lusis

Tango Developer Hand Book 2.9

Content

1 DOCUMENT PURPOSE

The purpose of this document is to define development norms and procedures for TANGO C++.

2 DATA TYPES

2.1 Basic types

These types are to be preferred for passing methods parameters:

byte1	1 byte
byte2	2 bytes
byte4	4 bytes
byte8	8 bytes

For an unsigned variable, a ‘u’ must be in front of byte? (ex: ubyte4).

Comment: the “unsigned” have to be used only for variables which value range exceeds signed maximum.

2.2 Objet simple types

string	the C++ standard class of characters string.

3 WRITING NORMS

3.1 Header for names of variables or properties

2 or 3 characters in front of a "_" for all the names.

Warning: all characters are in lowercases.

First character:

p	Parameter
k	Constant
l	Local
g	Global
m	Member (class property)
s	Static (static property of a class)

Second character:

a	Array
s	Structure
c	Object
1	Byte1 (signed or not)
2	Byte2
4	Byte4
8	Byte8
v	Void
z	'string' (with a lowercases « s » template std c++)
i	Iterator
t	TG_TEMPS (TG_TIME)

Third character:

nothing or m	The variable
R	Reference
p	Pointer

See example here after.

3.2 Development repository

The root of a Development repository defines the following sub-repositories:

- include : for all the header files

- source : for all the source codes
    - if the product is « big », it is possible to set up other sub-repositories to spread the sources
- EXEC : for all files upon which a link edition must be done
- lib : for the compiled files
- bin : for the binaries

3.3 File names

3.3.1 Generals

C++ source: *.cxx
C source: *.c
Headers: *.h

Name syntax: <belonging>_<description>
Belonging: hierarchy of sub-systems to which it belongs
Description: free

Warning: only one underscore ("_") is possible in the file name.

Examples:
- tg_type.h,
- tg_cons.h

3.3.2 Classes Coding:

Name syntax: <product>_<class name>

product: sub-systems to which this class belongs
class name: the class defined in this file
Description: free text

3.3.3 Examples

Headers repository for the PRO_TMP class:

pro_tmp.h: class descriptor
protmp_cfg.h: class configuration descriptor.
protmp_con.h: class constant descriptor
protmp_bus.h: class bus access descriptor
protmp_ctx.h : class queries context elements descriptor
protmp_ses.h : class applicative session context elements descriptor

4 NAMING AND WRITING RULES

4.1 Generation repositories and runtimes

Various types of programs can be generated to run an application:

- Pure Tango runtimes (platform programs)
- Runtimes for functional area (payment systems, finance, loyalty…)
- Specific runtimes for a project.

Programs naming must be consistent with this.

Naming rule for generation repositories:

- prefix to identify the program family
  - tg_ for Tango (hypervisor, dispatcher,..)
  - ema_ for payment systems[1]
  - fin_ for finance systems
  - fid_ for loyalty systems
  - trs_ for transport systems
  - uty_ for utilities systems
  - proj_ for project specifics (eg : isim_, wb_, …)
- programs names must explicitly describe the function which is provided. Examples:
  - disp: Tango dispatcher
  - hyper : Tango hypervisor
  - visar : Visa requesting interface
- gen suffix is used if the program to be generated contains more than one object

Programs names must be consistent with the names of the generation repositories.

With the same list of programs:

- tg_loader is the generic loader of Tango applications which are generated in modules (libraries .so or .sl)
- tg_disp Tango dispatcher
- tg_hyper Tango hypervisor
- ema_visar Visa requesting interface
- isim_ifggabe GABE ATM interface for ISIM project.

4.2 Language

All the names for classes, methods, functions and variables are in English language.

4.3 Names

It is better to use full names than acronyms. If the name is too long, it is better too use full syllables than acronyms.

[1] EMA : « Electronic Money Applications »; used to qualify all the Payment system Tango modules/objects.

Example: instead of “translateMessageDictionnary” this method is named “transMesDic”.

Underscores are used only for prefixes.

If you want to append terms, you will use lowercases and uppercases for the first character of the new term (aaaBbbbCccc).

Ex: a method used to import files should be named **importFile**, not IMPORTFILE, nor importfile, and **never import_file**.

4.4 Functions and methods

All C + + and C methods functions must be coded into homogeneous groups.

The blocks of replicated must be avoided. It is preferable to define utility procedures in order to allow easy maintenance (bug fixing at a single location).

Details of various uses of comments are described below.

4.5 Variables and Includes

Variables and Includes management:

- Avoid useless inclusions in the sources
- Avoid multiple inclusions
- Pay attention to cross references (or circularly references):
    - Example: if A is included in B, and B is included in C, do not include A in C.
- Avoid useless declarations:
    - suppress **unused** variables
- If, in a class declaration, another class is presented on pointer form, it is better to use a « forward declaration » of this class than to include its header file. Doing so, the header file will just be included when the class is used (.cxx file).

4.6 Comments

JAVADOC formalism is used to generate a first level documentation using ‘doc++’. Therefore, comments must respect the following syntax.

