Table of Contents:

User Manual 1
RTB overview 3
What is it? 4
RTB is a software system that: 4
By using this system you can expect the following: 4
How does it work? 5
Functional vs. Unit testing 6
Quick start guide 7
Installation 7
Operation 8
Command line arguments: 8
Running your first test 9
Using the RTB 10
Main GUI screen 10
Time Based Entries filtering screen 11
Scripts creation screen 12
Operating the RTB at predefined times 13
Writing your first script 14
RTB commands 16
ASSERT 17
CATCH 17
CLEAR_CATCHES 18
CLOSE_SOCKET 18
DELAY 18
GOTO 18
IF_STATEMENT 19
READ_BLOCKING 20
READ_NON_BLOCKING 20
REOPEN_SOCKET 20
RUN_SCRIPT 20
END_SCRIPT 20
SEND 21
SEND_FILE 21
SEND_FLUSH 21
SET 22
START_CLIENT_SOCKET 23
START_SERVER_SOCKET 24
SYSTEM 25
Built-In-Functions (BIF) 26
Mathematical functions and operators: 26
MD5 27
Printf 27
Test environment functions: 28
RTB structure 29
Test Database 30
Voyager 31
BOT 32
Filter 33
Regular Expression engine 33
Expressions Evaluation engine 33

RTB overview

Dear customer!

We are proud to present to you this user manual.
You will find here instructions of how to install, get experience and use the RTB.
We are sure that the RTB will help your company in projects development and maintenance and will introduce to you a valuable, time and money saving methodology.

We consider your feedback as extremely valuable in our product development and refinement process. You are always invited to contact us through our website: http://www.siptele.com/RTB
Or by e-mail: rtb@siptele.com

We will do our best to provide you with the best service and support.

RTB team, SipTelecom

What is it?

RTB is a software system that:

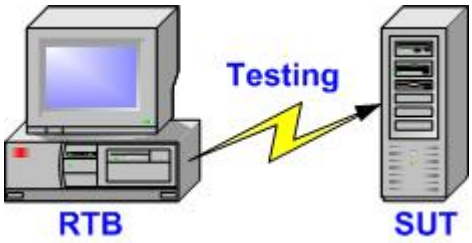
a. Accumulates and manages functional tests.
b. Runs selected tests at predefined times or on-demand.
c. Sends e-mail notification with problems pinpointing, or “all is well” to selected audience.

By using this system you can expect the following:

a. Increase of your systems’ overall up-time.
b. Steep reduce of manual testing to locate problems.
c. Implementation of business methodology for operations’ efficiency and enhancing business successes.
d. (For software manufacturers) Implementing testing procedures resulting in higher project success rates.

How does it work?

As a testing mechanism, the RTB tests the required “System Under Test” or “Software Under Test”.



The RTB opens TCP/IP communication sockets to the SUT and holds predefined sessions with it. The incoming session information is compared to the previous results that were approved as valid (“Gold”) in order to determine whether the test was successful. Any discrepancy in the session conversation, states the test as failed and the discrepancy information is sent to the owner pinpointing the problem.

The purpose of this operation is to:

a. Ensure full functionality of servers.
b. Validate that all changes in the SUT are backward compatible
c. Allow developers to locate bugs in-house rather than on customers’ premises.
d. Increase the overall up-time and business operational performance.

Functional vs. Unit testing

The RTB performs testing methodology that is known as “Functional testing” or “Black Box testing”. In this type of testing we do not have access to the inner structure of the system. We have specification documents that define expected system behaviors. Using this specification we “kick the tiers” to demonstrate the system behavior according to the specification. This methodology is in use to ensure that the system will behave in the same way as it designed, and that nothing can happen that will distract it from the way it is expected to operate. In this methodology, we do not check the actual code robustness and do not trigger the code with boundary conditions.

The other methodology is called “Unit testing” or “White Box testing”. In this methodology we have the system’s source code or blueprints and we can access them directly to check the system. In this type of testing we verify that the source code does not have flaws in it. However we do not check that the implementation is indeed what the designer meant, or what the specification dictates.

Both methods are in extensive use when developing software, and high-tech systems. Using them increases projects’ success rates and regulates GMP – Good Manufacturing Practice.

Quick start guide

Installation

a. Create a directory in your system that will house the RTB.
b. Copy RTB.jar from the cd-rom into that directory.
c. Verify that you have java runtime version 1.4 or later. To do that, open a command shell and type *"java –version*". If you don't see a java response, or you see an older version, please install java runtime on your system. You can find it at the Sun company site (http://www.sun.com), or on your installation cd-rom.
d. Change directory to the selected RTB runtime directory and type the command: *java –jar RTB.jar –e*

[This will start the RTB GUI editor]

e. Run the "Hello RTB world" predefined test.
f. Select RTB root directory for your own tests, fill the setup information by clicking on the root directory in GUI.

Operation

The RTB can be operated in two modes:

a. Interactive – This mode is mainly used for creating and maintaining tests.
b. Batch – This mode is for activating the tests at predefined times.

To start the RTB in the interactive mode type (in command shell):
java –jar RTB.jar –e
To start the RTB in batch mode type (in command shell):
java –jar RTB.jar

This command will run the RTB over the whole test base. You can use command line options to narrow the test population.

Command line arguments:

-t name	Name of specific test or group of tests to be done.
-v	Print version information and exit.
--mail <addresses>	Override RTB setup mail addresses. Used for manual RTB operations.
--nomail	Prevent sending summary mail to audience.
--rtbsetup <file>	RTB setup file. (Default: " rtbSetup.root.xml" in the current directory).
--help	Print usage information and exit.
-e or --editor	Invoke the RTB script editor.
--intermediate	Store the regular expression filtering phase in the output xml file.

