TTWORKBENCH ONLINE DEMO

Welcome to the online demonstration of TTworkbench.

TTworkbench is a technology-independent tool for testing products and services in a wide range of different industry sectors. This full-featured, integrated test development and execution environment is suitable for any kind of test automation project based on TTCN-3.
You can define your tests with a TTCN-3 core language editor or with an easy-to-use graphical editor including T3Doc. An integrated compiler compiles TTCN-3 modules into executables which can be executed and afterwards be analyzed in detail. TTworkbench also provides a TTCN-3 debugger and a Runtime Plugin Development Environment.

TEST CASE DEVELOPMENT

This demo shows a small example based on SIP, the Session Initiation Protocol. After installation, you find the SIP test suite in the Development View of TTworkbench. On the left hand side you see the TTCN-3 Projects View with all available projects. In the TTCN-3 sub-folder there are the TTCN-3 source modules which are standardized by the European Telecommunications Standard Institute, ETSI. Here, I will take out our SIP example.

Let me introduce some of the features available to help you to develop TTCN-3 source code.
In the Preferences mode, you can select your requested TTCN-3 edition. Testing Technologies makes sure to always be up-to-date with the latest edition of the TTCN-3 language.

To get an overview of all available projects, the Dependency View will be helpful. Go to window -> 'show view-> other. In the TTCN-3 development folder, select Graph module dependencies and press ok. On the right hand side, the Graphical Module Dependency View shows the dependencies of available projects.

After computing the dependencies you will see the complete tree structure of TTCN-3 modules correlating. On top, in the root folder of the SIP solution, you will see how the main module depends on all available TTCN-3 sources. It is possible to decrease the view to see only parts, for instance for the RFC3261 folder which includes standardized SIP TTCN-3 sources from ETSI.

Here on top is the main module and some other modules for call control, registration, querying capabilities and for messaging. All depend for example on the steps module which includes reusable functions for the different test cases. After getting an overview of the modules, you can

open up these TTCN-3 modules with the Core Language Editor. The basis of TTCN-3 is the type configuration which I start now. The generation of an outline helps you to easily step through the source code. Let us choose one of the message types, for example the Invite Request for inviting users to call.
The Invite Request is a record type, a TTCN-3 specific type similar to a struct in C. This invite request depends on several header fields, a request line, message headers and a message body. You can simply increase the Source Code View and jump to a specific declaration, for example the declaration of the request line or the message headers. A request line is apparently nothing more than a structure depending on a method, and a SIP URL. The SIP URL includes TTCN-3 base types like a charstring or user information.

In the next step, you can fill up these types with specific values. Finally, these are the messages to be used for sending, and also for checking the received statements. You see all previously defined TTCN-3 templates here. I will take out one of the templates, which is a parameterized template, and include specific constant fields, like the SIP name version, or hard coded fields like the integer value for forwarding the message.

Some fields are set to omit, if they are not relevant in this case. For others, the mandatory parameters are assigned or sub-templates are defined. To change a template or a message data structure, TTworkbench provides a GUI, the message building system, to change and edit TTCN-3 templates, or to build new TTCN-3 template structures.

Just fill in the values you might want to change. If you want to change the SIP name version for example, just enter the number, let us say 2.0, and press next. You see this change directly in the template structure. It is also possible to build a new template from an existing type. Click control + space to see all available types. Take for example the Invite Request Type, and generate the necessary values from here.

Another available feature is the code highlighting. Users can individually choose the formatting and the syntax of coloring, specific keywords and source code.

Last but not least, let us have a look at the test case behavior required for execution. I will take a very simple test case that just sends out an invite request and expects a preliminary receive message, a 180 Ringing, or a final message, a 200 OK, to establish a VoIP call. Here, you see the TTCN-3 core notation of this test case. There is some initialization at the beginning: Several keywords like ports which declare the interfaces to the IUT, and a template reference for the message that will be sent out or received.

Here you see the invite request reference as a template structure, just introduced before. Now, a timer will be started followed by an alternative behavior of what is expected from the system under

test. The first response branch in the alternative is supposed to receive a message with a status code integer value between 100 and 199. This is a template structure which includes some kind of inheritance from an originally defined default template. And here you see the TTCN-3 wildcard mechanism for the reasonphrase ?a question mark. It must be there, but a detailed value is not of interest.

