The TSP Design & Programming Guide

for TSP v0.8.0

Document Revision 1.0

provided by
The TSP Team Worldwide

Table of Content

1 Introduction

1.1 Purpose of this guide

This guide is a TSP design primer which explain the fundamentals of TSP design, not the details of the specifications. It may be used by TSP developers and users in order to understand

- ✔ What is TSP design and purpose,
- ✔ How use and integrate the TSP in a development environment using different programming languages (C, Java, Perl, Python, ...),
- ✔ An overview of TSP ready to use applications

As a matter of fact TSP is an active and evolving project so that the most up to date informations may be found at the source (always "*use the source, Luke*" [1]).

The two primary source of informations are:

1. The source code itself which may be browsed or downloaded at Savannah: https://savannah.nongnu.org/projects/tsp

   CVS Browse: http://cvs.savannah.nongnu.org/viewcvs/?root=tsp

   Download Area: http://download.savannah.nongnu.org/releases/tsp/
2. The TSP mailing lists, including the archives

   https://savannah.nongnu.org/mail/?group=tsp

1.2 Reader's guide

The guide may be read linearly. Nevertheless, the guide fall in 3 parts which may be read separately.

- **Part 1** describes ***the TSP Design and Principle*** (Chapter 2), without reference to any

1 http://catb.org/~esr/jargon/html/U/UTSL.html

programming language.

- **Part 2** describes ***the way to program with TSP***. Chapters 4 to 9 explain how to use TSP in different programming language. Chapter 3 explains how to get and install the source code from Internet repository. Chapter 10 gives pointers to TSP documentation.
- **Part 3** describes ***the use of TSP application***. Chapter 11 lists the currently available TSP applications and their usage.

TSP observer should read Part 1.

TSP user should read Part 1 then Part 3.

TSP programmer should read Part 1 then Part 2 and Part 3.

1.3 Glossary

You will find hereafter a list of terms used throughout this document with a small definition. We advise the interested reader to go to Wikipedia (http://www.wikipedia.org) for finding more detailed definition.

Name	Definition
API	**A**pplication **P**rogramming **I**nterface (http://en.wikipedia.org/wiki/Application_programming_interface)
Blackboard	The TSP Blackboard is a structured shared memory space. It realizes a local publish/subscribe Idiom.
Doxygen	A documentation generator tool used by TSP in C: http://www.doxygen.org. See http://en.wikipedia.org/wiki/Documentation_generator for general information about documentation generator.
ONC-RPC	The **O**pen **N**etwork **C**omputing **R**emote **P**rocedure **C**all (http://en.wikipedia.org/wiki/ONC_RPC) which defined in RFC 1831 (http://www.ietf.org/rfc/rfc1831.txt) is a remote procedure call originally design by SUN for their NFS (http://en.wikipedia.org/wiki/Network_File_System) network file system.
QoS	**Q**uality **o**f **S**ervice.
TCP/IP	**T**ransport **C**ontrol **P**rotocol over **I**nternet **P**rotocol, a wide spread network protocol (http://en.wikipedia.org/wiki/TCPIP)
TSP	**T**ransport **S**ample **P**rotocol

Name	Definition
XDR	e**X**ternal **D**ata **R**epresentation (http://en.wikipedia.org/wiki/External_Data_Representation).
XML-RPC	A simple XML Remote Procedure Call (http://en.wikipedia.org/wiki/XML-RPC). This is a remote procedure call protocol whose encoding scheme is based on XML and transported by HTTP.

2 TSP Design

This chapter describes the overall TSP Design and Principle.

2.1 TSP History

TSP started in 2002 from the collaboration of BT C&SI (formerly Syntegra) and EADS-Astrium (formerly Astrium) in the domain of satellite avionics validation testbeds. TSP has now been used in different fields and is thus not bound to the space domain.

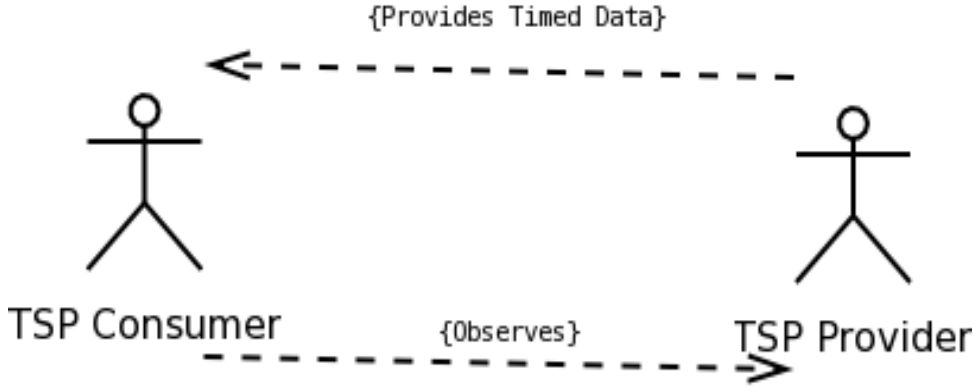
2.2 TSP Objectives

TSP stands for **T**ransport **S**ample **P**rotocol. TSP main goal is to provide an efficient mean to observe evolving data. The evolving data may be variable coming out of a simulation process or physical value taken from a numerical bus or physical captor (temperature, speed, etc...).

2.3 Provider/Consumer Principle

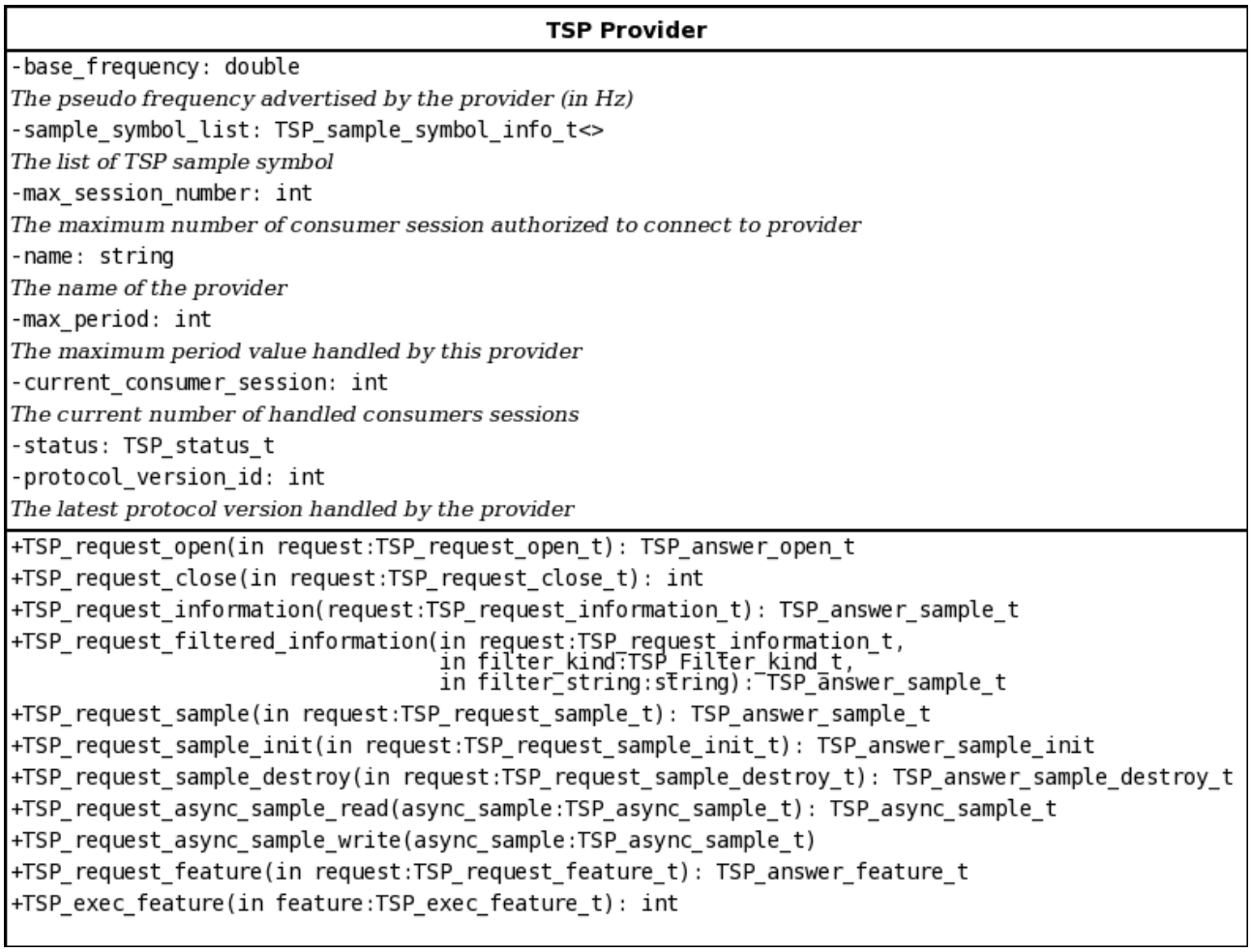
In TSP there is two main roles

- the ***provider*** which produces observable data and,
- the ***consumer*** which observes (draws, plots, views, writes to file, ...) the data produced by the provider



A TSP Provider may be considered as a cyclical or pseudo-cyclical process that produces

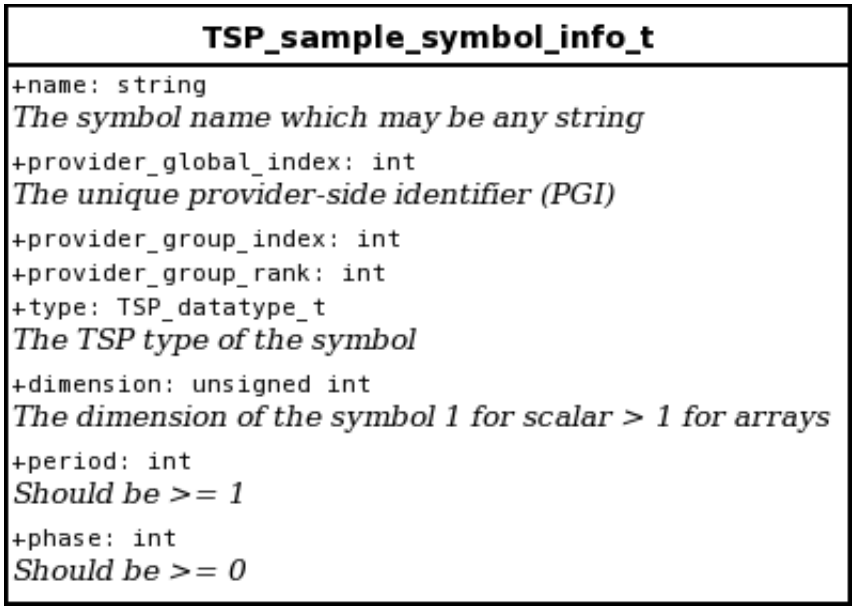
timestamped data. How the provider produces the data is out of the scope of TSP. The TSP only indicates how the provider is offering the produced data to TSP consumer. A TSP Provider has several attributes or properties as described in the following figure:



2.3.1 Cyclical nature of a provider

The aspect to understand about the TSP provider is its “pseudo-cyclical” nature. A provider should advertise its *base_frequency*, in Hz, this theoretically means that this provider will be able to produce a [set of] sample symbol value [s] at this specified rate. Nevertheless, as you will see later a TSP consumer should **NEVER** consider that this *base_frequency* is a real-time one. 1Hz may not be 1 second of wall clock time, it may be more, it may be less or even totally unrelated (mathematically speaking) to real time.

In fact a TSP consumer may ask a TSP provider for TSP sample symbols which are described by `TSP_sample_symbol_info_t`:



The sample `TSP_sample_symbol_info_t` structure entirely defines what the consumer will receive when asking for the symbol: `type` (floating point value, signed or unsigned integer, characters...) `dimension` (for arrays).

The `period` parameter specifies whether the consumer wants to receive ALL symbol values generated by the provider (`period=1`) or a subset of them (`period>1`). For example if `period = 4` the consumer will receive 1 value out of 4 generated by the provider. More precisely if the provider base frequency is 32Hz, the consumer will receive data at 8Hz = 32Hz / 4.

The phase argument specifies the offset in the cycle count of the provider.

The period and phase usage is better explained with an example. Let's say the provider has a 4Hz base frequency, with a symbol called '`time`' whose value is evolving as follow:

`time` values on provider						
Phase 0	***Phase 1***	***Phase 2***	***Phase 3***	***Phase 0***	***Phase 1***	***Phase 2***
0.000	0.250	0.500	0.750	1.000	1.250	1.500

Here are the values obtained by the consumer when asking for '`time`' symbol.

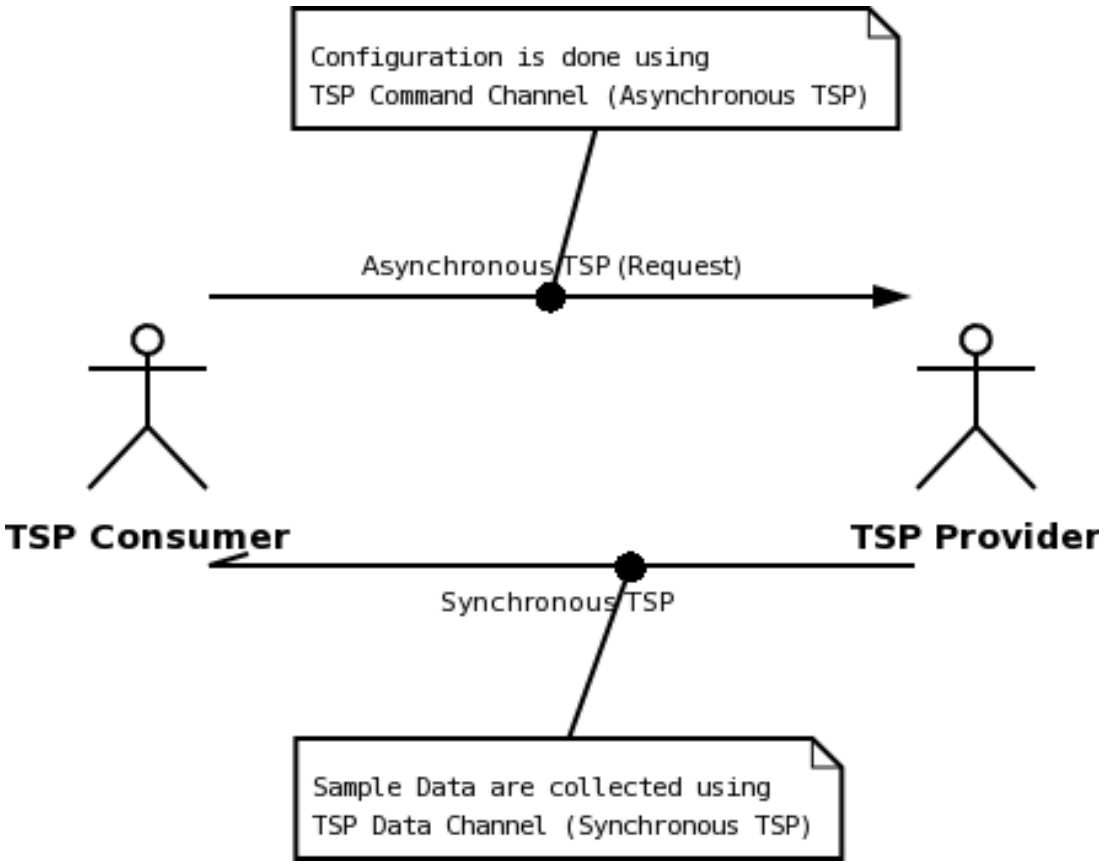
Period = 1, Phase = 0						
0.000	0.250	0.500	0.750	1.000	1.250	1.500

Period = 2, Phase = 0			
0.000	0.500	1.000	1.500

Period = 2, Phase = 1			
0.250	0.750	1.250	1.750

As you will see soon the consumer asks for a symbol using its name, period and phase, the provider will complete the other parameters when answering to the sample request.

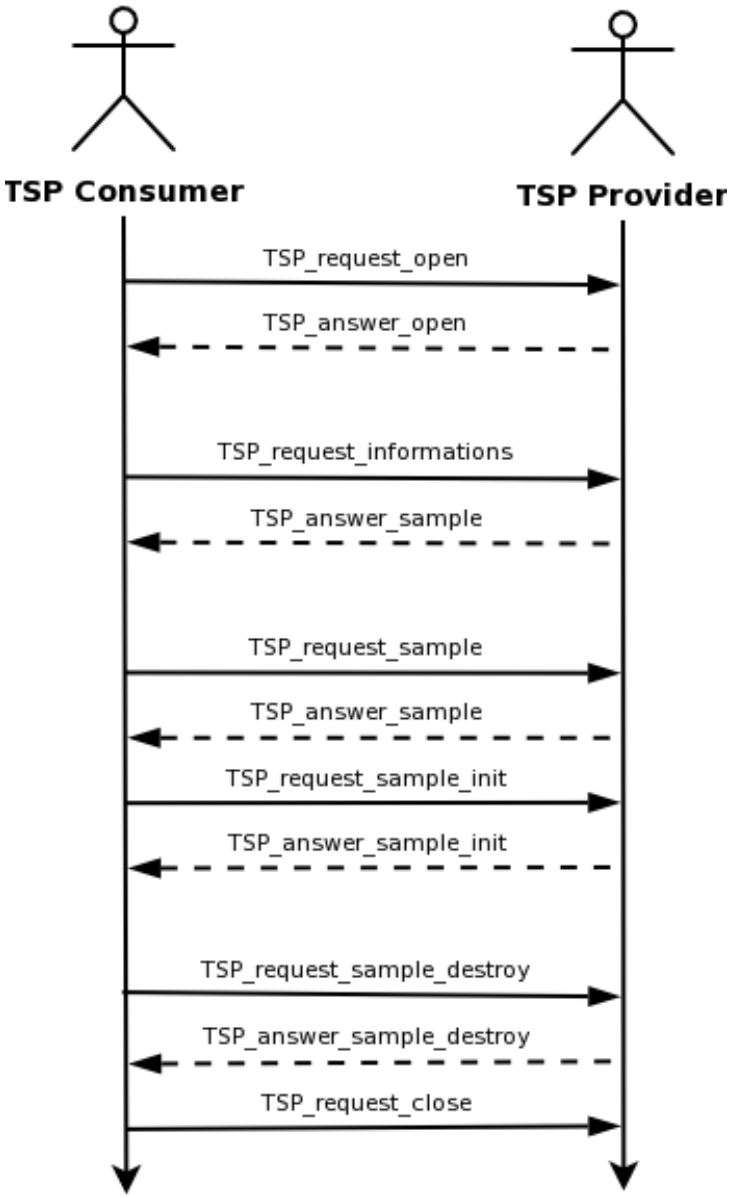
2.3.2 Consumer/Provider collaboration mean



The TSP provider and consumer collaborate using two communication means

1. the TSP Command Channel is the so-called "***asynchronous***" communication mean between TSP providers and consumers. Asynchronous TSP is used for:
    a) Getting informations about the provider,
    b) Opening a new TSP session, asking for sample configuration,
    c) Reading/writing one value etc...
2. the TSP Data Channel is the so-called "***synchronous***" communication mean between TSP providers and consumers. Synchronous TSP is used to sample data which are "sampled" on provider-side at a specified pace.

The typical TSP sequence call is described hereafter:



2.4 TSP Command Channel – Asynchronous TSP

TSP consumers and providers interact using the TSP Command Channel (Asynchronous TSP). This is the mean used to send configuration request describing what data is to be observed. The TSP Command Channel is a "***logically unconnected***" communication mean, which may be realized by different transport protocol (TCP/IP, XML-RPC, SOAP, CORBA, ONC-RPC, etc...). The default protocol used in the current TSP implementation is ONC-RPC (RFC1831, RFC1832).

2.4.1 TSP URL and Request Handler

A TSP consumer may contact a TSP provider using a TSP URL which has the following syntax:

request_handler_protocol*://*host*/*provider name*:*instance

For a Stubbed Server on tsp_demo the TSP URL would be:

```
rpc://tsp_demo/StubbedServer:0
```

For a BlackBoard provider running on bb_simu pseudo simulator it would be;

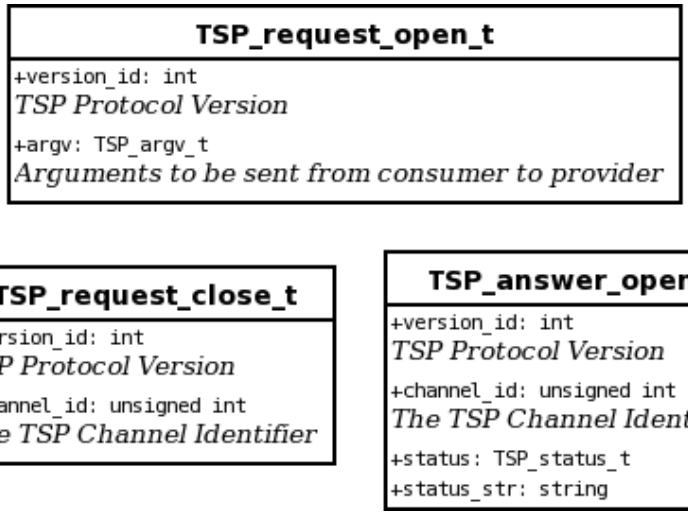
```
rpc://tsp_demo/bb_simu:0
```

Most of the consumer accept abbreviated TSP URL and try to find missing part, for example giving `rpc://tsp_demo` will try to connect to first provider on tsp_demo. The RPC URL scheme was designed with the idea that the Command Channel may be transported using different ***request handler protocols***. The first implementation is using ONC-RPC. There is an alpha XML-RPC (http://www.xmlrpc.com/) implementation in TSP C library.

2.4.2 Request Open/Close

When a consumer wants to negotiate a TSP Session with a TSP provider it has to send a TSP Request Open to the provider. The Provider answers with a TSP Answer Open which specifies success or failure. On success the consumer obtains a session identifier which is called ***channel identifier*** in TSP. The channel ID is used in each subsequent TSP request in order to identifies the TSP Session on provider side. Using this channel ID the provider maintains a set of provider-side consumer configuration state:

- Which TSP symbols where asked for this TSP session?
- Is the consumer currently receiving sample?
- ...