4.6.1 Files headers

All the files must contain a header presenting the global function of the program:

```
/*
```

```
*
* file content description
*
*/
```

4.6.2 Classes headers, methods, variables and constants declaration

Each of these elements must be described by a sentence. Each sentence is ended by a dot. It is possible to add complement information and possible parameters or values.

```
/**[2]
 * method concise description.
 * [complement]
 * @param parameter-name parameter description.
 * @param parameter-name parameter description.
 * @param parameter-name parameter description.
 */
```

4.6.3 Algorithms:

It is sometime useful to explain a complex branch of code. On the contrary it is more than useless to comment simple or even clearly readable code.
It is forbidden to "replicate" code in the commentary section.

Examples:
« nbcnx=nbcnxo+nbcnxf // init of nbcnx by summing nbcnxo+nbcnxf »
Is of no interest,
On the other side:
« nbcnx=nbcnxo+nbcnxf // total number of connections is the sum of open connections and closed connections »
Is valuable (would be useless if the variables were named correctly).

4.6.4 Bug fixing

Bug fixing identification by Author / date / [Bug number] / text

// M1 : 02/04/2002 type I adding: iterator
Or
// M1 : 02/04/2002 LU012AB type I adding: iterator

4.6.5 Example for each elements

```
/*
 *
 * file content description.
 *
 */
```

[2] /** is a necessity for the use of JAVADOC.

```
/**
 * concise constant description.
 * [complement]
 * value
 */
const TG_TRAMSK tg_traceMskNone     = 0x0000000000000000LL;


/**
 * concise class description.
 * [complement]
 */
class DLLSPEC TG_ROOT
{

public:

  /**
   * concise description.
   * [complement]
   *
   * name               : feature
   *
   */
  enum k4_log_type
  {
    k4_log_elementState,
    k4_log_clientState,
    k4_log_externalEvent,
    k4_log_command,
    k4_log_user
  };

private:

  /**
   * concise variable description.
   * [complement]
   */
  byte1 ma_tangoId[17];


protected :

  /**
   * method concise description.
   * [complement]
   * @return description of the possible values to be returned.
```

```
   */
  const TG_TRAMSK getMskTraObj()


public :

  /**
   * method concise description.
   * [complement]
   * @param parameter-name parameter description.
   * @param parameter-name parameter description.
   * @param parameter-name parameter description.
   */
  TG_ROOT(byte1*    pap_tangoId = 0,
          TG_TRAMSK pc_mskTraObj = 0);

  /**
   * method concise description.
   * [complement]
   */
  ~TG_ROOT();

};

#endif // TG_ROOT_H
```

4.6.6 Implementation example

```
/*
 * TANGO root class declaration file.
 */


// ////////////////////////////////////////////////////////////////////////// //
// Constants for basic trace filters //
// ////////////////////////////////////////////////////////////////////////// //

/**
 * trace mask for which all trace levels are inactive.
 * 0x0000000000000000
 */
const TG_TRAMSK tg_traceMskNone     = 0x0000000000000000LL;


/**
 * TANGO base Class. This class undertakes all common
 * tango client access methods for e_spy and e_log
 * e_log access methods for errors are implemented at the
 * exceptions class level.
```

```
 *
 */
class DLLSPEC TG_ROOT
{

public :

 /**
  * messages type Identifier sent to e_log.
  * messages type Identifier is transported in "TG_E_LOG type" field.
  * possible values are:
  *
  * k4_log_elementState: messages type Identifier for state change of an element
  *                      this element is not a TANGO client
  *                      but an entity of the processing string).
  *
  * k4_log_clientState   : messages type Identifier for state change of
  *                      a TANGO client application.
  *
  * k4_log_externalEvent: messages type Identifier for external event
  *                      signalization.
  *
  * k4_log_command       : messages type Identifier for command execution
  *                      acknowledgment.
  *
  * k4_log_user          : messages type Identifier for custom message (for these
  *                      messages, the extraction of the using fields
  *                      will be defined using the data base).
  *
  */
 enum k4_log_type
 {
   k4_log_elementState,
   k4_log_clientState,
   k4_log_externalEvent,
   k4_log_command,
   k4_log_user
 };

private :

 /**
  * each non volatile TANGO resource owns its TANGO identifier. this
  * identifier is supplied when the resource is constructed.
  */
 byte1 ma_tangoId[17];


protected :
```

```
 /**
  * this method is used to get the trace mask of the instance.
  * @return trace mask of the instance.
  */
 const TG_TRAMSK getMskTraObj()
 {
   return mc_mskTraObj;
 };


public :

 /**
  * this method is used to get the TANGO identifier of the instance
  * derived class from TG_ROOT.
  * @return TANGO identifier of the instance.
  */
 const byte1* const getTangoId();


 /**
  * TG_ROOT class constructor. It is called (implicitly or not)
  * when a derived class is built.
  * @param pap_tangoId instance identifier. If no parameter is
  *        supplied, the identifier is calculated using the pointer of
  *        the instance.
  * @param pc_mskTraObj trace Mask initialization.
  */
 TG_ROOT(byte1*    pap_tangoId = 0,
         TG_TRAMSK pc_mskTraObj = 0);

 /**
  * TG_ROOT class Destructor.
  */
 ~TG_ROOT();

};

#endif // TG_ROOT_H
```

4.7 Writing rules

4.7.1 C++ classes

4.7.1.1 General rules

- For class declarations: order items per litter with public first, followed by protected and private (except for classes generated and modified automatically.).