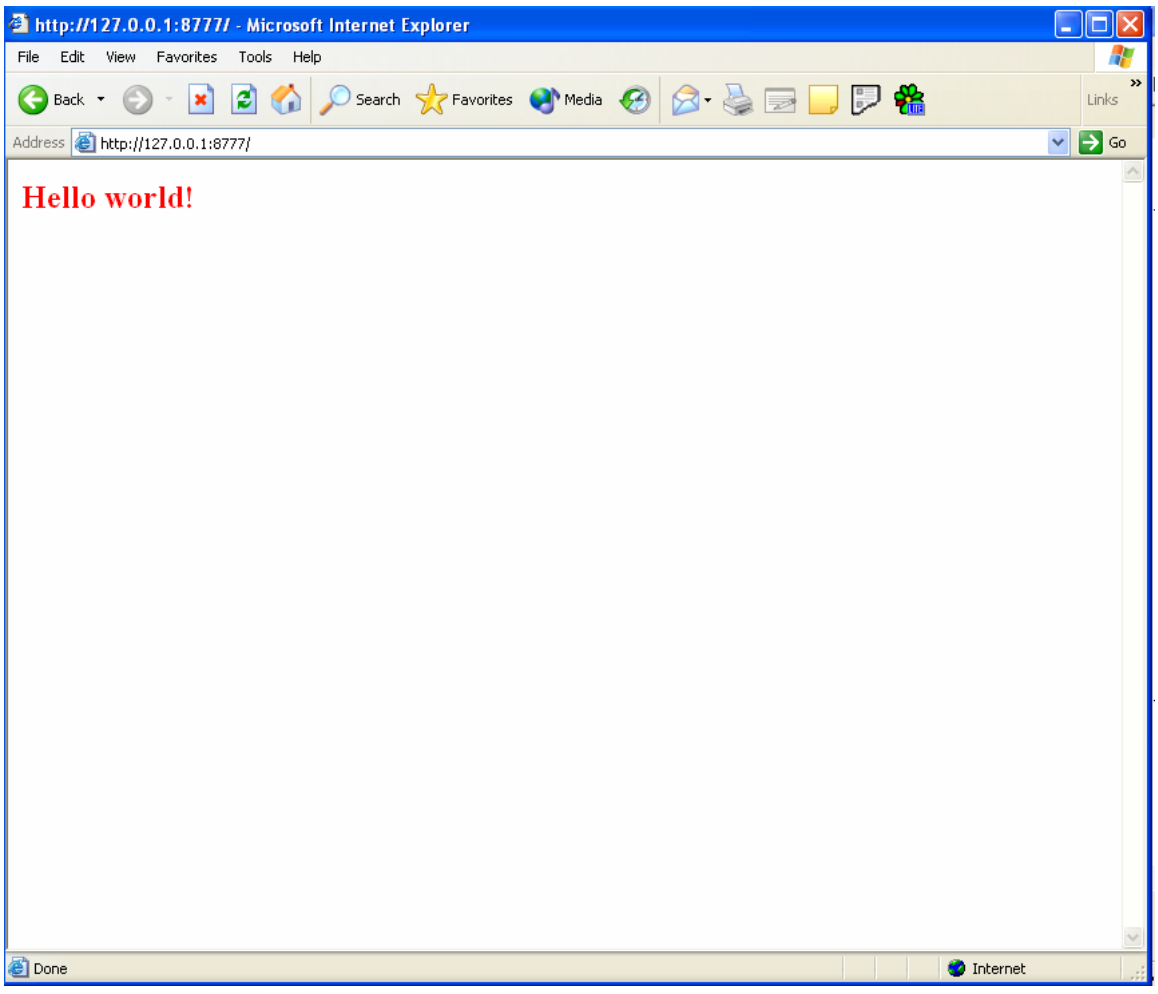
Running your first test

a. Start the RTB by command "*java –jar RTB.jar –e*" in a shell screen. You can see now the opening screen of the RTB.

b. Look for the test: **"your first test.test"** in the tree on the left side and click it.
Now you can see the "General Test Information" screen with three buttons on the top of the screen

c. Click the "Run" button.

A browser will open with the phrase "Hello world".

