GALAXY
SEMICONDUCTOR INTELLIGENCE

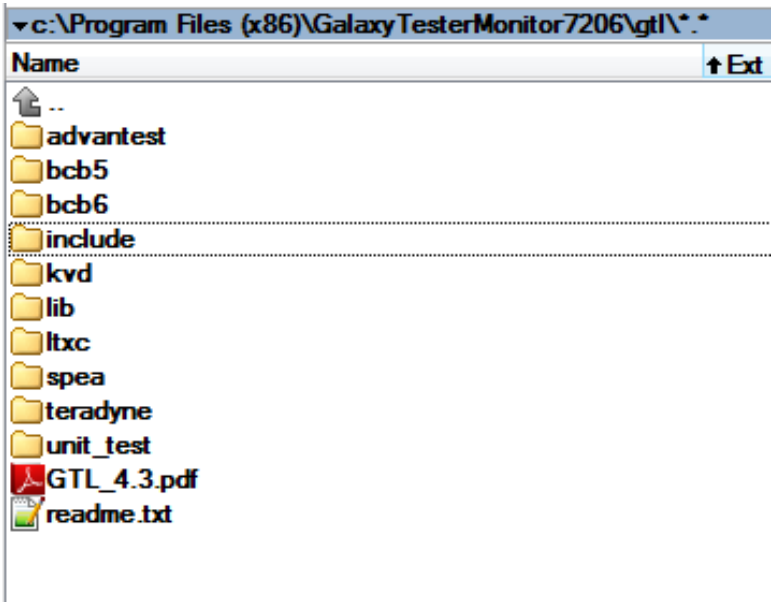
GTL
(GalaxySemi Tester Library)
version 6.0

Last update : 2015 Mar 10

Where are all the files ?

GalaxySemi currently does not provide a special package for the GTL but includes all the GTL related files into the GTM package, into the subdirectory:

(your GTM installation folder)/gtl

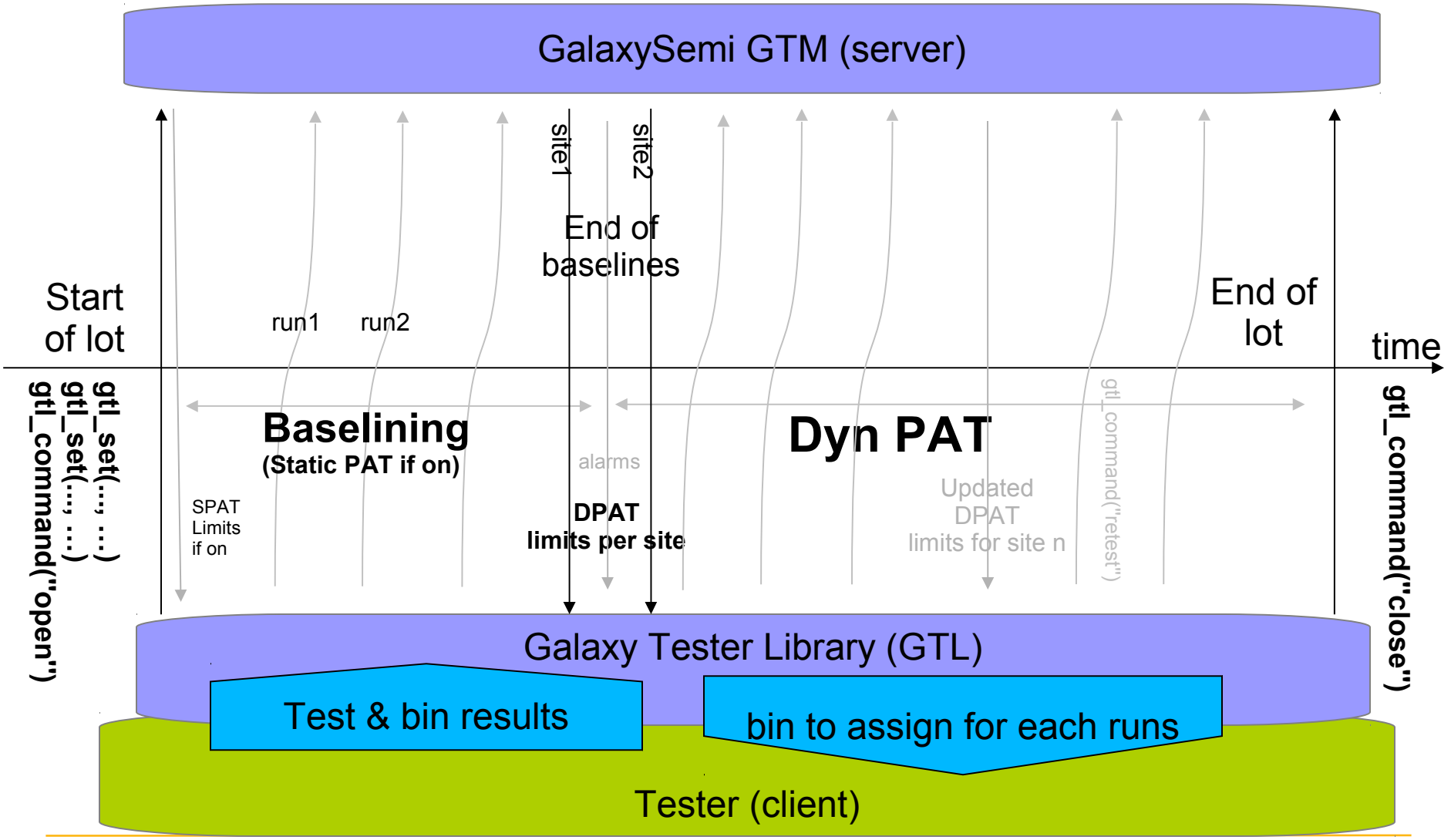


About the different builds of GTL

Here is the current official list of different builds of the GTL :

- wg1: for win32 based testers: ASL1k&3k, Magnum, ...
- wg2: for win64 based testers: Teradyne Eagle, ...
- GTL-IGXL: wrapper version for MicrosoftExcel/VisualBasic/Teradyne IGXL: available in (gtm install folder)/gtl/teradyne/igxl
- GTL enVision : wrapper version for all LTXC X-Series enVision family testers : MX, LX, EX, Fusion, ... available in (gtm install folder)/gtl/ltxc/envision
- lg1 : for Linux 32b based testers : Image,...
- lg2&3 : for Linux 64b based testers: LTXC Unison family (PAx-Dragon, …), ...

Global overview for real time PAT



GALAXY SEMICONDUCTOR INTELLIGENCE

General flow overview

time

```
gtl_set(tester_conf, …);
gtl_set(recipe_file, …);
gtl_set(max_number_of_sites, …);
...
Status=gtl_command("open");
```

For each run

```
    Status=gtl_command("beginjob");
    For all messages : gtl_pop_last_message(...);
```

For each test & site

```
        Status=gtl_test(site_no, test_no, test_name, test_result);
        Status=gtl_mptest(site_no, test_no, test_name, test_results, pin_indexes);
        Status=gtl_binning(
                site_no, original_hbin, original_sbin,
                &hbin_to_assign, &sbin_to_assign, part id);

    Status=gtl_command("endjob");   // end of run


Status=gtl_command("close")
```

GTL interfaces

Is GTL ABI (Application Binary Interface) compatible?

Yes, GTL is offering a standard ABI interface in order to be used by any system able to open binary libraries. Consequently, GTL can be used by any environment able to open a standard ABI library (VisualBasic, Ch, JNI, ...).

http://en.wikipedia.org/wiki/Application_binary_interface

Does GTL have a C or C++ API ?

GTL has a 100% C API that can be used either in a C or in a C++ test program or hybrid C/C++.