- Focus on assigning a value to the properties of a class rather than by direct initialization by copying.
- For any class which manipulates resources, it is mandatory to define a constructor for copy, a copying operator and a destructor.
- Constructors, copying operators and destructors must at most avoid generating exceptions.
- Functions defined inline must be simple and not virtual.
- Do not use the inline keyword. Prefer the definition of the functions inside the class declaration.
- Declare as constant, the set of functions that do not alter the state of the instance.
- Do not declare an operator conversion to basic types. This would introduce uncontrollable implicit conversions.
- Always report an output operator (operator <<) to an ostream. This operator is used for debugging and testing.
- Each class must at least define the following methods: default constructor, copy constructor, copying operator and destructor. Depending on the case these operators are defined as protected or private to restrict their use.

4.7.1.2 Constructors and destructors

- Check that all the constructors call a constructor for each basic class and initialize all non-static members of the class.
- Write the initialization list of data members in their order of definition.
- Check that abstract classes have no public constructor
- Do not use the delete operator on “this”.

4.7.1.3 Inheritance

- Use only public derivations
- Check that the polymorphic base classes have virtual destructor.
- If a virtual function is not overloaded in a derived class, then it should not be a virtual function.
- When a function is overloaded, it should have the same default settings that the function of the base class.
- For abstract classes, the copying operator must be protected.

4.7.1.4 Object oriented design

- Data members should be private
- The "const" functions should not return non-constant data or on accessors non-constant data
- Ensure that classes or methods “friends” are justified in terms of design.

4.7.1.5 Operators overloading

- Do not overload operators ', ' '& & ' and'||'.
- The role of overloaded operators must be consistent with the role of these operators for other entities (e.g. an overloaded operator '==' is to test equality).
- Binary operators should not be members, to allow implicit conversions from the left operand.
- When the operator '[]' is implemented, it is necessary to implement the const version and non-const version.
- Whenever possible, do not overload the operator '()'.

4.7.2 Code complexity

- Do not write functions with a McCabe Cyclomatic Complexity3 indicator greater than 10.
- Do not write functions with a number of branches greater than 200.
- Do not write functions with more than 6 parameters.

4.7.3 Control structures

- Follow each flow control instruction (if, else, while, for, do, switch) by a block of statements delimited by parentheses, even if the statement block is empty or contains only one statement.
- For the test instructions (if, for, while, and '?:'), always use an explicit test that returns a Boolean.
- Do not implement tests that return a constant value.
- If a switch block ends without a break statement at the end of treatment box (case): this should always be explained with a comment.
- Never modify the control variable in a statement 'for' inside of the statement block of the loop.
- Do not change more than once per iteration, the control variable of the loop.
- For a loop, the control variable must be compared to a constant (or variable with a constant value over the duration of the loop) and not an expression neither a result of a function.
- With the exception of switch statements and software exceptions, each code element should have a single entry point and one exit point.
- Include an explicit statement block for each path of a multi-paths control structure.
- “for” instructions: Declare the control variable loops inside the “for” statement (do not use a pre-existing variable).

4.7.4 Constants

- Use the L and UL suffix for all constants and byte4 ubyte4.
- For float and double constants, you must show the decimal part even when is zero and use the suffix L for double.

[3] McCabe Cyclomatic Complexity

- For strings, use the escape sequence defined by the ISO standard C + +.
- Store ASCII strings in variables of type char (or char from) and not elsewhere.
- Store Unicode strings in variables of type wchar (standard ISO/C99 - wchar.h) or from wchar (std:: wstring = std:: basic_string <wchar>)[4] and not elsewhere
- Prefix Unicode constant strings with L
- Escape sequences within a Unicode string should be in Unicode (\ uXXXX xxxx with value in hexadecimal).
- Constants other than integers must be declared as extern const in header files and defined in a source file. Cxx.

4.7.5 Conversions

- Always use the forms of conversion 'statis_cast', 'const_cast', 'dynamic_cast', 'reinterpret_cast' or an explicit call to the constructor of the desired type. Do not use other types of conversions.
- Minimize the use of casts.
- Never remove the cast for the modifier 'volatile' or 'modifier 'const' to a variable.
- Do not cast pointers or references.
- For conversion ASCII to Unicode and vice versa, always use the system API or if necessary the libiconv library[5].