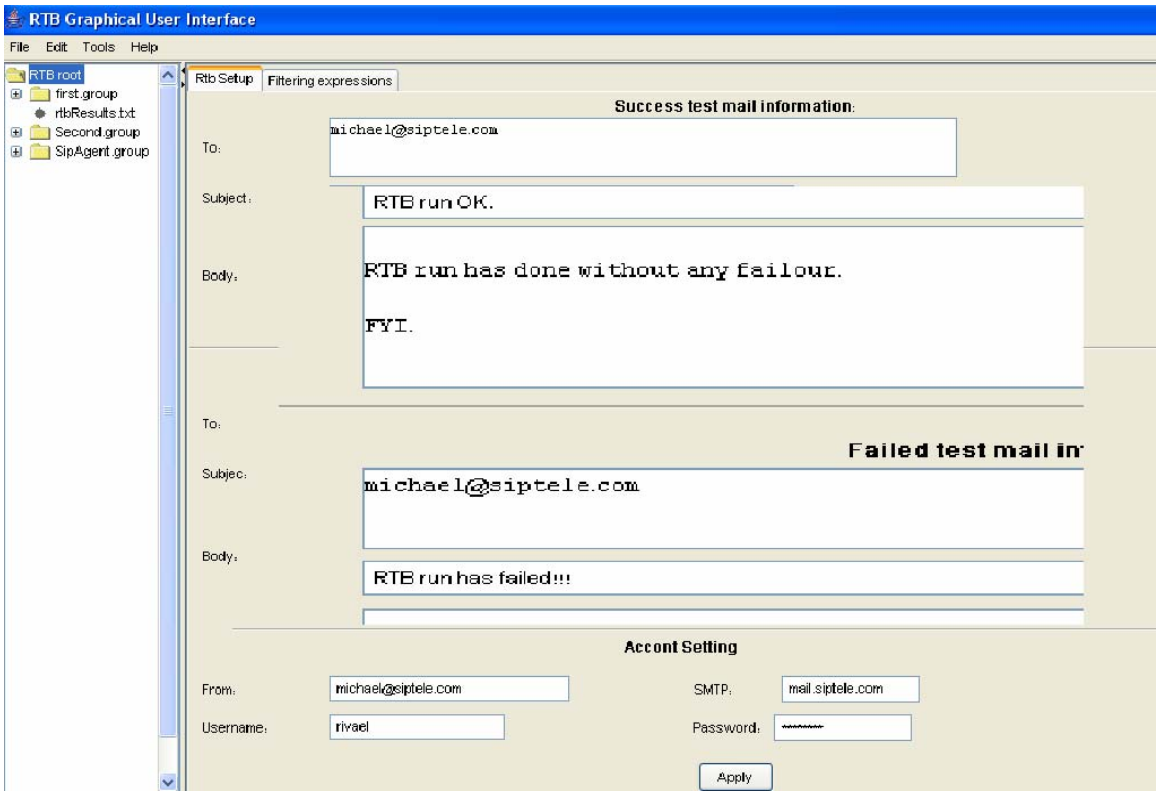


This script demonstrates opening a browser and sending text to it.
When you close the browser you will be able to see the run flow of the test on the screen.
You can look at the log files and the output files to learn more about this test.

Using the RTB

The easiest way to create and run the tests is to use the RTB GUI screens, you will find them user-friendly and easy to use:

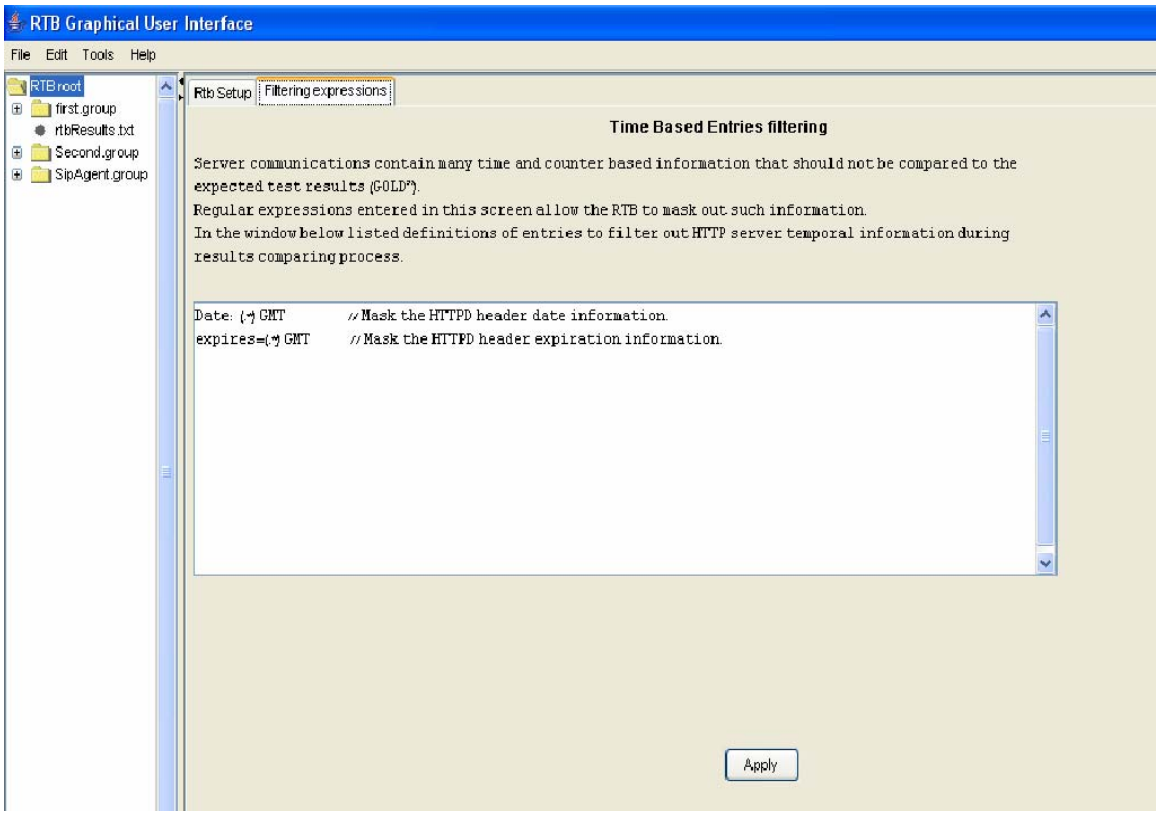
Main GUI screen



This screen appears when you click the “root” of the tree on the left side.
In this screen you can fill your e-mail parameters setting.
The left side contains a test tree. The tree shows your current location in it.
Right-clicking a tree node will open a menu with different operations that can be performed on that node.