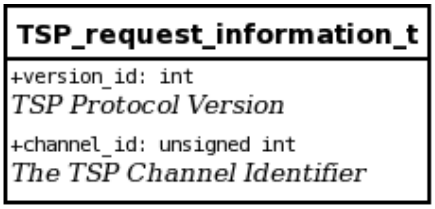
When the consumer wants to terminate its TSP session it sends the TSP Request Close and the provider frees any provider-side data related to this session. The TSP provider may garbage collect the TSP session if 'broken link' is detected during sample.

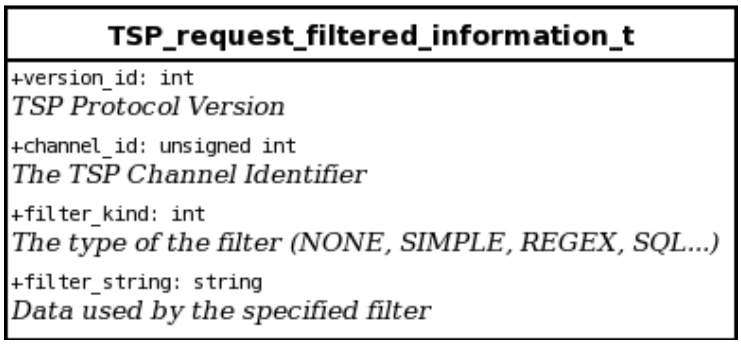
A TSP Request Open may fail if the concerned provider does not want to accept more session. The current C implementation limits the number of sessions to 100. One may customize this limit for its own purpose or set-up some other kind of quality of service (QoS).

The TSP Request Open is a MANDATORY request, it MUST be sent to provider before sending any other TSP Request.

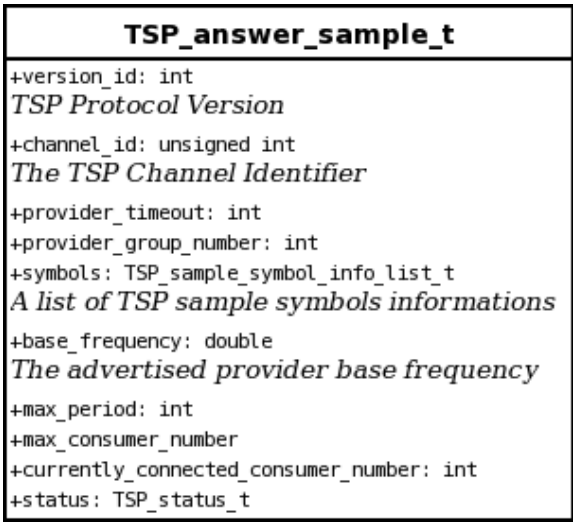
2.4.3 Request [Filterered] Informations

After opening a TSP Channel a consumer may (optionally) ask the TSP provider for informations. This is done with the TSP Request Informations or TSP Request Filtered Informations.





The TSP provider will answer with a TSP Answer Sample containing the complete list of available symbols or a filtered list if Request Filtered Informations was used.



Filtered Request is very useful for provider which have a huge number of available symbols. This specific request was introduced of a “real-life” bb_tsp_provider offering more than 1 000 000 of symbols. *Note here that offering a huge number of symbols is easy as long as consumer* ***do not ask to effectively sample all of them***.

The TSP Request Filtered Information has more arguments than TSP Request Information:

- `filter_kind`: for specifying the kind of filter you want to use. There are 2 kinds of filter implemented `NONE` and `SIMPLE`.
- `filter_string`: which is a string representing the data to be used by the selected filter kind. For the `SIMPLE` filter kind the string must be the pattern used to match the symbol

name.

An example of TSP Request Filtered usage is given hereafter using the generic_consumer.
In this first example we want to see what symbol containing the string '99' in their name is offered by a Stubbed Server provider:

```
1. $ tsp_request_generic -u rpc://tsp_demo/StubbedServer
   tsp_request_filtered_information SIMPLE 99
2. tsp_request_generic: TSP provider URL is <rpc://tsp_demo>
3. Request Open successfully sent to : <rpc://tsp_demo/StubbedServer:0>
4. Obtained channel Id : <0>
5. Provider::base frequency      = 100.000000
6. Provider::max period          = 100000
7. Provider::max consumer        = 100
8. Provider::current consumer nb = 1
9. Provider <symbols list begin>
10.    pgi = 00000099, Symbol99, type = TSP_TYPE_DOUBLE, dim = 1
11.    pgi = 00000199, Symbol199, type = TSP_TYPE_DOUBLE, dim = 1
12.    pgi = 00000299, Symbol299, type = TSP_TYPE_DOUBLE, dim = 1
13.    pgi = 00000399, Symbol399, type = TSP_TYPE_DOUBLE, dim = 1
14.    pgi = 00000499, Symbol499, type = TSP_TYPE_DOUBLE, dim = 1
15.    pgi = 00000599, Symbol599, type = TSP_TYPE_DOUBLE, dim = 1
16.    pgi = 00000699, Symbol699, type = TSP_TYPE_DOUBLE, dim = 1
17.    pgi = 00000799, Symbol799, type = TSP_TYPE_DOUBLE, dim = 1
18.    pgi = 00000899, Symbol899, type = TSP_TYPE_DOUBLE, dim = 1
19.    pgi = 00000990, Symbol990, type = TSP_TYPE_DOUBLE, dim = 1
20.    pgi = 00000991, Symbol991, type = TSP_TYPE_DOUBLE, dim = 1
21.    pgi = 00000992, Symbol992, type = TSP_TYPE_DOUBLE, dim = 1
22.    pgi = 00000993, Symbol993, type = TSP_TYPE_DOUBLE, dim = 1
23.    pgi = 00000994, Symbol994, type = TSP_TYPE_DOUBLE, dim = 1
24.    pgi = 00000995, Symbol995, type = TSP_TYPE_DOUBLE, dim = 1
25.    pgi = 00000996, Symbol996, type = TSP_TYPE_DOUBLE, dim = 1
26.    pgi = 00000997, Symbol997, type = TSP_TYPE_DOUBLE, dim = 1
27.    pgi = 00000998, Symbol998, type = TSP_TYPE_DOUBLE, dim = 1
28.    pgi = 00000999, Symbol999, type = TSP_TYPE_DOUBLE, dim = 1
29. Provider <symbols list end>.
30. Request Close successfully sent to <rpc://tsp_demo/StubbedServer:0>
```

In this second example we want to see symbols offered by a provider answering at TSP URL `rpc://tsp_demo/bb_simu` and containing the string '**_0_**' in their name:

```
1.  $ tsp_request_generic -u rpc://tsp_demo/bb_simu
    tsp_request_filtered_information SIMPLE _0_
2.  tsp_request_generic: TSP provider URL is <rpc://tsp_demo/bb_simu>
3.  Request Open successfully sent to : <rpc://tsp_demo/bb_simu:1>
4.  Obtained channel Id : <4>
5.  Provider::base frequency      = 32.000000
6.  Provider::max period          = 100000
7.  Provider::max consumer        = 100
8.  Provider::current consumer nb = 1
9.  Provider <symbols list begin>
10.     pgi = 00000017, DYN_0_d_qsat, type = TSP_TYPE_DOUBLE, dim = 4
11.     pgi = 00000018, ORBT_0_d_possat_m, type = TSP_TYPE_DOUBLE, dim = 3
12.     pgi = 00000019, ECLA_0_d_ecl_sol, type = TSP_TYPE_DOUBLE, dim = 1
13.     pgi = 00000020, ECLA_0_d_ecl_lune, type = TSP_TYPE_DOUBLE, dim = 1
14.     pgi = 00000021, POSA_0_d_DirSol, type = TSP_TYPE_DOUBLE, dim = 3
15.     pgi = 00000022, POSA_0_d_DirLun, type = TSP_TYPE_DOUBLE, dim = 3
16.     pgi = 00000023, Sequenceur_0_d_t_s, type = TSP_TYPE_DOUBLE, dim = 1
17. Provider <symbols list end>.
18. Request Close successfully sent to <rpc://tsp_demo/bb_simu:1>
```

2.4.4 Request Extended Informations

The `TSP_sample_symbol_t` contains all the necessary informations for defining a TSP symbol; nevertheless sometimes some other informations may be interesting for specific provider or consumer. Example of extended informations are:

- unit : second, meter, etc...
- profile, order : TSP only support 1-dimensional arrays so if your application wants to map 1-dimensional array to multi-dimensional the provider may indicates that such 1-D array of dimension 9 should map to a 3*3 matrix in a row-major column ordering. Here `profile="3*3"` and `order="row"`.

The extended informations ARE NOT normalized by the TSP protocol, so consumer/provider pair should agree on their semantic. Generic Consumer should be able to display those informations but nothing more.

For this need, some TSP providers may provide "**extended informations**" for the concerned TSP symbols. The extended information is a list of "key/value" pair which is attached to each symbol that needs it. It is up to the provider to add extended informations to symbols. On the same provider,

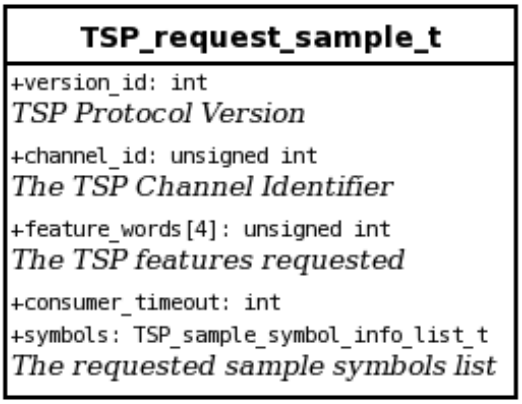
some symbols may have extended information and other may not.

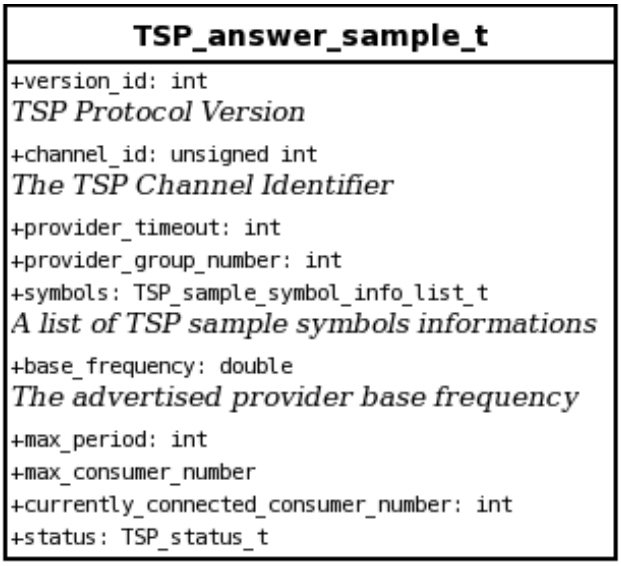
2.4.5 Request Sample/SampleInit/SampleDestroy

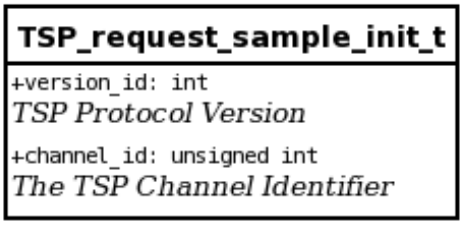
The sampling process is the heart of TSP. Using TSP you are able to receive sample symbols values, i. e. values of a TSP symbols varying over time. Those “sample” values are provided by a TSP provider in a flexible and efficient way. When a consumer wants to receive sample symbols value it has to send:

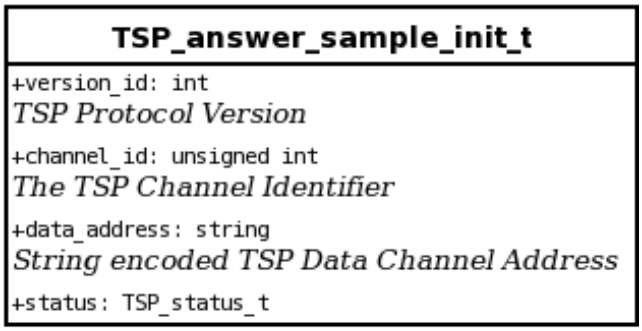
1. one or several TSP Request Sample in order to describe and negotiate what it wants to “sample”
2. a Request Sample Init[ialization] in order to start the sampling and obtain a data address of the TSP Data Channel used to receive sample
3. a Request Sample Destroy when he wants to terminate the sampling process.

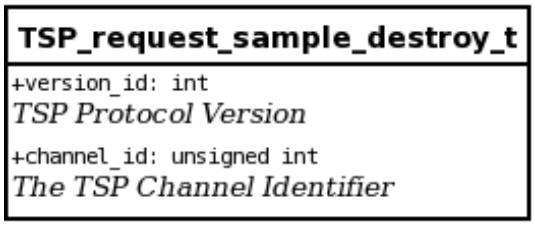
The structure of the Sample Requests and Answer is illustrated in the following figure:

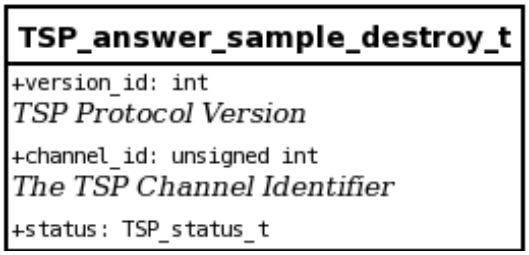












When the consumer sends a TSP Sample Request it receives a TSP Sample Answer. The answer mostly consists in a global status (`TSP_status_t`) which may be `TSP_STATUS_OK` or `TSP_ERROR_XXX`[2] and the list of validated requested symbols, `TSP_sample_symbols_info_list_t`.

When status is `TSP_STATUS_OK` the consumer may proceed with Request Sample Init.
When status is `TSP_STATUS_ERROR_SYMBOLS` the consumer hast to check the symbols list contained in the answer. The symbols whose provider global index set to **-1** are the ones that may not be satisfied by the provider because they are unknown.

2 consult the API documentation for the list of possible ERROR codes

When status is **not** `TSP_STATUS_OK` the consumer has to resend an updated TSP Request Sample until he gets an OK status.

> The consumer **should not** send a TSP Request Sample Init unless ***the last TSP Request Sample*** he sent triggered a TSP Answer Sample with `TSP_STATUS_OK`.

Once the status is OK the consumer sends the TSP Request Sample Init and gets a TSP Answer Sample Init which contains the address of the TSP Data Channel to use to receive the sampled data. The string representing the data address has the form **<host>:<port>.** The consumer hast to open a TCP socket on this **<host>:<port>** in order to receive the TSP sample values..

As soon as the consumer is connected to the TSP Data Channel he will receive the samples.

When the consumer doesn't want to receive samples anymore he has to send the TSP Request Sample Destroy which terminates the sample process. The provider will release the TSP Data Channel and close the socket.

2.4.6 Request Asynchronous Read/Write

If a consumer needs to pick only one value of a symbol it may use the TSP Request Asynchronous Read.

Using this request the consumer does receive a stream of sample data but only one value. Another difference with Request Sample is that the provider cannot ensure **WHEN** the value was collected on provider side. If a consumer sends 2 Request Asynchronous Read he cannot assume anything useful about the time it will take to get the answer.

It is to be compared with a Request Sample for which the provider guarantees the sample values sent over the TSP Data Channel respect the TSP Request Sample timing contract (period, phase). Different values of the **same** symbol on the TSP Data Channel MUST have been sampled by the provider at the rate specified by the request sample.

The TSP Request Asynchronous Write may be used to ask the TSP provider to write a value onto a symbol.

This is up to the provider to accept or not asynchronous read and/or write request. The default GLU implementation does not implements asynchronous read and write operation. The Blackboard GLU

used by the Blackboard provider does.

The generic consumer may be used to send TSP Request Asynchronous Read or Write request to a TSP Provider. Here follows an example of use on a `bb_tsp_provider` running on `localhost`. The TSP URL is not specified on command line so the implicit value `rpc://localhost` is used.

```
1. $ tsp_request_filtered_information SIMPLE Sequenceur
2. Provider::base frequency      = 30.000000
3. Provider::max period          = 100000
4. Provider::max consumer        = 100
5. Provider::current consumer nb = 1
6. Provider <symbols list begin>
7.     pgi = 00000049, Sequenceur_0_d_t_s, type = TSP_TYPE_DOUBLE, dim = 1
8. Provider <symbols list end>.
9. $ tsp_request_async_sample_read 49
10. 2054.760000
11. $ tsp_request_async_sample_read 49
12. 2055.150000
13. $ tsp_request_filtered_information SIMPLE disp
14. Provider::base frequency      = 30.000000
15. Provider::max period          = 100000
16. Provider::max consumer        = 100
17. Provider::current consumer nb = 1
18. Provider <symbols list begin>
19.     pgi = 00000000, bb_simu_display_level, type = TSP_TYPE_UINT32, dim
    = 1
20. Provider <symbols list end>.
21. $ tsp_request_async_sample_read 0
22. 0.000000
23. $ tsp_request_async_sample_write 0 5
24. $ tsp_request_async_sample_read 0
25. 5.000000
26. $ tsp_request_async_sample_write 0 0
27. $ tsp_request_async_sample_read 0
28. 0.000000
29. $
```

We use request filtered information (lines 1—8) in order to find the PGI (provider global index) of the symbol. Then we send request asynchronous read and write. The first argument of those

requests is the PGI, the second argument of the write request is the value to be written.

2.5 TSP Data Channel – Synchronous TSP

When a TSP Consumer wants to observe data it negotiates the list of the concerned [sample] symbols. After that, it will receive ***in a predefined order*** the data he asked for. The communication mean used to received data is the TSP Data Channel (Synchronous TSP). The data transmitted over the TSP Data Channel is encoded in XDR (RFC1832) in order to avoid endianness (http://en.wikipedia.org/wiki/Endianness) issue between providers and consumers. The TSP Data Channel is a "***logically connected and lossless***" communication mean which may be implemented over different transport protocol. The default data channel of the current TSP implementation is TCP/IP socket.

3 Understanding TSP modules

The TSP project consists in several modules which may be used for different needs or environments.

3.1 The TSP modules

The TSP project is divided into the following modules:

CVS module name	*Role*
tsp	The TSP in C language. This module includes the core TSP protocol libraries, the TSP Blackboard, some ready to use provider and consumer written in C.
jtsp	The 100% Java TSP, which is the way to use TSP in Java.
tsp_docs	The TSP documentation module including specifications, this guide and more.
perltsp	The TSP Perl binding, which is the way to use TSP in Perl.
pytsp	The TSP Python binding, which is the way to use TSP in Python.
tcltsp	The TSP Tcl binding, which is the way to use TSP in TCL.
rubytsp	The TSP Ruby binding, which is the way to use TSP in Ruby.

3.2 Accessing the TSP sources

TSP is hosted as a Savannah non-Gnu project; thus the different TSP modules may be publicly accessed on Savannah at https://savannah.nongnu.org/projects/tsp .

The TSP released version may be downloaded using the download/file section of the Savannah TSP project.
Bleeding edge TSP snapshot may be retrieved through (anonymous) CVS access. Savannah offers anonymous read-only access to CVS and full read-write access to registered TSP Project members.

4 TSP in C

The TSP C module is written in ANSI C, using only standard library either C standard library or POSIX API (pthread). TSP in C is meant to be as portable as possible and is currently running on Linux (32bit Intel, Power PC), DEC OSF, Solaris 2.5+ (Sparc, Intel), Free BSD and VxWorks.

4.1 Setting up your TSP

TSP in C comes as a software toolkit. You may use TSP in your application by using a binary TSP distribution or by building your own TSP using the TSP source distribution. The primary TSP distribution format is the source distribution. It is out of the scope of the TSP project to build binary distribution for all TSP supported platforms.

4.1.1 TSP Binary distribution

If you get a binary TSP distribution such as pre-packaged RPM for your favorite Linux distribution, you may install it as usual, for example:

```
rpm -i tsp-0.7.3-1.i586.rpm
```

If you did get a Source RPM you may rebuild the binary RPM before installing by doing:

```
rpm -i tsp-0.7.3-1.src.rpm
rpmbuild -bb /usr/src/RPM/SPECS/tsp.spec
rpm -i /usr/src/RPM/RPMS/i586/tsp-0.7.3-1.i586.rpm
```

The exact path for `/usr/src/RPM` may vary depending on your Linux distribution. The resulting binary RPM does depend on your target architecture too. If you are not familiar with RPM, please go to http://www.rpm.org/ or any other RPM resources in order to find detailed informations about RPM usage.

The TSP should be installed under `TSP_HOME` whose value may depends on the packager of the RPM. A typical binary distribution will have:

TSP_HOME	SUBDIR	What's inside
/opt/tsp	/bin	The binary TSP executable such as ready-to-use providers (tsp_stub_server, bb_tsp_provider, ...), ready-to-use consumers (tsp_gdisp, tsp_gdisp+, tsp_ascii_writer, tsp_request_generic) blackboard tools command line....
	/include	The TSP public includes to be used within your application using TSP public API.
	/lib	The TSP library to link with when using TSP API.
	/scripts	Helper scripts like bb_tools or tsp_request_xxx wrapper scripts. This directory contains tsp_profile.sh and tsp_profile.csh . On Linux systems those file may be added to your /etc/profile.d/ directory in order to set up path and environment variables for the TSP user on the system. Those files are sourced on each shell startup (see your Linux manual for more informations).

You may want to rebuild your own TSP binary distribution tailored for your system using a *tarball* source distribution, or a your private CVS extracted source tree. You may get both on Savannah: http://savannah.nongnu.org/projects/tsp .

4.1.2 TSP Source distribution

If you get a tarball source TSP distribution such as you may found in the download section of the Savannah project (http://download.savannah.nongnu.org/releases/tsp/), you should follow these steps:

- untar the archive: `tar zxvf tsp-<version>.tar.gz`
  this should create a `tsp` directory

```
$ tar zxvf tsp-0.7.3.tar.gz
... wait for tar ending ...
```

- configure your TSP (`configure --help` for more options)

```
$ cd tsp
$ ./configure
... wait for configure ending ...
```

- set-up some environment variables as indicated at the end of the configure scripts

```
$ source src/scripts/tsp_dev.login.sh
Using host target <linux>
Using TSP_BASE=/home/noularde/TSP/tsp_all/tsp
Using DEVBASE=/home/noularde/TSP/tsp_all/tsp
Using STRACE_DEBUG=1
$
```

The `tsp_dev.login.sh` script sets-up some environment variables. When you are using a TSP source toolkit, you need to source this file in each shell you want to use the TSP source kit. The environment variables defined or modified by the script are:

1. `TSP_BASE` is the variable defining the 'base' directory of your TSP development source toolkit,
2. `STRACE_DEBUG`, is the variable which controls the amount of trace the TSP libraries are sending to standard output when running a program using the TSP libraries.
3. `PATH` which is updated in order to include the path to the TSP binary executable when they are built

- compile your TSP

```
$ make
... wait the compilation end ...
```

After that you will have a compiled and usable TSP source kit.

