in the constructor. This block must define items that correlate to the class attributes defined (only the ones that are prefixed with 'dw' or 'fl'). You simply assign your status information to an object of this class and call SendBlock() whenever you want to publish your decoder status.

Parameter	Description
strBlkStatus	Name of the status block to generate.
TError:	An exception class derived from Exception. This exception will be raised in case of error.

Returns

The new decoder status object.

Comments

Make sure that your block definition has all the items that are defined in your class definition.

Example

The following example shows a simple TDecoderStatus derived class and the according block definition:

```
class MyError(Exception): pass
```

```
class TDecStatus_Sample(GseosDecoder.TDecoderStatus):
    #
    # * ctor()
    *
    # *
    #
    # * Initialize the decoder status. All attributes of this class need to
    #
    # * be defined as items of the destination block (without the dw prefix).
    #
    # *
    #
    # * Parameters: strDestBlk: Name of the destination status block.
    #
    #
    ****
    def __init__(self, strDecStatusBlock, TError=TMyError):
        #
        ----- #
        # - Initialize our member data.
        - #
        # - These must be defined in the destination block.
        - #
        #
        ----- #
        self.dwSrcBlkCnt    = 0
        self.dwSrcByteCnt   = 0
        self.dwDestBlkCnt   = 0

        #
        ----- #
        # - Initialize base class.
        - #
        #
        ----- #
        GseosDecoder.TDecoderStatus.__init__(self, strDecStatusBlock, TError)

    #
    # The according block definition
    #
    DecStatus_Sample
    {
        SrcBlkCnt    ,,, 32;
        SrcByteCnt   ,,, 32;
        DestBlkCnt   ,,, 32;
    }
```

8.4.1.6.6 Examples

8.4.1.6.6.1 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](#)^[150] 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 GseosDecoder  
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:  
        Gseos.Message('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.SendBlock()
```

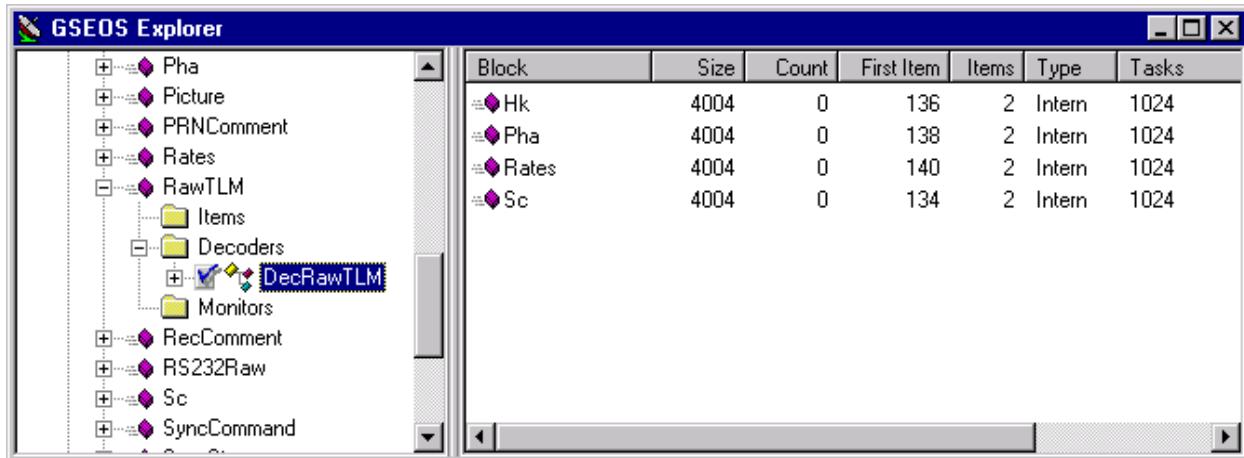
Once we have the decoder function defined we have to assign it to the RawTLM block to get executed:

```
# Create a new decoder.
#
oDecRawTLM = GseosDecoder.TDecoder('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](#)^[126]. 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.

8.4.1.6.6.2 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](#)^[171] file in the [Config/BlkFiles](#)^[179] 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 GseosDecoder

#
#
# 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. Then 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.SendBlock()

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

```

```
[TOFSpin]))
```

8.4.1.6.7 GseosDecoder.TDecoder.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.

8.4.1.7 GseosError

All exceptions GSEOS can raise (besides general Python exceptions of course) are exported in the GseosError module. The exceptions are structured hierarchically and you can catch any GSEOS specific exception by handling TGseosError. Most exceptions fall into groups related to a specific GSEOS module:

Gseos

TGseosError:	The base class of all GSEOS exceptions.
TWindowNotFoundError:	Any functions that work on specific windows will raise this error if the specified window can not be found.
TConsoleError:	Base class for console function errors.
TConsoleFileError:	Exception if there are problems with console files.
TLogError:	Base class of all log function errors.
TLogFileError:	File related log errors.
TLogNotFoundError:	The requested log could not be found.
TStatusMapError:	Base class for all status map errors.
TStatusMapRangeError:	The range is not valid.
TStatusMapColorError:	The color is not valid.
TStatusMapSyntaxError:	There is a syntax error in the status map file.
TStatusMapNoMatchError:	The value did not match a defined value or range.
TStatusMapImageNotDefinedError:	The image is not defined.
TStatusMapImageNotFoundError:	The image file can not be found.
TStatusMapImageLoadError:	The image file could not be loaded.

GseosBlocks

TItemDescriptionError:	The item description is not valid.
TBDBMBlockError:	Base class for block errors.

GseosCmd

TCmdError:	Base class for command errors.
TCmdParseError:	The command definition file could not be parsed, you can also get XML specific exceptions if your command definition is not well formed.
TCmdFormatError:	The command is not formatted properly.
TCmdUnknownCmdError:	The command is unknown.
TCmdArgValueError:	A command argument has an invalid value.
TCmdArgMissingError:	A command argument is missing.
TCmdTypeError:	A command argument has an invalid type.
TCmdBatchError:	Command batch file errors.
TCmdMenuError:	Base class for all command menu related errors.

TCmdMenuSyntaxError:	Syntax error in command menu file.
TCmdMenuItemError:	Error including menu file.
TCmdMenuNestingError:	Error nesting command menus, this could be a circular include.

GseosFileUpload

TFileUploadError:	Base class for all File Upload errors.
TFileUploadConfigError:	Error in the File Upload gseos.ini configuration.
TFileUploadCallbackError:	Either the StartHandler or TimerHandler callback function raised an exception.

GseosSequencer

TSequencerError:	Base class for all sequencer related errors.
TSeqAbortError:	The sequencer was aborted by the user.
TSeqTimeoutError:	The sequencer timed out.

8.4.1.8 GseosExport

The GseosExport module exposes a Python API to control the GSEOS Data Export tool from Python script. For more information about the Data Export tool refer to [The Data Export Dialog](#)¹³⁸ chapter.

The GseosExport module exposes the following functions:

- [IsEnabled\(\)](#)²⁵⁷
- [Enable\(\)](#)²⁵⁷
- [GetActiveConfig\(\)](#)²⁵⁸
- [SetActiveConfig\(\)](#)²⁵⁸

8.4.1.8.1 GseosExport.IsEnabled**.IsEnabled()**

Determine if the Data Export tool is currently enabled.

Parameters

-

Returns

bEnabled: True, if the Data Export is enabled.

8.4.1.8.2 GseosExport.Enable**Enable(bEnable)**

Enabled or disable the Data Export too. If the Data Export tool is not properly configured

(e.g. no export file is configured) enabling the tool won't start the Data Export. You can use [IsEnabled\(\)](#)²⁵⁷ to verify if the Data Export tool is active.

Parameter **Description**
bEnable True to turn on the Data Export tool.

Returns
None

8.4.1.8.3 GseosExport.GetActiveConfig

GetActiveConfig()

Return the currently active Data Export configuration.

Parameter **Description**
-

Returns
strConfigName The name of the currently active Data Export configuration.

8.4.1.8.4 GseosExport.SetActiveConfig

SetActiveConfig()

Set a new configuration for the Data Export.

Parameter **Description**
strConfigName: Name of the new configuration. You have to make sure this matches the name of a defined configuration.

Returns
-

8.4.1.9 GseosIni

The GseosIni module allows you easy access to the GSEOS configuration options that can be set via the [gseos.ini](#)¹⁷¹ file.

That way you can either check system settings or parse your own configuration options that you might want to add to gseos.ini. Since there can be multiple configuration files and they can contain include statements on top level or on section level and/or use

various instance settings it is not trivial to simply parse a configuration file since the settings might be imported from a different file. The GseosIni module helps in resolving this issue and will operate on the correct section and read/write the settings to the correct effective file.

You can read a simple entry with [GseosIni.fGetEntry\(\)](#)²⁵⁹. If your setting is a boolean value (Enabled, Disabled, yes, no, enable, disable) you can use the [GseosIni.fIsEnabled\(\)](#)²⁶⁰ function instead and don't have to parse the result string of GseosIni.fGetEntry(). To find out the configured sections you can use [GseosIni.fListSectionNames\(\)](#)²⁶¹. If you want to know all the keys/entries within a section call [GseosIni.fListEntries\(\)](#)²⁶⁰. Since a single entry can contain one or more values you can use the [GseosIni.fGetEntries\(\)](#)²⁶⁰ function to retrieve all values for a specific Section/Entry. If you need to remove specific entries from a section you can do so with [GseosIni.fRemoveEntries\(\)](#)²⁶¹. To remove an entire section use [GseosIni.fRemoveSection\(\)](#)²⁶¹. To add a new entry use [GseosIni.fAddEntry\(\)](#)²⁵⁹. If you want to modify an existing entry you can do so with [GseosIni.fSetEntry\(\)](#)²⁶¹. This will replace the first matching entry in the specified section with the new value. After you have modified a section you have to write it back to the according underlying file. You do that by calling [GseosIni.fWriteSection\(\)](#)²⁶².

8.4.1.9.1 GseosIni.fAddEntry

fAddEntry()

Add a new entry to a section. If the entry exists a new value will be added.

Parameter	Description
strSection	The section name.
strEntry	The entry to remove.
strValue	The value to add.

Returns

-

8.4.1.9.2 GseosIni.fGetEntry

fGetEntry()

Get a specific entry. If the entry does not exist return the default value instead. If this entry has multiple values return the first one. If the strSection parameter is None all section names are returned. If the strEntry parameter is None all entry names in the specified section are returned. If section names or entry names are returned they are separated by 0 and the string is terminated with two zeros.

Parameter	Description
strSection	The original section name.
strEntry	The entry to look up.
strDefault	The default entry to return if no match is found.

Returns

strValues The return string as discussed above.

8.4.1.9.3 GseosIni.fGetEntries

fGetEntries()

Get a list of values for the specific section and entry.

Parameter

strSection
strEntry

Description

The section name.
The entry to look up.

Returns

ctValues

A list of values. Can be empty if no match is found.

8.4.1.9.4 GseosIni.fIsEnabled

fIsEnabled()

This is a convenience function to get a boolean value from a setting.

If the entry is one of: Yes, yes, Enabled, Enable, enabled, enable we consider the entry as True, otherwise as False.

Parameter

strSection
strEntry
bDefault

Description

The section name.
The entry to look up.
The default value to return if no entry was found.

Returns

bEnabled

True, if the setting is enabled.

8.4.1.9.5 GseosIni.fListEntries

fListEntries()

List the entry names of the requested section. If the section does not exist an empty list is returned.

Parameter

strSection

Description

The section name.

Returns

ctEntries

A list of entries in the section.

8.4.1.9.6 GseosIni.fListSectionNames

fListSectionNames()

Get a list of all section names that match the passed in regular expression.

Parameter	Description
strMatch	A regular expression string. If not specified matches all sections.
Returns	
ctSections	A list of section names that match the regular expression.

8.4.1.9.7 GseosIni.fRemoveEntries

fRemoveEntries()

Remove all entries that match strEntry from the section. If you want to commit this change to the actual ini file use [GseosIni.fWriteSection\(\)](#) 262 afterwards.

Parameter	Description
strSection	The section name.
strEntry	The entry to remove.
Returns	-

8.4.1.9.8 GseosIni.fRemoveSection

fRemoveSection()

Remove an entire section. If you want to commit this change to the actual ini file use [GseosIni.fWriteSection\(\)](#) 262 afterwards.

Parameter	Description
strSection	The section name.
Returns	-

8.4.1.9.9 GseosIni.fSetEntry

fSetEntry()

Replace the first occurrence of an entry with the same name with the new value, delete all subsequent entries with the same name. This will only leave this one entry. If no entry with this name exists add the entry to the end of the section.

Parameter	Description
strSection	The section name.
strEntry	The entry to modify.
strValue	The value to set.

Returns

-

8.4.1.9.10 GseosIni.fWriteSection**fWriteSection()**

After modifying a section by either removing entries or [adding new ones](#) [259] or [updating existing ones](#) [261] call fWriteSection() to make the changes permanent. The section is written back to the file where this section originated.

Parameter	Description
strSection	The section name.

Returns

-

8.4.1.10 GseosFileUpload

The GseosFileUpload module exposes a Python API to control the file upload functionality to Python script. For more information on the configuration of file uploads refer to the [\[FileUploads\]](#) [180] section in [gseos.ini](#) [171]. The [FAQ](#) [332] also illustrates on how to implement a file upload.

The GseosFileUpload module exposes the following functions:

- [StartUpload\(\)](#) [262]
- [AbortUpload\(\)](#) [263]
- [IsUploadActive\(\)](#) [263]