Time Based Entries filtering screen

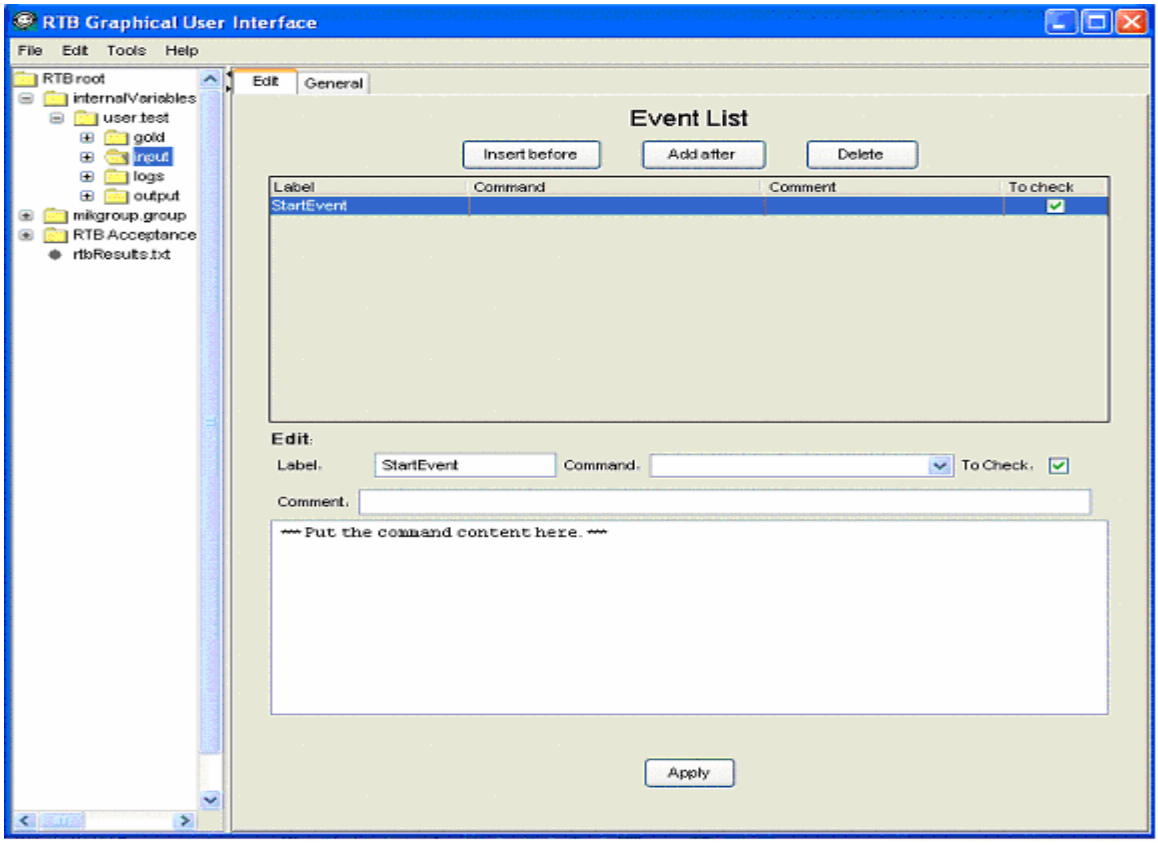
This screen appears when you click the “Filtering expressions” tab on the main screen.



This screen allows the user to define the time and counter entries that should be filtered out from the input stream.
We have put, as default, time expressions that appear in the HTTPD protocol. You will probably need to add some more expressions according to your own system. The entries to be filtered out are identified by regular expressions.
For more information on regular expressions and how to use them, read this tutorial: http://www.regular-expressions.info/tutorial.html

We must filter out such information, as we check the success of a test by comparing it to the previous runs’ test results that are marked as valid (“GOLD”). Unless the comparison of date and time information appearing in the Input\Output is prohibited, the test will be marked as “failed”.

Scripts creation screen



This screen is used to create test scripts. The GUI allows you to create test scripts easily and effectively.
At the top frame, you can see a list of commands in the script. Selecting a command allows you to edit it in the bottom section – the edit area.
The ***label*** field allows you to mark a command with a label. You can use labels as “GOTO” commands targets.
The ***command*** field allows you to select a command from the available commands domain. You do not need to remember the commands – a drop down menu lists the available commands for you. You can type the first letter of the command and the drop down pane will select the next command that starts with this letter.
The “***to check***” entry indicates that IO streams related to this command are to be checked with the predefined “Gold” test results. There are times where initial or pre-test operations are not important and are excluded from the comparison process.

The **apply** button must be clicked in order to save the changes from editing to the script.

Operating the RTB at predefined times

It is recommended to have a nightly RTB run that performs the entire test base and reports the results. This way the next morning, when people come to work, they can be assured that everything is okay.