List of GTL functions

GTL provides a minimalist API with only 6 functions.
The full list and description is in gtl_core.h:

gtl_core.h <Select Symbol>

```
/* PUBLIC Functions                                                    */
/*
    gtl_set(key, value):
        This function allows to set some variables of the GTL. Examples:
        - logfile_name : the name of the log file if desired
        - node_name : must be set before init

    gtl_get(key, value):
        This function allows to retrieve the value of a variable of the GTL.
        The keys are the same as the set function even if some variables are read-only i.e. not allowed to be set.
        Warning: The value string must be allocated by the user and large enough to receive the value of the variable:
        example: char MyValue[2048]=""; gtl_get(GTL_KEY_XXX, MyValue);


    gtl_command (work in progress):
        Some arguments can be passed to the function to interact with the GTL allowing dynamic toggling of many features,
        as well as forcing actions (eg: reset baseline learning phase, enable/disable datalog, etc):
        - all commands as defined in all GTL_COMMAND_XXXXXX defines
        -gtl_debugon : sets the debug mode; advanced trace and information details are reported
        -gtl_debugoff : disables debug mode
        -gtl_quieton : Set 'Quiet' mode, so no messages reported to console
        -gtl_quietoff : Disable 'Quiet' mode, display all messages to console
        -gtl_info : report full status info (PAT running mode, status, etc)
        -gtl_help : report all available options and their meaning to the console
        -gtl_status : report status mode (eg: running baseline, PAT disabled, etc . . .)
        -gtl_test <test number> : display PAT settings for given test#
        -gtl_testlist <test list> : display PAT settings for given tests

    gtl_test(site_no, test_number, test_name, test_result):
        This function set the measurement value to/inside the GTL buffer for the specified test on specified site.
        It should be called for each test under PAT.
        If called several times in the same run for the same test number/name, the previous result is overwritten.
        It can be called for non PAT test but then the GTL will just ignore the call,
            not using unusefull memory nor sending unusefull packets over the network but these tests.
        This function in any cases will have to search for test index in the current tests list
            to be able to write testresults into the internal results buffer. This search has a complexity (big O) of n.
        Does not send results to server immediately (will be done later) and does not do any network tasks.
        Could fail if:
        - beginjob not call prior (GTL_CORE_ERR_BEGINJOB_NOT_EXEC)
        - lib not initialized (GTL_CORE_ERR_LIB_NOT_INITIALIZED)
        - test does not exist in the test list (GTL_CORE_ERR_INVALID_TEST)
        If fail, GTL does not change state, it will just continue as usual.
        Returns usual status code.

    gtl_mptest(site_no, test_number, test_name, test_results, pin_indexes, nb_of_pins)
        This function set the measurement values to/inside the GTL buffer for the specified test on specified site
        and given pin indexes (as given in the recipe, usually starting from 0).
```

How stable is the GTL interface ?

From 7.1, GalaxySemi has reviewed and simplified the interface in order to have a stable one allowing insertion of new features at the same time.

How ? Thanks to 2 main “design patterns”:

- Most options/parameters are now get/setable through get/set :

gtl_set('my 1st option', 'value1'); gtl_set('my 2nd option', 'value2');

gtl_get("secret_variable", result); ...

- Most commands are now executed through gtl_command('my command') :

example: gtl_command('open'); gtl_command('begin_job'); gtl_command("close"); .....

any new commands can be added without changing the interface, just a new command key :

example: gtl_command('do_this'); gtl_command("http_get"), gtl_command("send_email"); gtl_command("http_post"); gtl_command("ftp_get");...

GTL status code

All GTL functions (whatever the platform) return a status code.

Examples:

- GTL_CORE_ERR_OKAY (0) : no error.