```
'IsEnabled',
'Enable',
'GetActiveConfig',
'SetActiveConfig',
```

8.4.1.10.1 StartUpload**StartUpload()**

Start a new file upload for the specified file upload type.

Parameters

strFileUpload:	The name of the file upload in question. This is the name of the File Upload as specified in the [FileUploads] section.
strFile:	The file to upload. This must match the extension configured for the File Upload specified with the first argument.
ctArgs:	A list of arguments to pass along to the StartHandler() callback function. This must match the number of arguments configured in the File Upload specified with the first argument.

Returns

None

8.4.1.10.2 AbortUpload

AbortUpload()

Abort an active file upload. If there is no file upload active for the selected File Upload type this is a no-op.

Parameters

strFileUpload:	The name of the file upload to abort. This is the name of the File Upload as specified in the [FileUploads] section (not the file name of the file upload!).
----------------	--

Returns

None

8.4.1.10.3 IsUploadActive

IsUploadActive()

Determine if the specified file upload is active.

Parameters

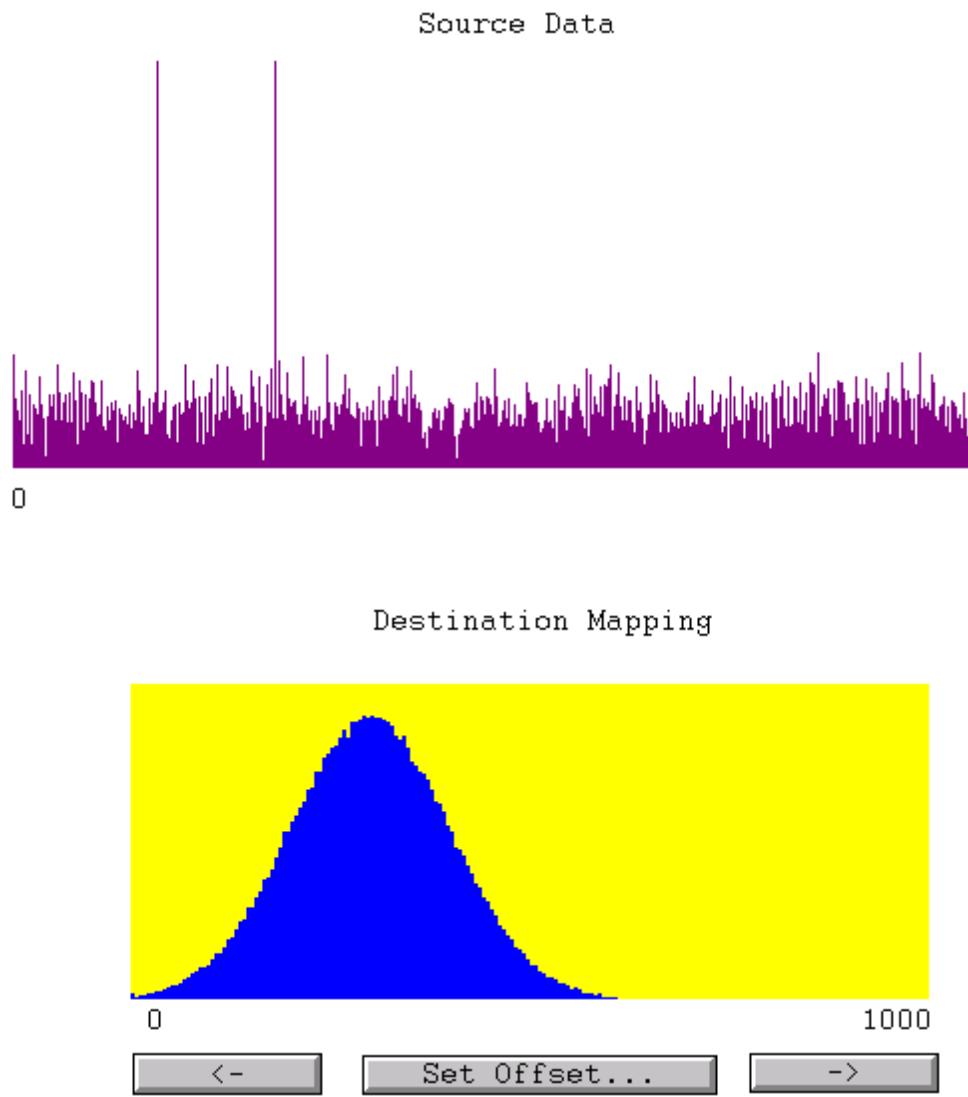
strFileUpload:	The name of the file upload in question. This is the name of the File Upload as specified in the [FileUploads] section.
----------------	---

Returns

bActive:	True if a file for this File Upload is currently being uploaded.
----------	--

8.4.1.11 GseosHistogram

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 a sample of a 1D 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:

```
Destination Position = x + 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. The YScale item in the destination block will be set to the current scale value. 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
 strSourceXItem
 strDestinationItem
 YMax
 ClipLeft
 ClipRight
 strEventCntItem
 bAutoScaleMode
 bLogMode
 strCalcMode

2D: Name
 strSourceXItem
 strSourceYItem
 strDestinationItem
 HorzDimension
 ClipLeft
 ClipRight
 ClipBottom
 ClipTop
 Range
 EventCnt

The strSourceXItem and strSourceYItem 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. The strDestinationItem parameter specifies the destination item name and must be an array item.

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!!!

The Left, Right, Bottom, and Top 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 Top 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 YMax parameter allows you to set a maximum value for the histogram. If auto scaling is turned on the entire histogram is scaled down by a factor of 2 if any histogram bin reaches the YMax value. If auto scaling is off the destination item is simply not incremented once it reaches YMax.

The strEventCntItem parameter is the name of a data item that is read at runtime to determine the number of input values to read. By default, that is if no strEventCntItem item is specified all values in the strSourceXItem 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 strEventCntItem item. It is assumed that the strEventCntItem will hold the currently valid number of items in the strSourceXItem (and strSourceYItem in the 2D case) data item. Only these items contribute to the histogram.

The histogram decoder exports status data to the destination block. If you define the following elements in your destination block they will be populated by the histogram decoder:

EventCnt:	The total number of histogram events.
YMax:	The configured maximum Y value.
YScale:	The current Y scale factor.
AutoScaleMode:	Flag that indicates if auto scale mode is on.
LogMode:	Flag that indicates if log mode is on.
CalcMode:	0: Min, use the minimum value, 1: Max, use the maximum value, 2: Mean, use the mean of all values.
Zoom:	The current zoom factor in percent.
HorzOffset:	The current horizontal offset.
VertOffset:	The current vertical offset (2D histograms only).
Left:	The left position of the histogram.
Right:	The right position of the histogram.
Top:	The top position of the histogram (2D histograms only).
Bottom:	The bottom position of the histogram (2D histograms only).

Here a sample block definition for a destination block:

```
Histo1Dest
{
    EventCount    ,,, 32;
    YMax          ,,, 32;
```

```

Zoom      ,,, 16;
HorzOffset ,,, 32;
VertOffset ,,, 32;
Left       ,,, 32;
Right      ,,, 32;
Top        ,,, 32;
Bottom     ,,, 32;
LogMode    ,,, 16;
CalcMode   ,,, 16;
AutoScaleMode ,,, 16;
YScale     ,,, 32;
Data[200]   ,,, 16;
}

```

For more information please refer to the 1D and 2D histogram decoders:

- [1D Histogram](#)
- [2D Histogram](#)

8.4.1.11.1 GseosHistogram.Clear

Clear()

Clear the histogram and reset all control data.

Parameter

Description

strName The histogram name as defined in the histogram constructor.

Returns

None

Example

```
import GseosHistogram
```

```
GseosHistogram.THistogram1D('Histol', 'HistolSrc.Data', 'HistolDest.Data',
400, 0, 1000)
GseosHistogram.Clear('Histol')
```

8.4.1.11.2 GseosHistogram.DownYScale

DownYScale()

Half the Y scale factor if possible.

Parameter

Description

strName The histogram name as defined in the histogram constructor.

Returns

None

8.4.1.11.3 GseosHistogram.MoveLeft

MoveLeft([wPercent=10])

Move to the left by a certain percentage. This only takes effect if the zoom factor is greater than 100%.

Parameter

strName

wPercent

Description

The histogram name as defined in the histogram constructor.

Percent to move to the left. The default is 10%.

Returns

None

8.4.1.11.4 GseosHistogram.MoveRight

MoveRight([wPercent=10])

Move to the right by a certain percentage. This only takes effect if the zoom factor is greater than 100%.

Parameter

strName

wPercent

Description

The histogram name as defined in the histogram constructor.

Percent to move to the right. The default is 10%.

Returns

None

8.4.1.11.5 GseosHistogram.MoveUp

MoveUp([wPercent=10])

Move to the top by a certain percentage. This only takes effect if the zoom factor is greater than 100%. Applies only to 2D histograms.

Parameter

strName

wPercent

Description

The histogram name as defined in the histogram constructor.

Percent to move to the top. The default is 10%.

Returns

None

8.4.1.11.6 GseosHistogram.MoveDown

MoveDown([wPercent=10])

Move to the bottom by a certain percentage. This only takes effect if the zoom factor is

greater than 100%. Applies only to 2D histograms.

Parameter

strName
wPercent

Description

The histogram name as defined in the histogram constructor.
Percent to move to the top. The default is 10%.

Returns

None

8.4.1.11.7 GseosHistogram.SetAutoScaleMode

SetAutoScaleMode(bEnable)

Turn auto scale mode on/off.

Parameter

strName
bEnable

Description

The histogram name as defined in the histogram constructor.
Enable/Disable auto scale mode.

Returns

None

8.4.1.11.8 GseosHistogram.SetCalcMode

SetCalcMode(strCalcMode)

Set the calculation mode to 'Min', 'Max', or 'Mean'. If the histogram can't display all values (for example the histogram is zoomed out) the value to be displayed is determined from all values given the calculation mode configured. In Min mode the minimum value of the group of values is taken, in Max mode the maximum value and in Mean mode the mean value is used.

Parameter

strName
strCalcMode

Description

The histogram name as defined in the histogram constructor.
One of 'Min', 'Max', or 'Mean'.

Returns

None

8.4.1.11.9 GseosHistogram.SetHorzOffset

SetHorzOffset(dwOffset)

Sets the horizontal offset of the left border of the histogram. If the offset is out of

boundary this function has no effect.

Parameter

strName

dwOffset

Description

The histogram name as defined in the histogram constructor.

The offset to set for the left side of the histogram.

Returns

None

8.4.11.10 GseosHistogram.SetVertOffset

SetVertOffset(dwOffset)

Sets the vertical offset of the bottom border of the histogram. If the offset is out of boundary this function has no effect. This only applies to 2D histograms.

Parameter

strName

dwOffset

Description

The histogram name as defined in the histogram constructor.

The offset to set for the bottom side of the histogram.

Returns

None

8.4.11.11 GseosHistogram.SetLogMode

SetLogMode(strLogMode)

Set the log mode to logarithmic ('Log') for linear ('Lin'). In logarithmic mode the logarithm of the histogram values is taken, in linear mode the histogram bin are displayed as is.

Parameter

strName

strLogMode

Description

The histogram name as defined in the histogram constructor.

One of 'Log', or 'Lin'.

Returns

None

8.4.11.12 GseosHistogram.SetYScale

SetYScale(dwScale)

Sets the Y scale factor. The Y scale factor is used to scale the Y values of the histogram. If auto scale mode^[269] is turned on the histogram will automatically double the Y scale

factor every time the largest bin in the histogram reaches YMax. You can manually set the Y scale factor, however, if the autoscale mode is on it will ensure that the histogram will fit into the configured YMax range.