Another view of the TTCN-3 source code can be shown with our graphical editor. There are two different views, the textual editor and the graphical editor to generate the graphical format of TTCN-3 used for documentation or better overview. Within this editor you have a full round trip engineering possibility. On the one hand side you can define test cases from scratch, in a graphical way. Then you can switch to the textual notation and change your test cases textually. Afterwards you can load them up in the graphical notation again to make modifications.

Now I will import the TTCN-3 core notation to show you the import mechanism. I will take only the first test case introduced before. TTworkbench is now analyzing the complete source code. It provides the complete type system and the template structures already available, with the same group structure given in the core language editor. By selecting the first test case, a diagram from the textual notation is automatically being generated.

To see the same representation of this test case in a graphical way, just open the graphical editor. You see the invite request for sending out the message, the start of a timer and the definition of several alternatives. If the alternatives match, meaning the response from the system under test is the expected one, the test case passes.

In some cases, the verdict is set to fail at the end. Below you see the test case as TTCN-3 core notation. If you want to define your own test case, just give it a new name, for example test case 1. Now you can choose between all available templates. Change not to send out an invite request but an option request, and receive an acknowledgement. The TTCN-3 core notation will be changed automatically. Now you can export the TTCN-3 core language back into the editor. I just change the TTCN-3 file, and open it in the core language editor. Here, you see the test case name which was changed, plus changed send and receive event.

TEST EXECUTION

After developing the TTCN-3 source code, the generated test cases can be executed. Before execution, the TTCN-3 source code has to be compiled into executable Java classes. TTworkbench provides three buttons, the build, rebuild, and validation button. If you have changed only little things, use the build button to compile only changed modules. Press the rebuild button to compile all modules from scratch. The validation button is used to validate the TTCN-3 code without

generating the executable jar files.

Before you can start to execute test cases it is required to generate a test campaign loader file. The test campaign loader file is an intermediate format written in XML for the execution management. It includes all references of available test cases and module parameters, the parameters the user can change directly at runtime.

I have already prepared a test campaign file called SIP test. When you double click on it, TTworkbench automatically jumps into the execution view and opens up the file in the management perspective. There are several ways to generate a test campaign file. It is possible to generate a default one for all available test cases, or to generate only individual test cases inside a new test campaign file. You can store the configuration you previously defined.
Test cases are now available in the execution management. There is a selection of 10 test cases from the terminating endpoint group, where the test system starts to establish calls and the SUT (System Under Test) answers. When you click on the root directory, you see all available parameters in the parameters view below. They can be changed individually. At these parameters you can change the under lying protocols - UDP, TCP, TLS - and include specific IP addresses and domain names for the configuration of the test.

In this example we use loopback IP address 127.0.0.1 because the system under test runs on the same machine as our test system does. We distinguish in two different ports, port 5060 and port 5061. In case your system under test runs on a different machine just include the correct IP address of the test system and change the IUT IP address parameter to the IP address of the machine the system under test runs on.

There are further configuration options for the SIP messages itself: include the domain name in the request URI, in the FROM- and the TO fields. Or register the User Agents at the proxy server (if you are running proxy test cases). There are also default timer settings from RFC 3261 that can be changed.

In the SIP messages group, answer questions about checksum usage (e.g. AKA V1), authentication, or if the User Agent is required to authenticate at the server.

The SDP body for RTP streaming can be changed individually too. In addition to changing parameter values directly in the parameter view, TTworkbench provides a message building system GUI, where required fields can easily be changed. So, if you want to change the SDP body you can do this individually for the required IP address and for the required RTP codec which your system under test understands.

After configuring these parameters, let us have a look at the Properties View. You see several

statistics e.g. the number of test cases and what kind of test adaptation is being used. Now, I start the system under test, in this case a simple IP phone, to test its conformance against the first 10 test cases. All test cases now run one after the other. Here you see the online graphical logging which is promptly being generated for the specific test cases. This solution is also being provided in a command line version enabling you to run regression tests scheduled at a specific time. Let them run over night and see the results in the next morning.