The obtained TSP source tree is the following:

TSP_BASE	SUBDIR	What's inside
`$TSP_BASE`	`/src`	The sources of the TSP core libraries and consumers and providers applications
	`/make`	The makefiles fragments used by the TSP Makefile system
	`/external`	Some external source codes or binaries which may be useful to build some part of the TSP. They are not part of the TSP but are delivered here for convenience. Some of them are contribution from the TSP Team for improving TSP portability, like the `external/VxWork/posix` module.
	`/tests`	The tests sources code (may be scripts or C code ...)
	`/exec`	The directory where all products of compilation goes (the structure of this directory is explained later in this document).
	`/dist`	The directory where “ready to be distributed TSP modules” are generated, like RPM or tar.gz or API documentation)

When you compile TSP all the compilation result goes in `$TSP_BASE/exec` and its sub-directories

`$TSP_BASE/exec`	
SUBDIR	What's inside
`current`	Symbolic link pointing to the currently compiled version usually this is DEV.
`DEV`	The current tree used when compiling
`DEV/include`	The public TSP includes (exported includes)
`DEV/scripts`	The public TSP scripts (exported scripts)
`DEV/<arch>/<mode>/bin`	The binaries executables for this specific architecture (for example `linux`) in this compile mode (`debug` or `opt`)
`DEV/<arch>/<mode>/lib`	The libraries (static or shared) for this specific architecture.
`DEV/<arch>/<mode>/obj`	The repository for object files used to build static libraries or executable programs.
`DEV/<arch>/<mode>/shobj`	The repository for object files used to build shared libraries.

The `$TSP_BASE/exec/current` directory contains almost all what is needed to build a TSP RPM as you will see if you read the `$TSP_BASE/src/scripts/tsp.spec` file.

The only exception is the `$TSP_BASE/dist/doc` directory where the API documentation is generated.

When you develop inside TSP you have to understand the structure of the `$TSP_BASE/src` directory and its sub-directories

`$TSP_BASE/src`	
SUBDIR	What's inside
`consumers`	Directory containing the ready-to-use TSP consumers applications (ascii_writer, gdisp, gdisp+, generic...)
`core`	Directory containing the Core TSP libraries used to build consumers and providers.
`doxy`	The directory containing doxygen configuration files and Makefile.

`providers`	Directory containing the ready-to-use TSP providers applications (stub, bb_provider, res_reader,...).
`scripts`	Directory containing some useful shells scripts beginning with the configure generated `tsp_dev.login.sh` script.
`utils`	Directory containing utility libraries (Blackboard, XML configuration file reader/writer, etc...) used by some consumers or providers.

4.2 Provider side programming

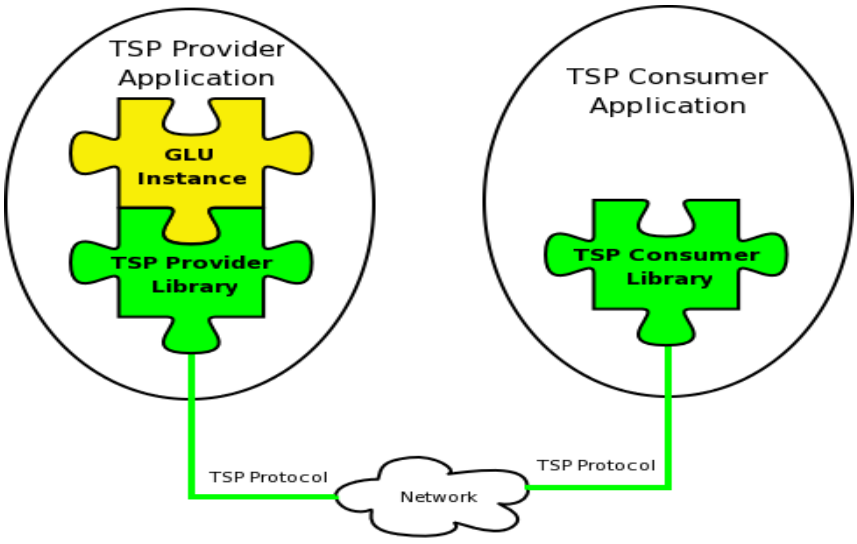
TSP Provider-side programming globally means enabling your favorite application to provided TSP symbols. Nevertheless there are different ways to achieve this goal:

1. write your home made provider conforming to the GLU interface
2. re-use some ready-to use provider libraries

We will examine hereafter the 2 options.

4.2.1 Writing a GLU type provider

Developing a new TSP provider is very simple you only need to write a set of functions implementing the ***TSP GLU interface***. The GLU is the part of a TSP provider application which is ***specific*** to the concerned provider. The TSP provider library will **call the GLU interface** in order to answer to the TSP Requests, using TSP protocol.

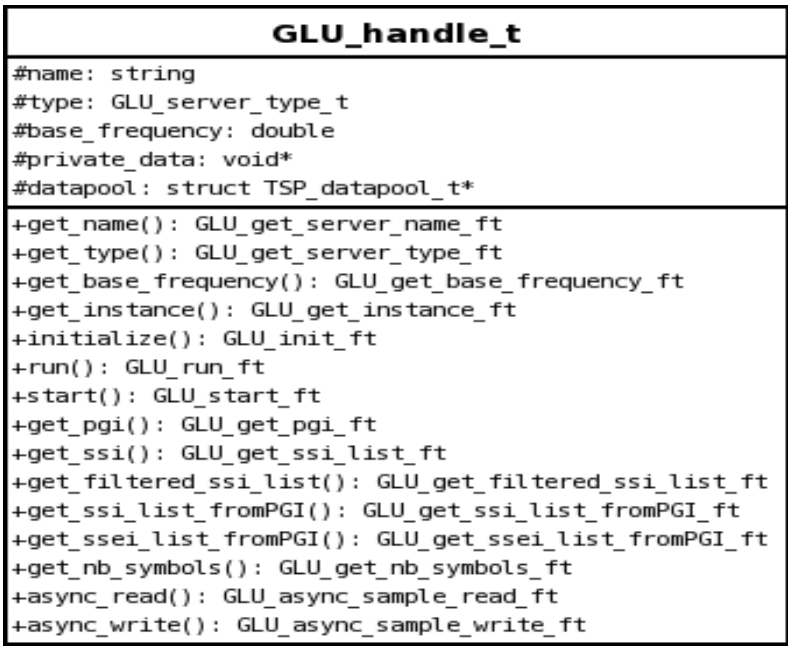


Every TSP provider must create a GLU object and pass it has an argument of the TSP provider library initialization. Here is an example of the STUB provider initialization (see `$TSP_BASE/src/provider/stub/server_main.c`)

```
1.  GLU_handle_t* GLU_stub = GLU_stub_create();
2.
3.    /* Init server */
4.    if(TSP_STATUS_OK==TSP_provider_init(GLU_stub,&argc, &argv)) {
5.          if  (TSP_STATUS_OK!=TSP_provider_run(TSP_ASYNC_REQUEST_SIMPLE  |
    TSP_ASYNC_REQUEST_NON_BLOCKING)) {
6.        return -1;
7.      }
8.      TSP_provider_urls(TSP_PUBLISH_URLS_PRINT | TSP_PUBLISH_URLS_FILE);
9.      sigwait(&allsigs, &whatsig);
10.     TSP_provider_end();
11. }
```

A GLU handle is created at **line 1** and passed for TSP provider library initialization at **line 4**.

The GLU may be considered as a structured callback. The GLU Interface is an Object-Oriented C interface using plain ANSI C structure and ANSI C function pointers. It is specified as a C structure named `GLU_handle_t` in the `$TSP_BASE/src/core/include/tsp_glu.h` include. This C structure is the C implementation of the following GLU_handle_ class:



Every **GLU_xxxx_ft** type is a C function pointer typedef, defined and documented in the $TSP_BASE/src/core/include/tsp_glu.h include file. A new GLU must not implement ALL the functions of the GLU interface.

There is mandatory GLU method (C function pointer) that MUST be implemented and there is optional method for which default implementation may be provided by the default GLU (see $TSP_BASE/src/core/ctrl/tsp_default_glu.c). The default GLU may be considered as an Abstract Base class for GLU.

Here is the example of the STUB GLU creation
(see $TSP_BASE/src/provider/stub/glue_stub.c):

```
1. /* create the GLU handle instance for STUB */
2. GLU_handle_t* GLU_stub_create() {
3.
4.   /* create a default GLU */
5.   GLU_handle_create(&stub_GLU,"StubbedServer",GLU_SERVER_TYPE_ACTIVE,TS
   P_STUB_FREQ);
6.
7.   stub_GLU->initialize              = &STUB_GLU_init;
8.   stub_GLU->run                     = &STUB_GLU_thread;
9.   stub_GLU->get_ssi_list            = &STUB_GLU_get_ssi_list;
10.  stub_GLU->get_ssei_list_fromPGI  = &STUB_GLU_get_ssei_list_fromPGI;
11.
12.  return stub_GLU;
13. } /* GLU_stub_create */
```

The mandatory GLU methods are:

- `initialize`: this function is called only once at the end of `TSP_provider_init` function. It is supposed to do whatever initialization the further calls to the GLU will need
- `run`: this function is called at the end of the `TSP_provider_run` function. The run GLU method will be launched in a separate and is supposed to never return the GLU sampling activity is terminated.
- `get_ssi_list`: (get **S**ample **S**ymbol **I**nformation List) this function is used by the TSP provider library in order to obtain the complete list of symbols provided by the GLU.

All other GLU methods are optional since default implementation may be built either by using the 3 mandatory method or by giving a default simple behavior. Implementing non-mandatory method may be useful since the specific GLU may provide a far better (in terms of performance) implementation than the default one.

This the case of the Blackboard provider which overrides the `get_pgi` method with a more efficient method
(see `$TSP_BASE/src/provider/bb_provider/bb_tsp_provider.c`).
The Blackboard provider overrides `async_read` and `async_write` too since the default implementation simply refuses asynchronous read or write.

The GLU method uses TSP data types in their prototype like:

- SSI: **S**ample **S**ymbol **I**nformation (see `TSP_sample_symbol_info_t` and `TSP_sample_symbol_info_list_t`)
- SSEI: **S**ample **S**ymbol **E**xtended **I**nformation (see `TSP_sample_symbol_extended_info_t` and `TSP_sample_symbol_extended_info_list_t`)

Those structures are defined and their usage documented in the source code (doxygen comments), see `$TSP_BASE/src/core/common/tsp_common_*.`

There is one more thing to say about GLU, a GLU may be ACTIVE or PASSIVE. An ACTIVE GLU has only one instance and does not wait for the TSP consumer to produce the sample data. It is driven by a process which may not be suspended like real time simulation or external world. ***This is the most common case***. A PASSIVE GLU may suspend its sample data flow, this is the case of the TSP provider which reads their sample data from a file (see generic_reader or res_reader Provider).

The better way to understand how a GLU is working and how to implement the GLU interface is to go to the examples in `$TSP_BASE/src/provider/*` subdirectories.

The STUB server is a good starting point for writing a new ACTIVE GLU, see
`$TSP_BASE/src/provider/stub`
The Generic Reader s a good starting point for writing a new PASSSIVE GLU, see
`$TSP_BASE/src/provider/generic_reader.`

4.2.2 Using a Blackboard provider

If you already have a nice C application and you want it to provides sample symbols using TSP, but you don't want to code your own GLU, you may use a TSP Blackboard.

Adding TSP to an existing C/C++ application using a Blackboard is usually done in less than 2 days of work including basic TSP learning.

The C TSP comes with a utility library implementing the Blackboard idiom. The TSP Blackboard is a structured shared memory area where data may be published.
The Blackboard library itself is independent of the TSP, the Blackboard library source code is located at: `$TSP_BASE/src/util/libbb` .

You may examine an example of pseudo-simulator using a TSP Blackboard at: `$TSP_BASE/src/util/libbb/bbtools/bb_simu.c` .

Here is an extract of the bb_simu code:

```
1.  /* Create Blackboard */
2.   /***************/
3.   n_data = 1000;
4.   data_size = n_data*8 + 500*30*4 + 200000*8;
5.   if (BB_NOK==bb_create(&mybb,basename(argv[0]),n_data,data_size)) {
6.     bb_attach(&mybb,basename(argv[0]));
7.   }
8.
9.   /* Publish data in the BB */
10.  /*************************/
11. display_level = (uint32_t*)
    bb_simple_publish(mybb,"display_level",basename(argv[0]),-1,
    E_BB_UINT32, sizeof(uint32_t),1);
12.
13. Titi= (double*) bb_simple_publish(mybb,"Titi",basename(argv[0]),1,
    E_BB_DOUBLE, sizeof(double),1);
14.
15. [...]
16. /* use the published data */
17. *display_level = 0;
18. *Titi = 3.14159;
19. [...]
20. /* send synchro for Blackboard TSP provider
21. bb_simple_synchro_go(mybb,BB_SIMPLE_MSGID_SYNCHRO_COPY);
```

In the previous example we see that publishing a data in a TSP Blackboard (lines 11 and 13) is equivalent to a call to `malloc(3)` . If the returned address is non NULL you may use it in your program (lines 17 and 18). Afterwards if you want to distribute the values of the published data suing TSP you only have to send synchronization (line 21).
The symbols values may be distributed using TSP by using the ready-to-use Blackboard provider as explained in section 11.1.2.

4.3 Developing a new consumer

The C TSP comes with several TSP consumers in the `tsp/src/consumers` sub-directories. The simplest are either the tutorial consumer (`tsp/src/consumers/tutorial`) or the stdout (`tsp/src/consumers/stdout`).

A full featured console consumer is the AsciiWriter (`tsp/src/consumers/ascii_writer`).

The most efficient way to develop a new consumer is to read the source code of one or several consumers, in order to understand the practical use of the consumer C API. Do not forget to use the doxygen generated API documentation which is a handy TSP developer tool for using TSP C API.

Developing a new TSP consumer (in C) is as simple as understanding the TSP design and the TSP C consumer API (tsp_consumer.h).

Here is a shortened example taken from the tutorial client:

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  /* All what you need for creating a TSP consumer */
4.  #include <tsp_consumer.h>
5.  #include <tsp_time.h>
6.
7.  /* Just for fast exit */
8.  void perror_and_exit(char *txt)
9.  { perror (txt); exit (-1);}
10.
11. /* Everthing must begin somewhere */
12. int main(int argc, char *argv[]) {
13.
14.   const TSP_answer_sample_t*  information;
15.   TSP_sample_symbol_info_list_t symbols;
16.   int i, count_frame, wanted_sym=10, t = -1;
17.   TSP_provider_t provider;
18.   char* url;
19.   /* Initialisation for TSP library. */
20.   if(TSP_STATUS_OK!=TSP_consumer_init(&argc, &argv))
```

```
21.       perror_and_exit("TSP init failed");
22. [... handle program argv/argc ...]
23.   /* Connects to all found providers on the given host. */
24.   provider = TSP_consumer_connect_url(url);
25.   if(0==provider)
26.     perror_and_exit("TSP_consumer_connect_url failed ");
27.
28.   /* Ask the provider for a new consumer session.*/
29.   if(TSP_STATUS_OK!=TSP_consumer_request_open(provider, 0, 0))
30.     perror_and_exit("TSP_request_provider_open failed");
31.
32.   /* Ask the provider informations about several parameters, including
33.    * the available symbol list that can be asked. */
34.   if(TSP_STATUS_OK!=TSP_consumer_request_information(provider))
35.     perror_and_exit("TSP_request_provider_information failed");
36.
37.   /* Get the provider information asked by
    TSP_consumer_request_information */
38.   information = TSP_consumer_get_information(provider);
39.   if (wanted_sym > information-
    >symbols.TSP_sample_symbol_info_list_t_len)
40.       wanted_sym = information-
    >symbols.TSP_sample_symbol_info_list_t_len;
41.
42.   /* Will use only the "wanted_sym" first symbols of provider */
43.   symbols.TSP_sample_symbol_info_list_t_val =
    (TSP_sample_symbol_info_t*)calloc(wanted_sym,
    sizeof(TSP_sample_symbol_info_t));
44.
45.   symbols.TSP_sample_symbol_info_list_t_len = wanted_sym;
46.   for(i = 0 ; i < wanted_sym ; i++)
47.     {
48.       symbols.TSP_sample_symbol_info_list_t_val[i].name = information-
    >symbols.TSP_sample_symbol_info_list_t_val[i].name;
49.       symbols.TSP_sample_symbol_info_list_t_val[i].period = 1; /* at
    max frequency */
50.       symbols.TSP_sample_symbol_info_list_t_val[i].phase = 0;         /*
    with no offset */
51.       printf ("Asking for symbol = %s\n",
    symbols.TSP_sample_symbol_info_list_t_val[i].name);
52.     }
53.
```

```
54.  /*-------------------------------------------------------------------
     ----------------------------------*/
55.  /* Ask the provider for sampling this  list of symbols. Should check
     if all symbols are OK*/
56.  if(TSP_STATUS_OK!=TSP_consumer_request_sample(provider, &symbols))
57.    perror_and_exit("TSP_request_provider_sample failed");
58.
59.  /* Start the sampling sequence. */
60.  if(TSP_STATUS_OK!=TSP_consumer_request_sample_init(provider, 0, 0))
61.    perror_and_exit("TSP_request_provider_sample_init failed");
62.
63.  /* Loop on data read */
64.  for (count_frame = 0; count_frame<100; )
65.    {
66.      int new_sample=FALSE;
67.      TSP_sample_t sample;
68.
69.      /* Read a sample symbol.*/
70.      if (TSP_STATUS_OK==TSP_consumer_read_sample(provider, &sample,
     &new_sample))
71.      {
72.        if(new_sample)
73.          {
74.            if (t != sample.time)
75.            {
76.              count_frame++;
77.              t = sample.time;
78.              printf ("======== Frame[%d] ======== Time : %d
     ========\n", count_frame, t);
79.            }
80.            i = sample.provider_global_index;
81.            printf ("# Sample nb[%d] \t%s \tval=%f\n", i,
     symbols.TSP_sample_symbol_info_list_t_val[i].name,
     sample.uvalue.double_value);
82.          }
83.        else
84.          {
85.            /* Used to give time to other thread for filling fifo of
     received samples */
86.            tsp_usleep(100*1000); /* in S, so about 100msec */
87.          }
```

```
88.       }
89.       else
90.       {
91.         perror_and_exit ("Function TSP_consumer_read_sample failed !!
    \n");
92.       }
93.     }
94.
95.   free (symbols.TSP_sample_symbol_info_list_t_val);
96.   /* Stop and destroy the sampling sequence*/
97.   if(TSP_STATUS_OK!=TSP_consumer_request_sample_destroy(provider))
98.     perror_and_exit("Function TSP_consumer_request_sample_destroy
    failed" );
99.
100.  /* Close the session.*/
101.  if(TSP_STATUS_OK!=TSP_consumer_request_close(provider))
102.    perror_and_exit("Function TSP_consumer_request_close failed" );
103.  /* call this function when you are done with the library.*/
104.  TSP_consumer_end();
105.  return 0;
106.}
```

4.4 Source code documentation (API doc)

The TSP API is documented using special comment in the source code itself. Doxygen (http://www.doxygen.org) is the tool used by TSP in order to generate readable documentation (HTML, PDF, etc...) from source code special comment.
You may browse latest TSP generated documentation from http://www.ts2p.org/tsp/API_doc/html.

Every TSP developer ***must*** document his source code. You will find hereafter some recommendations and example of code documentation using doxygen tags.

It is not the purpose of this guide to explain all the features and objectives of Doxygen, you may find all needed informations about doxygen from http://www.doxygen.org.

4.4.1 General Doxygen usage

The Doxygen documentation is governed by the **tsp** Doxygen configuration file which may be found here : `$TSP_BASE/src/doxy/tsp`.

> Note that this file is a `configure` output that is produced out of `$TSP_BASE/src/doxy/tsp.in` configure input file. This is done with the intent to maintain the TSP version information only in a single place (the `$TSP_BASE/configure.ac` file). If you ever need to change something in the TSP doxygen configuration file, do it in `$TSP_BASE/src/doxy/tsp.in` and re-run ./configure.

The `$TSP_BASE/src/doxy/tsp.in` file may be edited by any text editor with the Doxygen manual in the other hand (http://www.doxygen.org/manual.html) or using the `doxywizard` tools which is distributed with the Doxygen software.

The TSP C API documentation may be generated using either command:

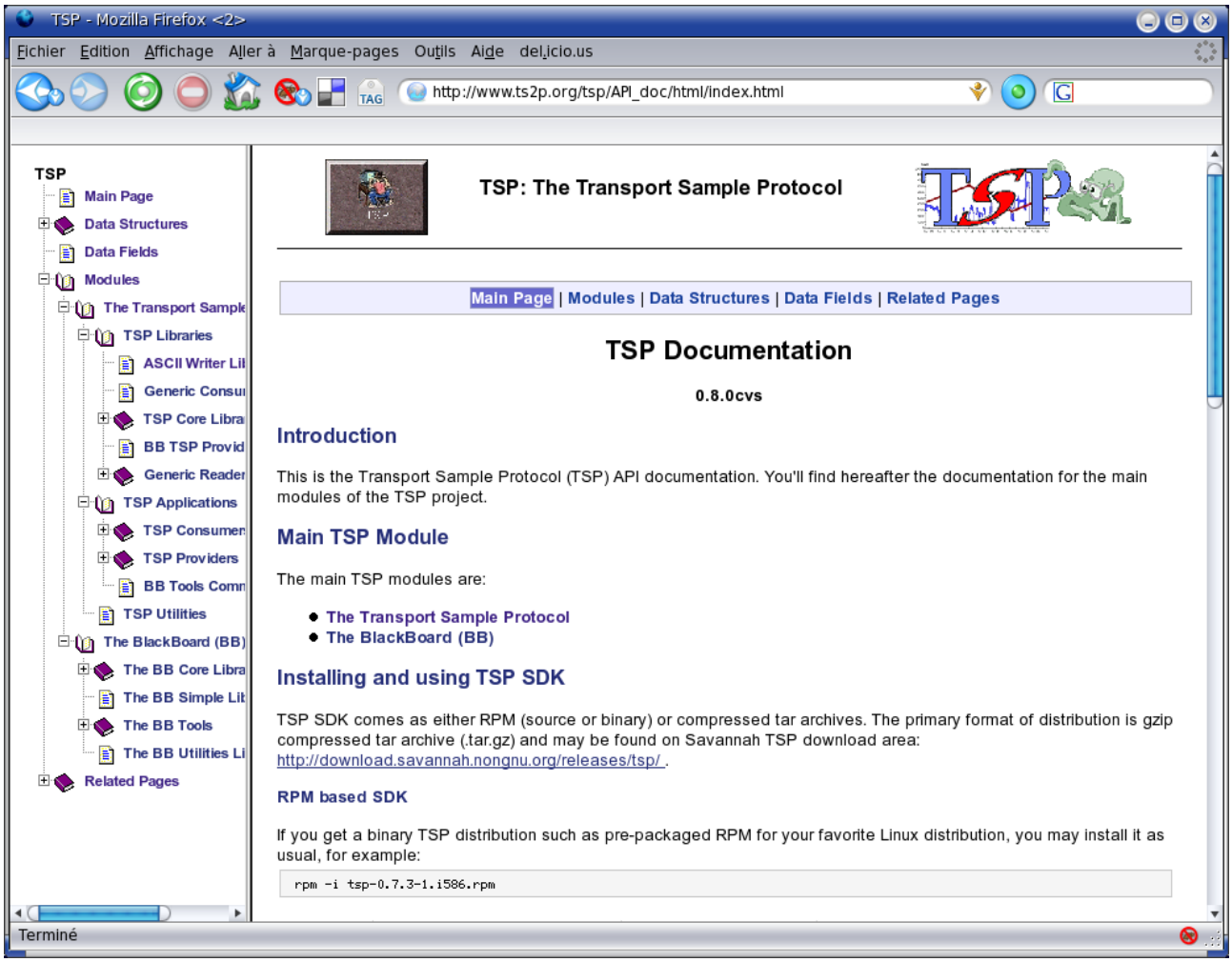
- ant tool (http://ant.apache.org/)

