

My First TTCN-3 Project with TTworkbench

A first steps guide including instructions for download and installation, detailed description of functionalities, test case behavior, test case execution, and evaluation of test results



Testing Technologies

My First TTCN-3 Project

This user's guide gives you a clearer understanding of TTworkbench and TTCN-3 and helps you to easily start using test tool and test language. Besides instructions for download and installation you will be guided through a complete TTCN-3 project starting from scratch - with a detailed description of functionalities, test case behavior, test case execution, and evaluation of test results.

Contents

1. Requirements	2
2. Download	2
3. Installation.....	2
4. AddressBookExample Intro	4
5. TTCN-3 Source Code Development*	4
5.1. Core Language Editor (CL Editor).....	4
5.2. Test Case Definition Analysis.....	9
5.2.1. Test Case TC1 Details	9
5.2.2. Test Case TC2 Details	10
5.3. TTCN-3 Compiler (TTthree)	10
6. TTCN-3 Execution Perspective	10
6.1. Campaign Loader File Generation	10
6.2. Using Module Parameters.....	11
6.3. Running Test Cases.....	11
6.4. Checking Test Cases with Graphical Logging and Test Data View.....	12
6.5. Loading and Saving of Test Campaigns and Logging	13
6.6. Generate Test Reports.....	13
7. Test Adapter Code.....	13
8. Codec Code.....	14
9. Plugin Concept**.....	15
10. Download and Installation of Available Plugins.....	15
11. Appendix.....	16

Please note!

Chapters marked with * are only applicable to TTworkbench Basic or Professional. Chapters marked with ** are only applicable to TTworkbench Professional.

Support

If you need any assistance, please contact our customer care department:

Mr. Dirk Borowski

Phone: +49 30 726 19 19 0

Email: ttn3-support@testingtech.com

For bug and error reports please use our ticket system at <http://support.testingtech.com>.

1. Requirements

Operating Systems: Windows 7 (32bit or 64bit) / Vista / XP / 2000
 Linux Red Hat - v7.1 (x86/GTK)
 Linux SuSE - v9.1 (x86/GTK)

Memory: 2 GB (4 GB recommended)

Java 2 Platform (JRE): Version J2SE 6.0 (1.6.x)
 Download at www.oracle.com/technetwork/java/javase/downloads/index.html
 Please note that we **strongly** recommend to use the above Java JDK. With the OpenJDK/IcedTea for Linux the TTworkbench license will **not work correctly**.

Reference ID and License File

Before you download TTworkbench, make sure you received a valid Reference ID and license. Otherwise please contact our sales team at sales@testingtech.com.

2. Download

Please use Testing Technologies' download portal at www.testingtech.com/support/downloads.php.

Step 1

Select the latest version of TTworkbench Express, Basic or Professional for your platform.
 Please note: TTworkbench 32bit is NOT compatible with Java 64bit. However you can install TTworkbench 32bit or 64bit on an OS of either 32bit or 64bit.

Step 2

Enter your download reference ID.

Step 3

Download/save file in your favored directory.
 Please note: The Microsoft Internet Explorer saves the linux version (*.jar file) as *.zip.
 Just rename it back to *.jar.

Step 4

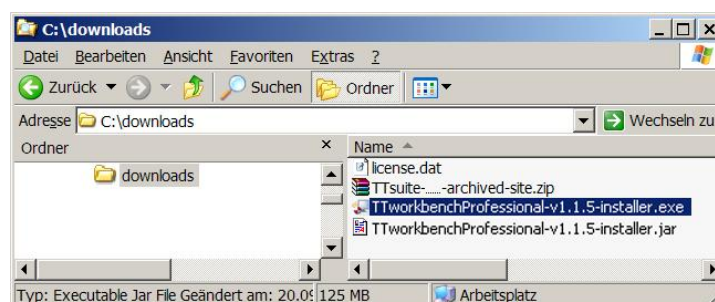
Save the license file [license.dat](#) in your favored directory.