Parameter	Description
strName	The histogram name as defined in the histogram constructor.
dwYScale	The Y scale factor.
Returns	
None	

8.4.1.11.13 GseosHistogram.SetZoom

SetZoom(wPercent)

Sets the zoom factor in percent. The zoom factor changes your horizontal view. 100% means the entire histogram is visible, larger zoom factors will show smaller portions of the histogram at higher resolution. You can also set the zoom factor to less than 100, in this case the histogram will be compressed at the possible loss of resolution.

Parameter	Description
strName	The histogram name as defined in the histogram constructor.
wPercent	The zoom factor in percent.
Returns	
None	

8.4.1.11.14 GseosHistogram.ToggleAutoScaleMode

ToggleAutoScaleMode()

Toggle the auto scale mode. This is a convenient function to assign to a button for quick switching between auto scale modes.

Parameter	Description
strName	The histogram name as defined in the histogram constructor.
Returns	
None	

8.4.1.11.15 GseosHistogram.ToggleLogMode

ToggleLogMode()

Toggle the log mode between Lin and Log. This is a convenient function to assign to a button for quick switching between log modes.

Parameter	Description
strName	The histogram name as defined in the histogram constructor.
Returns	
None	

8.4.1.11.16 GseosHistogram.UpYScale

UpYScale()

Double the Y scale factor if possible.

Parameter	Description
strName	The histogram name as defined in the histogram constructor.
Returns	
None	

8.4.1.11.17 THistogram1D

```
THistogram1D(strName, strSourceXItem, strDestItem, [YMax], [ClipLeft],
[ClipRight], [strEventCntItem=None], [bAutoScaleMode=False],
[strLogMode='Lin'], [strCalcMode='Mean'])
```

The Histogram1D constructor creates a new 1D histogram decoder. You have to specify a unique name (amongst all decoders), the source data item, the destination data item, and the dimensions of the clipping area. Optionally you can also specify an event count item and several mode flags.

Parameter	Description
strName	The unique name identifies the histogram decoder. This is the name you will see in the GSEOS Explorer.
strSourceXItem	The strSourceXItem 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! The source item has to be an array item. If you only have a single value declare an array item with dimension

	1.
strDestItem	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).
YMax	The maximum Y value any bin in the histogram can take on. If auto scaling is enabled the histogram will be scaled down, otherwise it will be clipped at the YMax value.
ClipLeft	The ClipLeft and ClipRight 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-Left = 7-6 = 1 to be updated (incremented). Therefore the clipping area allows you to map a certain area of your source data onto the destination. The ClipRight value is not included in the range. E.g.: If you specify 0 for ClipLeft and 256 for ClipRight the indices from 0..255 will be mapped. This parameter is optional, if not specified it will be set to 0.
ClipRight	See ClipLeft. This parameter is optional, if not specified it will be set to the number of elements in the destination item (the histogram buffer).
strEventCntItem	The strEventCntItem 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 strEventCntItem item is specified all values in the strSourceXItem 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 strEventCntItem item. It is assumed that the strEventCntItem will yield the currently valid number of items in the strSourceXItem data item. Only these items contribute to the histogram. The strEventCntItem item does not have to be located in the source block but can be any item.
bAutoScaleMode	If set the histogram will operate in auto scale mode. The YScale item indicates the current scaling factor.
strLogMode	Can be 'Log' or 'Lin'. If log mode the source items are computed using the log function. In linear mode the values are added up verbatim.
strCalcMode	In the case that multiple source values map into one destination bin the calculation mode allows you to control how the mapping is performed. The available modes are: 'Min', the minimum value, 'Max', the maximum value, 'Mean', the mean value.

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](#)^[246].

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: GseosHistogram.THistogram1D('MyHisto',
 'Block.Source',)
 Do NOT use: GseosHistogram.THistogram1D('MyHisto', Block.Source,)

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.

```
GseosHistogram.THistogram1D('My1DHistogram',
                            'SrcBlock.SrcItemX',
                            'DestBlock.DestItem',
                            20,
                            0, 128,
                            'Block.EventCnt')
```

8.4.1.11.17.1 Example

The following example demonstrates a simple 1D histogram. There are different files involved in setting up this sample. First we have to define the source and destination blocks of the histogram data. We do this in a [block definition file](#) Histo.blk:

```
Histo1Src
{
    Data[500]    ... 16;
}

Histo1Dest
{
    EventCount    ... 32;
    YMax          ... 32;
    Zoom          ... 16;
    HorzOffset    ... 32;
    Left           ... 32;
    Right          ... 32;
    LogMode        ... 16;
    CalcMode       ... 16;
    AutoScaleMode ... 16;
    YScale         ... 32;
    Data[200]      ... 16;
}
```

The first block Histo1Src simply has one data item that contains the data we want to histogram. The second block is the histogram block. The Data item will receive the histogram data, in this case 400 bins. The other items are status items that will get populated by the histogram decoder and that you can add to the screen for some additional status information.

Our histogram decoder is then created with the following Python script:

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:

```
#*
*****
*#
## * Histogram Sample.
* *#
#*
*****
*#
import GseosHistogram

GseosHistogram.THistogram1D('Hist01', 'Hist01Src.Data', 'Hist01Dest.Data',
300, 0, 1000)
GseosHistogram.SetAutoScaleMode('Hist01', True)
GseosHistogram.SetCalcMode('Hist01', 'Min')
```

The above script sets the maximum Y value to 300. We also turn auto scale mode on. This means if we set the Y max value of the display range to 300 the histogram will always remain within the limits of the display item. The Y scale factor will change as we add up the bins. The clip region is defined from 0 to 1000. That means that source values from 0 through 999 will be binned into our destination histogram. Since we only display 400 bins (the destination item has been defined as an array of 400) we will on average combine 2.5 bins into one for display. Since we have set the calculation mode to 'Min', the minimum value of the two or three values used to generate the destination bin is taken.

In order to invoke operations on the histogram we use the name that we have assigned to the histogram when we constructed it. The sample display screen defines some buttons with operations on the histogram.

To run this example you can restart GSEOS with the /ini Demo/Histo1D/gseos.ini command line switch. If you click the Start button on the Histo1DDemo screen a random generator will generate the source data with a gauss distribution and you can use the histogram commands to change the view of the histogram.

8.4.1.11.18 THistogram2D

**THistogram2D(strName, strSrcHorzItem, strSrcVertItem, strDestItem,
HorzDimension, [YMax], [Left], [Right], [Bottom], [Top], [strEventCntItem],
[bAutoScaleMode=False], [strLogMode='Lin'], [strCalcMode='Mean'])**

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 horizontal dimension of the destination matrix. Optionally

you can specify a clipping area, event count item, and some histogram modes.

Parameter	Description
strName	The unique name identifies the histogram decoder. This is the name you will see in the GSEOS Explorer.
strSrcHorzItem	The strSrcHorzItem parameter specifies the item name of the horizontal source item. You have to specify the name of a data item here (string) and not the data item itself. If you specify an array item you can specify the number of elements to use for histogramming: Histo1Src.HorzData[:12]. If the number of elements to be evaluated is dynamic you can use the EventCntItem (see below).
strSrcVertItem	The strSrcVertItem parameter specifies the item name of the vertical source item. You have to specify the name of a data item here (string) and not the data item itself. If you specify an array item you can specify the number of elements to use for histogramming: Histo1Src.VertData[2:4]. If the number of elements to be evaluated is dynamic you can use the EventCntItem (see below). The dimension of the horizontal and vertical source items should match. In most cases the horizontal and vertical items will be from the same source block. However, if you configure the items from different source blocks the histogram will only be evaluated on arrival of the horizontal source item block.
strDestItem	This parameter specifies the destination item name (note again that this is a string, e.g.: 'Histo.Data').
HorzDimension	The HorzDimension parameter specifies the dimension of the horizontal axis of the histogram matrix. The vertical dimension is determined by the length of the destination item divided by the horizontal dimension.
YMax	The maximum Y value any bin in the histogram can take on. If auto scaling is enabled the histogram will be scaled down, otherwise it will be clipped at the YMax value.
ClipLeft	The ClipLeft and ClipRight parameters specify the horizontal dimensions of the clipping area. The horizontal 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-Left = 7-6 = 1 to be updated (incremented). Therefore the clipping area allows you to map a certain area of your source data onto the destination. The ClipRight value is not included in the range. E.g.: If you specify 0 for ClipLeft and 256 for ClipRight the indices from 0..255 will be mapped. This parameter is optional, if not specified it will be set to 0.
ClipRight	See ClipLeft. This parameter is optional, if not specified it will be set to the horizontal dimension as specified in HorzDimension.
ClipBottom	Similar to ClipLeft but for the vertical dimension. The ClipBottom and ClipTop parameters specify the vertical dimensions of the clipping area. The vertical source values are only mapped into the destination item if they fall within the clipping area. E.g. if the clipping interval is Bottom=6, Top=12 a source value of 5 would be discarded, a source value of 7 would cause the destination item at vertical position 7-Bottom = 7-6 = 1 to be updated (incremented), given the horizontal dimension is within the horizontal clipping area. Therefore the clipping area allows you to map a certain area of your source data onto the destination. The ClipTop value is not included in the range. E.g.: If you specify 0 for ClipBottom and 256 for ClipTop the indices from 0..255 will be mapped. This parameter is

ClipTop	optional, if not specified it will be set to 0. See ClipBottom. This parameter is optional, if not specified it will be set to the number of elements in the vertical direction, that is the lenght of the destination item divided by the horizontal dimension as configured with HorzDimension.
strEventCntItem	The strEventCntItem 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 strEventCntItem item is specified all values in the strSrcHorzItem, strSrcVertItem items 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 strEventCntItem item. It is assumed that the strEventCntItem will yield the currently valid number of items in the source data items. Only these items contribute to the histogram. The strEventCntItem item does not have to be located in the source block but can be any item.
bAutoScaleMode	If set the histogram will operate in auto scale mode. The YScale item indicates the current scaling factor.
strLogMode	Can be 'Log' or 'Lin'. If log mode the source items are computed using the log function. In linear mode the values are added up verbatim.
strCalcMode	In the case that multiple source values map into one destination bin the calculation mode allows you to control how the mapping is performed. The available modes are: 'Min', the minimum value, 'Max', the maximum value, 'Mean', the mean value.

Returns

The new histogram object.

Comments

The histogram is automatically registered for its source block (only the horizontal source block will trigger the histogram computation). There is no need to append it to the Decoder list of the source block like you have to do for an ordinary [decoder](#)^[246].

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: GseosHistogram.THistogram2D('MyHisto', 'Block.HorzSource',
)
 Do NOT use: GseosHistogram.THistogram2D('MyHisto',
 Block.HorzSource,)

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 which clips the input data to the area 0, 400, 0, 100.

```
GseosHistogram.THistogram2D('My2DHistogram',
                           'SrcBlock.SrcHorzItem[:20]',
```

```
'SrcBlock.SrcVertItem[:20]',
'DestBlock.DestItem',
400,
256,
0, 400, 0, 100)
```

8.4.1.11.18.1 Example

The following example demonstrates a simple 2D histogram. There are different files involved in setting up this sample, they can be found in the Demo\Histo2D folder. First we have to define the source and destination blocks of the histogram data. We do this in a [block definition file](#) 150 Histo.blk:

```
Histo2Src
{
    X[20]    ... 16;
    Y[20]    ... 16;
}

Histo2Dest
{
    EventCount    ... 32;
    YMax          ... 32;
    YScale         ... 32;
    YMaxScaled    ... 32;
    Zoom           ... 16;
    Left            ... 32;
    Right           ... 32;
    Bottom          ... 32;
    Top             ... 32;
    HorzOffset     ... 32;
    VertOffset     ... 32;
    LogMode         ... 16;
    CalcMode        ... 16;
    AutoScaleMode   ... 16;
    Data[80000]     ... 16;
}
```

The first block Histo2Src has the horizontal and vertical source items that contain the data we want to histogram. The second block is the destination histogram block. The Data item will receive the histogram data, in this case 80000 bins. This item will be displayed as a bitmap and we will specify the horizontal dimension of the bitmap in both the Range property of the bitmap display as well as the histogram constructor as HorzDimension. We will use 400 for the horizontal dimension, which will result in a vertical dimension of 200. The other items are status items that will get populated by the histogram decoder and that you can add to the screen for some additional status information.