```
$ cd $TSP_BASE
$ ant apidoc
Buildfile: build.xml

init:

apidoc:
     [exec] ------TSP--Documentation Generation BEGIN----------
     [exec] -------TSP--Documentation Generation END------------

BUILD SUCCESSFUL
Total time: 22 seconds
$
```

- make

```
$ cd $TSP_BASE/src/doxy
$ make
------TSP--Documentation Generation BEGIN----------
-------TSP--Documentation Generation END------------
$
```

The generated HTML documentation is located in `$DEVBASE/dist/doc/api/html/` and you may open the `$TSP_BASE/dist/doc/api/html/index.html` file with your favorite browser in order to browse the documentation:



4.4.2 TSP Doxygen structure and usage example

We define here some rules for using Doxygen within TSP. Doxygen has a wealth of feature to define

documentation blocks. We describe here some recommendation for basic doxygen usage, please consult the doxygen manual (http://www.stack.nl/~dimitri/doxygen/manual.html) for more details.

Using the TSP doxygen configuration file the following files type will be parsed by doxygen:

File Patterns	Supposed File Content
`*.h, *.c`	Header and Implementation ANSI C files. The public API documentation must be done in the `.h` file with NO duplicate in corresponding `.c` implementation. All C public API should be documented: • structure, enumeration, typedef etc... • functions • MACRO definition (#define)
`*.x`	ONC-RPC IDL file. All items of the IDL should be documented: • structure, enumeration, typedef etc... • functions • MACRO definition (#define)
`*.dox`	Doxygen "free" documentation files. Those files may be used to create doxygen documentation blocks or structures which are not directly linked to source code. There is at least one mandatory `.dox` file `$TSP_BASE/src/doxy/tsp_doc_tree.dox`, which is used to define the main TSP doxygen documentation tree. Other file may be added since doxygen may be used to write documentation as you usually do it with pure HTML or LaTeX, docbook etc...

4.4.2.1 TSP Doxygen main structure

The TSP doxygen documentation structure is defined in the `$TSP_BASE/src/doxy/tsp_doc_tree.dox` file. This file defines the documentation main page for the project and the high level documentation groups for TSP. Those high level documentation groups may be used later to attach new documentation group with `@ingroup` doxygen facility.

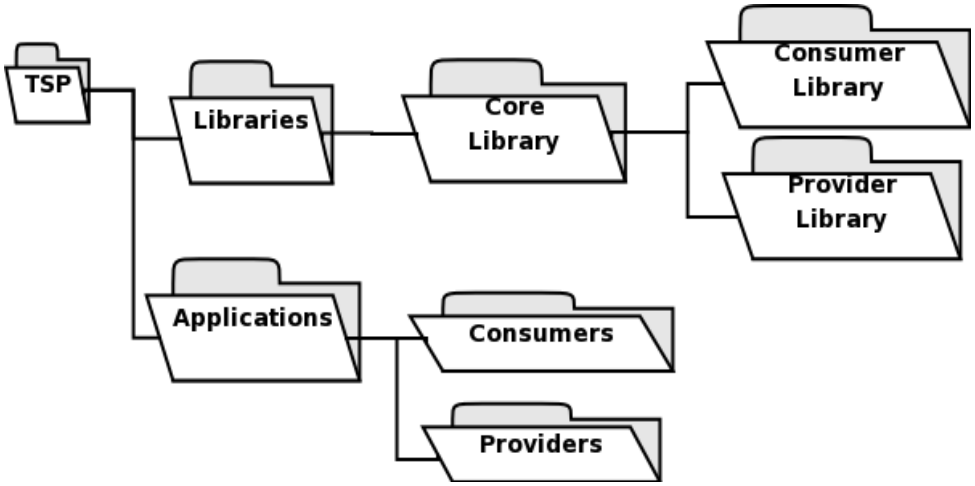
4.4.2.2 Grouping

Doxygen has a flexible and lazy way to build "groups" of documentation. When used with

“module” structured languages like C++, the groups definition may be inferred by the source code structure, for example, the: `Class Provider` would generate a “Provider” documentation group.

In TSP in C, we are using C language :)), hence we do not have implicit grouping information besides the file and directory structure. For that we decided (for TSP doxygen documentation) to explicitly define doxygen documentation groups that reminds the TSP design and C implementation.

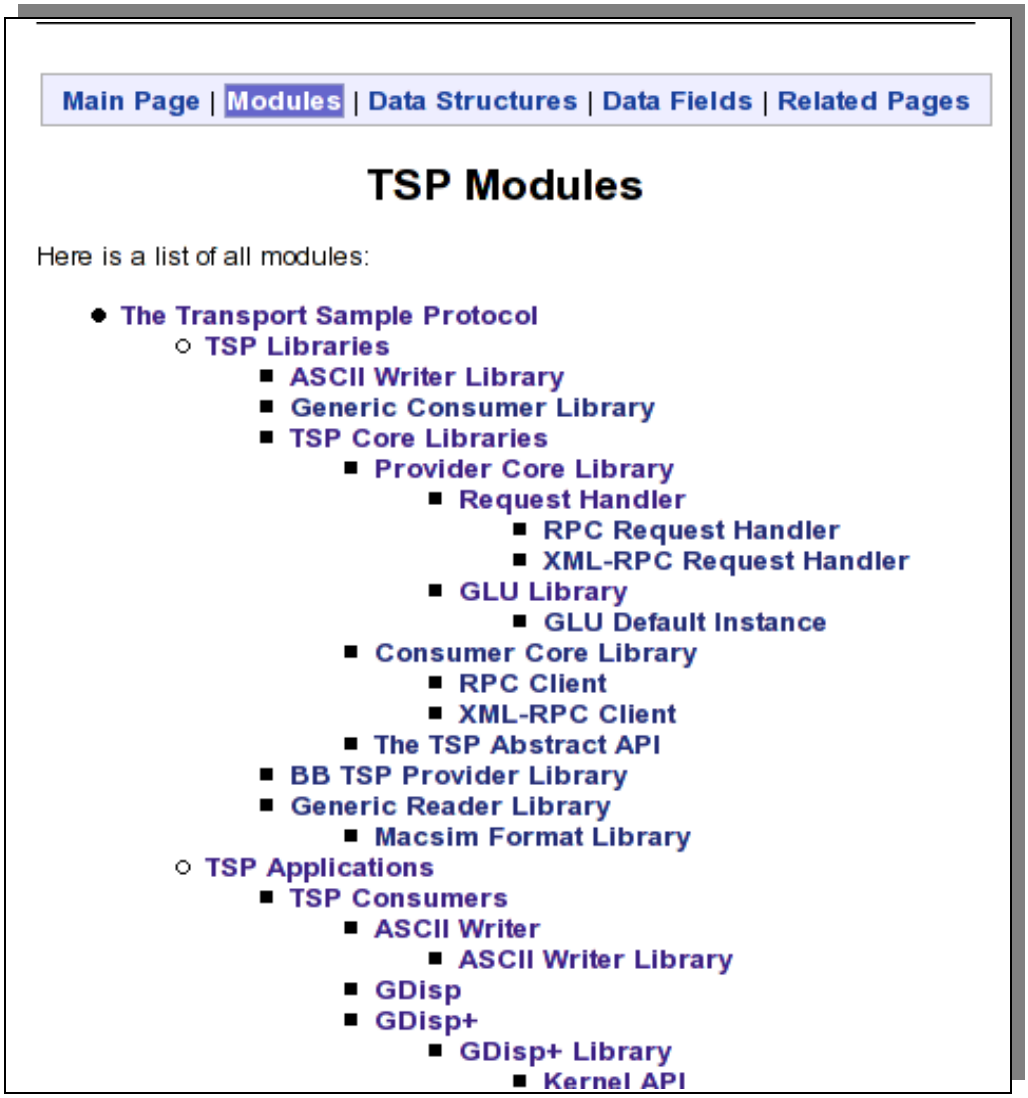
The overall (not detailed) documentation groups defined by the TSP design are shown below:



The rules for defining a new doxygen documentation group are the following:

- A properly defined design module must define a new group: TSP Core, TSP Consumers, TSP Providers, Blackboard, etc...
- An application must have its own module attached to a sub-group of TSP Applications
    - Consumer Applications should be put inside the TSP Consumers documentation module
    - Provider Applications should be put inside the TSP Providers documentation module
- A Library used by any other application or library should have its own documentation module

Using these rules you should be able to add or understand the documentation groups shown in the currently generated documentation.



When you want to create a group you have to use the **`@defgroup`** doxygen special command, as in the following example (excerpt from `$TSP_BASE/src/core/ctrl/tsp_provider.h`)

```
1. /**
2.  * @defgroup TSP_ProviderLib Provider Core Library
3.  * @ingroup  TSP_CoreLib
4.  * The Provider module is the set of all
5.  * provider library interface.
6.  * @{
7. */
```

The `@defgroup` command defines the group using a key and a displayed name on a single line (see line 2 of the previous example). `@ingroup` specifies that this group is a sub-group of the first

argument of `@ingroup`. The `@{` opening prefix specify that the following doxygen comments will be put in the previous `@defgroup` until a `@}` closing suffix is encountered. Most of the time the closing prefix is located at the end of the C header file.

`@defgroup` is lazy in the sense that multiple `@defgroup` is silently ignored by doxygen such that the first effectively `@defgroup` and other are equivalent to `@addtogroup` (see doxygen documentation for this).
You may use multiple `@ingroup` in case you want some groups of documentation to appear as sub-group of different groups just as in the following example:

```
/**
 * @defgroup TSP_AsciiWriterLib ASCII Writer Library
 * The TSP ascii writer consumer library API.
 * @ingroup TSP_AsciiWriter
 * @ingroup TSP_Libraries
 * @{
 */
```

The currently defined `TSP_AsciiWriterLib` group will be seen as a subgroup of both `TSP_AsciiWriter` and `TSP_Libraries`. This is a handy way to refer to the same group in different places without duplicate information.

4.4.2.3 Structure, Enumeration, Macros, Typedef

Every single “type” defined in public C header should be documented using doxygen. This is true for C structure, enumeration or typedef; the pre-processor constructs such as Macros. You will find hereafter an example for each kind of documented constructs.

C Structure documentation example

```
/**
 * BlackBoard data descriptor.
 * Each data published in a blackboard is described using
 * one such structure.
 */
typedef struct S_BB_DATADESC {
  /** Variable name */
  char name[VARNAME_MAX_SIZE+1];
  /** The Variable type */
  E_BB_TYPE_T type;
  /**
   * Dimension. 1 if scalar, > 1 for single dimension array.
   * There is no multidimensionnal array type.
   */
```

C Structure documentation example

```
  uint32_t dimension;
  /**
   * Type size (in byte).
   * This size enables the appropriate computation
   * of the data offset in the raw data BlackBoard area.
   */
  size_t type_size;
  /**
   * Data offset (in bytes) in the raw data BlackBoard area.
   */
  unsigned long data_offset;

  /**
   * The index of the aliases published (@ref bb_alias_publish)
   * data in the BlackBoard data descriptor array
   * -1 if genuine published data (not an alias).
   */
  int  alias_target;

} S_BB_DATADESC_T ;
```

C Enum + Typedef documentation example (small comment)

```
/**
 * BlackBoard publishable data type.
 * Any data published with @ref bb_publish, @ref bb_alias_publish
 * or @ref bb_simple_publish should be specified with its type.
 */
typedef enum {E_BB_DISCOVER=0, /*!< Discover is used by @ref
bb_subscribe when discovering data type */
      E_BB_DOUBLE=1,    /*!< An IEEE double precision floating point  */
      E_BB_FLOAT,       /*!< An IEEE simple precision floating point  */
      E_BB_INT8,        /*!< An 8bit signed integer                    */
      E_BB_INT16,       /*!< A 16bit signed integer                    */
      E_BB_INT32,       /*!< A 32bit signed integer                    */
      E_BB_INT64,       /*!< A 64bit signed integer                    */
      E_BB_UINT8,       /*!< An 8bit unsigned integer                  */
      E_BB_UINT16,      /*!< A 16bit unsigned integer                  */
      E_BB_UINT32,      /*!< A 32bit unsigned integer                  */
      E_BB_UINT64,      /*!< A 64bit unsigned integer                  */
      E_BB_CHAR,        /*!< An 8bit signed character                  */
      E_BB_UCHAR,       /*!< An 8bit unsigned character                */
      E_BB_USER         /*!< A user type of any size (should be
```

C Enum + Typedef documentation example (small comment)

```
      supplied) in @ref bb_publish */
   20. } E_BB_TYPE_T;
```

Note that if your different enumeration values needs lengthly comments you may use `/** lengthly comment */` **BEFORE** the value instead of `/*!< small comment */` after as in the previous example.

C Enum + Typedef documentation example (lengthly comment)

```
/** GLU server type */
typedef enum GLU_server_type_t
{
  /**
   * GLU is active. Means that the data are continuously produced
   * and must be read at the same pace (or faster) by the provider.
   * When GLU is active their shouldn'tr be more that one
   * GLU instance running by provider.
   * @see GLU_get_instance.
   */
  GLU_SERVER_TYPE_ACTIVE,
  /**
   * GLU is passive. Means that the data are produced only when the
   * provider ask for them (typically File Based Glu/Provider)
   */
  GLU_SERVER_TYPE_PASSIVE
} GLU_server_type_t;
```

Macro documentation example

```
/**
 * The BlackBoard version identifier.
 * Since the BlackBoard is evolving, the BlackBoard structure
 * itself may change from time to time.
 * If suche change occurs the BB_VERSION_ID is changed
 * such that @ref bb_check_version may be called in order
 * to check if the BlackBoard version used by an application
 * is compatible with the process currently trying to use
 * BlackBoard.
 */
#define BB_VERSION_ID       0x0002000
```

4.4.2.4 Functions

Every public function must be documented as in the following example. Note that even if doxygen does not force you to do it, it is recommended to indicate the intent of the function parameters (in out or in,out).

Function documentation example

```
1. /**
2.  * function to encode double
3.  * @param[in] v_double data to encode.
4.  * @param[in]  dimension of the data
5.  * @param(out] out_buf buffer to write the data
6.  * @param[in]  out_buf_size size of the buffer
7.  * @return TRUE or FALSE. TRUE = OK
8.  */
9. uint32_t TSP_data_channel_double_encoder(void* v_double,uint32_t
   dimension,  char* out_buf, uint32_t size);
```

4.4.2.5 Main and Programs

Every main program must define its own documentation group using `@defgroup <program>` which is `@ingroup <TSP_Application_subgroup>.` Unlike other doxygen documentation, this one should be written in the C file where the main program is implemented.

Main program documentation example
from src/consumers/ascii_writer/tsp_ascii_writer_main.c

```
1. /**
2.  * @defgroup TSP_AsciiWriter ASCII Writer
3.  * A TSP ascii writer consumer.
4.  * a TSP consumer which is able to output symbols values in different
   ASCII file format.
5.  * It's output may be standard output or file with other options to
   chose file format and eventual
6.  * size limit. It's main purposes is to be able to export TSP
   distributed symbols and value to
```

Main program documentation example
from src/consumers/ascii_writer/tsp_ascii_writer_main.c

```
7.  * some kind of CSV (Comma Separated Value) format in order to be
    easily post processed by
8.  * spreadsheet softwares or simpler plotting software like
9.  * <a href="http://www.gnuplot.org/">Gnuplot
    (http://www.gnuplot.org/)</a>.
10. * You may specify different file format output which essentialy change
    the header of the file.
11. *
12. * \par tsp_ascii_writer [-n] -x=\<sample_config_file\> [-o=
    \<output_filename\>] [-f=\<output file format>] [-l=\<nb sample\>] [-u
    TSP_provider URL ]
13. * \par
14. * <ul>
15. *   <li> \b -n  (optional) will check and enforce no duplicate
    symbols</li>
16. *   <li> \b -x  the file specifying the list of symbols to be
    sampled</li>
17. *   <li> \b -f  (optional) specifying the format of output file.
    Recognized file format are
18. *                <ul>
19. *                  <li> \b simple_ascii  tabulated ascii no header</li>
20. *                  <li> \b bach          tabulated ascii with BACH
    header</li>
21. *                  <li> \b macsim        tabulated ascii with MACSIM
    header</li>
22. *                </ul>
23. *                Default is \b simple_ascii.
24. *   </li>
25. *   <li> \b -o  (optional) the name of the output file. If not
    specified standard out is used</li>
26. *   <li> \b -l  (optional) the maximum number of sample to be stored
    in file. Unlimited if not specified.</li>
27. *   <li> \b -u  (optional) the TSP provider URL. If not specified
    default is localhost.</li>
28. * </ul>
29. * @ingroup TSP_Consumers
30. */
```

If the main itself is using its own library then the library used by this main program should trigger a new documentation subgroup. That group should be a `@ingroup` of the main program group and TSP libraries groups as in the following example:

Main program library documentation example
from src/consumers/ascii_writer/tsp_ascii_writer.h