3. Installation

Step 1

Windows Platform:

Double click on [TTworkbench-v1.x.x-installer.exe](#) to be found on your desktop or in selected directory.



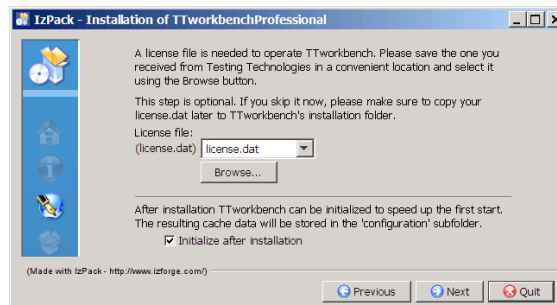
Linux Platform:

Use command line `java -jar TTworkbench-v1.x.x-installer.jar`.

Step 2

Follow the pop up installation wizard...

After request, browse for the valid license file [license.dat](#) already saved in your favored directory. (TTworkbench requires a valid license file for execution, which was sent to you by mail.)



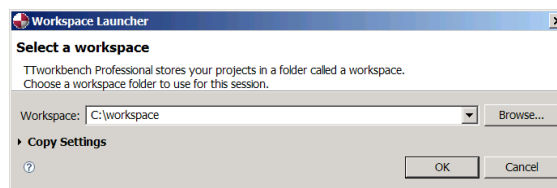
Step 3

Start TTworkbench from created desktop icon or menu entry.



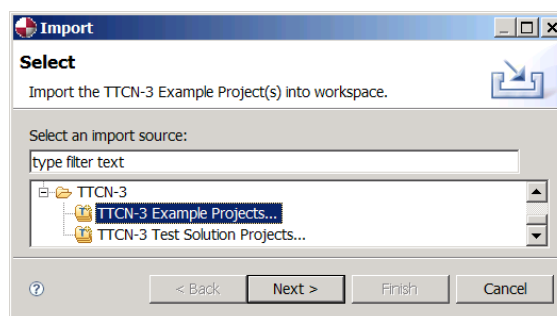
Step 4

Start a new workspace by accepting the default workspace location after request or choose an existing one.



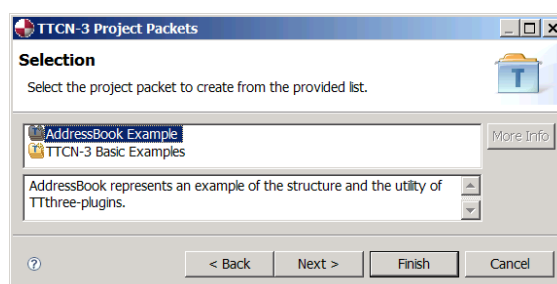
Step 5

Click on the menu item *File*, choose *Import...* and select *TTCN-3 Example Projects...* to import them.



Step 6

Select *AddressBook Example* and follow the instructions. Afterwards you will find the provided example as tree structure in the Package Explorer.



Further details about installation you can find in Chapter 2 of the integrated Users Guide. Click on: *Help* → *Help Contents* → *Testing Tech TTworkbench Users Guide*.

4. AddressBookExample Intro

This example is a simplified version of testing an addressbook database. It sends specific ENTRY messages to this database which acts as SUT (System Under Test). The ENTRY message contains user information like surname, first name, gender etc. The database answers with a reply which includes exactly this message.

Within this simple example, the test case sends a specific ENTRY message via UDP and will receive the same message in the next step. Here we use loop back IP Address 127.0.0.1 and the same port (6061) for sending and receiving the messages. In this case, no real SUT is required. The direct enqueue of the message after sending emulates the SUT itself.

There are two different test cases available to demonstrate a verdict 'pass' and a verdict 'fail' behavior. The 'fail' behavior happens if an incoming message differs from the expected one. Here, we use one template as send message and a different one which represents the expected answer (receive template).