Our histogram decoder is then created with the following Python script:

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:

```
#*
*****
*#
#* * Histogram Sample.
* *#
#*
*****
*#
import GseosHistogram

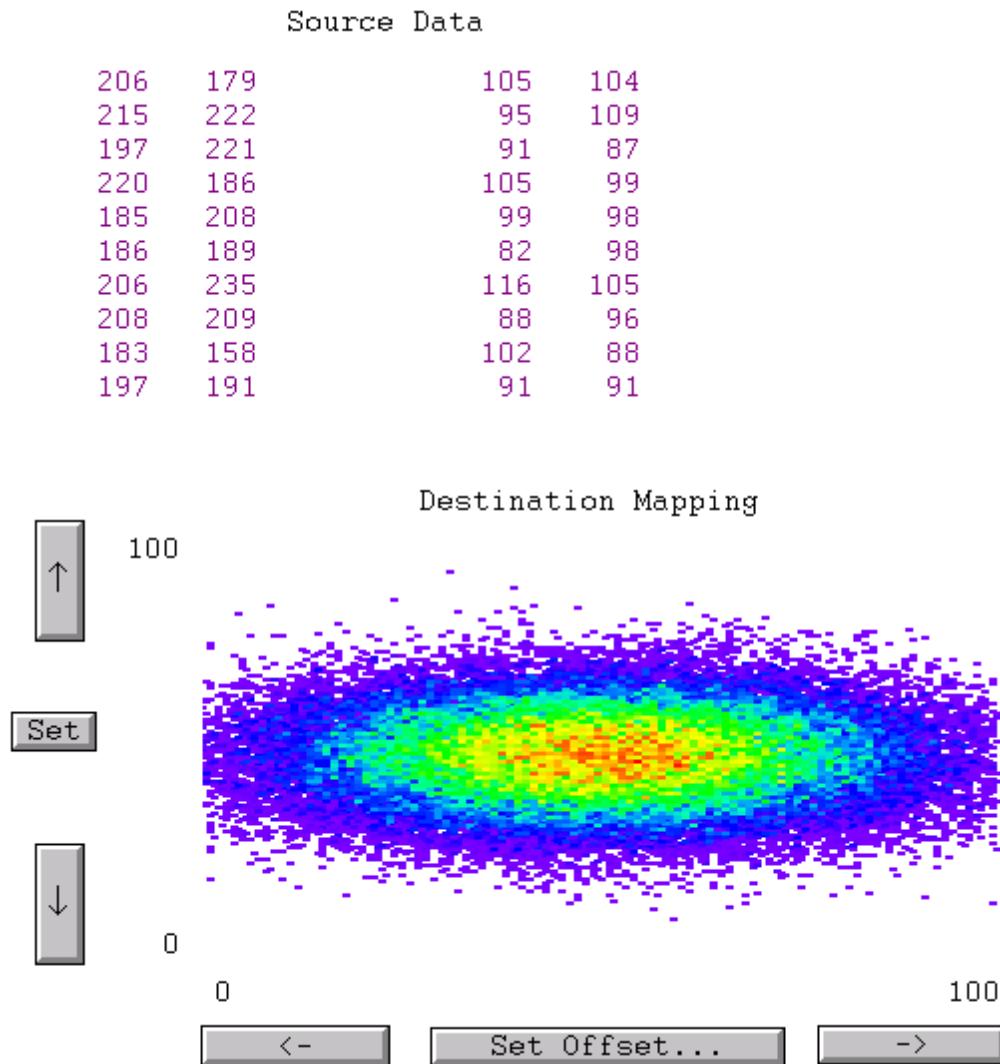
GseosHistogram.THistogram2D('Histo2', 'Histo2Src.X[:10]',
'Histo2Src.Y[:10]', 'Histo2Dest.Data',
        400, 1000,
        150, 250, 50, 150)
GseosHistogram.SetAutoScaleMode('Histo2', True)
```

The above script sets the horizontal dimension to 400 and the maximum Y value to 1000. We also turn auto scale mode on. This means if we set the Y max value of the display range to 1000 the histogram will always remain within the limits of the display item. The Y scale factor will change as we add up the bins. The clip region is defined from 150 to 250 in the horizontal direction and from 50 to 150 in the vertical. That means that horizontal source values from 150 through 250 and vertical source values from 50 to 150 will be binned into our destination histogram.

In order to invoke operations on the histogram we use the name that we have assigned to the histogram when we constructed it. The sample display screen defines some buttons with operations on the histogram.

To run this example you can restart GSEOS with the /ini Demo/Histo2D/gseos.ini command line switch. A random generator with will generate the source items with a gauss distribution around (200, 100). You can use the histogram commands to change the view of the histogram.

The screenshot below shows the histogram after accumulating some data.



8.4.1.12 GseosLog

The GseosLog module gives you access to the logging API. For an overview of the logging functions check [here](#)^[18].

8.4.1.12.1 GseosLog.Log

Log(strLogFile, strText, [wColorID=None], [byAttr=0])

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. 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 wColorID	The text to be appended to the the log file.. Optional, a color ID specifying the color of the text to insert. The module defines several color definitions:BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, LTGRAY, GRAY, LTBLUE, LTGREEN, LTCYAN, LTRED, LTMAGENTA, YELLOW, WHITE. If not specified (None) the configured foreground color will be used.
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:

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

8.4.1.12.2 GseosLog.Reload

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.

ParameterstrLogFile **Description**

The file name or the name of the AutoLog of the log file to reload.

Returns

None

Example

Reload the log file from the previous example:

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

8.4.1.12.3 GseosLog.Print

Print(strLogFile)

Print the log file. For automatic logs only the current log file is printed.

Parameter	Description
strLogName	The file name of the log file to print or the name of the AutoLog.

Returns

None

Example

Print the log file:

```
GseosLog.Print('testprotocol.log')
```

8.4.1.12.4 GseosLog.AddAutoLog

AddAutoLog(strLogFile)

Add a new auto log. This must be a new log that doesn't exist already. The log doesn't get added to the gseos.ini file on termination. If you want a permanent AutoLog you must add it to the gseos.ini file.

Parameter	Description
strLogName:	The log name.
strPathTemplate:	The path template.
strFileTemplate:	The file template.
strPathRolloverSession:	'Enabled', 'Disabled', or 'Only'. (Default is Enabled.)
strPathRolloverMaxFiles:	Maximum number of files in path. (Default is 0, no limit.)
strFileRolloverSession:	'Enabled', 'Disabled', or 'Only'. (Default is Enabled.)
strFileRolloverMaxSize:	Maximum log file size. Can be specified in KB, MB, or GB. (Default is 0, no limit.)

Returns

None

Example

Add a new AutoLog:

```
GseosLog.AddAutoLog('MyAutoLog', 'LogPath_%Y%m%d', 'CmdLog')
```

8.4.1.12.5 GseosLog.AutoLogUpdateSettings

AutoLogUpdateSettings(strLogFile)

Update the settings of an existing Auto Log. This allows you to reconfigure the log settings at runtime.

Parameter	Description
strLogName:	The log name.
strPathTemplate:	The path template.
strFileTemplate:	The file template.
strPathRolloverSession:	'Enabled', 'Disabled', or 'Only'. (Default is Enabled.)

strPathRolloverMaxFiles:	Maximum number of files in path. (Default is 0, no limit.)
strFileRolloverSession:	'Enabled', 'Disabled', or 'Only'. (Default is Enabled.)
strFileRolloverMaxSize: (Default is 0, no limit.)	Maximum log file size. Can be specified in KB, MB, or GB.

Returns

None

Example

Update an existing AutoLog:

```
GseosLog.AutoLogUpdateSettings('MyAutoLog', 'LogPath_%Y%m%d', 'CmdLog',
'Disabled', 1000)
```

8.4.1.12.6 GseosLog.ListAutoLogNames

ListAutoLogNames()

List the names of all AutoLogs.

Parameter	Description
-	

Returns

ctNames: A list of all automatic logs, ordered alphabetically.

8.4.1.12.7 GseosLog.GetAutoLogInfo

GetAutoLogInfo()

Get information about an AutoLog. The information is returned as an object of TAutoLogInfo which simply lists all the configuration attributes of the AutoLog.

Parameter	Description
strLogName:	The log name

Returns

oInfo: AutoLog info object.

8.4.1.12.8 GseosLog.AutoLogNew Session

AutoLogNewSession()

Manually roll over an automatic log. The rollover is determined by the session rollover settings. This allows you to programmatically start a new session.

Parameter	Description
-	

strLogName: The log name

Returns

None

8.4.1.12.9 GseosLog.AutoLogShow

AutoLogShow()

Show the AutoLog window in floating (dialog) format. The window can also be pinned to it's desktop tab as a regular window. Use the [GseosLog.AutoLogPin\(\)](#) function for that. To close an AutoLog window use the [GseosLog.AutoLogClose\(\)](#) function.

Parameter

strLogName:

Description

The log name

Returns

None

8.4.1.12.10 GseosLog.AutoLogPin

AutoLogPin()

Show the AutoLog window in pinned format (only if previously in floating (dialog) format, it doesn't open the window if it is not already shown). The window can also be shown in floating format using the [GseosLog.AutoLogShow\(\)](#) function. To close an AutoLog window use the [GseosLog.AutoLogClose\(\)](#) function.

Parameter

strLogName:

Description

The log name

Returns

None

8.4.1.12.11 GseosLog.AutoLogClose

AutoLogClose()

Close the AutoLog window. In order to move the AutoLog window to a different tab on the desktop you have to do the following:

- a) Close the Window. (`GseosLog.AutoLogClose()`).
- b) Show the window in floating format. ([GseosLog.AutoLogShow\(\)](#)).
- c) Go to the tab you want the log window pinned to.
- d) Pin the log window. ([GseosLog.AutoLogPin\(\)](#)).

Parameter	Description
strLogName:	The log name

Returns

None

8.4.1.13 GseosMonitor

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](#)^[246] 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](#)^[286] 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](#)^[288]
- [Counter Check](#)^[287]

8.4.1.13.1 GseosMonitor.TMonitor.bEnable**Property: bEnable**

Enable or disable the monitor.

Example

Disable the monitor from a script:

```
oMyMon.bEnable = 0
```

8.4.1.13.2 GseosMonitor.TMonitor

TMonitor(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 GseosMonitor
import Gseos

def MyMonitor1(oBlock):
    if oBlock.Temp1 > 200:
        Gseos.Message('Monitor1: Sensor temp out of range')

HK.Monitors.append(GseosMonitor.TMonitor('Sensor Temp Check', MyMonitor1))
```

8.4.1.13.3 GseosMonitor.TMonitor.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
```

8.4.1.13.4 GseosMonitor.TMonitor.dw Cnt

Property: dwCnt

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

8.4.1.13.5 Examples

8.4.1.13.5.1 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](#)¹⁵⁰ 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 GseosMonitor
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 of 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 = GseosMonitor.TMonitor('MonSeq', fMonSeq)

#
# Assign it to the TLM block.
#
TLM.Monitors.append(oMonSeq)

```

8.4.1.13.5.2 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 GseosMonitor
import Gseos

# 12V Monitor
#
def fMon12V(oBlock):
    "12V Monitor"

    if oBlock.Plus_12V_Mon > 15:
        Gseos.Message('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:

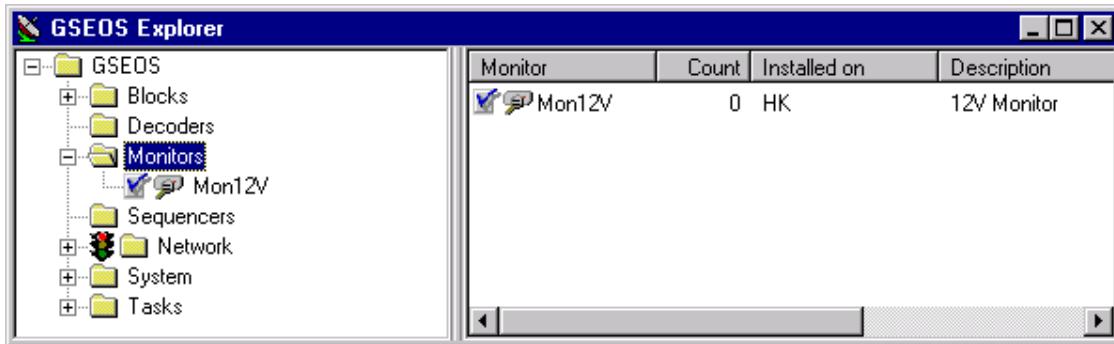
```
# Create a new monitor.
#
oMon12V = GseosMonitor.TMonitor('Mon12V', fMon12V)

# Assign it to the HK block.
#
HK.Monitors.append(oMon12V)
```

After we execute this script we can examine the monitor in the [GSEOS Explorer](#)^[127]. 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.

Note

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 [GseosConversion.signed\(\)](#) function e. g.:

```
#  
# -5V Monitor  
#  
def fMonNeg5V(oBlock):  
    """-5V Monitor"  
    Minus5V = GseosConversion.signed(oBlock.Minus_5V_Mon, 7)  
  
    if Minus5V < -7:  
        Gseos.Message('-5V Monitor: Limit exceeded, current voltage %d' %  
        Minus5V)
```

8.4.1.13.6 GseosMonitor.TMonitor.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.

8.4.1.14 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.

8.4.1.14.1 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.

Parameter Description

strClientName The name of the client connection to connect.

Returns

None.

8.4.1.14.2 GseosNet.ClientDisconnect

ClientDisconnect(strClientName)

Disconnect the client connection.

Parameter Description

strClientName The name of the client connection to disconnect.

Returns

None.

8.4.1.14.3 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.

8.4.1.14.4 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.

8.4.1.14.5 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.

8.4.1.14.6 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.

8.4.1.14.7 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.

8.4.1.14.8 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.

8.4.1.15 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\(\)](#)²⁹⁶. Use [StartRecording\(\)](#)²⁹⁸ and [StopRecording\(\)](#)²⁹⁸ 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\(\)](#)²⁹⁴, [RemoveRecordBlock\(\)](#)²⁹⁷, [AddPlaybackBlock\(\)](#)²⁹⁴, and [RemovePlaybackBlock\(\)](#)²⁹⁶. [GetRecordBlocks\(\)](#)²⁹⁶ and [GetPlaybackBlocks\(\)](#)²⁹⁵ 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\(\)](#)²⁹⁵, [SetPrefix\(\)](#)²⁹⁷, and [GetDataPath\(\)](#)²⁹⁵, and [SetDataPath\(\)](#)²⁹⁷. The SetPrefix() and SetDataPath() functions will also write the new value to the [gseos.ini](#)¹⁷¹ 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.

8.4.1.15.1 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\(\)](#)²⁹⁵. To remove blocks from the playback list use [RemovePlaybackBlock\(\)](#)²⁹⁶.

Parameters

strBlockName: Name of the block to put on the playback list.

Returns

None.

8.4.1.15.2 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\(\)](#)²⁹⁶. To remove blocks from the record list use [RemoveRecordBlock\(\)](#)²⁹⁷.

Parameters

strBlockName: Name of the block to put on the record list.

Returns

None.

8.4.1.15.3 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\(\)](#)²⁹⁷.

Returns

The current data path.

Example

```
>>> GseosRecorder.GetDataPath()
'c:\temp'
```

8.4.1.15.4 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\(\)](#)²⁹⁵. To add or remove blocks from the playback list you can use the Recorder interface or the functions [AddPlaybackBlock\(\)](#)²⁹⁴ and [RemovePlaybackBlock\(\)](#)²⁹⁶.

Returns

Tuple of the block names of blocks on the recording list.

Example

```
>>> GseosRecorder.GetPlaybackBlocks()
('CMDSTRING', 'RS232Raw', 'TLM', 'HK')
```

8.4.1.15.5 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\(\)](#)²⁹⁷ to modify the prefix.

Returns

The current file prefix.

Example

```
>>> GeosRecorder.GetPrefix()  
'EM_1'
```

8.4.1.15.6 GeosRecorder.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\(\)](#)²⁹⁶. To add or remove blocks from the record list you can use the Recorder interface or the functions [AddRecordBlock\(\)](#)²⁹⁴ and [RemoveRecordBlock\(\)](#)²⁹⁷.

Returns

Tuple of the block names of blocks on the recording list.

Example

```
>>> GeosRecorder.GetRecordBlocks()  
( 'RecComment', 'DSACtrI', 'RS232Raw', 'PeriodicMsg' )
```

8.4.1.15.7 GeosRecorder.isPlayingBack

IsPlayingBack()

Determine if the Recorder is in playback mode.

Returns

True if the Recorder is in playback mode, false otherwise.

8.4.1.15.8 GeosRecorder.isRecording

IsRecording()

Determine if the Recorder is in recording mode. In order to put the Recorder into recording mode use [StartRecording\(\)](#)²⁹⁸, to stop it use [StopRecording\(\)](#)²⁹⁸.

Returns

True if the Recorder is in recording mode, false otherwise.

8.4.1.15.9 GeosRecorder.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\(\)](#)²⁹⁵. To add blocks to the playback list use [AddPlaybackBlock\(\)](#)²⁹⁴.

Parameters

strBlockName: Name of the block to remove from the playback list.

Returns

None.

8.4.1.15.10 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\(\)](#)²⁹⁶. To add blocks to the record list use [AddRecordBlock\(\)](#)²⁹⁴.

Parameters

strBlockName: Name of the block to remove from the record list.

Returns

None.

8.4.1.15.11 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\(\)](#)²⁹⁵. 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\(\)](#)²⁹⁶, to stop the Recorder from recording use [StopRecording\(\)](#)²⁹⁸. The changes to the prefix are made permanent by writing them to the [gseos.ini](#)¹⁷¹ configuration file. On the next invocation of GSEOS you will have the prefix preserved.

Parameters

strDataPath: The new data path.

Returns

None.

8.4.1.15.12 GseosRecorder.SetPrefix

SetPrefix()

Set the file prefix to be used for multi file mode, see also [GetPrefix\(\)](#)²⁹⁵. 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](#)¹⁷¹ configuration file. On the next invocation of GSEOS you will have the prefix preserved.

Parameters

strPrefix: The new file prefix.

Returns

None.

8.4.1.15.13 GseosRecorder.StartPlayback

StartPlayback()

Switch the Recorder into playback mode. This makes the Recorder the active data source (switching all mutually exclusive data sources off) and starts 'fast forward' playback with the current recorder settings. To determine if the Recorder is in playback mode use [IsPlayingBack\(\)](#). To stop playback you can use [StopPlayback\(\)](#).

Returns

None

8.4.1.15.14 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

8.4.1.15.15 GseosRecorder.StopPlayback

StopPlayback()

Stop playback from the recorder and switch back to the previous data source. To determine if the Recorder is in playback mode use [IsPlayingBack\(\)](#), to start playback you can use [StartPlayback\(\)](#). Note that once we reach the end of the recorded data playback automatically stops.

Returns

None

8.4.1.15.16 GseosRecorder.StopRecording

StopRecording()

Stop recording. To determine if the Recorder is in recording mode use [IsRecording\(\)](#).

Returns

None

8.4.1.16 GseosSequencer

The GseosSequencer module allows you to run and control test sequences. Using the GseosSequencer 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 or log the success or 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](#)^[285] and Sequencer modules 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\(\)](#)^[309] 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 have the Wait() function 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 raises a TSeqTimeoutError exception. If you stop the Sequencer the Wait() function will raise a TSeqAbortError exception. For trivial condition functions (one liners) you can make use of the lambda function of Python.

Each sequencer is run in its own thread of execution and you have to be aware of threading issues when accessing shared resources. Especially if you plan to generate GSEOS blocks from a sequencer you have to make sure to use thread safe block access if any other thread might write to the same block. Please refer to the [GseosBlocks.TThreadSafeBlock](#)^[235] documentation for more details on how to access GSEOS blocks in a thread safe manner. There are also specific requirements for the Wait() condition function. Please see the note below.

Let's just start with a simple example to demonstrate the use of the Sequencer:

```
import GseosSequencer

# 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: oSeq.TriggerBlock().Temp == 20)

    # Stop the heater
    fStopHeater()
```

```
# Start the Sequencer
GseosSequencer.TSequencer('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 at the beginning or we don't get a block with HK.Temp equal to 20 with we will never exit the Wait().

Note

We use the TriggerBlock() function of the sequencer to access the HK block instead of writing: HK.Temp(). The reason is that when accessing the block HK directly we always see the last sample. Assume that a number of HK blocks are being generated and are sitting in the queue, oSeq.TriggerBlock() would access the sample that actually triggered the wait monitor, whereas HK would access the last sample. So to be correct you should use the TriggerBlock() function of the sequencer class instead of directly reading the item.

Threading Note

The Wait() function is implemented using a regular Monitor. The Monitor function registers for the blocks you are interested in (as indicated in the Wait() call) and gets called on the main thread when one of these blocks arrives. To synchronize with the Sequencer Wait() function that runs in its own thread we have to block the main thread waiting for the Wait() condition function to complete before we can release the Monitor running on the main thread. That means that your condition function should be very short and fast (same requirement as any Monitor or Decoder function) it also must not yield the processor or perform any blocking operations since this will cause a deadlock. Especially functions like: print, Gseos.Log, Gseos.Message, MessageBox, and anything that interacts with the GUI can not be used. Keep in mind that this is only required for the condition function of the Wait() statement. The sequencer function in general can use all these constructs.

Your sequencer function (fWarmUp() in the above sample) gets the sequencer object passed as the first parameter. Additional parameters can be added when the sequencer object is created and are passed through to your sequencer function.

The last line then finally starts the Sequencer. To start the Sequencer you have to provide a unique name, the Sequencer function, optional arguments as a tuple, 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 its 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 GseosSequencer

# 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:
oSeq.TriggerBlock().Temp[Heater] == Temp)

# Stop the heater
fStopHeater(wHeater)

# Fire up the Sequencer for heater 2 and 30C.
GseosSequencer.TSequencer('Heater Control 2', fWarmUp, (2, 30))

# Fire up the Sequencer for heater 3 and 25C.
GseosSequencer.TSequencer('Heater Control 3', fWarmUp, (3, 25))

# Fire up the Sequencer for heater 4 and 45C.
GseosSequencer.TSequencer('Heater Control 4', fWarmUp, (4, 45))

```

The above script expects two additional parameters besides the Sequencer reference that is passed in as the first parameter. In our case we pass the heater number and the destination temperature. When we invoke the Sequencer we have to provide the parameters as the third parameter of the TSequencer() constructor. 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](#)^[121]).

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:

```

# Create a Sequencer
oHeaterCtrl = GseosSequencer.TSequencer('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\(\)](#)^[309] to stop a running Sequencer, and [Delete\(\)](#)^[304] to delete it. Besides the Wait() function there are three other functions the Sequencer provides: [MessageBox\(\)](#)^[306], [InputDialog\(\)](#)^[305], and [Sleep\(\)](#)^[308]. 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: oSeq.TriggerBlock().Temp <=
wTemp)

    elif HK.Temp <= wLowTemp:
        fStartHeater()
        oSeq.Wait(HK, lambda wTemp=wHighTemp: oSeq.TriggerBlock().Temp >=
wTemp)

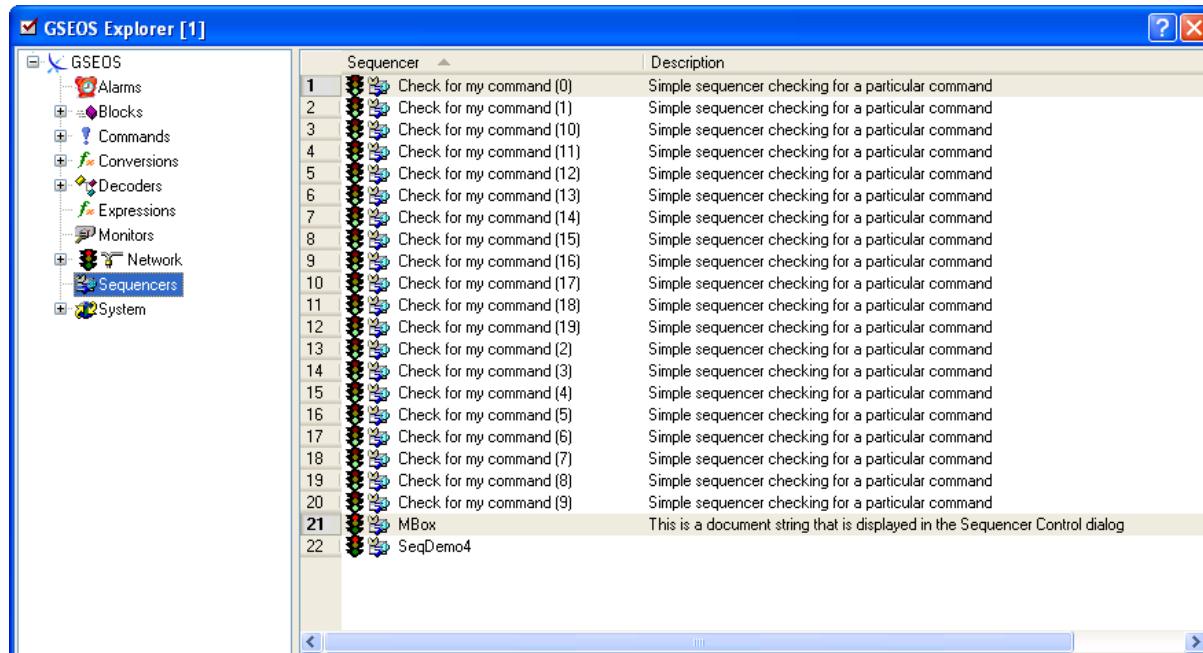
    # If we are in the right range just wait for a while
    else:
        oSeq.Sleep(300)

# Fire up the two-point controller.
GseosSequencer.TSequencer('Two-point controller', fTwoPointHeater, (20,
30))

```

Although this functionality could also be implemented with a Monitor it demonstrates some of the Sequencer functions.

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 function 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. 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.

The sequencer module implements the TSequencer class which exports the following methods:

[TSequencer](#)³⁰³
[Start](#)³⁰⁸
[Stop](#)³⁰⁹
[Delete](#)³⁰⁴
[Wait](#)³⁰⁹
[Sleep](#)³⁰⁸
[MessageBox](#)³⁰⁶
[InputDialog](#)³⁰⁵
[GetStatus](#)³⁰⁴

Most of the above functions also have a doc string which you can access with:
`GseosSequencer.function.__doc__`

8.4.1.16.1 GseosSequencer.TSequencer

TSequencer(strName, fSequencer, [oArgs=()], [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.
oArgs	Optional, any arguments you want to pass to your Sequencer function. This argument must be specified as a tuple, even if you only want to pass a single argument. Make sure you have matching parameters in your fSequencer function for every argument you list. The default is no arguments. To specify a single argument (a one item tuple) use: (MyArg,). E.g.: (22,)
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 Explorer 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 = GseosSequencer.TSequencer('MySeq', fMySeq)
```

8.4.1.16.2 TSequencer.Delete

Delete()

Delete a Sequencer.