```
/**
 * @defgroup TSP_AsciiWriterLib ASCII Writer Library
 * The TSP ascii writer consumer library API.
 * @ingroup TSP_AsciiWriter
 * @ingroup TSP_Libraries
 * @{
 */
```

5 TSP in Java

TSP programming in Java is possible using the 100% Java `jtsp` module. As with C implementation, TSP in Java includes:

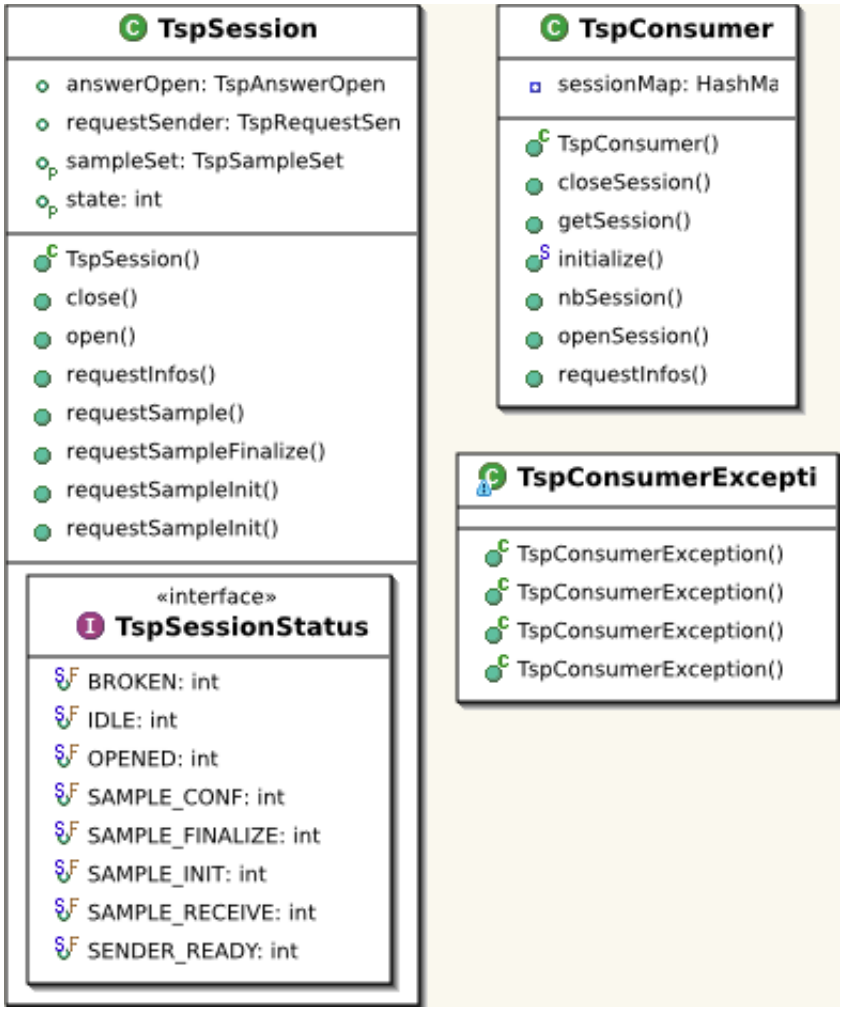
- a library which may be used to develop your own TSP consumer in Java,
- some “ready to use” TSP consumers (jstdout, jsynoptic plugins, ...)

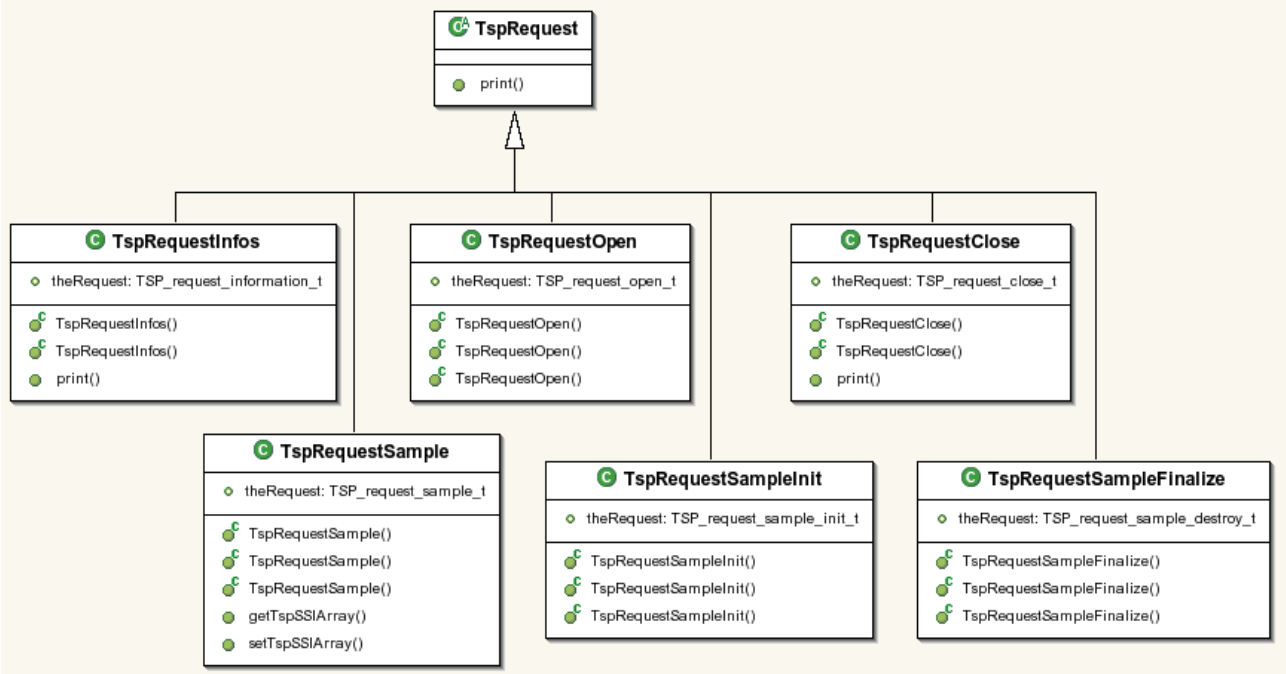
The main difference with TSP in C is that the current jtsp only implements the consumer side of the TSP protocol. In fact, at the time of the writing there is no need for provider-side programming in Java.
Another difference is the fact that there is less ready-to-use consumers because most of jtsp users embed their own jtsp-based consumer directly in their application.

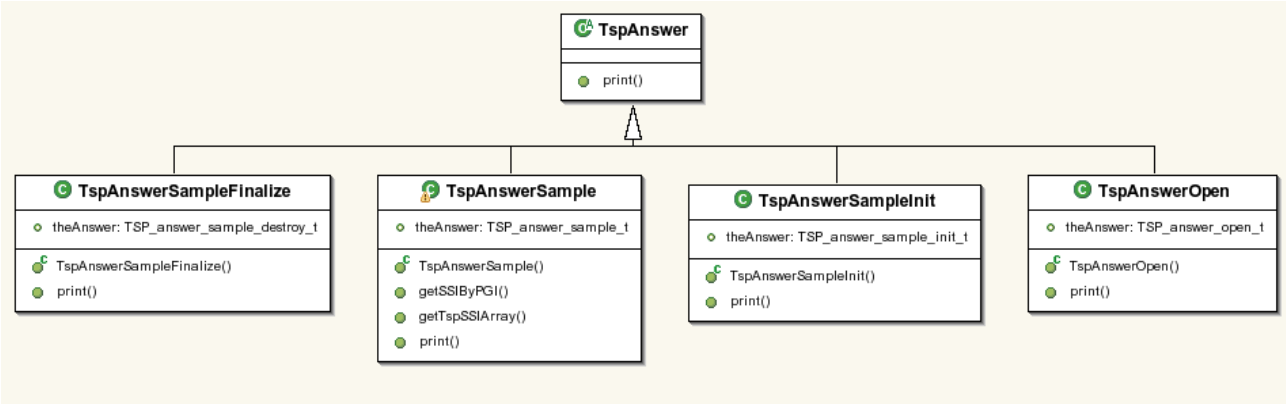
5.1 Using the JTSP API

The design of jtsp follows the previously presented design. It is even more simple to understand the Java TSP due to the object-orientation of the TSP design and the object oriented support of the Java language.

You will see hereafter a part of the JTSP class hierarchy:

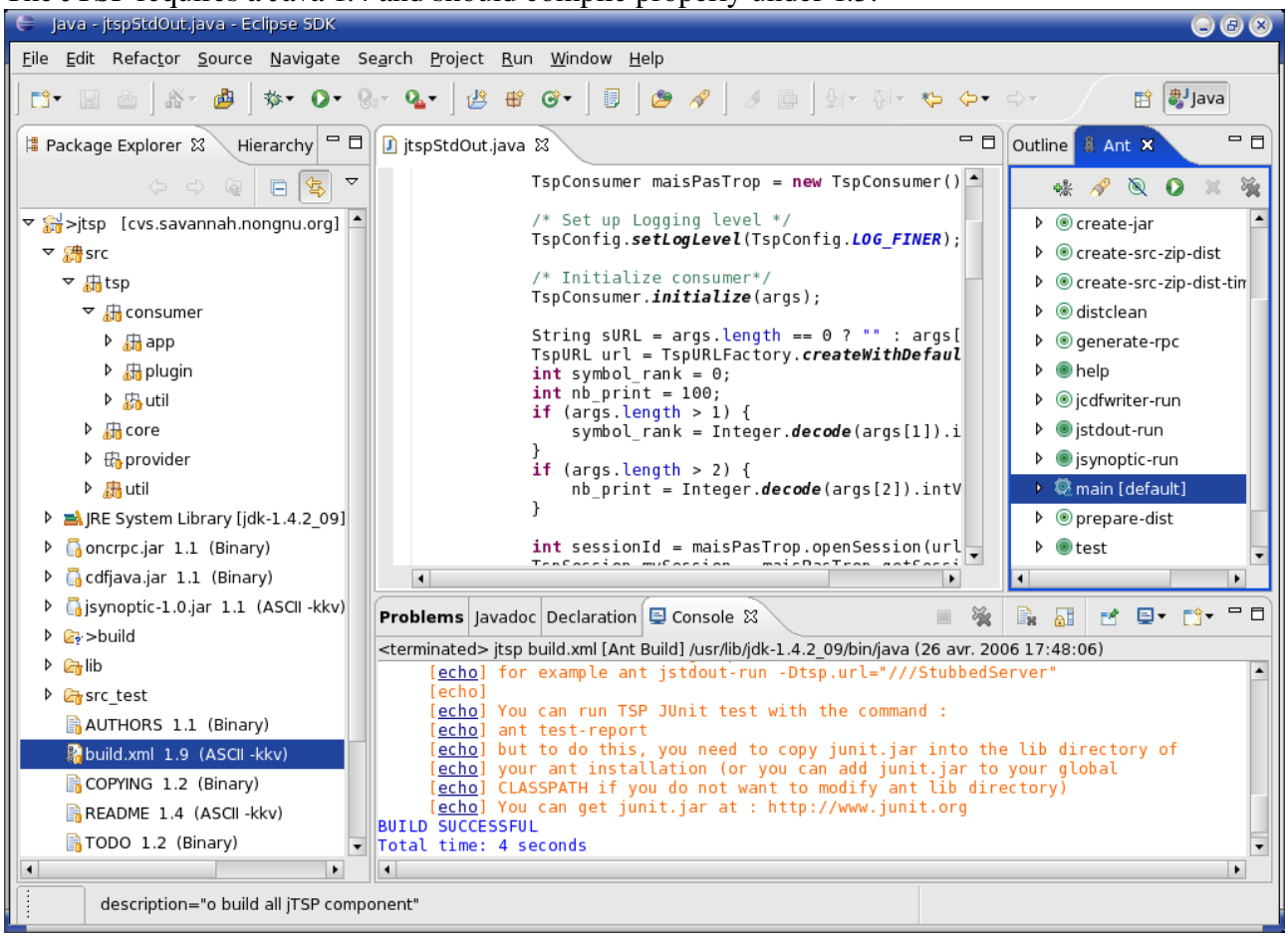

TspSession
answerOpen: TspAnswerOpen
requestSender: TspRequestSen
sampleSet: TspSampleSet
state: int
TspSession()
close()
open()
requestInfos()
requestSample()
requestSampleFinalize()
requestSampleInit()
requestSampleInit()
«interface»
TspSessionStatus
BROKEN: int
IDLE: int
OPENED: int
SAMPLE_CONF: int
SAMPLE_FINALIZE: int
SAMPLE_INIT: int
SAMPLE_RECEIVE: int
SENDER_READY: int
TspConsumer
sessionMap: HashMa
TspConsumer()
closeSession()
getSession()
initialize()
nbSession()
openSession()
requestInfos()
TspConsumerExcepti
TspConsumerException()
TspConsumerException()
TspConsumerException()
TspConsumerException()


TspRequest
print()
TspRequestInfos
theRequest: TSP_request_information_t
TspRequestInfos()
TspRequestInfos()
print()
TspRequestOpen
theRequest: TSP_request_open_t
TspRequestOpen()
TspRequestOpen()
TspRequestOpen()
TspRequestClose
theRequest: TSP_request_close_t
TspRequestClose()
TspRequestClose()
print()
TspRequestSample
theRequest: TSP_request_sample_t
TspRequestSample()
TspRequestSample()
TspRequestSample()
getTspSSlArray()
setTspSSlArray()
TspRequestSampleInit
theRequest: TSP_request_sample_init_t
TspRequestSampleInit()
TspRequestSampleInit()
TspRequestSampleInit()
TspRequestSampleFinalize
theRequest: TSP_request_sample_destroy_t
TspRequestSampleFinalize()
TspRequestSampleFinalize()
TspRequestSampleFinalize()



5.1.1 Ant and Eclipse usage

The favorite way to build JTSP is to use Ant (http://ant.apache.org/). The Ant build file for JTSP is located at the root of the jtsp source tree: `jtsp/build.xml`. Using the Ant build file you may build jtsp with or without eclipse with a simple command line and a properly installed JDK.
The JTSP requires a Java 1.4 and should compile properly under 1.5.



5.1.2 Source code documentation: javadoc

The JTSP is as usual available as source code commented using javadoc[3] comment. Thus the API documentation may be generated using the javadoc tool.

5.2 Jstdout example

Jstdout is the simpler example for using TSP in Java. This is an example of a minimal 100% Java TSP Consumer written using JTSP. You'll find hereafter a some shortened sample of the Jstdout java consumer code. You may find the entire code in `jtsp/src/tsp/consumer/app/jtspStdOut.java`.

```
1.  package tsp.consumer.app.jstdout;
2.  import tsp.core.*
3.
4.  class jtspStdOut {
5.  public static void main(String[] args) {
6.        try {
7.              /* Create a consumer object */
8.              TspConsumer maisPasTrop = new TspConsumer();
9.              /* Initialize consumer*/
10.             TspConsumer.initialize(args);
11. [... handle main arguments ...]
12.             /* Open a TSP Session */
13.             int sessionId = maisPasTrop.openSession(url);
14.             TspSession mySession = maisPasTrop.getSession(sessionId);
15.
16.             /* request Infos */
17.             TspAnswerSample asi = maisPasTrop.requestInfos(sessionId);
18.
19.             /* build request sample */
20.             TspSampleSymbols sampleSymbols = new TspSampleSymbols(asi);
21. [... include the symbol you want in request sample ... ]
22.
23.             TspRequestSample rqs = new TspRequestSample(
24.                                 mySession.answerOpen.theAnswer.version_id,
```

3 http://java.sun.com/j2se/javadoc/writingdoccomments/

```
                                mySession.answerOpen.theAnswer.channel_id,
                                fw,
                                1,
                                new TSP_sample_symbol_info_list_t());
            rqs.setTspSSIArray(sampleSymbols.toTspSSIArray());
            /* send the requestSample */
            mySession.requestSample(rqs);
            /* begin sampling */
            mySession.requestSampleInit();
            /* print 50 sample value */
[... wait for first sample then ...]
      for (int k = 0; k < nb_print; ++k) {
            if (mySession.getSampleSet().nbSample() == 0) {
                  try {Thread.sleep(100);}
                  catch (InterruptedException e) {}
            }
            sample = mySession.getSampleSet().getSample();
            System.out.println(
                  "Sample <"
                        + k
                        + "> = { time_stamp ="
                        + sample.time_stamp
                        + ",  provider_global_index ="
                        + sample.provider_global_index
                        + ", value="
                        + sample.value);
      }
      /* end sampling */
      mySession.requestSampleFinalize();
      /* close Session */
      maisPasTrop.closeSession(sessionId);

[... catch some exceptions ...]
      } /* end of main */
}
```

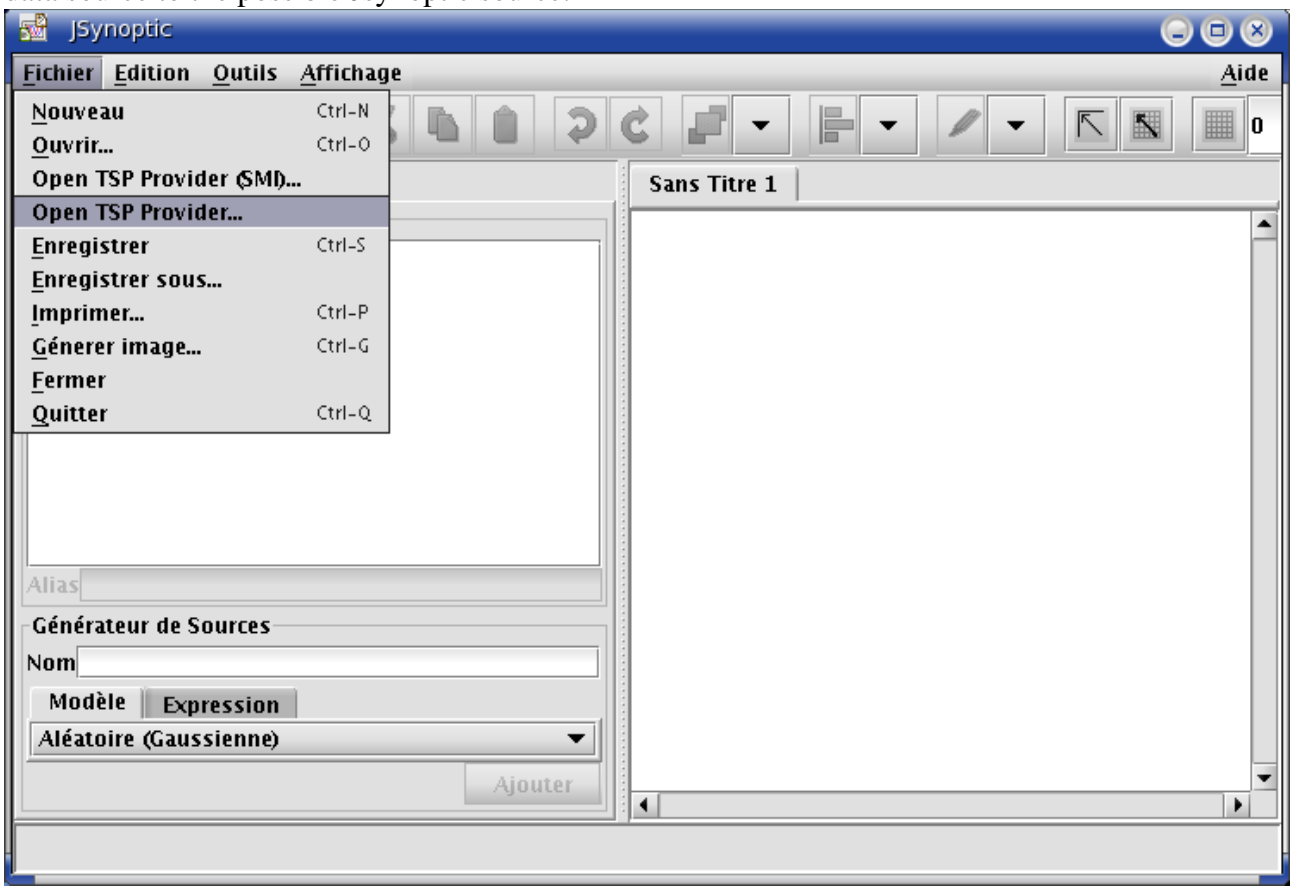
As you can see writing a TSP consumer in java is really simple thanks to the object-orientation and the simplicity of the TSP protocol and jtsp API.

5.3 JCDFWriter

The JCDFWriter is a proof of concept consumer which has not been used a lot until now. Nevertheless, the main idea is to experiment with the Nasa CDF file format (http://cdf.gsfc.nasa.gov/) which has valuable properties such as storing "sparse" value and varying values. Nowadays TSP consumers are not used for ***storing*** huge amount of data, since most of TSP users interactively and dynamically display a relatively small amount (500) of symbols picked-up in a large number (1 000 000) of ***possible*** samples. The CDF experiment was a way to prepare future use.

5.4 Jsynoptic TSP plugin

Included in the JTSP module there is a TSP Plugin for Jsynoptic, an Open Source framework for building nice synoptic. For more informations about Jsynoptic visit the Jsynoptic project at SourceForge: http://jsynoptic.sourceforge.net .
The jsynoptic plugin source is located at `jtsp/src/tsp/consumer/plugin/jsynoptic`.
You should refer to the Jsynoptic documentation for using it. The Jtsp plugin "only" adds a TSP data source to the possible Jsynoptic source.



As the time of the writing the Jsynoptic framework has some performance limitation. Jsynoptic is working well with TSP data source but it's difficult to render TSP samples at high frequency rate. Your experience may vary but trying to display TSP samples at more than 4Hz will probably leads to hieratic behavior of Jsynoptic.

The problem does not really come from JTSP plugin but more from the toolkit used to render the value which was designed for displaying static data collection and not high rate data stream. An optimization effort is necessary; JTSP Team is waiting for contribution or funding for this aspect.

6 TSP in Perl

Perl TSP binding enable the use of TSP consumer API within a Perl script. It has been done by Pierre MALLARD with SWIG interface generator (http://www.swig.org/).

The CVS module is: `perltsp`

7 TSP in Python

Python TSP binding enable the use of TSP consumer API within a Python script. It has been done by Julien BRUTUS with SWIG interface generator (http://www.swig.org/).

The CVS module is: `pytsp`

8 TSP in Ruby

The Ruby TSP binding has been contributed recently by Stéphane GALLES. It is in ***alpha*** stage since it uses a bleeding edge XML-RPC command channel. Using this development feature enables a 100% Ruby implementation.

The CVS module is: `rubytsp`

9 TSP in Tcl

The TCL TSP binding is not available as the time of the writing but it may be done as Perl or Python using SWIG interface generator.

10 TSP documentation modules

Most of TSP documentations are published under the GNU FDL (Free Documentation License). This is the case of the document you are currently reading.

10.1 TSP Specifications

The TSP specifications are exposed in the TSP User's Requirement Document or TSP URD.

This document explains the features of the TSP protocol without explaining any implementation aspect. Using TSP specifications one should be able to realize a TSP implementation from scratch.

The document may be found here: `tsp_docs/tsp_specs/TSP_URD.sxw`

10.2 TSP White paper

The TSP white paper is a short introduction to TSP design and possibilities. It may read easily and quickly in order to have an overall idea of what is TSP.

The document may be found here: `tsp_docs/tsp_whitepaper/tsp_whitepaper.xml`

The document is written in Docbook XML format and may be easily transformed into PDF, HTML or other document format.

10.3 TSP Design and programming guide

This is the guide your are currently reading.
The document may be found here:
`tsp_docs/tsp_progguide/tsp_programming_guide.odt`

The document is written in OASIS OpenDocument format using OpenOffice 2.x. You may find a ready to read/print PDF version of the document at:
`tsp_docs/tsp_progguide/tsp_programming_guide.pdf`

PDF version is generated with the built in capability of OpenOffice to generate PDF. The reference version is the OO one. You should check if PDF version is not outdated.

10.4 Blackboard Design and Programmers guide

The Blackboard is a versatile TSP utility library which deserves its own guide. The blackboard clearly ease the use of TSP. Adding a TSP provider to your application may be done in less than a day by:

1. Adding a blackboard to your application and,
2. Use the ready-to-use Blackboard TSP provider.

The Blackboard may well be useful without TSP too, since it may be used within an application as a simple multi-process publish/subscribe library.

The document may be found here: `tsp_docs/tsp_bbguide/tsp_bbguide.*`

11 TSP Applications

You will find hereafter some user documentation for working with the ready-to-use TSP applications bundled with the TSP distribution. The present documentation has some nice screens or console shots but you may find more up to date usage documentation in the Doxygen generated documentation in the **TSP Applications** section and **TSP Consumers** and/or **TSP Providers subsections.**

11.1 TSP Providers

The TSP distribution includes ready-to-use TSP Providers. If you get a binary TSP distribution or a

properly compiled TSP source kit you should have the binary executable of those TSP providers in your PATH. Unlike ready-to-use TSP consumers which may have GUI (Graphical User Interface), TSP providers are command line executable.

Some of them like bb_provider come both as a command line tool (`bb_tsp_provider`) and as a library (`libbb_tsp_provider`) usable from your favorite application.

Other are command-line only tools which may help to develop and/or debug TSP enabled applications.

11.1.1 Generic Reader

The generic reader is a TSP provider which pick its symbol definition and values from file. It is called generic_reader since it is designed to be easily extensible for reading several file format.

The TSP Generic Reader is located in `tsp/src/provider/generic_reader`.

At the time of the writing the generic reader only support one file format which is the “macsim” file format. An example of use of the generic reader is:

```
$ tsp_generic_reader -x test_macsim.res -f macsim
#=========================================================================#
# Launching <generic reader server> for generation of Symbols from a generic
file #
#=========================================================================#
TSP Provider on PID 11387 - URL #0 : <rpc://tsp_demo/GenReaderServer:0>
GLU: source file is <test_macsim.res>
GLU: format file is <macsim>
```

The corresponding ascii wtiter session is:

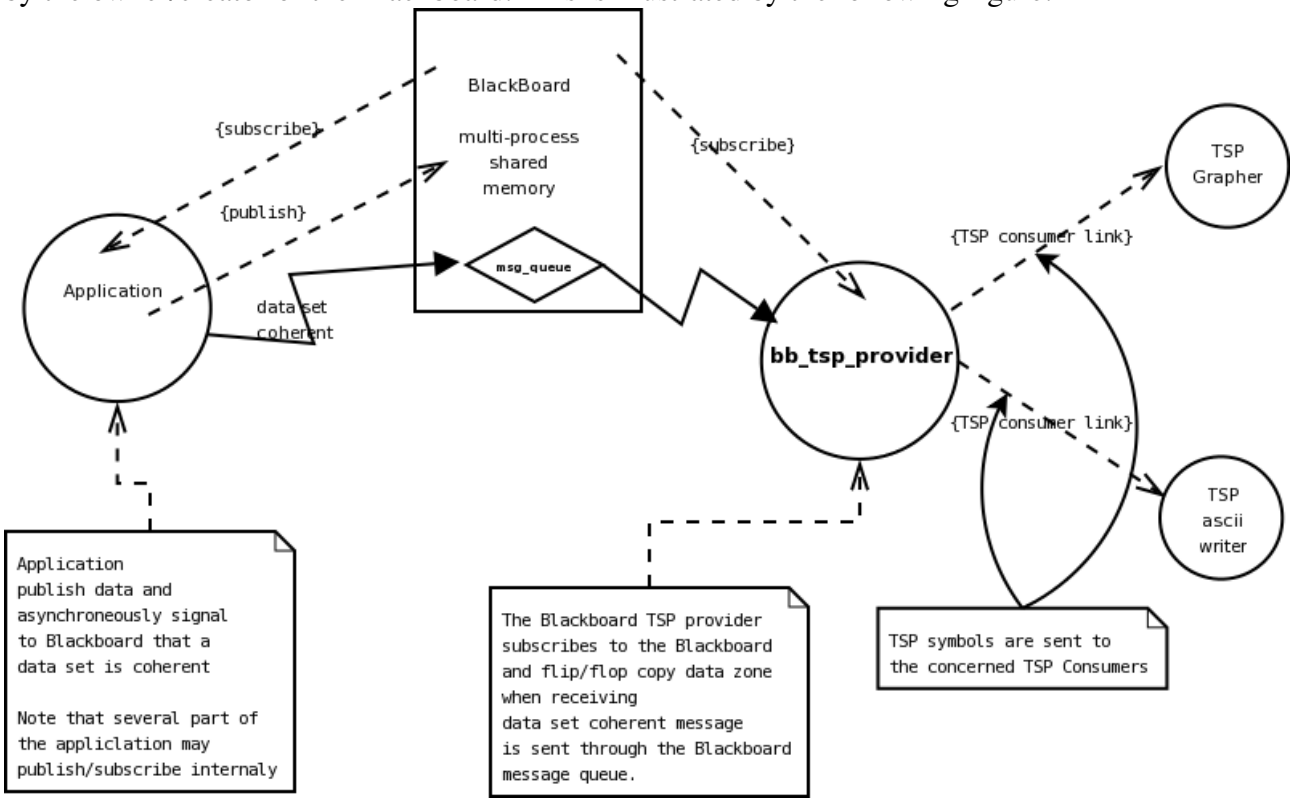
```
$ tsp_ascii_writer -x src/providers/generic_reader/test_macsim.dat
tsp_ascii_writer: sample config file is
<src/providers/generic_reader/test_macsim.dat>
tsp_ascii_writer: selected output file format is <Simple tabulated ASCII
format>
tsp_ascii_writer: Load config file...
tsp_ascii_writer: Validate symbols against provider info...
AsciiWriterLib:: Initially asking for <3> symbol(s)
AsciiWriterLib:: Enforcing same period for every symbols <begin>...
AsciiWriterLib:: Enforcing same period <done>.
AsciiWriterLib:: Finally asking for <3> symbol(s)
tsp_ascii_writer: Ascii writer running...
67.6    -0.997838725116478      0.615156387812678
[...]
68.9    -0.203604735112341      0.786401448233788
69      -0.104845336357333      0.84414139157743
tsp_ascii_writer: Ascii writer stopped...
$
```

The generic provider will serve the “whole” file from the beginning each time a new consumer is connected. Each consumer connection (i.e. TSP session) is handled by a separate PASSIVE GLU instance such that:

- each GLU instance may wait (PASSIVE GLU) for the consumer to read sample, which ensures the consumer won't lose any sample
- each GLU instance may only handle a single consumer.

11.1.2 Blackboard provider

The Blackboard provider is located in `src/providers/bb_provider` directory and is compiled to the `bb_tsp_provider` command line and `libbb_tsp_provider` library. The Blackboard provider attaches itself to an existing Blackboard, then parses symbols definition from the published data in the Blackboard. The BB provider then waits for synchronization request sent by the owner/creator of the Blackboard. This is illustrated by the following figure:



Blackboard enable application and bb_tsp_provider collaboration

Using a Blackboard is the fastest way to bring TSP to an existing application. The only thing the application has to do is to create a Blackboard and publish the data it wants to distribute. After that the application must send synchronization each time a sample set is ready.

The TSP Blackboard utility libraries are located in `src/util/libbb` (and its sub directories).

You may experiment the `tsp_bb_provider` command using the `bb_simu` example application which create a Blackboard named "**bb_simu**" with some symbols published in it.
In order to try this

1. Run the `bb_simu -s` (bb_simu with synchro blackboard)

```
$ bb_simu -s
Run with synchro ACTIVE
@Toto = 0xb7bd5144, Toto[0] = 0
@Toto = 0xb7bd5148, Toto[1] = 1
@Toto = 0xb7bd514c, Toto[2] = 2
@Titi = 0xb7bd5150, Titi = 3.141590
INFO : BB_PUBLISH DYN_0_d_qsat[4] type double
INFO : BB_PUBLISH ORBT_0_d_possat_m[3] type double
INFO : BB_PUBLISH ECLA_0_d_ecl_sol[1] type double
INFO : BB_PUBLISH ECLA_0_d_ecl_lune[1] type double
INFO : BB_PUBLISH POSA_0_d_DirSol[3] type double
INFO : BB_PUBLISH POSA_0_d_DirLun[3] type double
INFO : BB_PUBLISH Sequenceur_0_d_t_s[1] type double
Toto[0] = 0
Toto[1] = 1
Toto[2] = 2
Titi = 3.141590
Tata[0] = -1.000000
Tata[1] = 0.000001
Tata[2] = 0.500001
Tata[3] = 0.707107
Tata[4] = 0.809017
Tata[5] = 0.866026
Tata[6] = 0.900969
Tata[7] = 0.923880
Tata[8] = 0.939693
....
```

2. Run the `bb_tsp_provider` on the"bb_simu" Blackboard created by the application (bb_simu and bb_tsp_provider should run on the same host)

```
$ export STRACE_DEBUG=3
$ bb_tsp_provider bb_simu 32
   Info||tsp_provider.c##TSP_cmd_line_parser##214: No GLU stream init
provided on command line
   Info||bb_tsp_provider.c##BB_GLU_init##310: Skipping unhandled symbol type
<13> name <bb_simu_MyType_t_var>
   Info||bb_tsp_provider.c##BB_GLU_init##310: Skipping unhandled symbol type
<13> name <bb_simu_MyType_t_var.insider>
   Info||bb_tsp_provider.c##BB_GLU_init##310: Skipping unhandled symbol type
<11> name <bb_simu_astring>
```

```
   Info||tsp_datapool.c##TSP_global_datapool_init##216: No More datapool
thread
   Info||bb_utils.c##bb_logMsg##197: bb_tsp_provider::GLU_thread : Provider
thread started with <50> symbols
```

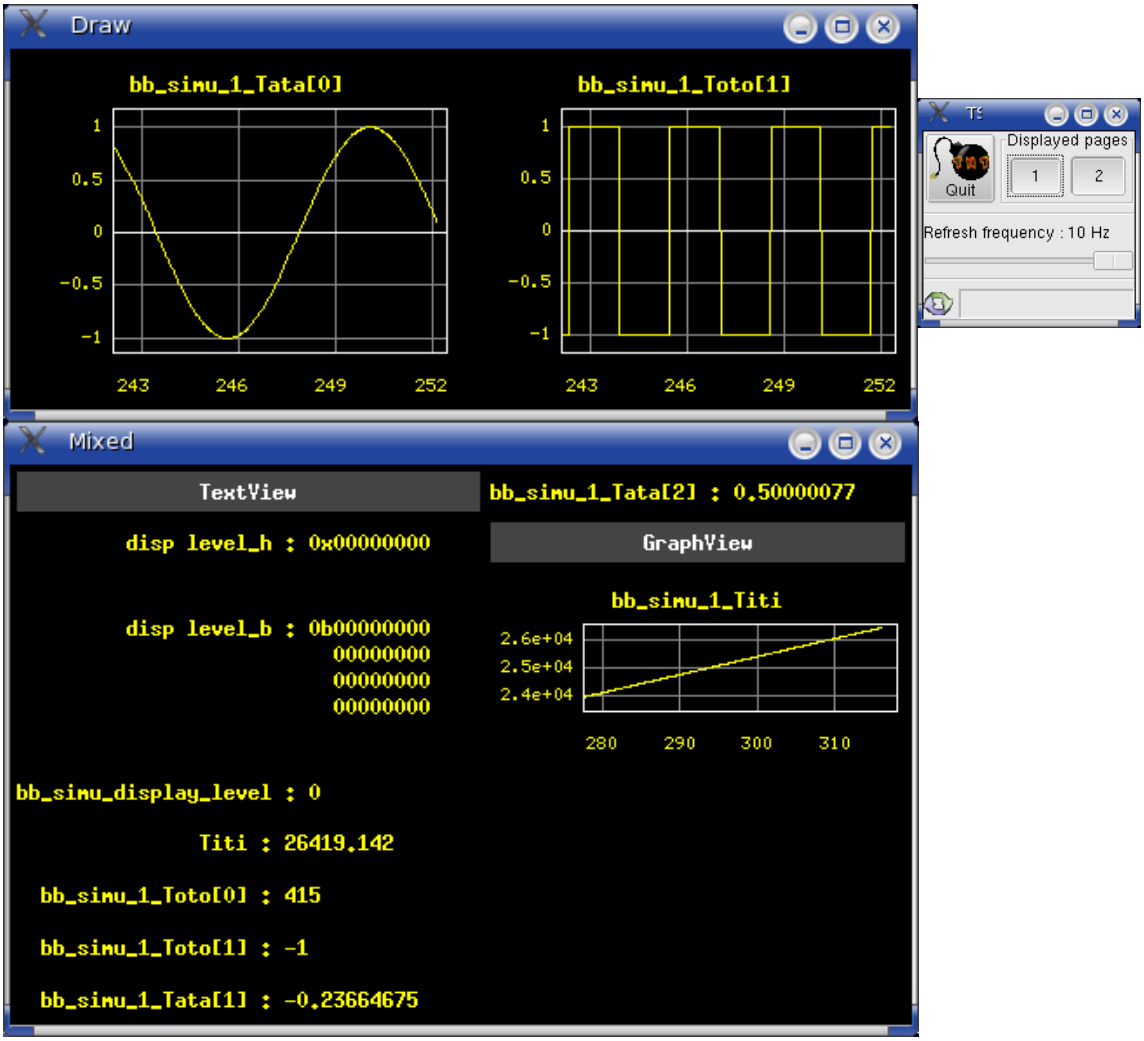
As you can see the Blackboard provider has seen 50 distributable symbols in the Blackboard. Some symbols may not be distributed using TSP since they are user defined structure (there is a way to distribute user type consult TSP Blackboard guide for those precise issues).

3. Launch a TSP consumer
   tsp_ascii_writer example

```
$ tsp_ascii_writer -x src/util/libbb/bbtools/bb_simu.dat -u rpc://tsp_demo
tsp_ascii_writer: sample config file is <src/util/libbb/bbtools/bb_simu.dat>
tsp_ascii_writer: TSP provider URL is <rpc://tsp_demo>
tsp_ascii_writer: Load config file...
tsp_ascii_writer: Validate symbols against provider info...
AsciiWriterLib:: Asking for symbol <bb_simu_1_Titi> with period <1>
AsciiWriterLib:: Asking for symbol <bb_simu_1_Tata[0]> with period <2>
AsciiWriterLib::  ---> [period forced to <1>]
AsciiWriterLib:: Asking for symbol <bb_simu_1_Tata[1]> with period <2>
AsciiWriterLib::  ---> [period forced to <1>]
AsciiWriterLib:: Checking for symbol like <unknown_var2> on provider side.
AsciiWriterLib:: Symbol <unknown_var2> not found on provider side.
tsp_ascii_writer: Ascii writer running...
 1.197514159E+04  8.744105840E-01  4.851866966E-01
 1.197714159E+04  8.849757282E-01  4.656371553E-01
 1.197914159E+04  8.951038654E-01  4.458576792E-01
 1.198114159E+04  9.047899942E-01  4.258580355E-01
 1.198314159E+04  9.140293315E-01  4.056481002E-01
 1.198514159E+04  9.228173148E-01  3.852378531E-01
 1.198714159E+04  9.311496047E-01  3.646373729E-01
 1.198914159E+04  9.390220865E-01  3.438568322E-01
 1.199114159E+04  9.464308728E-01  3.229064928E-01
tsp_ascii_writer::Captured signal<2>
tsp_ascii_writer: Ascii writer stopped...
[noularde@tsp_demo tsp]
```

tsp_gdisp example:

```
$ tsp_gdisp -x src/util/libbb/bbtools/bb_simu.xml -u rpc://tsp_demo
Loading 'src/util/libbb/bbtools/bb_simu.xml' conf file
nb_page = 2
nb_var = 13
```



11.1.3 Stubbed Server

The TSP Stub Server is located in `tsp/src/provider/stub`.

It is the simplest example of TSP provider implementation. It may be run with no argument as following:

```
$ export STRACE_DEBUG=3
$ tsp_stub_server
#=====================================================================#
# Launching <StubbedServer> for generation of 1000 Symbols at 100Hz #
#=====================================================================#
   Info||tsp_provider.c##TSP_cmd_line_parser##214: No GLU stream init provided on
command line
   Info||tsp_datapool.c##TSP_global_datapool_init##216: No More datapool thread
TSP Provider on PID 2944 - URL #0 : <rpc://tsp_demo/StubbedServer:0>
   Info||glue_stub.c##STUB_GLU_thread##114: TOP 1000 : t=0      Symbol1=0
Symbol2=0       Symbol3=0
   Info||glue_stub.c##STUB_GLU_thread##114: TOP 2000 : t=0      Symbol1=0
Symbol2=0       Symbol3=0
```

When launched the stub server produces 1000 TSP Symbols at 100Hz pseudo frequency.

The symbol of PGI=0 is named 't' and symbols with PGI ranging from 1 to 999 are named SymbolNNN, that is to say Symbol1, Symbol2, ... Symbol999.

We may check this fact using the `tsp_request_filtered_information` command (generic consumer wrapper command) as follow:

```
$ tsp_request_filtered_information -u rpc://tsp_demo/StubbedServer:0 SIMPLE
Symbol17
tsp_request_generic: TSP provider URL is <rpc://tsp_demo/StubbedServer:0>
Provider::base frequency      = 100.000000
Provider::max period          = 100000
Provider::max consumer        = 100
Provider::current consumer nb = 1
Provider <symbols list begin>
    pgi = 00000017, Symbol17
    pgi = 00000170, Symbol170
    pgi = 00000171, Symbol171
    pgi = 00000172, Symbol172
    pgi = 00000173, Symbol173
    pgi = 00000174, Symbol174
    pgi = 00000175, Symbol175
    pgi = 00000176, Symbol176
    pgi = 00000177, Symbol177
    pgi = 00000178, Symbol178
    pgi = 00000179, Symbol179
Provider <symbols list end>.
$
```

The 't' symbol represents a sort of time which is strictly increasing and other symbols are some nice plottable functions like, sinus, cosines, square functions, constant values, etc...

11.1.4 Res Reader

The `tsp_res_reader` (ResReader) is a TSP provider which provides symbols and value from a file which respects the 'Res' file format.
The TSP Res Reader is located in `tsp/src/provider/res_reader`.

The ResReader is very simple. After you launch it with the file as argument, it will provide the sample values contained in the file until the end of the file. This is a "***one shot***" provider. After the ResReader has delivered all the values it terminates itself.
You find hereafter an example of use of the ResReader. The consumer asking for symbols on this example session is the ResWriter, a TSP consumer asking the provider for all its symbols and storing the values in a file which respects the 'Res' file format.

```
$ export STRACE_DEBUG=3
$ tsp_res_reader tests/auto/file.res
#=========================================================================#
# Launching <res reader server> for generation of Symbols from .res file #
#=========================================================================#
   Info||tsp_provider.c##TSP_cmd_line_parser##134: Tsp ARG : '--tsp-stream-init-start'
   Info||tsp_provider.c##TSP_cmd_line_parser##134: Tsp ARG : '--tsp-stream-init-stop'
   Info||glue_res.c##RES_GLU_init##164: stream_init = 'tests/auto/file.res'
   Info||glue_res.c##RES_GLU_init##172: Total number of records = 1001
   Info||glue_res.c##RES_GLU_init##173: Total number of variables = 82
   Info||glue_res.c##RES_GLU_init##174: Data type = FLOAT
   Info||tsp_session.c##TSP_add_session##250: New consumer connected : channel_id=0
   Info||tsp_provider.c##TSP_provider_request_filtered_information##406: Requested
filter NONE
   Info||tsp_provider.c##TSP_provider_request_sample##447: Consumer No 0 asked for 82
symbols
   Info||tsp_group_algo.c##TSP_group_algo_get_groups_summed_size##194:
groups_summed_size is 82
   Info||glue_res.c##RES_GLU_loop##125: New record : time=0, val[0]=0
   Info||glue_res.c##RES_GLU_loop##125: New record : time=1, val[0]=0.03125
[...]
   Info||glue_res.c##RES_GLU_loop##125: New record : time=1000, val[0]=999.031
   Info||tsp_datapool.c##TSP_datapool_push_commit##112: GLU sent EOF
$
```

The corresponding ResWriter session is following.

```
$ export STRACE_DEBUG=3
$ $ tsp_res_writer -f out.res
   Info||client_res.c##main##142: Autodetect CPU : 32 bits
   Info||tsp_consumer.c##TSP_consumer_connect_url##499: Trying to connect to
<rpc://localhost/:0>
   Info||tsp_client.c##tsp_remote_open_progid##94: CONNECTED to server localhost
   Info||tsp_client.c##TSP_remote_open_server##150: Server opened : 'ResServer'
   Info||tsp_consumer.c##TSP_consumer_store_informations##178: Provider base frequency =
32.000000 Hz
   Info||client_res.c##main##222: Id=0 Sym='t'
   Info||client_res.c##main##222: Id=1 Sym='wz'
   Info||client_res.c##main##222: Id=2 Sym='wy'
   Info||client_res.c##main##222: Id=3 Sym='wx'
[...]
   Info||client_res.c##main##222: Id=80 Sym='thp_rouex'
   Info||client_res.c##main##222: Id=81 Sym='pto_zpyp'
   Info||tsp_consumer.c##TSP_consumer_request_sample##1006: Total group number = 1
   Info||client_res.c##main##286: file=out.res
   Info||tsp_consumer.c##TSP_request_provider_thread_receiver##1055: Receiver thread
started. Id=3059940272
   Info||tsp_stream_receiver.c##TSP_stream_receiver_receive##277: Received socket EOF
WarninG||tsp_data_receiver.c##TSP_data_receiver_receive##370: Unable to receive group
size and time stamp
   Info||tsp_consumer.c##TSP_request_provider_thread_receiver##1069: function
TSP_data_receiver_receive returned FALSE. End of Thread
   Info||tsp_consumer.c##TSP_consumer_read_sample##1217: Received status message
FFFFFFFF
   Info||tsp_consumer.c##TSP_consumer_read_sample##1221: status message EOF
   Info||tsp_consumer.c##TSP_consumer_end##416: End...
$
```

11.2 TSP Consumers

11.2.1 Generic Consumer

The generic consumer purpose is to have an handy command line tool to test TSP providers and to provide a reference C source code usage of the TSP Request on Consumer side.

Just like BB Tools Command Lines the generic consumer has a BusyBox-like design (http://www.busybox.net). The generic interface is;

`tsp_request_generic [generic_opts] <tsp_request> [request_opts]`
`generic_opts`

- **-u** (optional) TSP Provider URL. Default is localhost
- **-s** (optional) silent mode (may be used for silent scripting)
- **-v** (optional) verbose mode
- **-n** (optional) no newline read mode

`tsp_request`

- tsp_request_open
- tsp_request_close
- tsp_request_information
- tsp_request_filtered_information
- tsp_request_feature
- tsp_request_sample
- tsp_request_sample_init
- tsp_request_sample_destroy
- tsp_request_async_sample_write
- tsp_request_async_sample_read

And there is some predefined wrapper scripts for the 4 most used requests:

- tsp_request_information
- tsp_request_filtered_information
- tsp_request_async_sample_read
- tsp_request_async_sample_write

When used with the wrapper name the command line is:
tsp_request_<specific_req_name> [generic_opts] [specific_request_opts]

Some usages examples are explained hereafter.

11.2.1.1 tsp_request_information

The TSP request informations may be sent to show ALL the list of symbols which are offered by a provider; some minimal informations about symbols are shown.

Example, request send on bb_tsp_provider running on localhost and attached to bb_simu:

```
$ tsp_request_information -u rpc://localhost/bb_simu
tsp_request_generic: TSP provider URL is <rpc://localhost/bb_simu>
Provider::base frequency      = 10.000000
Provider::max period          = 100000
Provider::max consumer        = 100
Provider::current consumer nb = 1
Provider <symbols list begin>
    pgi = 00000000, bb_simu_display_level
    pgi = 00000001, bb_simu_int8[0]
    pgi = 00000002, bb_simu_int8[1]
    pgi = 00000003, bb_simu_uint8[0]
    pgi = 00000004, bb_simu_uint8[1]
    pgi = 00000005, bb_simu_MyType_t_var.a
    pgi = 00000006, bb_simu_MyType_t_var.d
    pgi = 00000007, bb_simu_MyType_t_var.byte
    pgi = 00000008, bb_simu_MyType_t_var.insider.ai[0]
    pgi = 00000009, bb_simu_MyType_t_var.insider.ai[1]
    pgi = 00000010, bb_simu_MyType_t_var.insider.f
    pgi = 00000011, bb_simu_1_Toto[0]
```

```
    pgi = 00000012, bb_simu_1_Toto[1]
    pgi = 00000013, bb_simu_1_Toto[2]
    pgi = 00000014, bb_simu_1_Titi
    pgi = 00000015, bb_simu_1_Tata[0]
    pgi = 00000016, bb_simu_1_Tata[1]
    pgi = 00000017, bb_simu_1_Tata[2]
    pgi = 00000018, bb_simu_1_Tata[3]
    pgi = 00000019, bb_simu_1_Tata[4]
    pgi = 00000020, bb_simu_1_Tata[5]
    pgi = 00000021, bb_simu_1_Tata[6]
    pgi = 00000022, bb_simu_1_Tata[7]
    pgi = 00000023, bb_simu_1_Tata[8]
    pgi = 00000024, bb_simu_1_HugeArray[0]
    pgi = 00000025, bb_simu_1_HugeArray[1]
    pgi = 00000026, bb_simu_1_HugeArray[2]
    pgi = 00000027, bb_simu_1_HugeArray[3]
    pgi = 00000028, bb_simu_1_HugeArray[4]
    pgi = 00000029, bb_simu_1_HugeArray[5]
    pgi = 00000030, bb_simu_1_HugeArray[6]
    pgi = 00000031, bb_simu_1_HugeArray[7]
    pgi = 00000032, bb_simu_1_HugeArray[8]
    pgi = 00000033, bb_simu_1_HugeArray[9]
    pgi = 00000034, DYN_0_d_qsat[0]
    pgi = 00000035, DYN_0_d_qsat[1]
    pgi = 00000036, DYN_0_d_qsat[2]
    pgi = 00000037, DYN_0_d_qsat[3]
    pgi = 00000038, ORBT_0_d_possat_m[0]
    pgi = 00000039, ORBT_0_d_possat_m[1]
    pgi = 00000040, ORBT_0_d_possat_m[2]
    pgi = 00000041, ECLA_0_d_ecl_sol
    pgi = 00000042, ECLA_0_d_ecl_lune
    pgi = 00000043, POSA_0_d_DirSol[0]
    pgi = 00000044, POSA_0_d_DirSol[1]
    pgi = 00000045, POSA_0_d_DirSol[2]
    pgi = 00000046, POSA_0_d_DirLun[0]
    pgi = 00000047, POSA_0_d_DirLun[1]
    pgi = 00000048, POSA_0_d_DirLun[2]
    pgi = 00000049, Sequenceur_0_d_t_s
Provider <symbols list end>.
$
```

11.2.1.2 tsp_request_filtered_information

The TSP request ***filtered*** informations is just the same as request informations with filtering capability option.

Here follows an example of filtered information request sent on TSP URL `rpc://localhost/bb_simu`, and showing symbols whose name contains '`qsat`':

```
$ tsp_request_filtered_information -u rpc://localhost/bb_simu SIMPLE qsat
tsp_request_generic: TSP provider URL is <rpc://localhost/bb_simu>
Provider::base frequency      = 10.000000
Provider::max period          = 100000
Provider::max consumer        = 100
Provider::current consumer nb = 1
Provider <symbols list begin>
    pgi = 00000034, DYN_0_d_qsat[0]
    pgi = 00000035, DYN_0_d_qsat[1]
    pgi = 00000036, DYN_0_d_qsat[2]
    pgi = 00000037, DYN_0_d_qsat[3]
Provider <symbols list end>.
$
```

11.2.1.3 tsp_request_async_sample_read

The TSP request asynchronous sample read may be used to read on symbol value. When you want to asynchronous read or write a symbol you should first gets is PGI (Provider Global Index) by using request [filtered] informations first.

Here is an example for reading the value of 'Sequenceur_0_d_t_s':

```
$ tsp_request_filtered_information  -u rpc://localhost/bb_simu SIMPLE d_t
tsp_request_generic: TSP provider URL is <rpc://localhost/bb_simu>
Provider::base frequency      = 10.000000
Provider::max period          = 100000
Provider::max consumer        = 100
Provider::current consumer nb = 1
Provider <symbols list begin>
    pgi = 00000049, Sequenceur_0_d_t_s
Provider <symbols list end>.
$ tsp_request_async_sample_read  -u rpc://localhost/bb_simu 49
tsp_request_generic: TSP provider URL is <rpc://localhost/bb_simu>
1170.690000
$ tsp_request_async_sample_read  -u rpc://localhost/bb_simu 49
tsp_request_generic: TSP provider URL is <rpc://localhost/bb_simu>
1171.420000
$
```

Note that TSP request asynchronous sample read (or write) is not a mandatory request that must be honored by all TSP Providers, if you try this on StubbedServer for example:

```
$ tsp_request_filtered_information  -u rpc://localhost/StubbedServer SIMPLE
Symbol230
tsp_request_generic: TSP provider URL is <rpc://localhost/StubbedServer>
Provider::base frequency      = 100.000000
Provider::max period          = 100000
Provider::max consumer        = 100
Provider::current consumer nb = 1
Provider <symbols list begin>
    pgi = 00000230, Symbol230
Provider <symbols list end>.
$ tsp_request_async_sample_read  -u rpc://localhost/StubbedServer 230
tsp_request_generic: TSP provider URL is <rpc://localhost/StubbedServer>
tsp_request_generic::tsp_request_async_sample_read: async read refused (or
not handled) by provider
$
```

11.2.1.4 tsp_request_async_sample_write

The TSP request asynchronous sample write may be used to write (if Provider authorized it) on symbol value. When you want to asynchronous read or write a symbol you should first gets is PGI (Provider Global Index) by using request [filtered] informations first.

Here is an example for writing the value of 'bb_simu_display_level':

```
$ tsp_request_filtered_information  -u rpc://localhost/bb_simu SIMPLE dis
tsp_request_generic: TSP provider URL is <rpc://localhost/bb_simu>
Provider::base frequency      = 10.000000
Provider::max period          = 100000
Provider::max consumer        = 100
Provider::current consumer nb = 1
Provider <symbols list begin>
    pgi = 00000000, bb_simu_display_level
Provider <symbols list end>.
$ tsp_request_async_sample_read  -u rpc://localhost/bb_simu 0
tsp_request_generic: TSP provider URL is <rpc://localhost/bb_simu>
0.000000
$ tsp_request_async_sample_write  -u rpc://localhost/bb_simu 0 45
tsp_request_generic: TSP provider URL is <rpc://localhost/bb_simu>
$ tsp_request_async_sample_read  -u rpc://localhost/bb_simu 0
tsp_request_generic: TSP provider URL is <rpc://localhost/bb_simu>
45.000000
$ tsp_request_async_sample_write  -u rpc://localhost/bb_simu 0 0
tsp_request_generic: TSP provider URL is <rpc://localhost/bb_simu>
$ tsp_request_async_sample_read  -u rpc://localhost/bb_simu 0
tsp_request_generic: TSP provider URL is <rpc://localhost/bb_simu>
0.000000
$
```

11.2.2 Res Writer

`tsp_res_writer` is a simple file writer TSP consumer. It asks for all the symbols of the provider it is connected to and stores the values in a "res" file format file until the provider gives him EOF or the user interrupts its execution (hit `Ctrl-C`). See the `tsp_res_reader` example.

11.2.3 ASCII Writer

`tsp_ascii_writer` is a TSP consumer which is able to output symbols values in different ASCII file format. Its output may be standard output or file. Several file format output may be chosen using the **-f** command line option. Its main purposes is to be able to export TSP distributed symbols and values to some kind of CSV (Comma Separated Value) format in order to be easily post processed by spreadsheet softwares or simpler plotting software like Gnuplot (http://www.gnuplot.org/).

The ASCII writer command line arguments are the following:

tsp_ascii_writer [-n] -x=<sample_config_file> [-o=<output_filename>] [-f=<output file format>] [-l=<nb sample>] [-u TSP_provider URL]

- **-n** (optional) will check and enforce no duplicate symbols
- **-x** the file specifying the list of symbols to be sampled
- **-f** (optional) specifying the format of output file. Recognized file format are
  - **simple_ascii** tabulated ascii no header
  - **bach** tabulated ascii with BACH header
  - **macsim** tabulated ascii with MACSIM header
- **-o** the name of the output file
- **-l** (optional) the maximum number of sample to be stored in file
- **-u** (optional) the TSP provider URL, default is localhost.

Example of use (default ASCII file format output):

```
$ cd $TSP_BASE
$ tsp_ascii_writer -x src/util/libbb/bbtools/bb_simu.dat -u rpc://tsp_demo
tsp_ascii_writer: sample config file is <src/util/libbb/bbtools/bb_simu.dat>
tsp_ascii_writer: TSP provider URL is <rpc://tsp_demo>
tsp_ascii_writer: selected output file format is <Simple tabulated ASCII
```

```
format>
tsp_ascii_writer: Load config file...
tsp_ascii_writer: Validate symbols against provider info...
AsciiWriterLib:: Asking for symbol <bb_simu_1_Titi> with period <1>
AsciiWriterLib:: Asking for symbol <bb_simu_1_Tata[0]> with period <2>
AsciiWriterLib::  ---> [period forced to <1>]
AsciiWriterLib:: Asking for symbol <bb_simu_1_Tata[1]> with period <2>
AsciiWriterLib::  ---> [period forced to <1>]
AsciiWriterLib:: Checking for symbol like <unknown_var2> on provider side.
AsciiWriterLib:: Symbol <unknown_var2> not found on provider side.
tsp_ascii_writer: Ascii writer running...
 5.298291416E+05 -3.895918475E-01  9.209876179E-01
 5.298311416E+05 -3.690309488E-01  9.294171070E-01
 5.298331416E+05 -3.482878202E-01  9.373876436E-01
 5.298351416E+05 -3.273727047E-01  9.448952917E-01
 5.298371416E+05 -3.062959302E-01  9.519363441E-01
 5.298391416E+05 -2.850679048E-01  9.585073237E-01
tsp_ascii_writer::Captured signal<2>
tsp_ascii_writer: Ascii writer stopped...$
$
```

Another example of use with MACSIM header

```
$ cd $TSP_BASE
$ tsp_ascii_writer -f macsim -x src/util/libbb/bbtools/bb_simu.dat -u
rpc://tsp_demo
tsp_ascii_writer: provided output file format is <macsim>
tsp_ascii_writer: selected output file format is <CNES MACSIM file format>
tsp_ascii_writer: sample config file is <src/util/libbb/bbtools/bb_simu.dat>
tsp_ascii_writer: TSP provider URL is <rpc://tsp_demo>
tsp_ascii_writer: Load config file...
tsp_ascii_writer: Validate symbols against provider info...
AsciiWriterLib:: Asking for symbol <bb_simu_1_Titi> with period <1>
AsciiWriterLib:: Asking for symbol <bb_simu_1_Tata[0]> with period <2>
AsciiWriterLib::  ---> [period forced to <1>]
AsciiWriterLib:: Asking for symbol <bb_simu_1_Tata[1]> with period <2>
AsciiWriterLib::  ---> [period forced to <1>]
AsciiWriterLib:: Checking for symbol like <unknown_var2> on provider side.
AsciiWriterLib:: Symbol <unknown_var2> not found on provider side.
tsp_ascii_writer: Ascii writer running...
bb_simu_1_Titi : 1 : double : s
bb_simu_1_Tata : 2 : double : s
 ========================================
bb_simu_1_Titi  bb_simu_1_Tata(1)       bb_simu_1_Tata(2)
 5.203611416E+05  9.370072612E-01  3.493098802E-01
 5.203631416E+05  9.445377138E-01  3.284029646E-01
 5.203651416E+05  9.516017473E-01  3.073338813E-01
 5.203671416E+05  9.581958733E-01  2.861130344E-01
 5.203691416E+05  9.643168356E-01  2.647509030E-01
 5.203711416E+05  9.699616116E-01  2.432580358E-01
tsp_ascii_writer::Captured signal<2>
tsp_ascii_writer: Ascii writer stopped...
$
```

The TSP provider used is `bb_tsp_provider` attached to the `bb_simu` pseudo simulator. The `bb_simu.dat` ASCII Writer configuration file is the following:

```
#a comment
##another one
##ccc
##ddd
bb_simu_1_Titi           1
#unknown_var1             15
# tiot
#bb_simu_1_Toto          1
bb_simu_1_Tata[0]        2
bb_simu_1_Tata[1]        2
unknown_var2             1
```

11.2.4 GDisp

`tsp_gdisp` is the first generation GUI TSP consumer. It's written using GTK+1.2. It has limited capabilities but it is small and very efficient at drawing a huge amount of TSP symbols. It has a simple XML configuration file which may be used to described the TSP Symbols you want to draw or view. The `tsp_gdisp` command line is the following:

tsp_gdisp [-u TSP_provider URL] -x config.xml

Example of use of TSP GDisp on a StubbedServer running on localhost:

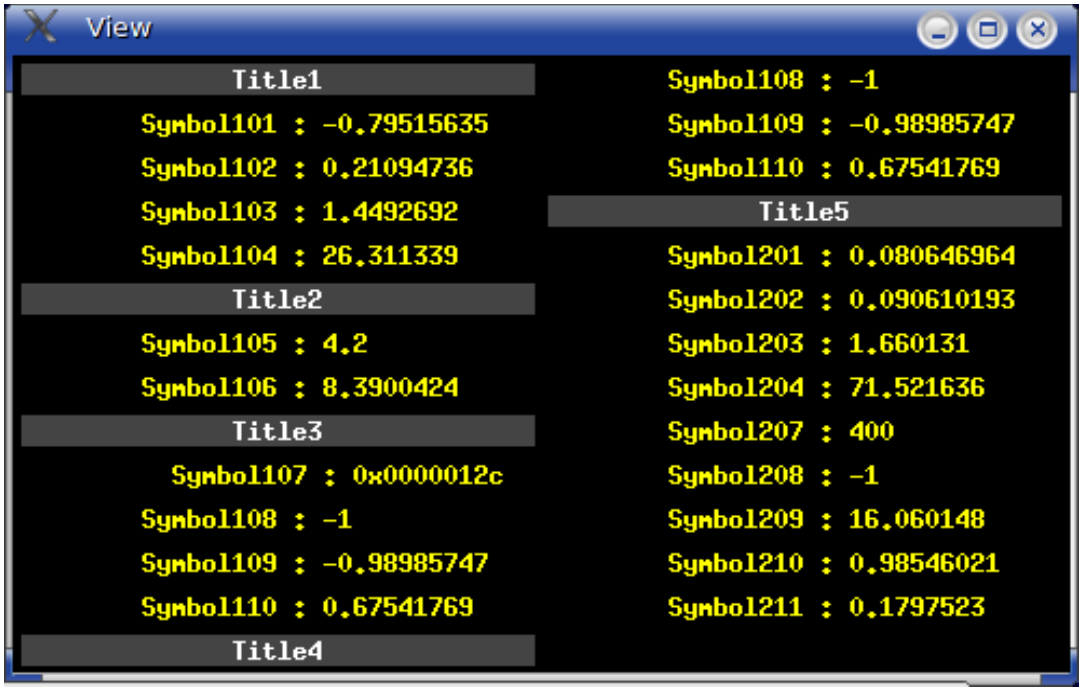
Launch the `tsp_stubbed_server`:

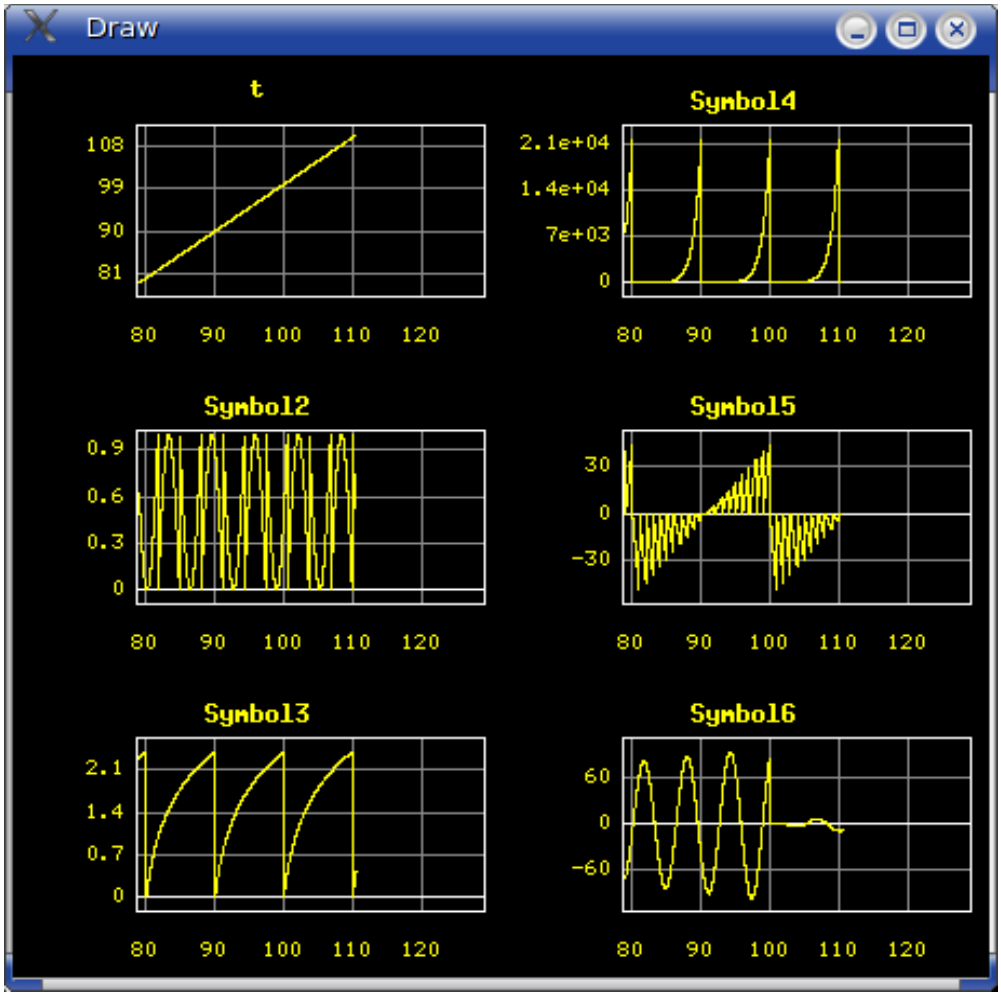
```
$ cd $TSP_BASE
$ tsp_stub_server
#=====================================================================#
# Launching <StubbedServer> for generation of 1000 Symbols at 100Hz #
#=====================================================================#
TSP Provider on PID 7160 - URL #0 : <rpc://tsp_demo/StubbedServer:0>
```

Launch `tsp_gdisp`

```
$ cd $TSP_BASE
$ tsp_gdisp -x src/consumers/gdisp/sexy.xml
Loading 'src/consumers/gdisp/sexy.xml' conf file
nb_page = 3
nb_var = 41
```

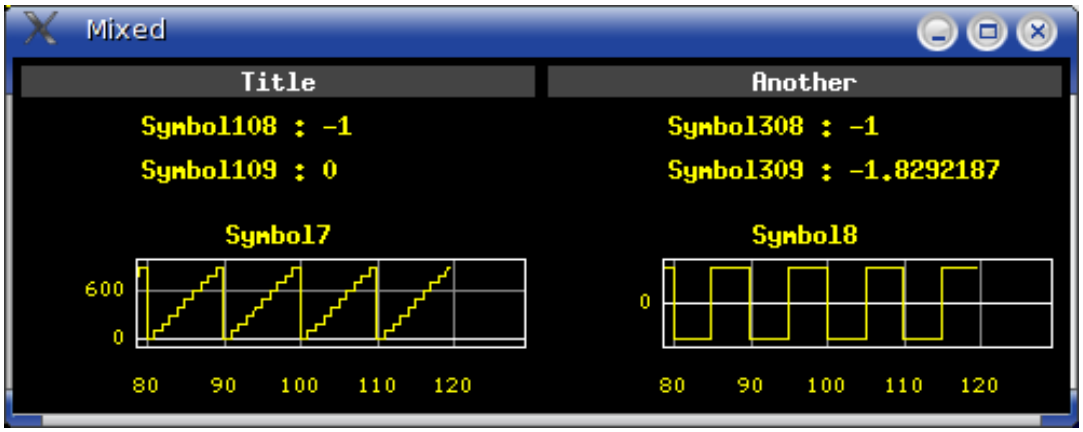
`tsp_gdisp` may display TSP sample symbol as “View” which is textual display

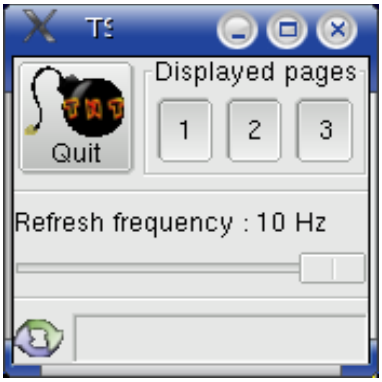




Or “Draw” which is plot display:

Every set of `tsp_gdisp` display is considered as a “page” which may mix “View” and “Draw”:





The page may be hidden or displayed using the `tsp_gdisp` Control Panel;

The corresponding “sexy.xml” XML configuration file is:

```
<?xml version="1.0"?>
<page_config display_frequency="10.0" period="1"  widget="draw"
visible="true" no_border="false" rows="3" duration="50.0">