4.7.6 Declarations and definitions

- All functions “not inline”, and all variables and enumerations that are declared outside of classes must be declared in a namespace whose name is suffixed by the module of their membership.
- If namespaces are not supported by the compiler (C API, ...), variables, constants, typedefs and enums must have their name prefixed by the module of their membership.
- Global variables should be avoided. If they are still used, a comment should explain why.
- Instructions using namespace should be present in source files .cxx and must be positioned after the inclusion of header files.
- Do not use the keyword "auto" and "register ".
- Each variable must be declared on a line (with the variable declaration followed by an explanatory comment).
- Initialize each object or variable at its declaration. Do not use objects or uninitialized variables.
- Use the keyword "const" whenever possible.
- Best use signed integers for arithmetic operations and unsigned integers for logical operations.

4.7.7 Exceptions

- Never exit a destructor by an exception.

[4] C support for Unicode and UTF-8
[5] TG_SYS has now the functions wstringtostring and stringtowstring.

- Wherever possible, exception classes must derive from std:: exception.
- Always catch exceptions by reference
- Always handle the release of temporary resources in the exception handler.

4.7.8 Expressions

- Always use symbolic names instead of numeric values in code.
- Always test the bounds of an array before accessing an item by its serial number (this remark also applies to objects with operator []).
- Do not play on the evaluation order of operands in an expression.
- Use parentheses to make explicit the order of evaluation of operands of an expression.
- In a test of equality between a variable and a constant, always put the constant on the left of the test.
- Never use side effects in the right operand operators '&&', '||', 'sizeof' and 'typeid'.
- Do not use the following operators to operands signed offset to the left and right and ('<<''>>'), '&', '|', '^'.
- Do not mix signed and unsigned integers in a single operation.
- Do not use floating point numbers if the result of calculation must be exact.
- Always test the non nullity of the divider before using the operator '/' or operator '%'.
- Do not use the operator ','.
- When using the operator '? ', always check the return type if the expression is true, is the same as that returned type when the expression is false.

4.7.9 Functions

- All functions with the same name must have the same functionality. Only vary the number and type of parameters.
- In source files, put all non-member and not "extern" functions in a namespace anonymous.
- Specify the name of the function parameters at their statement and their definition (using the same names).
- Declare the read only parameters of objects type as const reference. For basic types, pass constant parameters by value (warning types from the STL, are objects and not basic types).
- The functions should not return pointers or references to local objects, but must return a copy of these objects.
- Only the trivial functions can be declared "inline".

4.7.10 Memory management

Memory management is an absolute key point of Tango programming.

- For dynamically allocated memory, do not use the C API malloc, realloc, alloca and free, but the API C + + new and delete.
- Ensure that the form used to delete (either an array or an element) is the same as that used for the new.
- Never specify the number of elements in the release of an array of items (delete []).
- Never return a dynamically allocated pointer in a function.

- Always put a pointer to 0 after freeing the memory associated with it.
- Avoid memory leak when manipulating containers:
    - Elements of the container must always be deleted before the container is destroyed
    - When elements of a container are also containers, start destruction from the inner part of the structure and finish by the outer part.
- Maximize the lifetime of dynamically allocated resources.
- While using dynamically allocated objects with short lifetime, use TANGO pool allocators to limit the impact of their allocations/deallocations on the system.

4.7.11 Portability

- Limit the use of non-ANSI API to objects in charge of Operating System abstraction.
- Never make any assumptions about the internal representation of a value or object.
- Never cast a pointer to a base type to a pointer to a base type whose alignment is more restrictive.

4.7.12 Pre-compilation Directives

- Always start the pre-compilation directives on the first column of a line of code and do not put spaces or tabs between the character '# ' and the pre-compile directive.
- Always use the identifier "__cplusplus" to differentiate the C ++ structures from C structures
- Always put a blank line at the end of the source files.
- For inclusions of files, always use
    - <> for standard header files and
    - "" for header files for applications.
- Do not include paths in # include directives.
- Always put the guards in header files to avoid multiple inclusions.
- Do not put comments in the definition of macros.
- Put parentheses around the arguments and body macros.
- Do not use "# define" to define constants but on the contrary use const integers or enumerations.

4.7.13 Structures, Unions and Enumerations

- Limit the use of structures variable (union) to C structures used to optimize memory or communications.
- Do not make assumptions about the value assigned to an element of enum (compelling value if necessary).
- Never convert an integer variable to a variable of enum type.

4.7.14 Templates

- Do not set a template class with methods potentially ambiguous.
- Always instantiate templates with types that meet the constraints of templates

- Only use templates when the functionality of the template class or function template is totally independent of the type which it is applied.

4.7.15 STL

- Use the header files defined by the standard and not those suffixed by. h (<iostream> is ok, <iostream.h> is not).
- Use preferentially the containers and algorithms from the STL to locally defined elements.
- Always ensure the effectiveness of the copying operator to fill the objects used in STL containers.
- When it is not possible to implement an efficient copying operator, use containers of pointers.
- Never attempt to insert an instance of derived class in a container of base class.
- Use the operator "empty ()" rather than comparing the size of the container to 0.
- Ensure that resources are released well in the destructor when using an STL container as a base class.
- Never create containers of self pointers.
- Use containers vector or string instead of arrays allocated dynamically.
- Use the notation '& v [0]' to pass a vector as a parameter to a C function
- Always use the method "c_str ()" to get the constant pointer on the ASCIIZ string stored in a string object.
- Never change the key of an entry of container set or multiset.
- Avoid mixing types of iterators.