Returns

None

Comments

The call to Delete() removes the reference held by the Explorer 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
```

8.4.1.16.3 TSequencer.GetStatus

GetStatus()

Get the status of a Sequencer.

Comments

The status is a string and can have three states: 'Running', 'Stopped', 'Deleted'.

Example

Query the state from the Sequencer from the above example:

```
if oMySeq.GetStatus() == 'Running':
```

```

print 'The sequencer is running'

elif oMySeq.GetStatus() == 'Stopped':
    print 'The sequencer is stopped'

else:
    print 'The sequencer is deleted'

```

8.4.1.16.4 TSequencerInputDialog

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 a TSeqAbortError exception is raised. The input dialog is displayed as a modal dialog box, that means you have to acknowledge the dialog before you can interact with GSEOS. If you want to use a modeless dialog that lets you interact with the rest of the system use [InputDialogModeless\(\)](#)³⁰⁵. If you want your sequencer to continue running even if the user presses Cancel you have to handle the TSeqAbortError exception.

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.InputDialog('Please set a new HV level in the range 1 .. 4!')
    oSeq.Wait([HK], fCheckHV)

```

8.4.1.16.5 TSequencerInputDialogModeless

InputDialogModeless(strText)

Same as [InputDialog\(\)](#)³⁰⁵ except that it displays the dialog box as modeless.

If you want your sequencer to continue running even if the user presses Cancel you have to handle the TSeqAbortError exception.

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):
    oSeqInputDialogModeless('Please set a new HV level in the range 1 .. 4!')
    oSeq.Wait([HK], fCheckHV)
```

8.4.1.16.6 TSequencer.MessageBox

MessageBox(strText, [wButtons], [wIcon=MB_ICONEXCLAMATION])

Displays a message box. The text strText is displayed as the message prompt. The standard windows buttons can be specified with the wButtons parameter. If the user chooses Cancel the function raises a TSeqAbortError exception.

The message box dialog is displayed as a modal dialog box, that means you have to acknowledge the dialog before you can interact with GSEOS. If you want to use a modeless dialog that lets you interact with the rest of the system use

[MessageBoxModeless\(\)](#)³⁰⁶.

If you want your sequencer to continue running even if the user presses Cancel you have to handle the TSeqAbortError exception.

Parameter**Description**

strText	The message prompt text..
wButtons	Optional, specifies the buttons to display. One of the following attributes can be specified: MB_OK, MB_OKCANCEL, MB_RETRYCANCEL, MB_YESNO, MB_YESNOCANCEL.
wIcon	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)
```

8.4.1.16.7 TSequencer.MessageBoxModeless

MessageBoxModeless(strText, [wButtons], [wIcon=MB_ICONEXCLAMATION])

Displays a modeless message box. This function behaves identical to [MessageBox\(\)](#)³⁰⁶, except that the dialog is shown in a modeless fashion which lets you interact with the rest

of GSEOS. The text strText is displayed as the message prompt. The standard windows buttons can be specified with the wButtons parameter. If the user chooses Cancel the function raises a TSeqAbortError exception.

If you want your sequencer to continue running even if the user presses Cancel you have to handle the TSeqAbortError exception.

Parameter	Description
strText	The message prompt text..
wButtons	Optional, specifies the buttons to display. One of the following attributes can be specified: MB_OK, MB_OKCANCEL, MB_RETRYCANCEL, MB_YESNO, MB_YESNOCANCEL.
wIcon	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.MessageBoxModeless('Please make sure the HV value is in the range 1
.. 4!', MB_OKCANCEL)
```

8.4.1.16.8 TSequencer.TriggerBlock

TriggerBlock()

Returns the current trigger block. This function should be used in the conditional function of the Wait statement. In order to access the block that triggered the evaluation of the Wait statement you must use TriggerBlock() and read any data items from the block returned by TriggerBlock(). The reason is that there could be multiple samples of the block queued up and reading the block directly with: BlockName.ItemName will read the last sample which is not necessarily the one that triggered the Wait statement.

Parameter	Description
-	

Returns

oBlock: The block that triggered the evaluation of the wait statement. Use this block to read your data items to ensure you don't skip over any blocks.

Example

```
def fMyCond(oSeq):
    HK = oSeq.TriggerBlock()

    return HK.Value == 5

def fMySeq(oSeq):
    oSeq.Wait([HK], fMyCond, (oSeq, ))
```

8.4.1.16.9 TSequencer.Sleep

Sleep(fITimeout)

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 a TSeqAbortError is raised.

Parameter

fITimeout Timeout value in seconds, can be floating point number.

Returns

None

Comments

The Sleep() function puts the Sequencer thread in an effective state where it consumes only very little processor time.

8.4.1.16.10 TSequencer.Start

Start()

Start (restart) a Sequencer.

Returns

None

Comments

If the Sequencer is still running or if it has been deleted a RuntimeError exception is thrown.

Example

Restart the Sequencer from the above example:

```
oMySeq.Start()
```

8.4.1.16.11 TSequencer.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 has been deleted a RuntimeError exception is thrown.

8.4.1.16.12 TSequencer.Wait

Wait(ctBlocks, fCondition, [oArgs=()], [fTimeout=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 a TSeqAbortError. If an optional timeout value is specified and the timer expires before the condition evaluates to true raise a TSeqTimeoutError.

Parameter

ctBlocks

Description

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.

fTimeout

Optional, timeout value in seconds. A TSeqTimeoutError 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: oSeq.TriggerBlock().byChan == 123, (),
20)

except GeosSequencer.TSeqTimeoutError, oX:
    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

```

8.4.2 3rd Party Packages

One of the changes over GSEOS 7 (where it was difficult to install 3rd party packages with extension modules without recompiling those modules), GSEOS 7.1 makes it very easy to install 3rd party packages. For one, numpy, one of the most common numerical packages that is required by many add-ons already comes pre-installed with GSEOS 7.1

For your convenience there is a GeosLib module that bundles a number of common 3rd party modules like pandas, lxml, openpyxl, and PIL.

You can install GeosLib by simply extracting the archive into the GSEOS root folder. The packages are installed in the Lib folder. This includes all the necessary code for both Windows and Linux.

GeosLib does specifically not include matplotlib.

We do not recommend to install matplotlib within GSEOS. Matplotlib runs its own message loop which conflicts with the GSEOS message loop, also there is no backend that can be run in GSEOS. Lastly matplotlib was not designed as a real-time plotting tool. Therefore it makes more sense to run matplotlib, if so desired, in its own process and to send data over a socket from GSEOS to the matplotlib process.

For more information please refer to the matplotlib documentation.

Besides the pre-packaged GeosLib you can also freely install your own custom packages. To do so you have to specify a folder that will take these packages. You can specify your package folder with an entry in the [\[PyStartup\]](#)^[19] section:

```
[PyStartup]
LibDir = UserLib
LibDir = OtherLib
```

You can add one or more directories. However, to avoid clutter we recommend to only set up one such directory. Your packages then should go into subfolders within this folder:

```
UserLib
- PackageA
- PackageB
```

Please make sure that the package is compiled with the correct unicode settings. The

GSEOS uses UCS2 on Windows and UCS4 on Ubuntu. In general we use the platform specific default. If you use a Linux distribution that uses UCS2 you will need to compile those modules with UCS2 (or grab them from Ubuntu).

The recommended way to get 3rd party packages is to use a tool like conda, or anconda and install the according packages in a virtual environment since that will gather all necessary prerequisites. Make sure to do this for Python 2.7.12 since that is the version GSEOS 7.1 uses. You can then simply copy the packages into your configured folder. GSEOS scans your LibDir folders on startup and will try to import the packages. If there is an error you will get an error message in the message window.

8.5 Recorder File Format

Note

The recorder file format has changed with version 5.2. We now use an enhanced format that allows indexing and compression during recording, therefore there is no simple access from an API level. However, the tool RecExport can dump all recorder files to a flat ASCII or binary file.

The format described below is for GSEOS versions < 5.2.

The [GSEOS Recorder](#)¹⁴⁵ 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 "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	

abyData	dwSize	Jan/01/70 00:00:00.
dwLen	4	data bytes
		wSize+20
		Block data
		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 ffblk ffblk ;

    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 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 opended.
\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") ;
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
           ----- */
    }

    if (fread(&(sBlkBody.wBlockID),sizeof(sBlkBody)-2,1,pFile) !=
1)
{
    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 ;

}
```

9 Cross Platform Deployment

GSEOS can be deployed on multiple platforms: Microsoft Windows (Windows 2000, XP, Vista), Linux (Fedora, Ubuntu, openSUSE), Mac OS X (11.4, Tiger). Most applications use the Windows platform, however there are several missions using Linux. While we don't recommend switching between Windows and Linux on a regular basis you can move your GSEOS configuration back and forth between Windows and Linux. This chapter outlines some things that might be helpful in the transition between different OSs or different distributions.

File System

Besides the GSEOS kernel a typical mission has custom extensions, these must be available for the target platform. Also, when writing end-user code you have to ensure to be platform portable. Avoid absolute file names like c:\CASSINI\MIMI or /home/Pluto/GSEOS. GSEOS can be x-copy deployed, that means you can copy the GSEOS folder to any location on your machine or to different machines without breaking the installation. However, that requires that you don't use any absolute paths (when moving between different machines). GSEOS treats any path that is within the GSEOS root folder (that is the folder where the gseos executable resides) as a relative path. So as long as you keep all your configuration files within the GSEOS root folder (or subfolders) you avoid absolute paths which might lead to portability problems.

Fonts

When creating screen files you must choose fonts that are available on all target platforms. If you select a font that is not available on a given system a replacement font is chosen which results in a different alignment. Even if you only plan to deploy on one system for performance reasons you might want to choose a fixed pitch font for larger arrays of numeric data displays. Fixed pitch fonts are significantly faster than variable pitch fonts. An example of a fixed pitch font is Courier New, Verdana and Times New

Roman are variable pitch fonts.

The most important parameter is the DPI scaling. GSEOS uses the default of 96 DPI. Most Windows systems are configured with this value. You can adjust this with the Display Properties dialog, choose Settings and click on Advanced. You should see a tab called General that allows you to configure the DPI setting. Make sure this is set to 96.

For Linux distributions 96 DPI is not always the standard. Please see below how to set the DPI on the individual Linux distributions:

Ubuntu 6.06

Go to System > Preferences > Font. Select Details... In the dialog box select 96 DPI for the resolution.

Fedora 10

The default setting for Fedora 10 is 96 DPI.

openSUSE 11.1

Go to Applications > Desktop Configuration. On the General Tab under the section Look&Feel choose Appearance. This opens the Appearance - System Settings dialog. Choose Fonts on the left hand. On the right hand set Force fonts DPI to 96.

Now that we have set the proper DPI settings we have to make sure the fonts we use are available on all target platforms. Microsoft has made available cross platform portable true type fonts. These can be installed on the various Linux platforms. When using the Microsoft true type fonts you should restrict yourself to the following fonts: Arial Black, Arial, Courier New, Georgia, Impact, Times New Roman, Trebuchet, Verdana, and Webdings. Webdings can be useful to display various symbols.

There are other cross platform fonts available, if you choose to go that route you have to make sure to install the fonts on all the platforms you plan on using.

See below for notes on how to install the Microsoft true type fonts on the various Linux distributions:

Ubuntu 6.06

Go to System > Administration > Synaptic Package Manager. In the package manager install the msstcorefonts package. To register the fonts you have to execute the following command: update-ms-fonts

Fedora 10

There is no readily available msstcorefonts package. You can search on the internet how to install the msstcorefonts in Fedora. There is also an alternative solution called liberation fonts that you can install through the package management console.

openSUSE 11.1

Install package: fetchmsttfonts.

10 How do I...

10.1 How do I display alarm information?

Setting up an alarm item is a three step approach:

- Create your [alarm definition](#) ^[149] and load an alarm file.
- [Place the data item](#) ^[51] on a screen.
- Select the alarm and [configure the alarm properties](#) ^[68].

Once you have the alarm loaded you will be able to verify it in the [GSEOS Explorer](#) ^[121].

The image below shows a screen shot of a data item that has an alarm set up.



10.2 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:

- a) Write message to the message window.
- b) Write message to a log file.
- c) Send command.
- d) Execute Python function.
- e) 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](#) ^[163].

10.3 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 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.)
AutoConn	Only for Client connections. Allows to automatically connect to a server.
ct	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 were 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
```