To run the RTB:

In Linux (UNIX) environment, you need to use cron mechanism.
Use the command:
>crontab –e

A vi editor screen will be opened allowing you to insert information.
Insert the command in the following format:
Minute Hour Day Month Weekday Command
0 05 * * * java -jar RTB.jar –rtbsetup <your rtb setup file>

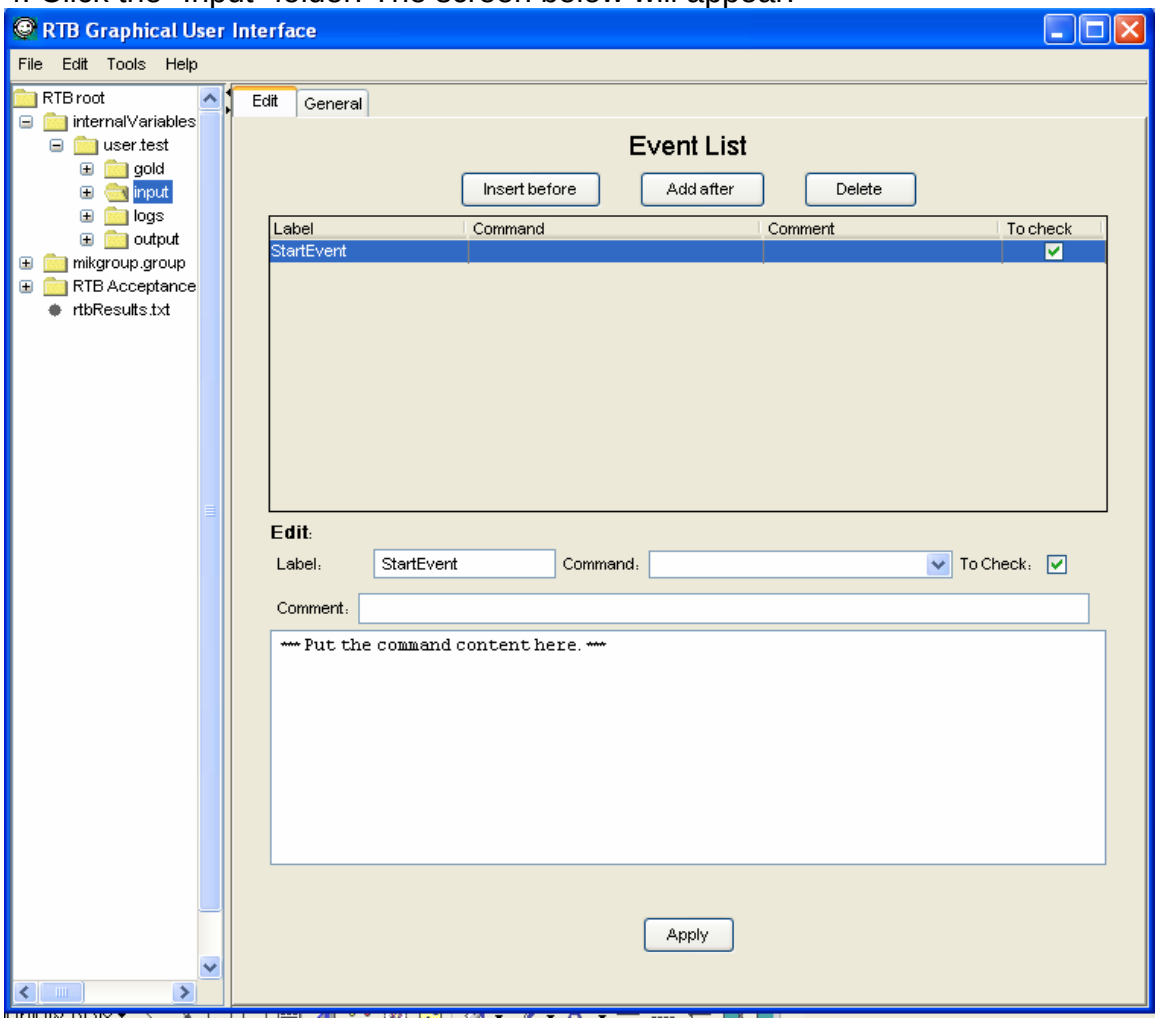
Press (capital) '*ZZ*' to save.
Now you have the RTB run every night at 0:05 AM.

In Microsoft Windows environment:
Click "Start"->"Control Panel"->"Scheduled Task"
Add the RTB using the "add scheduled task" wizard.

Writing your first script

This chapter will guide you step by step in writing your first script.

1. Start the RTB.
2. Right-click the tree root on the left side of the screen and select "new".
Now you can choose between two options:
Test – to create a new test.
Group – to create a directory of tests to be run together.
If you choose to create a group, perform the same as in stage 2, and choose a test (or a sub–group).
3. Name your test/group and save it. Now you can see your directory in the tree. If you extend this directory you will find that it holds four different folders:
Gold – holds the output you declared as "Gold".
Input – holds all your testing scripts.
Logs – holds all log messages sent during test execution.
Output – holds scripts' outputs.
4. Click the "Input" folder. The screen below will appear:



The screen contains the following objects:

- Three buttons to help you organize the commands in script
- A table that holds the script lines
- Command editing area: Label; Command; Comments; Text area

A script always starts with a command labeled "StartEvent". You can easily change this by changing the text in the label.

5. Writing the commands:
   a. Choose the command "system" from the list in the commands box.
   b. Optional – choose a suitable label. The label enables you to identify the line in the script. This is useful if you want to refer to that line later on.
   c. Optional – you can write a comment for this command.
   d. In the text area write the needed parameters. In this case they would be (in windows environment): cmd.exe /C echo "Hello world!"
   e. Click the Apply button.
   f. If you wish to add or delete commands from the script use the navigation buttons.
6. To run the test, click the test icon and the "Run" button.
7. Use the outputs and logs to learn more about the test.

RTB commands

The RTB simulator has an extensive command vocabulary allowing users to create complex testing and scenarios.

This section explains every command.

ASSERT

Parameters: <logical expression> <message>

Verify that the logical condition is true. If not, send the <message> parameter out to explain a discrepancy. The assert command usually fails the test, indicating an abnormal situation.