4.7.16 Miscellaneous

- Always think about how to test the program before coding.
- Create (or identify) the unitary testing tool in the time than the code is the best way to produce an efficient code
- Always treat the case 'out of memory'.
- Prefer compiling and linking errors to execution errors.
- Observe all warnings by the compiler products.
- Organize the code so that non-portable parts are simple to locate and replace.
- Use macros to replace the unsupported "key words".
- Always put a comment when proof encoding rule is violated.
- If you do not follow the rules, be prepared to be questioned
- Leave no unreachable code.
- Do not duplicate fragments of code or data.
- Random bug do not exist, neither non understandable ones.

4.7.17 Control tools

- New releases of Code are controlled by Cppcheck provided by sourceforge (http://sourceforge.net/projects/cppcheck/).
- Run times are controlled (by the developer or by the project team) using Valgrind (http://valgrind.org/).

5 SECURE PROGRAMMING REQUIREMENTS

A lot of security rules are implemented by TANGO architecture itself. Therefore it is not told about that in this section. For instance the standard Input Validation rule “Identify all data sources and classify them into trusted and untrusted” is assumed by the fact that to enter data in a TANGO system you have to go through either a NPM (Network Protocol Manager), either a TANGO Web service, or through a batch. In the 3 cases the aim of the first level of the software is to check data validity and to refuse any message or record which does not match the “sanity test”.

All the following rules are to be strictly applied:

Payment related rules

- S-P-1. Never code a back door
- S-P-2. Never code even for debugging purpose any function which would break PA DSS rules (such as casting track 2 data…)
- S-P-3. Never write forbidden data in error messages or trace messages: PAN, T2, Pin block, CVx, T3. This recommendation concerns full data or any subpart.
- S-P-4. Never store test data on archived modules delivered to project teams (this includes database data, commented or disabled source blocks, data files, ...). Use a separately archived and clearly identified module to store test data (eg : test data of ema_xxxHandler module have to be saved on ema_xxxHandler_test module which is only used at development level and never delivered to project teams or final customers).
- S-P-5. Never store test accounts on archived modules delivered to project teams. As explained on S-P-4, use a separately archived module to store test accounts.
- S-P-6. Never use Live PAN for modules tests.

C++ implementation related rules

- S-C-1. section 4.7 must be strictly applied
- S-C-2. Protect your code against “buffer overflows” or “buffer overruns”. This is addressed by forbidding using strcpy and use strncpy instead of it.
- S-C-3. Avoid vulnerabilities and security flaws resulting from the incorrect use of dynamic memory management functions (see 4.7.10)
- S-C-4. Eliminate integer-related problems: integer overflows, sign errors, and truncation errors (see 4.7.4, 6, 8 and 12)
- S-C-5. Avoid I/O vulnerabilities, including race conditions. Mutual exclusion of conflicting race windows are managed directly by TGSYS (Tango platform layer) which handles the Mutex conditions.
- S-C-6. Correctly use formatted output functions without introducing format-string vulnerabilities (see 4.7.9).
- S-C-7. While using the “system” command with command parameter build from user defined data, always check the form of the user defined data using regular expressions (prevent usage of shell special caracters such as ‘;’,‘>’,’&’, ... on user inputs).

These recommendations will be checked using the following methods:

- Each time a change regarding dynamic memory management function will be done, the TANGO code will have to pass through “endurance tests” which objectives are to detect memory leaks by managing a large number of transactions. This will validate the respect of rules S-C-1 to 3.

- Before any delivery the LUSIS project leader will always check the respect of rules S-P-1 to 3 using CVS source modifications viewer.
- Rule S-C-4 will be checked as follows: compilation time and development unit testing
- Rule S-C-5 will be checked as follows: endurance testing with concurrent message flows
- Rule S-C-6 will be checked as follows: code review development manager.
- Rule S-C-7 will be checked as follows: code review development manager.

6 MAKING AN EFFICIENT AND PROFESSIONAL CODE

Keep in mind that:

- the development will be operated in a complex environment where useful information has to be provided so that identification of incident must be easy
- the development must be validated through acceptance tests executed by non-developers that need clear and complete information accessible from logs and traces.

Therefore, events as they are specified in the specifications documents are to implemented as carefully than management rules themselves. Moreover, a specification without any Log event must generate an alert for your manager and the corresponding project leader because it is strictly abnormal.

To achieve the two goals it is important that the following rules are applied systematically.

6.1 Development lifecycle

To insure software quality, developments have to be respect following lifecycle

6.1.1 Technical design

On new pieces of software, developers have to write a technical design describing how functionalities presents on functional specifications will be implemented.
While modyfing existing pieces of sotfware, developers have to update technical design in order to describe how software updates will be implemented.
This technical design has to be validated by a technical manager or a senior developer prior to start coding.