TEST ANALYSIS

After execution, you see passing and failing test cases in the Management View on the left hand side. The green bubble shows that everything is fine, the red one points out a fail. Now you can analyze in detail the graphical logging below to find out what went wrong in case of a failure. The first and the second test case were ok, nothing went wrong. But let us have a look at the description of the third test case. The test purpose says that the system under test is supposed to send back a 404 not found response in case of sending an invite request with a request URI that includes an incorrect address or an address which is not known.

If you double click on the message for sending, it is displayed in different views. This is the Dump View, where you see an invalid request URI requiring a 404 not found as answer. Another view is the Hex Dump View showing the message in Hex notation or showing the message directly in the data structure like it is represented in TTCN-3, with user type, specific name and values.

What we received from our system under test, is first of all a 100 trying. You see the assigned matches and mismatch boxes highlighted in blue. TTCN-3 has a check mechanism for all events happening. In this case it checks the receive event of a 100 trying in the queue. It evaluates from top to bottom all branches of the TTCN-3 alternative until it matches. Here, there is a first mismatch on the 404 not found, so the first template in the branch of the set of alternatives is not matching. But the second one matches against the 100 trying message.

TTworkbench enables you to directly jump into the source code showing you the respective step. In our call control module, we are at the response of 404 not found. The second branch of the alternative is used to capture out the 100 TRYING responses. TTCN-3 provides this filter mechanism to filter out messages of no interest. In this case, TTCN-3 uses the repeat statement to reuse the alternative again. The next message arriving at the queue was a 180 ringing, which matches correctly. The third message received was a 200 ok. To check what went wrong, I just click on the first mismatch box. Now you can see in the Test Data View where exactly the error occurs. Instead of receiving an expected data structure with a 404 not found we received a 200 ok. The call is shut down here, and in the end our test case fails.

You saw that within TTworkbench, you can easily jump from the graphical logging to the TTCN-3 source code. The tool provides a nice overview of matching and mismatching operations, of received messages against expected TTCN-3 templates. You can see directly what went wrong, not only for the first integer value but for all kinds of defined values.

SAVING TESTS AND RESULTS

In addition to analyzing test cases you can store the complete graphical logging and all changed parameters in a ZIP file to be used at a later date. You get the statistics on how many test cases you have executed, how many passed, how many failed and so on.

You can also generate test reports from a current run or from selected test cases or log files in HTML, PDF, Excel or Word formats. You can include the graphical logging, open the report after generation, and forward it directly via Email. It is also possible to create individual test reports, for instance by selecting only failed test cases. Here you can define properties for the test report: report number, report date, company, test lab and system under test. All these properties can be individually configured with your company logo and so on.

By pressing finish, TTworkbench starts initializing and generating a test report now for our ten test cases, with JPG files from every graphical log. Afterwards you can look at the report to see a pie chart, the number of test cases passed and failed, report number, date. Below, there is a list of all executed test cases. You see the time stamp, when it was executed, the test case name, the purpose description and if it passed or failed. Click on the test case name link to open a picture of the test case behaviour. All graphic files are linked in this report.

PLUG-IN GENERATION

With the TTworkbench plug-in mechanism users can add or change the used test adaptation. They can individually change the codec implementation for encoding and decoding messages, or change other features, for example include external functions. The Runtime Plug-in Development Environment (short RPDE) can be enabled in the settings of the test adaptation. Here you see the built-in test adapter which is a plug-in test adapter for SIP. This is the enabled codec, the already implemented codec, and here you can address your individual codec implementation.

You can also enable external functions and define new port plugins. Our SIP solution provides the UDP, the TCP and the TLS port plug-in implementation. If you require SCTP or other underlying

protocols, just write the source code into a Java class, or via our language plug-in in other languages like C or C++. Afterwards you are able to easily add the sources as plugins inside these settings.

In addition, you can change the parameter server settings or add SUT actions.If, for example, the system under test has to be triggered at the beginning for sending out messages or a proxy server has to be configured, you can implement these SUT action events here. Just define them according to your individual requirements and enable them in the test adapter settings. Within TTsuite-SIP, there are only up to 3 users enabled by default, interacting with each other here in the test parameter configuration. To enable more than 3 users, just define them in TTCN-3 as component instances, provide ports, and declare them in these parameter configurations.

Testing Technologies provides a wide range of plugins for particular access technologies, protocols, for performing external tasks, and for importing additional languages. All available plugins can be freely combined which each other.
Testing Technologies