- GTL_CORE_ERR_CONF (2) : GTL failed to read given config file

- GTL_CORE_ERR_CRRF (6): GTL cannot read recipe file

…

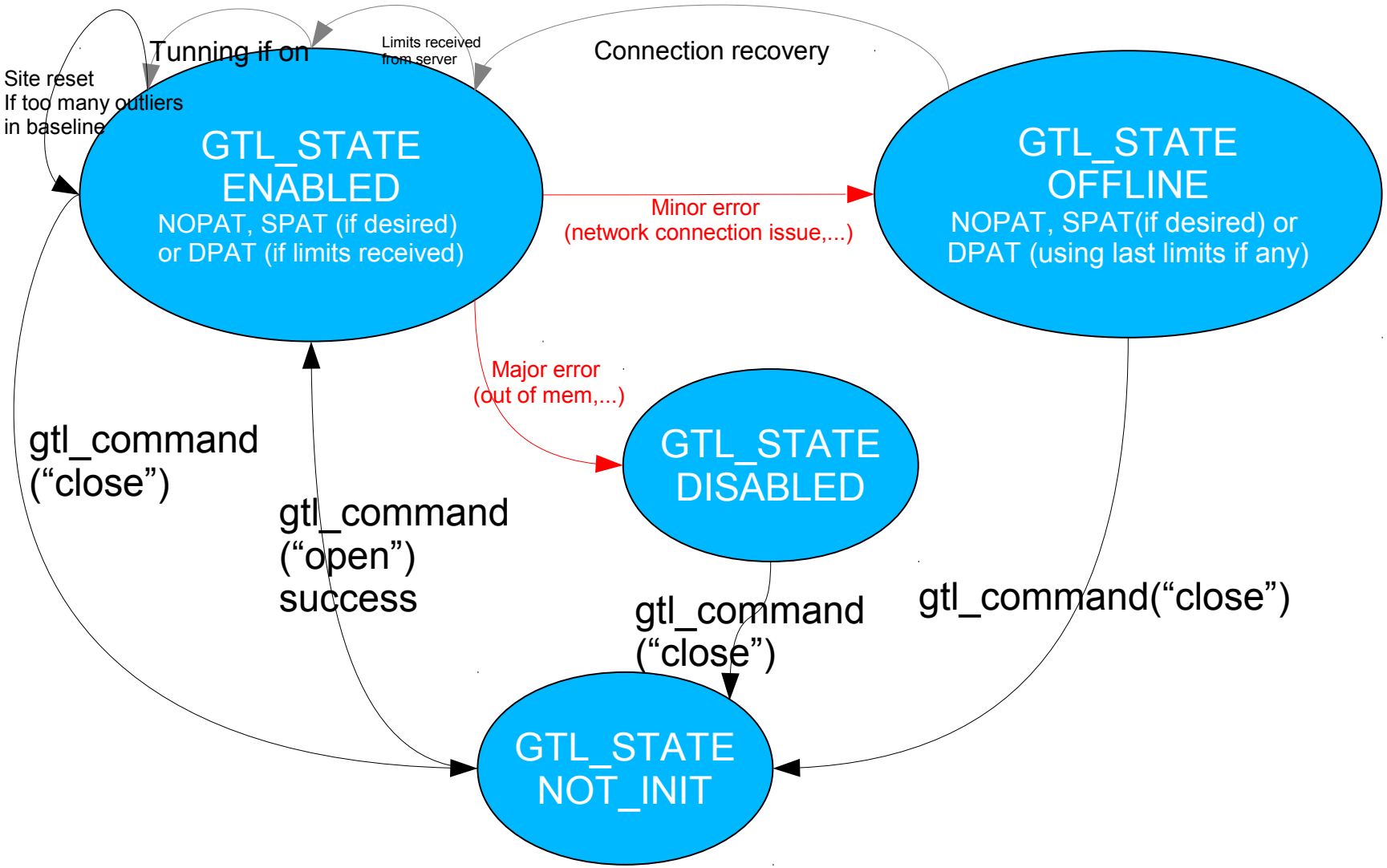
This is mandatory to check the status of all GTL functions after any call:

Example:

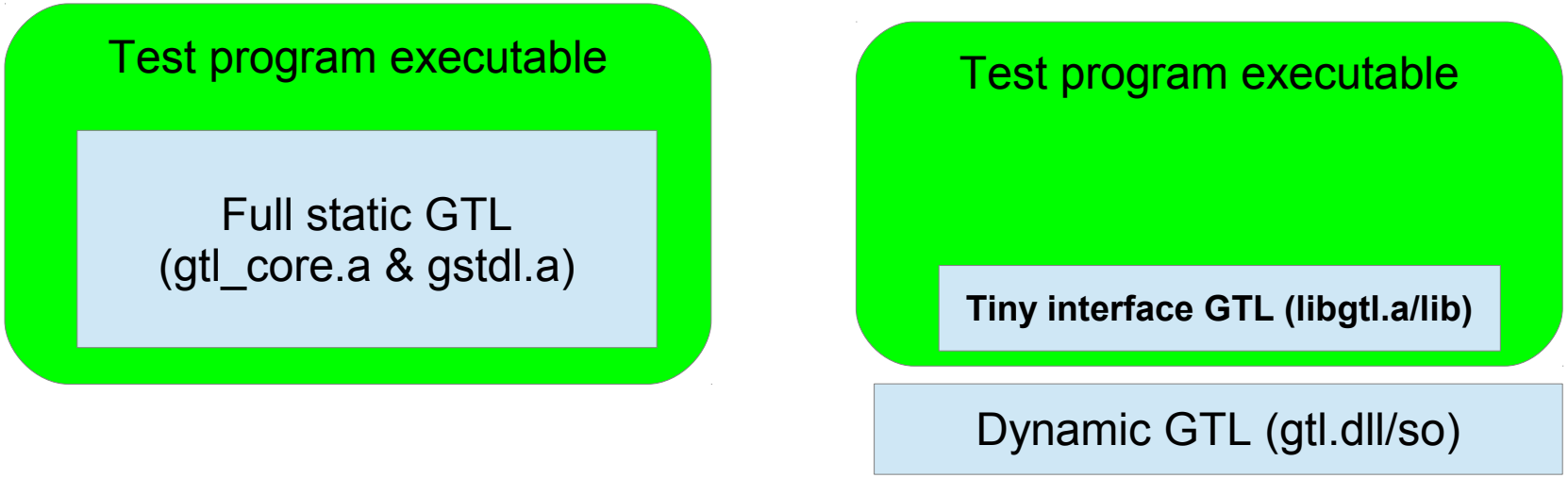
```
int Status = gtl_command("do_something");

if (Status!=GTL_CORE_ERR_OKAY)

then //you should better stop/pause....
```

The full list of status codes is in the file gtl/include/gtl_core.h

GTL logic


Tunning if on
Limits received from server
Connection recovery
Site reset
If too many outliers in baseline
GTL_STATE ENABLED
NOPAT, SPAT (if desired) or DPAT (if limits received)
Minor error (network connection issue,...)
GTL_STATE OFFLINE
NOPAT, SPAT(if desired) or DPAT (using last limits if any)
Major error (out of mem,...)
GTL_STATE DISABLED
gtl_command ("close")
gtl_command ("open") success
gtl_command ("close")
gtl_command("close")
GTL_STATE NOT_INIT

Static or dynamic GTL



In order to use the GTL dynamic library :
- define 'DYNGTL' in your project settings
- link with libgtl.a/.lib instead of gtl_core.a/so
- copy the dynamic lib (gtl.so/dll) in any executable registered folder, i.e. in your PATH

Note: on Teradyne IGXL (gtl-igxl.dll) and LTXC enVision (libGTLenVision.so), GTL is available as a dynamic lib only.
Warning: latest MSVC cannot use the static version of the GTL, only the dynamic.

Static or dynamic GTL ?

In order to use GTL in a compiled test program, the compiler must link with GTL:

- if using the static version of the GTL, the linker will input/insert all needed GTL codes directly into the executable so that no extra files are needed at execution time

- if using the dynamic version of the GTL, the compiler will just link against the GTL interface library, meaning the executable wont contain GTL codes, meaning the GTL dyn library file will be needed at execution time, on the tester.

It is usually advised to use the dynamic version of the GTL because it will ease the update process: the users would just have to overwrite the GTL dynamic lib file in order to upgrade.

About GTL version

GTL has its own versions in sync with GTM:

- GTM version 7.2 is in sync with GTL version 4.3
- GTM version 7.3 is in sync with GTL version 5.0
- GTM version 7.4 is in sync with GTL version 6.0

To get the version of the lib you are using:

either look in gtl_core.h:

```
#define GTL_VERSION_MAJOR 4
#define GTL_VERSION_MINOR 3
```

or call gtl_get(GTL_KEY_LIB_VERSION, ...)

Where is the revision history of GTL ?