6.1.2 Coding

Software developers have to respect rules presents on this document.
While working on existing software piece, and for all the points not described on this documents (indentation ...), developers also have to conform to the rules already used.

6.1.3 Testing

Prior to deliver software pieces, developers have to pass unitary tests.
Those tests must at least include all rules described on functional specifications.
They must also include all defects already found on software piece.
A document describing all unitary tests done have to be write.

6.2 Logging events

6.2.1 Systematic logging of incidents

When an unusual event occurs or when termination of the application event processing produces an error, an event log message must be produced. This message must contain relevant information that will help understanding of the incident or the application error. The relevant information is contextual to the application and must be described in the detailed specifications of the module.

Lower Tango layers like Database management or Tango Bus management manage errors through exception handlers: it is important to keep the exception handling provided even if exception is catched by new code as the Tango Exception handling provide error logging management and methods related to the error itself (ie: for DB error, the TG_CLIAPP exception handler manages to reset the DB connection if needed).

Each logging is associated to an logging event described by an event number and an event label. These must be defined in a logCfg_xxx.xml file. The event number is not the result code of the processing. The result code will be provided on each logging event as long as it is not an internal Tango exception.

6.2.2 Configurable logging of correct events

For testing purposes, even the correct message processing may be logged. The activation of logging is realized through a configuration parameter accessible to all TG_CLIAPP derived object. The parameter is called loginOut and is standard as a Boolean. It can be derived to precise the logging format (Ascii, Hexadecimal, etc.).

6.3 Tracing

Tracing is a facility accessible to all application class derived from TG_ROOT.
It is important that the Trace is called at least once in an application function. As a string field is optionally useable, it is recommended to provide information helping understanding the result of the function. Trace mask is a bitmap that give the possibility to the developer to categorise the trace per family. A set of 8 bits are available for user defined (application) level. Other bits are internally used by Tango Platform.
Here is the Trace bitmap description:

Object	*Hexadecimal mask*	*Decimal mask*
TG_ROOT	0x0000000000000001	1
TG_CFG	0x0000000000000002	2
TG_CLI	0x0000000000000004	4
TG_COM	0x0000000000000008	8

Object	*Hexadecimal mask*	*Decimal mask*
TG_TCP	0x0000000000000010	16
TG_MAIN	0x0000000000000020	32
TG_MUTEX	0x0000000000000040	64
TG_TASK	0x0000000000000080	128
TG_TIMER	0x0000000000000100	256
TG_X25	0x0000000000000200	512
TG_DB	0x0000000000000400	1024
TG_PRO	0x0000000000000800	2048
USER_1	0x0000000000001000	4096
USER_2	0x0000000000002000	8192
USER_3	0x0000000000004000	16384
USER_4	0x0000000000008000	32768
USER_5	0x0000000000010000	65536
USER_6	0x0000000000020000	131072
USER_7	0x0000000000040000	262144
USER_8	0x0000000000080000	524288
EVT_RCV_MSK	0x0000000000100000	1048576
CMD_CTRL_MSK	0x0000000000200000	2097152
TG_GROUP	0x0000000000400000	4194304