  <page title="Draw" x="50" y="50" width="450" height="440" rows="3" >
     <variable name="t" type="DOUBLE"   />
     <variable name="Symbol2" type="DOUBLE"   />
     <variable name="Symbol3" type="DOUBLE"   />
     <variable name="Symbol4" type="DOUBLE"   />
     <variable name="Symbol5" type="DOUBLE"   />
     <variable name="Symbol6" type="DOUBLE"   />
  </page>

  <page title="View" x="510" y="50" width="0" height="0" widget="view"
rows="14">
     <variable name="Title1" type="TITLE"   />
     <variable name="Symbol101" type="DOUBLE"   />
     <variable name="Symbol102" type="DOUBLE"   />
     <variable name="Symbol103" type="DOUBLE"   />
     <variable name="Symbol104" type="DOUBLE"   />
     <variable name="Title2" type="TITLE"   />
     <variable name="Symbol105" type="DOUBLE"   />
     <variable name="Symbol106" type="DOUBLE"   />
     <variable name="Title3" type="TITLE"   />
     <variable name="Symbol107" type="HEXA"   />
     <variable name="Symbol108" type="DOUBLE"   />
     <variable name="Symbol109" type="DOUBLE"   />
     <variable name="Symbol110" type="DOUBLE"   />

     <variable name="Title4" type="TITLE"   />
     <variable name="Symbol108" type="DOUBLE"   />
     <variable name="Symbol109" type="DOUBLE"   />
     <variable name="Symbol110" type="DOUBLE"   />
     <variable name="Title5" type="TITLE"   />
     <variable name="Symbol201" type="DOUBLE"   />
```

```
    <variable name="Symbol202" type="DOUBLE"   />
    <variable name="Symbol203" type="DOUBLE"   />
    <variable name="Symbol204" type="DOUBLE"   />
    <variable name="Symbol207" type="DOUBLE"   />
    <variable name="Symbol208" type="DOUBLE"   />
    <variable name="Symbol209" type="DOUBLE"   />
    <variable name="Symbol210" type="DOUBLE"   />
    <variable name="Symbol211" type="DOUBLE"   />
  </page>

  <page title="Mixed" x="510" y="330" width="0" height="0" widget="view"
rows="4" >
    <variable name="Title" type="TITLE"   />
    <variable name="Symbol108" type="DOUBLE"   />
    <variable name="Symbol109" type="DOUBLE"   />
    <variable name="Symbol7" type="DOUBLE" widget="draw"  />
    <variable name="Another" type="TITLE"   />
    <variable name="Symbol308" type="DOUBLE"   />
    <variable name="Symbol309" type="DOUBLE"   />
    <variable name="Symbol8" type="DOUBLE" widget="draw"  />
  </page>
 </page_config>
```

The configuration file must have `<page_config>` root node with any number of `<page>` sub nodes. Each page has a set of `<variable>` sub-node specifying the name and the displayed type of the variable.

GDisp may only be connected to one provider at one time. If you want to display different symbols coming from different providers you have to launch several `tsp_gdisp` and specify the appropriate `-u <TSP Provider URL>` option.

Every sample symbols specified in the configuration file will be asked by `tsp_gdisp` in a single TPS Request Sample.

11.2.5 GDisp+

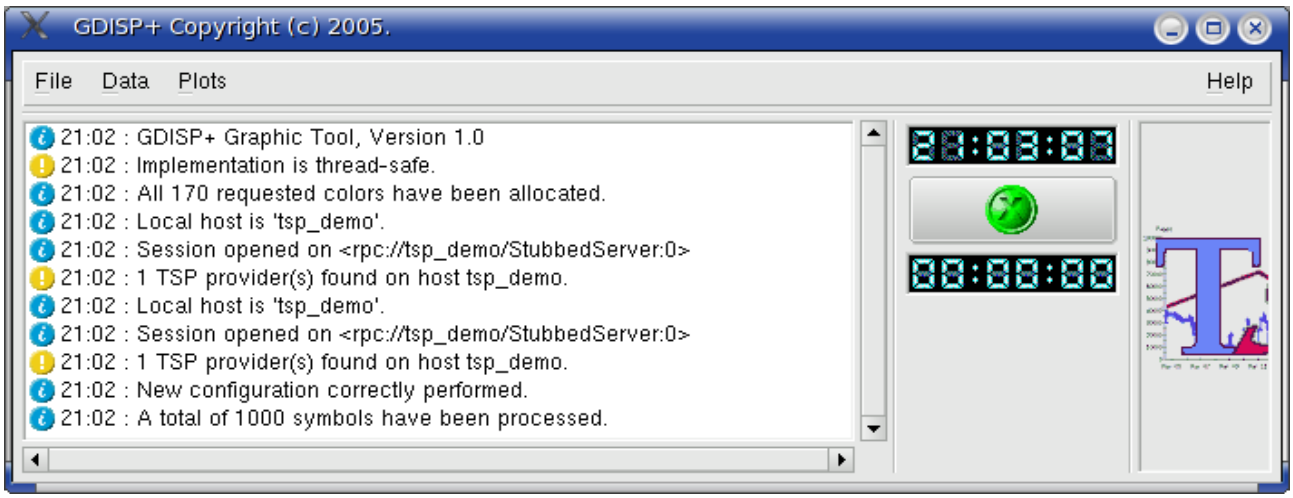
`tsp_gdisp+` is the second generation GUI TSP consumer which was designed to improve then replace `tsp_gdisp`. It's still written in GTK+1.2 and will be ported to GTK 2.x as soon as possible (contribution are welcomed). tsp_gdisp+ has a clean graphical kernel design with PlugIns architecture. It may load/save its configuration in an XML file, connect to several TSP provider at the same time.

The tsp_gdisp+ command line is:

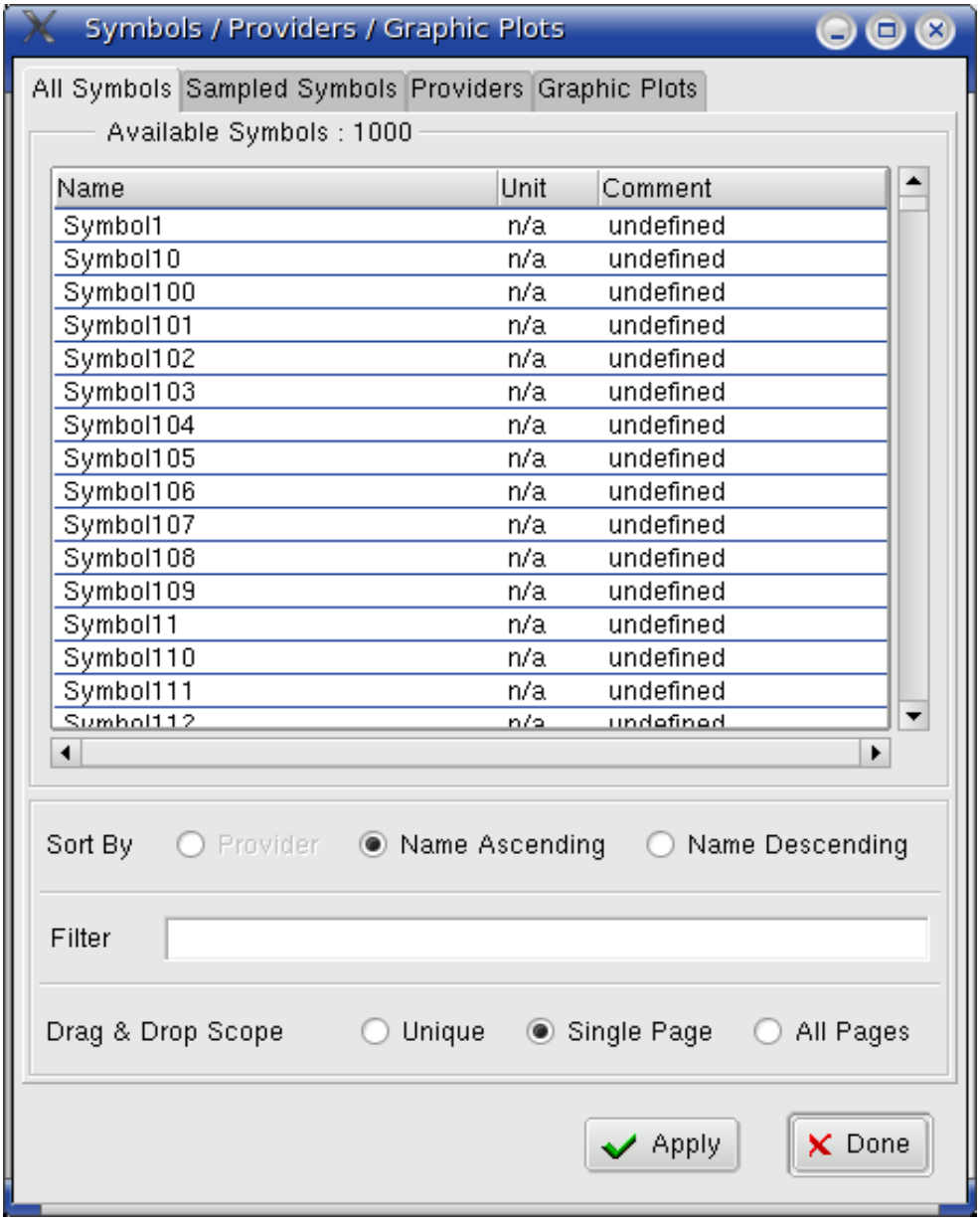
`tsp_gdisp+` [-u TSPurl] [-h host] [-x config.xml]

- `TSPurl` the TSP URL
- `host` the hostname or IP address
- `config.xml` the GDisp+ xml configuration file

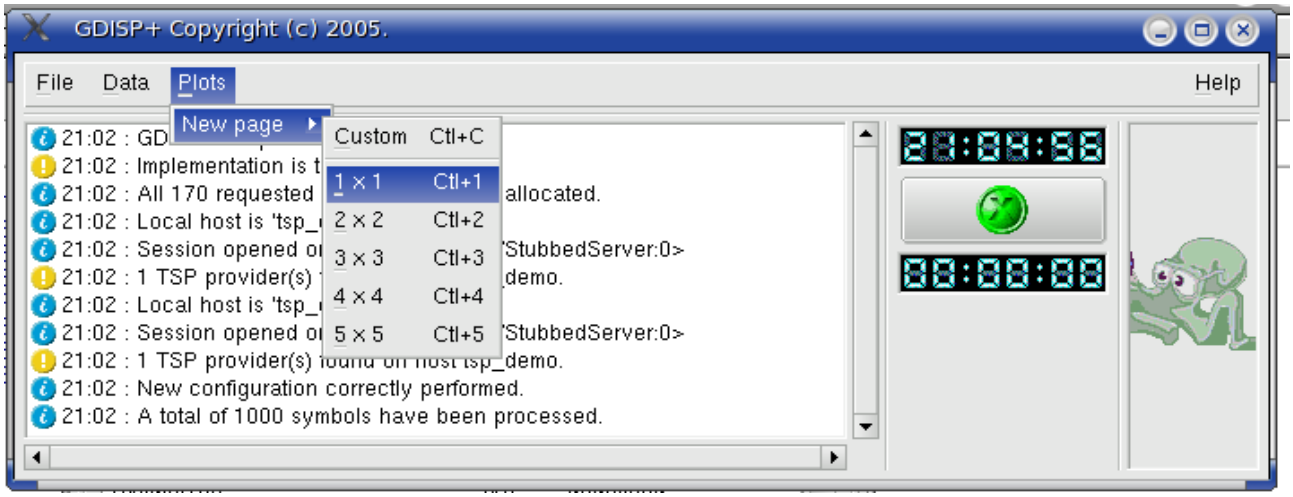
Launching tsp_gdisp+ on a machine where a TSP provider is running will bring the GDisp+ main board:



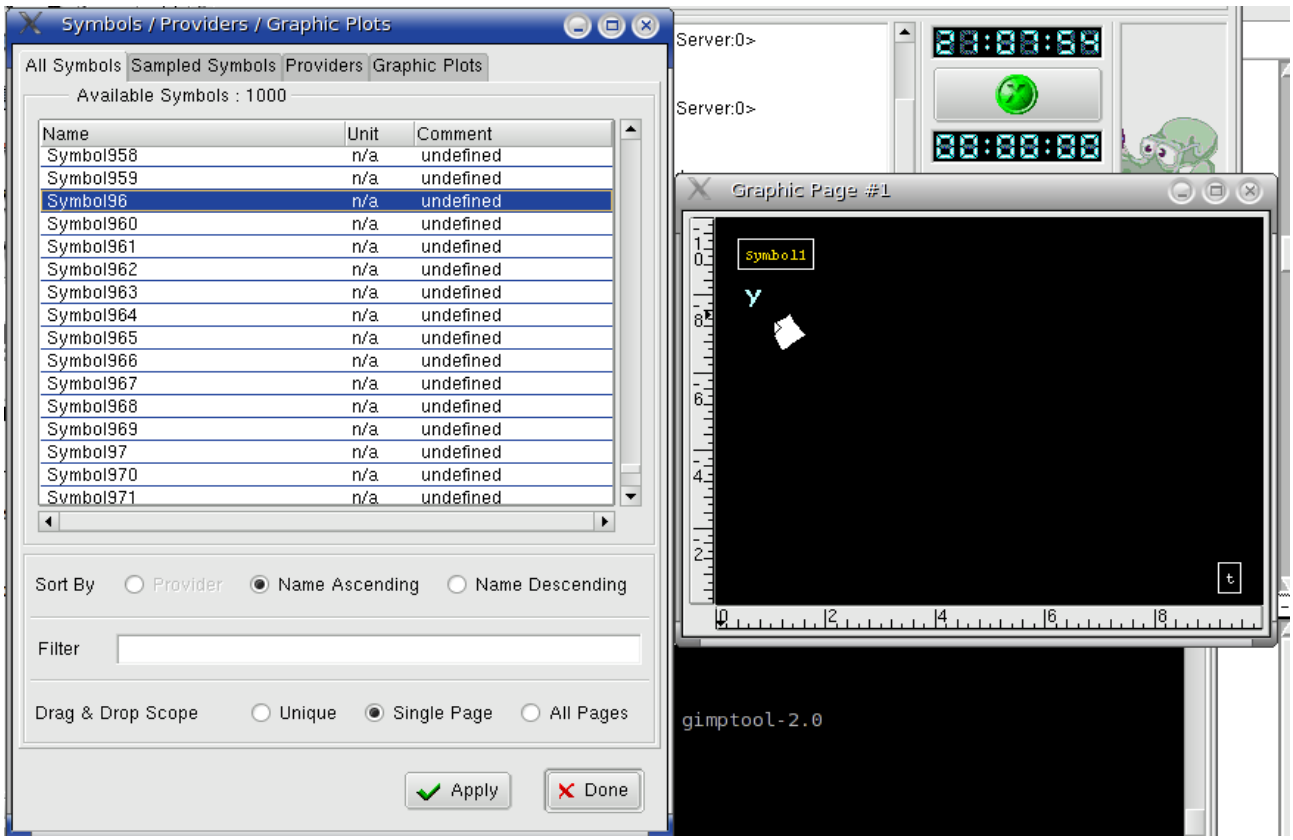
When you click on the Data Menu you obtain the list of symbol found by GDisp+:



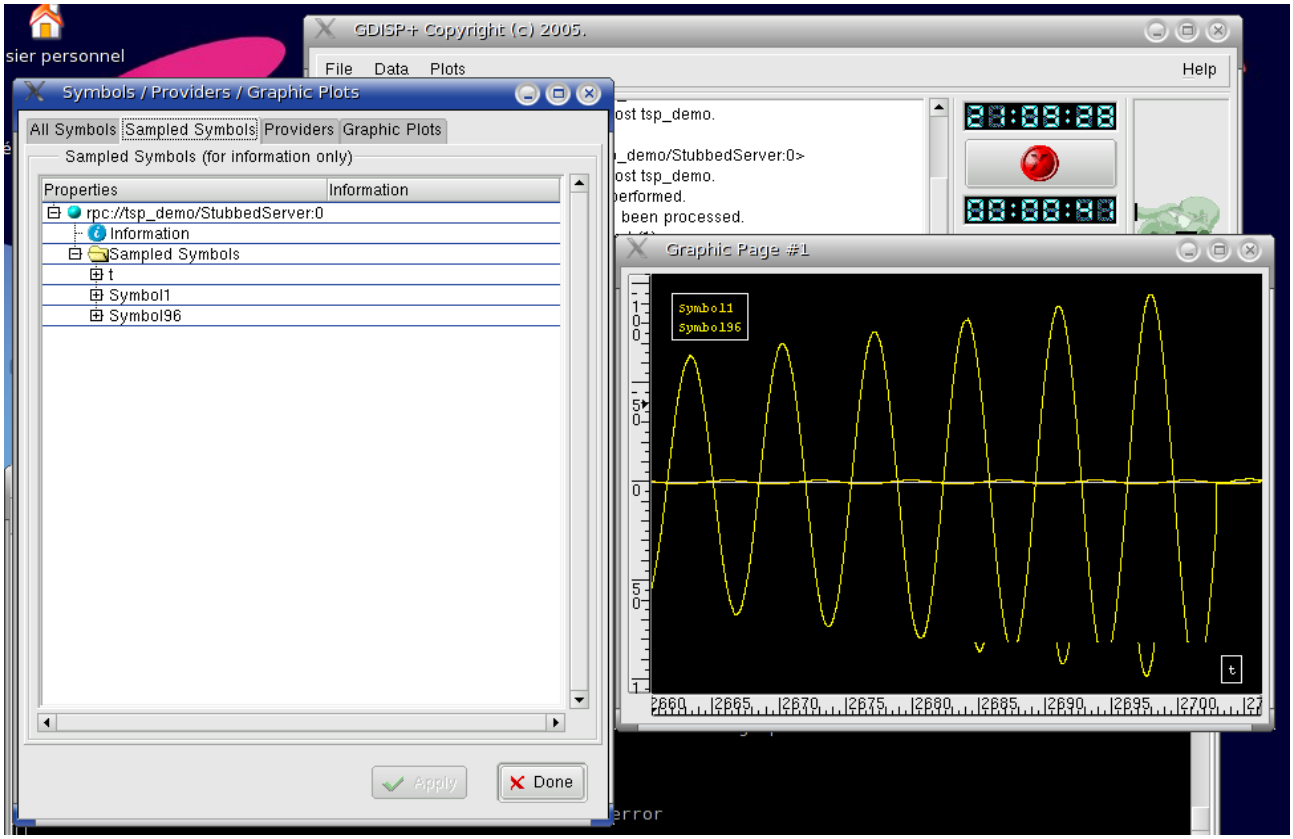
Using the Plot drop down menu you may build a new page:



After you have chosen the kind of page, you get a new graphical page on which you may drag'n'drop symbols:



Then you may click on the start sampling button of the GDisp+ main board (Green Cross) to begin the sampling process which will update [all] the graphical page[s]:


GDISP+ Copyright (c) 2005.
File Data Plots Help
Symbols / Providers / Graphic Plots
All Symbols Sampled Symbols Providers Graphic Plots
Sampled Symbols (for information only)
Properties Information
rpc://tsp_demo/StubbedServer:0
Information
Sampled Symbols
t
Symbol1
Symbol96
Apply
Done
Graphic Page #1
Symbol1
Symbol96

12 Developer Handbook

12.1 Common Problems (FAQ)

Q/A	Description
Q1	*My favorite provider refuse to start-up after a busy debugging/crashing period, what's up?*
A1	You may have exhausted the rpc id on your provider machine, stop all providers process on this machine and launch `tsp_rpc_cleanup`
Q2	*My consumer can connect to the provider but does not receive any symbols values?*
A2	Most of the time this is a name resolution problem see bug #14770 and bug #14783 on Savannah. Check if your provider may resolve its own name used by the consumer in the TSP URL properly. Check if your consumer knows how to reach provider by name and not by @IP.

12.2 Savannah Access

If you ever (want to) become a registered TSP at Savannah you'll find hereafter some useful recipes for clean TSP development process with Savannah.

If you want to use access Savannah SSH authenticated CVS access and you are behind an https proxy, you may use https tunneling to access the savannah cvs machine. Proceed as follow:

1. install a Proxy Tunnel command like http://proxytunnel.sourceforge.net , (*you don't need to have administrator right to install or use such tool*).
2. put those lines in your `~/.ssh/config`:

```
Host cvs.savannah.nongnu.org
  ProxyCommand proxytunnel -g <proxyserver> -G <proxyport> -d %h -D 443
```

if your proxy server requires username/password, use:

```
proxytunnel -g <proxyserver> -G <proxyport> -u <username> -s <password> -d %h -D 443
```

For example imagine you have a Squid proxy without authentication running on port 3128 whose @IP is 192.168.0.1 (private network address):

```
proxytunnel -g 192.168.0.1 -G 3128 -d %h -D 443
```

Note that if your proxy accepts to establish connections on port 22 (genuine SSH port) it is better to use port 22 (ssh) connection instead of 443 (https) since the Savannah CVS server may not answer to SSH connection on 443 (this feature is active at the time of the writing but may not be supported by Savannah in the future).

13 Support

13.1 Open Source Model Support

TSP is an Open Source project (LGPL license http://www.gnu.org/copyleft/lesser.html) and as such does not offer any guaranteed support to its users. Nevertheless the Open Source community using TSP is active and will be proud to provide you with their knowledge of the TSP on the TSP mailing lists: https://savannah.nongnu.org/mail/?group=tsp .

The favorite process is the following:

1. Check that your question has not already been answered on the list by browsing/searching the mailing list archives,
2. Subscribe to the mailing list, this is not mandatory but eases question/answer process since non-member cannot post non-moderated message and most people answer on the list and not to the sender,
3. Send your question to the list,
4. Wait for answers which will come through the list,
5. Unsubscribe if you are done with your TSP questions.

Using this process facilitates the communication between TSP actors and due to the mailing list archives , anyone coming after may browse/search the mailing archives for similar questions.

13.2 Professional Support

If your project needs professional support for TSP, some companies may provide you specific contract for this. Just ask for professional support on the TSP mailing list and any professional TSP stakeholder will come back to you **in private** with his/her proposal.

14 GNU Free Documentation License

see http://www.gnu.org/copyleft/fdl.html

GNU Free Documentation License
Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with

the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgments" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgments and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for

publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgments", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the

terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgments", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.