10.4 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](#)¹⁴⁸
- 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 [\[Config\]](#)¹⁷⁹ 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\ApiId_601 qlf
Load = i_LORRI\CMD_n_TLM\ApiId_601 alarm
Load = i_LORRI\CMD_n_TLM\ApiId_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 necessary to load Python scripts at startup time. There are several different approaches with different consequences:

1. Specify the scripts to load in the [gseos.ini](#)¹⁷¹ [\[Config\]](#)¹⁷⁹ 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 batch files can be used for delayed initialization at startup.

10.5 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\(\)](#)²⁰⁴ method of the [Gseos](#)²⁰⁰ 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](#)²⁶. 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\(\)](#)²¹⁴.

Once a screen is open you can change its appearance with [WindowMinimize\(\)](#)²¹⁸, [WindowMaximize\(\)](#)²¹⁸, or [WindowRestore\(\)](#)²²¹. You can also close it with [WindowClose\(\)](#)²¹⁸. All these functions are located in the Gseos 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() takes advantage of the fact that GSEOS raises a RuntimeError when it can't locate the window you try to restore. It then tries to load the file with [FileOpen\(\)](#)²⁰⁴.

Example

```

import Gseos

def fOpenScreen(strScreenFileName):
    try:
        Gseos.WindowRestore(strScreenFileName)

    except RuntimeError:
        Gseos.FileOpen(strScreenFileName)

```

10.6 How do I implement a file upload?

Many missions and instruments support a memory or file upload mechanism. This usually facilitates memory, table, or software uploads. There is a wide variety of input formats from simple binary files to record based (SFDU) files. On the output side each instrument has a different command or set of commands to accommodate a memory or file upload. The general mechanism of the file upload consists of multiple steps:

- 1) Select the upload file and specify any parameters (like start address) needed for the upload.
- 2) Issue any setup commands to prepare for the file load.
- 2) Loop over the file extracting chunks and issuing the proper memory upload command(s).
- 3) Issue any cleanup commands to finalize the file upload.

Due to the diverse requirements the file upload mechanism provides a basic framework that can be customized by on a per instrument basis.

The file type to be uploaded can be configured by file extension. Any parameters that need to be supplied can be configured also. When the user initiates the file upload he chooses the proper file type and a file selection dialog let's him choose the desired file. Then a dialog for the configuration parameters is presented. The configured FileUploadStartHandler() function is called and these parameters is passed along.

Then the FileUploadTimerHandler() function is called everytime a configured timer expires and the user code can read the next chunk of file data and format it into commands or BinCmd blocks as required.

The following section gives an overview of the various gseos.ini settings to configure the File Upload:

Configuration Settings

The file upload is configured in the [gseos.ini](#) configuration file. You can specify multiple file upload types. Each file upload type is configured with the [FileUploads] section. See the example below that specifies three different file types:

```

[FileUploads]
RAMLoad=FileUpload
EEPROMLoad=FileUpload
MemLoad=FileUpload

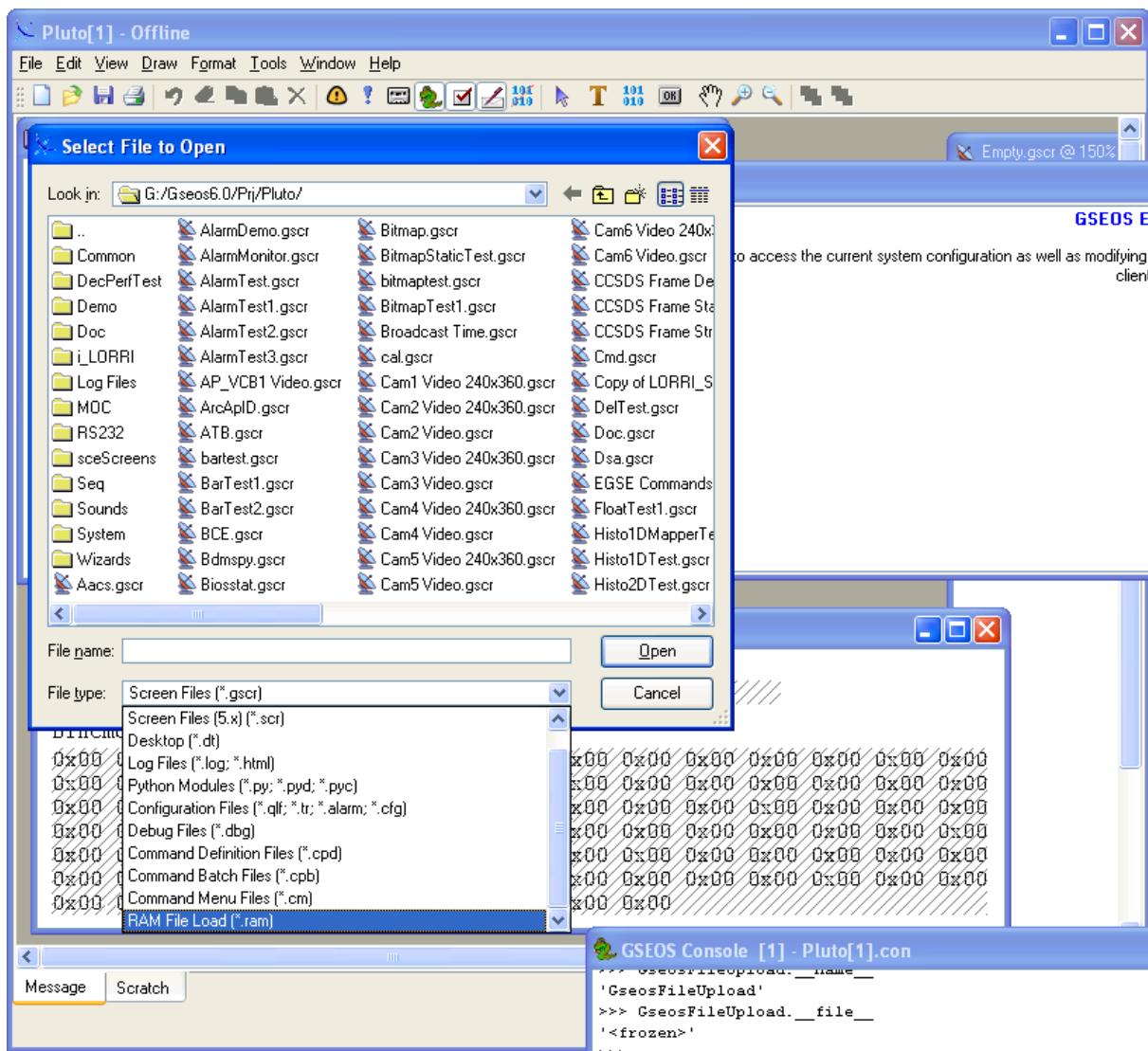
```

The name of each file upload type (RAMLoad, EEPROMLoad, MemLoad) is the name of a configuration section that further specifies the details of this file upload type:

```
[RAMLoad]
Description=RAM File Load
Extension=ram
Timer=100
FileUploadStartHandler=MyModule.fOnRAMLoadStart
FileUploadTimerHandler=MyModule.fOnRAMLoadProcess

Arg0 = StartAddr: 0x2000, 0x3FFF
Arg1 = Mode:      SWAP, BLANK, XOR
Arg2 = Verify:    ON, OFF
```

The above example configures the RAMLoad file upload. The description will show up in the GSEOS File Open dialog to describe the file upload type, the extension is used as the file filter. The user initiates a file load by simply selecting the desired file. The screenshot below shows the open File Select Dialog with the RAM File Load file type selected.



The Timer setting specifies the time out time of the upload timer. The FileUploadStartHandler and FileUploadTimerHandler settings specify your callback handler routines where you will fill in your code to process the file and issue the commands. The ArgX settings configure any additional arguments you want the user to specify when loading the file like start address etc..

The following paragraph describes the configuration settings in detail:

Key	Assignment
Description	This should be a brief description of the file upload type. This will show up in the File/Open dialog File Type setting.
Extension	The file extension, this should be a three character extension that doesn't conflict with any of the other GSEOS common extensions. The '.' (dot) must not be included in the file extension.
Timer	The time out interval in ms. The FileUploadTimerHandler routine (see below) is called whenever this timer expires.
FileUploadStartHand	This entry must specify your callback function that is called when the

ler	<p>user starts a file upload. This function must take two arguments, the first one is the absolute file name of the file the user has selected, the second is a list of arguments the user has selected. Even if there is only one argument it will be delivered as a list with one argument. If the user specifies arguments from a select list they will be provided as strings, numeric arguments will be checked against the proper range and be forwarded as integer.</p> <p>You can specify a module and package name in typical Python namespace syntax. You have to make sure that this name is accessible (i.e. the module is loaded).</p> <p>If this function returns False the file upload is aborted, if True the upload timer is started and the FileUploadTimerHandler function called. If it returns a string the string is interpreted as an error string and the argument query dialog is popped up again displaying the error and let the user correct the argument entry.</p>
FileUploadTimerHan dler	<p>This entry sets the timer callback. In this function you should service the file, read the next chunk of data and prepare and issue the command(s) as required. This function doesn't take any parameters, it is your responsibility to keep references to the open file, and any other data you need to process the file. Your function should return True if there is more data to be processed, it should return False when you are done with the file. Once this function returns False the file upload is finished and your routine won't be called until the user uploads another file.</p>
Arg0...Arg4	<p>You can configure up to five arguments the user can specify when choosing a file to upload. Each Argument takes the prompt followed by a colon and either a select list or a range. A select list is a comma delimited list. A range is indicated by two ellipses (..m). If neither a list nor a range is specified the argument will simply prompt for any string. If given a range the user input will be verified against the range (both ends are inclusive), if a select list is provided the user can choose one of the settings specified. If a range is specified only decimal integer values can be entered by the user. If you require hex values you can simply use a string argument and interpret the value in your handler. The user entries will be passed as a list as the second argument to the FileUploadStartHandler callback function you supplied.</p>

Simple File Upload Sample

In the following example we are going to set up a simple binary file upload. We make the following assumptions:

- a) The upload file is a simple binary file, that is we can take chunks of the file and upload the data as is.
- b) Each file upload command that we issue as part of the file upload is self-contained and has the upload opcode (one byte) and the 16-bit start address for the data following. The next parameter is the memory type: FLASH, RAM, EEPROM, the last parameter is the number of bytes to upload. The data portion can be up to 256 bytes.
- c) The start address must be in the range 0x4000..0x7FFF. We want the user to be able to enter a hexadecimal start address.

First we have to start with the gseos.ini configuration of the file upload we want to set

up:

```
[FileUploads]
BinLoad=FileUpload

[BinLoad]
Description=Binary File Load
Extension=bin
Timer=250
FileUploadStartHandler=BinFileLoad.fOnStart
FileUploadTimerHandler=BinFileLoad.fOnTimer

Arg0 = StartAddr:
Arg1 = Memory: FLASH, RAM, EEPROM
```

We will put all the file upload logic into the BinFileLoad module (which we will discuss below). So the Start and Timer handlers get assigned the according functions. The BinFileLoad module needs to be accessible so we will load it from the [PyStartup] gseos.ini section:

```
[PyStartup]
Import = BinFileLoad
```

We could also use a batch file to load the BinFileLoad module. However, if we use the [Config]/Load section to load the module it will not be 'imported' into the `__main__` namespace and would therefore not be found.