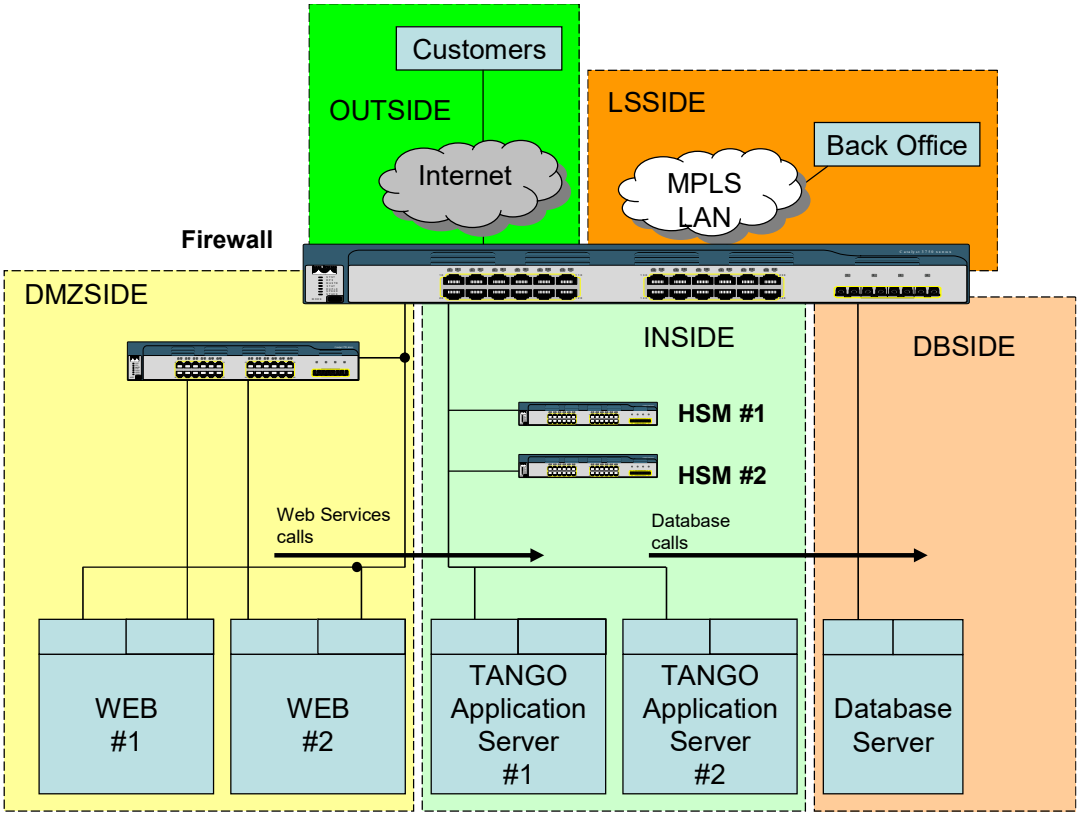
7 WEB DEVELOPMENTS

Although TANGO is a C++ application, some developments are realized using WEB technologies:

- Web services
- PHP

Before going to these 2 domains, it is important to remind the implementation of TANGO (server and WEB TANGO) and to keep in mind two main points:

- All value added services or features are on the TANGO server side. They are never coded on the web side (either in php, either in any other language)
- TANGO services and Web services are never exposed directly on the web.



7.1.1 Web services

On this area the reference document is the "Guide to Secure Web Services"[6] provided by the NIST (National Institute of Standards and Technology; U.S. Department of Commerce). No breach of security principles as they are exposed in this document is allowed.

Web services rely on the Internet for communication. Because SOAP was not designed with security in mind, SOAP messages can be viewed or modified by attackers as the messages traverse the Internet.

There are several options available for securing Web service messages:

- HTTP over SSL/TLS (HTTPS). Because SOAP messages are transmitted using HTTP, it is trivial to modify a Web service to support HTTPS.
- XML Encryption and XML Signature. These XML security standards developed by W3C allow XML content to be signed and encrypted. Because all SOAP messages are written in XML, Web service developers can sign or encrypt any portion of the SOAP message using these standards, but there is no standard mechanism for informing recipients how these standards were applied to the message.
- WS-Security. WS-Security was developed to provide SOAP extensions that define mechanisms for using XML Encryption and XML Signature to secure SOAP messages.

Each secure messaging option has its own strengths and weaknesses.

[6] http://csrc.nist.gov/publications/nistpubs/800-95/SP800-95.pdf

In the Tango architecture design, No WS is allowed to connect to Tango from a public address, thus only private network connections are allowed.

For some requests that would imply transport of Card data or User/password data, the choice is to use HTTP over SSL/TLS for transporting every message. The advantage is that this method is transparent to the application and can provide client authentication using X509 certificate for example when a strong authentication is requested for the client. This may be the case when accepting connections from Web Applications exposed to Internet.

WEB services are designed using Enterprise Architect and the design is directly used to get WSDL. Rules S-P-1 and 2 (see upper) are to be literally respected. Before any delivery the LUSIS project leader will always check the respect of rules S-P-1 and 2 using CVS source modifications viewer.

7.1.2 PHP

7.1.2.1 Data management

PHP is used for data presentation. Every user input should be checked and validate on php side to avoid cross side scripting (XSS) and SQL injection.

- Define a validation rule: length, type, regexp.
- Invalid input must be rejected, no data modification is allowed.
- Define API to access php pages.

SQL requests are never directly integrated into PHP files. PHP code only uses named request (eg: req-auto-1 to n). TANGO configuration does the mapping between this name and the real request and fills it with the parameters.

This technique has several advantages:

1. it clearly protects the TANGO server against SQL injection as there is no SQL request on Web side
2. it avoids any complex request or bad coding on web side as the developers has only a list of services which are provided by the server team. So there cannot be any programmer's improvisation such as "views", "joins"… without a strict performance control by the server team.

7.1.2.2 Data protection

Functions access and data vision or modifications are governed by the TANGO GUI rights definition. Developers have to use these features and are not allowed to overwrite them. This point will be checked for each delivery.

When the user right feature fails, then there is no right for the user and the connection is made unavailable.

7.1.2.3 Development recommendations

Although the scope of development is fairly reduced, the developer will apply as a general principle the recommendations of OWASP (http://www.owasp.org/), read carefully the reference document [7]and ask his manager every time he has any doubt about one implementation or recommendation.

Core pillars are:

- Confidentiality – only allow access to data for which the user is permitted
- Integrity – ensure data is not tampered or altered by unauthorized users
- Availability – ensure systems and data are available to authorised users when they need it

The specific architecture of TANGO and WEB TANGO largely address these points:

- Confidentiality – WEB TANGO: always use the user group / rights management feature.
- Integrity – same + named SQL technique
- Availability – TANGO high availability features.