CATCH

Parameters: <regular expressions>

This command allows users to extract dynamic information from the input to the socket.
The data extracted is preserved as an internal variable and can be manipulated or used in sessions with the “System Under Test”.

This command examines the incoming input stream, and sets internal variables to predefined values. This is the way to interact dynamically with the other side, based on the incoming data.

The command allows you to enter several lines of regular expression groups, in the format:
Regular expression
Action phrase

The action phrase can be:
$<internal variable name>,serial
This command extracts data from the input stream according to the regular expression and stores it in internal variable.

#<internal variable name>,value
This command assigns the internal variable with the value specified on preceding regular expression match.

In the parameter area you have to declare what kind of text you want to catch.
For example: to catch all data write:
(.*)
$var,0
Now you can use <var> to echo the content received to where it came from.

See also: CLEAR_CATCHES

CLEAR_CATCHES

Parameters: None.

Clear the regular expression table. No more data will be extracted from the input stream.

CLOSE_SOCKET

Parameters: <Local port number or 0>

This command closes the socket that runs on specified port number.
You must indicate the desired socket to close. If you select zero, the socket used in the current script is closed.

You should use this command mainly to stop server operation in the RTB.

DELAY

Parameters: <time in milliseconds>

Use this command to stop the test operation for a specific amount of milliseconds.
Use it for synchronization needs or to allow data from an external source to be processed.
Example: To stop the test run for 1 second use:
DELAY 1000

GOTO

Parameters: <label>

This command diverts the execution flow to the <label> location in the script.
When you write a script you have the option to give every command a label.
GOTO command uses that label in order to divert the execution to that command.
You will find this command useful if you want to create a loop in your script.

IF_STATEMENT

Parameters: <internal variable name> <logical condition> < object>

This command evaluates the expression and operates the next script line if the expression is evaluated to true.
You can apply logical condition on a predefined variable

Writing logical condition:

To check if two objects are equal:	"**==**".
To check if two objects are not equal:	"**!=**".
To check whether an object is larger:	"**>**" or "**<**"
To check is one object is larger than or equal to the other:	"**>=**" or "**<=**"
Boolean "**and**" logical condition:	"**&&**"
Boolean "**or**" logical condition:	"**||**"

Example:

To check if i is not less than 5 use: "i >= 5"
To check if i is not 5 use: "i != 5"
To check if i larger than 5 and j is not less than 4 use: "i > 5 && j >= 4".

READ_BLOCKING

Parameters: none.

This command stops the test execution and waits for data to come from a socket.
If no data arrives within a predefined time, a time-out exception is raised.
Use this command in cases of sockets communication.

READ_NON_BLOCKING

Parameters: none.

This command checks if there is any data available for reading, and performs "read" if such data exists.
When using this command, the test is not stopped. You might want to use DELAY before this command, to give time to the other side to send information.
Use this command in cases of sockets communication.

REOPEN_SOCKET

Parameters: None.

This command closes the current socket (If opened) and then reopens it.
Use it in cases when the remote system is supposed to close the connection, and you want to reopen the socket and continue testing.

RUN_SCRIPT

Parameters: <script name>

Run a separate script in the same thread. The current script is halted until the separate script ends. It is convenient to use after an IF_STATEMENT to group several instructions and perform them.

END_SCRIPT

Parameters: None.

End the currently running script.

SEND

Parameters <content to be sent>

This command is used to queue data to be sent over TCP/IP socket. The data is **accumulated and not sent** until the SEND_FLUSH command is used. That is to say, the content you write as a parameter will not be sent yet. This data is saved until the next SEND_FLUSH, which will actually send the accumulated content and the current command data.
If you want to send data immediately, use SEND_FLUSH.
This command is mainly used to ensemble content programmatically and send it on in one piece after the ensemble process is completed.

SEND_FILE

Parameters: <file name>

Opens a disk file and sends its content over the socket. This command is used mainly in order to send binary data such as images.

SEND_FLUSH

Parameters [content to be sent]

Immediately send data over TCP/IP socket. It takes any accumulated SEND command data, appends the data in this command and flushes it.

You can put content in this command to be appended to any accumulated data, or use it without content to flush the accumulated data only.

Conversion of content:

To allow usage of non-ascii values inside the content stream, a conversion function is in use. It takes the content-to-be-sent and inspect it. When it finds the character '\' it converts the following characters into another value:

\\	Converts into '\'
\0	Converts into binary zero
\r	Converts to carriage return.
\n	Converts to new line.
\nnn	Where nnn is numeric characters – converts to their numeric values. (|\123" => 123 binary).
\xhh	Where hh is hexadecimal value. Convert into their binary value.

The reverse operation occurs when data stream is captured using read.

SET

Parameters: <internal variable name> = <value> …

Set internal variables to predefined values. These values can be numeric or strings. There is no need to use ‘$’ sign as the internal variable prefix name. However reference to the internal variable in other commands is required.

START_CLIENT_SOCKET

Parameters: <local port> <remote host> <remote port> <script name>
This command starts a TCP/IP socket and connects it to a remote host.

Local port	Number indicating the local port to be used. Selecting 0 will let the operating system choose any available port.
Remote host	Hostname or IP address of the computer we want to connect to. If we want to connect to the RTB machine use “127.0.0.1” localhost address.
Remote port	The server port we want to use.
Script name	Once the socket is opened, this script is in use to run it.

Additional notes:

1. The script dealing with this socket should be created as follows: In the test directory, right-click the "Input" folder and choose "New test script"
2. Name your new script (the same name as defined to run the socket) and save it.
3. Click the script you created in order to write the commands for this socket.