Since we want the user to be able to enter hex start addresses we simply use a string type argument for the start address (not the range since that only allows decimal integers). We will parse the string ourselves in the Start handler. The Memory type can be specified using a select list, we will convert the selected string into the proper code in the Start handler as well.

After having configured the file upload we now have to write our actual command handling. The BinFileLoad module pretty much provides the two handlers for the file upload start and timer notifications. Besides these two routines we will keep some global state:

```
#
*****
# * BinFileUpload.py
# *
# *
# * Simple binary file upload sample.
# *
#
*****
#
-----
```

```
#  
# - Imports  
- #  
#-----  
#  
import struct  
import GseosBlocks  
  
#-----  
# = Defines.  
= #  
#-----  
#  
OPCODE_UPLOAD = 0x5F  
  
#-----  
# = Data  
= #  
#-----  
#  
oFile      = None  
wCurrAddr  = 0  
wMemType   = 0  
BinCmd     = GseosBlocks.Blocks['BinCmd']  
  
#*****  
# * fOnStart()  
* #  
# *  
* #  
# * File upload start handler, get's called when the user opens a file of  
* #  
# * type: *.bin.  
* #  
# * We open the file and keep a reference to the open file for further  
* #  
# * processing. We also check the user provided arguments and convert them  
* #  
# * accordingly.  
* #  
# * We don't have to check for exceptions since they will be handled by the  
* #  
# * file upload mechanism and abort the file upload.
```

```

* #
# *
* #
# * Parameters: strFile: The file name of the file upload.
* #
# * ctArgs: A list with user selected arguments.
* #
# * According to our configuration the first item is
* #
# * the start address (as a string) and the second
* #
# * is the memory type.
* #
# * Returns: oRet: True, if everything is ok and we need to continue
* #
# * the file load.
* #
# * False, if the file load can be completed here
* #
# * (that is the entire file is less than 256 bytes).
* #
# * An error string if there is a problem with the
* #
# * arguments and we want to prompt the user to
* #
# * correct his selection.
* #
#
*****#
#
def fOnStart(strFile, ctArgs):
    global oFile
    global wCurrAddr
    global wMemType

    #
----- #
    # - Check the arguments.
    - #
    #
----- #
    wCurrAddr = eval(ctArgs[0])

    if not 0x4000 <= wCurrAddr <= 0x7FFF:
        return 'The start address must be in range: 0x4000..0x7FFF'

    strMemType = ctArgs[1].strip()
    wMemType = {'FLASH': 0, 'RAM': 1, 'EEPROM': 2}[strMemType]

    #
----- #
    # - Open the file and read the first chunk of data.

```

```
- #
#
----- #
oFile = file(strFile, 'rb')

strData = oFile.read(256)
wLen     = len(strData)

#
----- #
# - Generate the command data. We pack the data using big endian.
- #
#
----- #
strCmdData      = struct.pack('>BHHH%ds' % wLen,
                               OPCODE_UPLOAD,
                               wCurrAddr,
                               wMemType,
                               wLen,
                               strData)
BinCmd.Len      = len(strCmdData)
BinCmd.Channel   = 0
BinCmd.CmdData[:] = strCmdData
BinCmd.SendBlock()

#
----- #
# - Update our current upload address.
- #
#
----- #
wCurrAddr += wLen

#
----- #
# - We are at the end of the file, terminate the file upload.
- #
#
----- #
if wLen < 256:
    oFile.close()
    return False

#
----- #
# - Continue file upload with next timer expiration.
- #
#
----- #
return True
```

```

#
*****
#
# * fOnTimer()
* #
#
# *
# *
# * We continue uploading data in chunks of 256 bytes. If we run out of
* #
# * data we terminate the upload.
* #
#
# *
# *
# * Parameters: -
* #
# * Returns:      bRet:      True, if everything is ok and we need to continue
* #
# *                  the file load.
* #
# *
# *                  False, if we are done with the upload.
* #
#
*****#
#
def fOnTimer():
    global oFile
    global wCurrAddr

    #
----- #
    # - Read the next chunk of data.
    - #
    #
----- #
    strData = oFile.read(256)
    wLen      = len(strData)

    #
----- #
    # - Generate the command data. We pack the data using big endian.
    - #
    #
----- #
    strCmdData      = struct.pack('>BHHH%ds' % wLen,
                                    OPCODE_UPLOAD,
                                    wCurrAddr,
                                    wMemType,
                                    wLen,
                                    strData)
    BinCmd.Len      = len(strCmdData)
    BinCmd.Channel   = 0
    BinCmd.CmdData[:] = strCmdData
    BinCmd.SendBlock()

```

```
#-----#
# - Update our current upload address.
- #
#
#-----#
wCurrAddr += wLen

#
#-----#
# - We are at the end of the file, terminate the file upload.
- #
#
#-----#
if wLen < 256:
    oFile.close()
    return False

#
#-----#
# - Continue file upload with next timer expiration.
- #
#
#-----#
return True
```

You have to save the module as BinFileUpload.py in a directory that is on the Python search path.

The code is generously commented. For simplicity we duplicated much of the code in the fOnStart() and fOnTimer() routines, the command generation could be split off it its own routine. Any exceptions in the script will be handled by terminating the upload and flagging the error in the GSEOS message window.

We first check the arguments provided by the user, since we want to allow hex arguments we have to convert the string ourselves. If we had used a range type argument the limit check would have been handled automatically but this limits us to decimal values only. After we have checked our settings we open the file and read the maximum amount of data allowed. If we are at the end of the file we are done with the file upload and return False, otherwise we return True and the upload will continue with the timeout value as specified in the configuration. The fOnTimer() routine then handles clocking out the remaining data. We have to keep the current address as a global since we have to feed this into every file upload command and increment accordingly.

10.7 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](#)^[168]. 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](#)^[57] 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](#)^[51].

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 GseosConversion  
GseosConversion.Calibrate(3, Rates.Mass[0], 0.9899)
```

Index

- [-

[Logs] 183

- 1 -

1D Histogram 272
1s event 201, 214

- 2 -

2D Histogram 275

- A -

active desktop page 205, 214
Add Autolog 282
Add event handler 201
Alarm 68, 149, 200
Alarm Limits 149
Alarm Window 142, 200
Alarms 142
Application 25
Architecture 14
Architecture Overview 14
ASCII 69
AutoLog new session 283
AutoLog Pin Window 284

- B -

batch 241
bin command 240
binary command 240
Binary Streamer 15
Binary Writer 15
Bitmap 71
Block 150
Block Count Stamp 228
Block Definition 150
Blocks 223

- C -

Casting 79, 196
charts 21
Close AutoLog Window 284
Close Window 218
Color 61
Command 133
Command batch 154
command doc 237, 239, 240
command mnemonics 237
command opcode 237
command range check 237
Command Window 133
commanding 332
Config 179
config file 171
configuration file 171
Console Window 135
ConsoleWrite 201
conversion 242, 243, 244, 245
Conversion Functions 168
convert 242
Count Stamp 228
cpb 154
Cross Platform 324

- D -

Data Blocks 223
Data Bus 15
Data Export 34, 138
Data Export Tool 34
Data Flow 15
Data Item 65
Data Items 223
Data Source 235
DataExport 138
Decoder 246
Decoder Explorer 126
Description 85
desktop page 205, 214
Desktop Print 213
Display format 43
double conversion 243, 244
double to long long 243

Draw menu 48
dtoll 244

- E -

EnableDataSource 235
EnableRangeCheck 237
End-to-end testing 299
event handler 201, 214
excute command 238
ExecCmd 238
Exit 202
Export 138
Expression 168

- F -

Features 2
fGetEntry 259
File Append 202
File Menu 202
File New 203
File Open 204
File Open Dialog 204
File Save 204
File Save Dialog 204
File Upload 127, 180, 262, 332
FileMenu 202
FileOpenDialog 204
FileSaveDialog 204
Float 74
float conversion 243
float to long 244
Font 60
Format 43, 59
Format Data Item 65
Format Description 85
Format Font 60
Format Orientation 63
Format Range 62
Format Window 85
Formula 168

- G -

Generate Block 231
Get active desktop page 205

Get AutoLog information 283
Get Window Position 206
Get Window Size 207
GetCmdArgDoc 239
GetCmdArgName 239
GetCmdDoc 240
GetCmdOpcode 237
GetInstance 206
GetProjectPath 205, 206
GetSystemLoad 206
GetWindowPos 206
GetWindowSize 207
Graph 182
GSEOS conversion 242
Gseos Main Window 25
GSEOS options 147
gseos.ini 171, 205
gseos.ini file name 205
GseosBlocks 223
GseosDecoder 246
GseosGraph 182
GseosGraph settings 182
GseosHistogram 264
GseosIni 259, 260, 261, 262
GseosSequencer 299

- H -

Help 207
Help Image 85
Help Text 85
Histogram 264, 272, 275

- I -

idle 208
idle processing 208
Input Dialog 208
InputDialog 208
Instance 206
IsIdle 208
IsRangeCheckEnabled 240
Item access 232
Item Description 85
Item Help Settings 85

- L -

Line 62
 Linux 324
 List AutoLog Names 283
 list status text map items 209
 List Status Text Maps 208
 ListCmdMnemonics 237
 ListStatusTextMapItems 209
 lltod 244
 Load startup files 179
 locking 235
 Log 120, 209, 210, 280
 Log File 210
 Log Print 281
 Log Reload 281
 Log Save 210
 Log windows 120
 Logging 120, 183, 209, 280
 LogReload 210
 LogSave 210
 long long to double 244
 long to float 245
 lookup status text 208, 211
 LookupStatusText 208, 211

- M -

Mac 324
 Main Window 25
 MakePathNormal 212
 MakePathRelative 211
 Maximize Window 218
 Memory Load 180, 262, 332
 Memory Upload 127
 Menu Draw 48
 Message 120, 212
 Message Box 213
 message processing 214
 Message Window 140, 212
 MessageBox 213
 Modules 199
 Monitor 285
 Move Window 219

- N -

New File 203
 normalize path 212

- O -

Open File 204
 Options 147
 Options... 32
 Orientation 63

- P -

Pin AutoLog Window 284
 play sound 221
 PlaySound 221
 Plot 182
 Plotting 21
 Print 201, 281
 Print Desktop 213
 Print Window 219
 PrintDesktop 213
 Project path 205, 206
 Publish 15
 PumpWaitingMessages 214
 Python 199

- Q -

QLook Format 59

- R -

Range 62
 range check 237, 240
 Read Block 228
 Read block item 229
 Read entire Block 228
 Recorder 145
 Recording 145
 Red Alarm 149
 relative path 211, 212
 Reload Log 210, 281
 Remove event handler 214

Resize Window 220
 Restore Window 221
 RGB 77
 RGB Bitmap 77

- S -

Scalar Item 228
 Screen 38
 send 231
 send bin command 240
 Send block 231
 SendBinCmd 240
 SendBlock 231
 Sequencer 299
 Sequencer Status 304
 sequencer trigger block 307
 SetSplashScreenText 215
 SetSplashScreenVersion 216
 SetStatusBarText 215
 ShellExecute 216
 Show AutoLog Window 284
 shutdown 214
 shutdown event 201, 214
 Sound 221
 spawn 216
 splash screen 215, 216
 splash screen text 215
 splash screen version 216
 Start Application 216
 start batch 241
 StartApplication 216
 StartBatch 241
 Startup files 179
 status bar 215
 status bar text 215
 Status Image 78, 196
 status text 79, 196, 208, 209, 211
 status text items 209
 Stripchart 81
 stripcharts 21
 Style 59
 Subscriber 15
 system load 206

- T -

Terminate 202
 text reference 79, 196, 208, 209, 211
 thread 235
 threading 235
 trigger block 307
 TriggerBlock 307

- U -

Update AutoLog settings 282

- V -

Version 217

- W -

Wait 217
 Wait Dialog 217
 WaitDialog 217
 Window 85
 Window Close 218
 Window Maximize 218
 Window Move 219
 Window Position 206
 Window Print 219
 Window Resize 220
 Window Restore 221
 Window Size 207
 WindowClose 218
 WindowMaximize 218
 WindowMove 219
 WindowPrint 219
 WindowResize 220
 WindowRestore 221
 Windows 324
 Write 201
 Write item 232

- Y -

y(t) 81
 Yellow Alarm 149

Back Cover