Security Principles have been classified by owasp and must be respected by developers or are de facto respected using TANGO architecture:

- Minimize Attack Surface Area
  - Surface area is minimized by the strict usage of "Client – Server" architecture (no value added feature is coded on the php area) and the named SQL technique (not SQL entry on the php area).
- Secure Defaults
  - It is forbidden to define any scree or function which would be out of the jurisdiction (for instance, accessible without user/pw and rights management)
- Principle of Least Privilege
  - This principle is fully respected by TANGO
  - Regarding the architecture, there is no way a php developer for WEB TANGO can breach this principle.
- Principle of Defense in Depth
  - Implementation guide for TANGO web
  - Systematic user/pw checking
  - User rights management
  - Client server architecture
  - Controls on the server side
- Fail securely
  - See 7.1.2.2
- External Systems are Insecure
  - TANGO and WEB TANGO embed all the security requirements into their own code.
  - External Systems can provide additional security levels but the the basic requirements
  - Data consistency is always checked by TANGO
- Separation of Duties
  - User groups definition and rights management
- Do not trust Security through Obscurity

[7] http://switch.dl.sourceforge.net/project/owasp/Guide/2.0.1/OWASPGuide2.0.1.pdf

    - The security should rely upon many other factors, including reasonable password policies, defense in depth, business transaction limits, solid network architecture, and fraud and audit controls.
- Simplicity
    - Security must rely on Simplicity
    - Developers should avoid the use of double negative and complex architectures when a simpler approach would be faster and simpler.
- Fix Security Issues Correctly
    - Once a security issue has been identified, it is important to develop a test for it, and to understand the root cause of the issue. When design patterns are used, it is likely that the security issue is widespread amongst all code bases, so developing the right fix without introducing regressions is essential.

A point on session management:

- Session management relies on using PHP session. As the session Id is the key for retrieving session data on each request, choice of saving session Id in a cookie was done rather than transferring it in the URL (GET method).
- The cookie shall be associated with the server plus the application path in order to be sure that the cookie applies only and exactly on the WebTango environment because Web servers may contain different Web applications.

Best practices

- Session lifetime control
- Expire session after a short period of inactivity - set the idle timeout to 5 minutes for highly
- Protected applications through to no more than 30 minutes for low risk applications.
- Enable logout option - explicitly expire and destroy the session on logout.
- Avoid persistent logins ("remember me" option) - optionally you can constrain the
- information retained and revealed by the application, i.e. force the user to re-log in before
- Performing any critical operations.
- Expire session on security error - any security error in the application should result in
- Termination of the session.
- Expire long lasting session – force re-authentication for session, which despite being active
- has reached the maximal allowed time, e.g. a few hours.
- Remove session cookie when destroying a session.
- Never write any technical comment in html format (ie sql request).
- In case of sql error never print the sql request unless a config flag is set.
- Any php pages must be submitted to access control check
- In case of identification failure never give any indication on what is going on to the attacker (ie “user not defined”, simply stat “Authentication failure”).
- While using the “system” command with command parameter build from user defined data, always check the form of the user defined data using regular expressions (prevent usage of shell special characters such as ‘;’,‘>’,’&’, ... on user inputs).

Session identifier

- Use only cookies to propagate session id, because when transmitted via a URL parameter, GET requests can potentially be stored in browser history, cache and bookmarks. It can be also easily viewable then Rotate session id:
    - regenerate (replace with new one) session id, at least whenever the user's privilege level changes. Generally it can be regenerated prior to any significant transaction,
    - after a certain number of requests or after a period of time.

- Check whether session id is valid (of expected size and type) and has been generated by the application and not provided by the user.
- Session id should be adequately long, unpredictable, hard to reproduce and created from high quality random sources.

Session cookie:

- Set the domain of the cookie to something more specific than the top level domain.
- Don't store in cookie anything (at least any sensitive data like username or password) but session id.
- Set http-only flag to disable capturing session id via XSS.
- When possible, use strong encryption (SSL) as well as "secure" attribute to allow transmitting cookies only over https.

Session data storage

- Determine where the application framework stores session data and if it is file system or
- database, determine who else might have access to these data.
- When you store session data in files, ensure the application is configured to use separate
- private directory (e.g. session.save_path directive). Use file system permissions to protect
- these files from observation or modification by users other than the accounts required to
- operate the web and application servers. If this is not possible, the session data needs to be encrypted or contain only non-sensitive data. Note that PHP uses public system temporary directory by default
- All session variables must be validated to ensure that is of right form and contains no

Unexpected characters.Caching

- Prevent client-side page caching on pages that display sensitive information.

Threats summary:

Threat	TANGO
Attacker may be able to read other user's messages - User may not have logged off on a shared PC	User management Rights management Time out for inactive sessions (not a developer concern)
Data validation may allow SQL injection	Impossible : no direct use of SQL
Implement data validation	On the TANGO server side
Authorization may fail, allowing unauthorized access	TANGO Web Monitor is protected against that by the way it uses user groups and attached rights
Implement authorization checks	User rights checking is systematic
Browser cache may contain contents of message	?
Implement anti-caching HTTP headers	To be done by developer
If risk is high, use SSL	SSL is systematic. WEB TANGO is never use as "pure Internet application". It is always an Intranet application.