If you are interested in server–client communication, make sure you create the server first, as the client will get “connection refused” response when there is no server waiting for it.

START_SERVER_SOCKET

Parameters: <port> <script name>

This command starts a server operation and binds it to a certain port in the local machine. You can use it to test client programs.

Port	The server listening port.
Script name	Once the socket is opened, this script name is in use to run it.

Additional notes:

1. The script dealing with this socket should be created as follows: In the test directory, right-click the "Input" folder and choose "New test script"
2. Name your new script (the same name as defined to run the socket) and save it.
3. Click the script you created in order to write the commands for this socket.

If you are interested in server–client communication, make sure you create the server first, as the client will get “connection refused” response when there is no server waiting for it.

SYSTEM

Parameters: <shell commands to be executed>

Use this command to operate shell command or program on the RTB machine.
In Windows environment, use: “cmd.exe /C <command>“ to start the shell.
In Unix environment: one might use “tcsh | sh | csh “ to start the desired shell.

This command is used for operating external testing tools or any test needed to be operated. The voyager compares any files in the “output” directory with the “gold” reference directory. This allows ensuring expected behavior from this command. Make sure that the stdout / stderr or the system command gets written to file in this directory.

Built-In-Functions (BIF)

In the RTB we have several build in functions aimed to help the users create the test cases they wants.

The Built-In-Function can be called in any set command. For example:
Set I = sin(x) calculate the sinus of x.

Mathematical functions and operators:

^	Exponent
*	Multiply
/	Divide
+	Add
-	Subtract
ln	Logarithms of base e
Sin	Sinus
Cos	Cosines
Tan	Tangent
ATOI	ASCII to Integer – convert string representation into numbers.

MD5

This command allows one to encode password according to the RFC2069 – Usage of “WWW-Authenticate” header field to encode username and password into authentication token.

Parameters:

User Name	
realm	Domain name in use.
password	
nonce	Parameter that varies in time.
url	

Usage:

set token = md5(“John”,”SipTele.com”,”John12345”,
"f84f1cec41e6cbe5aea9c8e88d359”,”qms.siptele.com”)

Printf

Allow printing of formatted text in the same way “C” printf operates, with the restriction that only one parameter is allowed.

Parameters:

Format	String representing the desired output format
Variable name	Name of internal variable to format.

Examples:

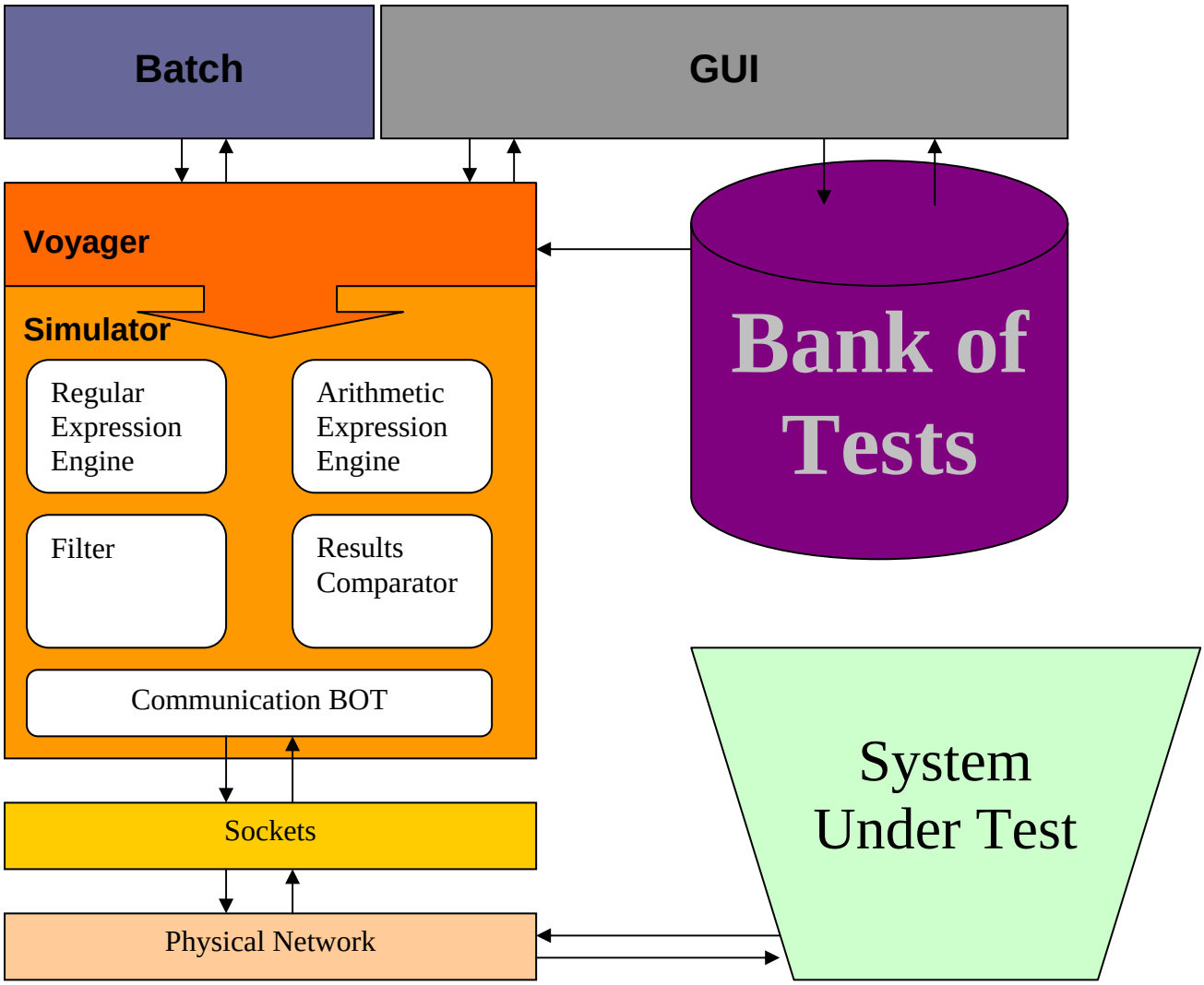
printf("%4.4d",324.123)	: 324.1230
printf("%4.1f",324.123)	: 324.1
printf("%4f",324.123)	: 324.123000
printf("%4e",324.123)	: 3.241230e+02
printf("|%7.1d|",324.123)	: | 324.1|
printf("|%-7.1d|",324.123)	: |324.1 |
printf("|%+7.1d|",324.123)	: | +324.1|
printf("|%+7.1d|",-324.123)	: | -324.1|
printf("|%-+7.1d|",-324.123)	: |-324.1 |
printf("|%07d|",324.123)	: |0000324|
printf("%o",324.123)	: 504
printf("%e",324.123)	: 3.241230e+02
printf("%E",324.123)	: 3.241230E+02
printf("%x",324.123)	: 144
printf("%X",324.123)	: 144