Directly into the gtl_core.h :

gtl_core.h | <Select Symbol> | # Line: 23, Col: 3

```
Revision history:
o 17 May 2005: Created.
o 2012 : version 2.2
o 2013 Q1 : version 2.3
o 2013 Q2 : version 3.0 to 3.1
o 2013 Q3 : version 3.1 to 3.3
o 2013 Q3 : version 3.3 to 3.4
o 2013 July : version 3.4 to 3.5
  - removed gtl_endlot() interface
  - protocol modification (ENDLOT message sent to GTM waits for a reply)
o 2013 July : 3.5 to 3.6 (RC1)
  - updated IGXL interface to adopt gtl_init() full interface with site numbers
  - added some hidden keys for blocking / non-blocking connection and communications
  - added some hidden keys for network timeout in non-blocking mode
o 2013 August : 3.6 to 3.7 (RC2)
  - changed protocol for Q_INIT to fix case 7484
o 2013 August : 3.7 to 3.8 (RC3)
  - fixed SPAT limits and stats insertion inside SQLite file
  - added transfert for both good hard and soft bin list from GTM to GTL
  - added nb_part and nb_parts good in SQLite splitlot table
  - added start_t and finish_t in SQLite splitlot table
o 2013 August : 3.8 to 3.9 (RC5 and 7.1 final):
  - now using good hard and soft bin list to identify Pass of Fail bin status for SQLite output update
o 2013 Sept : 3.9 to 3.91:
  - fixed test def insertion when LL but no HL and viceversa

o 2013 Nov : from 3.9 to 4.0:
  - added gtl_mptest(...) new function useful for MultiParametricResults tests.
  - added retest support: new command gtl_command("retest");
  - added resume support : if a previous SQLite exists on the tester, then GTL checks for same product/lot/sublot/tester and if same
  - added new command: "query" limited to 'select' queries. Exple: gtl_set(GTL_KEY_QUERY, "select MAX(run_id), * from ft_rollinglimit
  API deprecation:
  - gtl_get_lib_state(...) vs gtl_get(GTL_KEY_LIB_STATE, ...);
  - gtl_get_lib_version(...) vs gtl_get(GTL_KEY_LIB_VERSION, ...);
  - gtl_set_prod_info(...) vs gtl_set(GTL_KEY_PRODUCT_ID, ...), gtl_set(GTL_KEY_LOT_ID, ...), ...
  - gtl_set_node_info(...) vs gtl_set(..., ...), ...
  - gtl_get_number_messages_in_stack() vs gtl_get(GTL_KEY_NUM_OF_MESSAGES_IN_STACK, ...)
  - ...

o 2014 Jan : from 4.0 to 4.1 (7.2 preview)
  - added many keys for filling many splitlot table fields (case 7672)
o 2014 Feb : from 4.1 to 4.2 (7.2)
```

Which TCP port is GTL/GTM running through ?

- On the GTL client side:

  look at gtl_core.h: GTL_DEFAULT_SERVER_PORT

  look at gtl_tester.conf file

- On the GTM server side: look at the GTM options.

GTL logs

GTL is logging according to the environment variable GTL_LOGLEVEL :
- 1 : Alert : GTL cannot continue to work
- 2 : Critical : GTL cannot continue to work
- 3 : Error : An unusual error happen, GTL could continue but you better stop.
- 4 : Warning : GTL can continue to work even if something strange happened
- 5 : Notice : not unusual, the default level, acceptable for production.
- 6 : Information : useful detailed info and acceptable on production
- 7 : Debug : for developper/debug purpose only. Not adviced for production.

Example : a GTL log level of 5 (the default) will impose GTL to log messages of level 5 or less, excluding all logs of level 6 and 7.

The current 3 outputs of the GTL are :
- text based log file: generated in 'output_folder' set via gtl_set('output_folder',...)
- sqlite output : generated in folder 'output_folder' with filename 'outputfile_name'
- syslog output : using UDP port 514

GTL Configuration file

The configuration file needed for the GTL to (re)connect to a server is such :

```
[Server]
Name=first.mydomain.com
IP=192.168.0.1
SocketPort=4747

[Server]
Name=second.mydomain.com
IP=192.168.0.2
SocketPort=4748
```

1: GTL will first try to resolve the 'Name' via 'gethostbyname'.
2: If no success, GTL will try to use the given IP if any

Note : on Linux, if localhost, be sure to include your local domain if any as stated in /etc/hosts

Disconnection/Reconnection

In case of connection lost with the server, GTL will:

1 Swap to OFFLINE mode returning an error code as GTL_CORE_ERR_GTM_COMM, GTL_CORE_ERR_FAILED_TO_SEND_RESULTS, ...

2 At the next run, try to connect to a server starting from the first one until the last

3 If reconnection not successful, GTL will stay in OFFLINE mode

4 To provoke/request a reconnection trial, call: gtl_set(GTL_KEY_RECONNECTION_MODE, “on”);

Retest

Multi session retest: when initial test and retest are done in a different GTL/GTM session.

call gtl_command(GTL_COMMAND_RETEST) command BEFORE gtl_command(GTL_COMMAND_OPEN) for the retest session(s) so that GTL will try to reload any previous DPAT limits from previous splitlot data.

On the fly retest: when initial test and retest are done in the same GTL/GTM session.

call gtl_command(GTL_COMMAND_RETEST) command between 2 runs (before gtl_command(GTL_COMMAND_BEGINJOB) and after gtl_command(GTL_COMMAND_ENDJOB)) for GTL to reload any desired limits (tightest, largest, last)

Resume

Multi session resume: when both initial test batches are done in a different GTL/GTM session.

The multi session resume is automatic. Open GTL as usual, it will search for a previous session for this context (product,lot,sublot,tester,...) and if found, will augment the session and reload any last DPAT limits for any site. If the context is different, the previous data/lot will be overwritten.

- It is possible to disable limits reload by setting the key GTL_KEY_RELOAD_LIMITS to “off”: gtl_set(GTL_KEY_RELOAD_LIMITS, “off”). The default value is “on”.
- On the fly resume is not supported.

Matching context (1)

During multi session retest or multi session resume, the GTL will try to determine if the current testing context is the same as the previous one. This is done by comparing some parameters between the current and the previous context.

By default, these parameters are: Product, Lot, Sublot, Tester name, Tester type.

They can be customized by setting the GTL_KEY_FIELDS_TO_MATCH key:

gtl_set(GTL_KEY_FIELDS_TO_MATCH, “<field1>, <field2>,...”)

Matching context (2)

The following fields are supported: product_id, lot_id, sublot_id, tester_name, tester_type, station_number, retest_code, retest_hbins, job_name, job_revision, operator_name, test_code, facility_id, test_temperature, user_text, family_id, spec_name, datafile_name.

Each field has a define in the gtl_core.h header.

Example:
gtl_set(GTL_KEY_FIELDS_TO_MATCH, “product_id,lot_id,tester_name”);

Default value:
“product_id,lot_id,sublot_id,tester_name,tester_type”

About memory used by GTL

GTL is using internal memory to store:

- global information: num of tests, some indexes, num parts good, Pat bins, lib state, list of good soft/hard bins, message ring buffer, output folder, temp folder, about 10 file names, … about 10KB of mem

- station information: num of sites, site numbers, tester type, tester name,... about 1KB of mem

- prod information: operator name, lot, product, splitlot id, retest index, … about 1KB of mem

- tests results per site cache : (nb of sites * nb of tests * window size) KB

- historic information in a in-memory SQLite DB: previous limits received, … from 1MB : maximum defined by : number of tuning, number of devices in lots, number of tests, ... size of the SQLite file.

- run results : bins for previous runs of the buffer: 10KB

TOTAL: minimum:1MB maximum:depends on size of lots, recipes, params,...

About CPU used by GTL

GTL is not creating any thread. These estimations have been made on a 2GHz CPU.

The CPU cost of each function is given here:

- gtl_get/set : usually just setting/reading a variable. Constant cost (few nanosecs).

- gtl command open : probably the longest call in gtl: vary from 1sec (server on) to 1mn (server off, LAN out of service, bad IP, ...).

- gtl command beginjob : performs network communication with GTM (few microsecs).

- gtl_test : does not perform any network task, constant cost (few nanosecs).

- gtl_binning : does not perform any network task, constant cost (few microsecs).

- gtl command endjob : could perform network task if buffer full (few microsecs to few millisecs).

- gtl command close : writes the resulted SQLite file on local HDD: could cost several seconds according to the size of the lot, number of tests, number of tunning,...(few seconds).