Test environment functions:

isUnix()- Return true if the RTB is run in Unix (Linux) environment.
isWindows() Return true if the RTB is run in windows (DOS) environment.

RTB structure

The RTB is composed from several main elements:



Test Database

The database is saved in the RTB home directory on disk. We keep the following directory convention:

RTB root directory – Contains the RTB setup information.
<name>.group - directory contains other groups and tests. Used to allow grouping of tests and running the RTB on selected groups/tests.

<name>.test - A certain test. The test directory contains the following directories:

a. Input – directory housing tests scripts. The first script to run is “main”.
b. Output – directory containing the results of last RTB run. The output directory content is compared to the “Gold” directory content.
c. Logs – directory containing log files content of the last RTB run. Can be used to debug tests.
d. Gold – directory containing output results that were approved as correct. When performing RTB test, it compares results with the outputs saved in Gold.

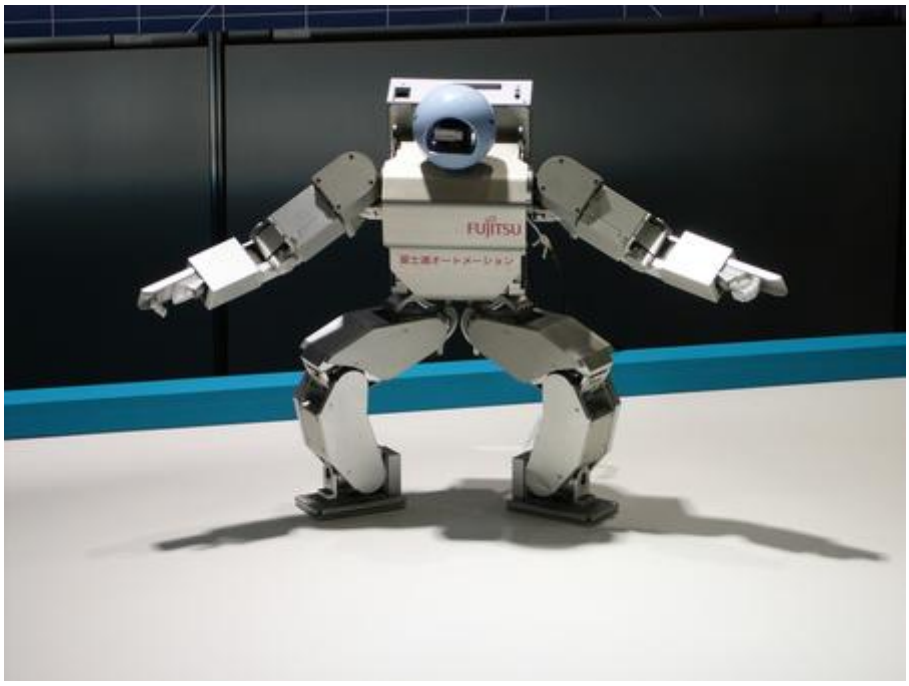
Voyager

This element travels over the RTB database and executes the tests according to the command line parameters.

After conducting all the tests, it sends a conclusion e-mail to the selected audience informing the results.



BOT



This element is the atom of the unit tests. Each test script is executed in a dedicated BOT.
The first main test BOT, does not have communication socket attached to it. Normally, child BOTS have TCP/IP sockets for conversations with the system under test.

The BOT executes the test script, writes log information about its activities in the log directory, and information about its IO activities in the output directory.

Filter

The Filter is used in order to filter out time, counter and other transitory information from the output test results.
The filtering process is necessary because test outputs are compared to previous runs and all temporal information is changed (such as time, counters etc.) and existence of such information will fail the test.
In order for proper verification to take place, the RTB should filter out all temporal information.

Regular Expression engine

This engine allows capturing data from the input test stream and using it in the test sequence. Items as session-id and cookies might be needed to successfully interact with the System Under Test and fully demonstrate complex scenarios.

Expressions Evaluation engine

This engine allows to make calculations inside the test, and to take actions based on these calculations. For example: loop counter needs to be initiated and updated on each iteration.