5. TTCN-3 Source Code Development*

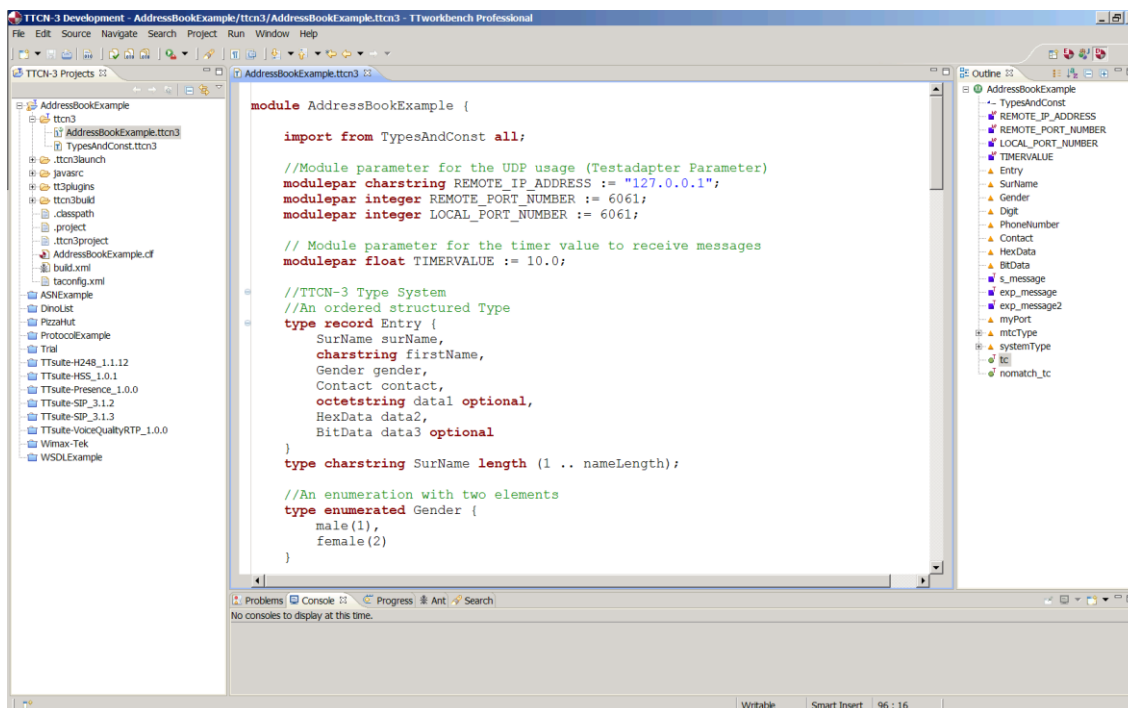
5.1. Core Language Editor (CL Editor)

Every TTCN-3 project contains a *ttn3* folder where all TTCN-3 code modules are stored.

Our **AddressBookExample** project contains two modules within the files: the **AddressBookExample.ttcn3** and **TypesAndConst.ttcn3** to be found in the *ttn3* folder.

STEP 1: TTCN-3 Type Description

To edit and view the source code, double click on *AddressBookExample.ttcn3* in the TTCN-3 Project View. The code will be shown central in the CL Editor.



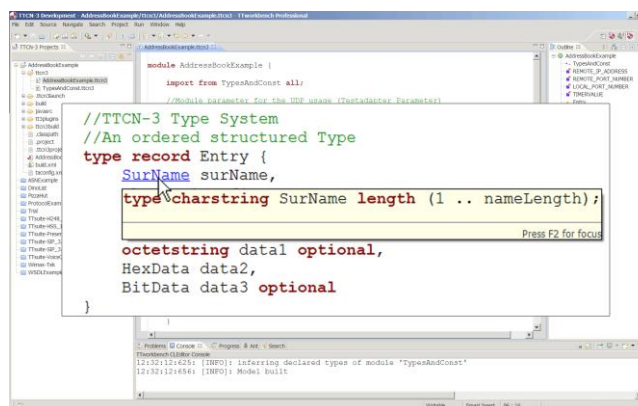
In order to implement a TTCN-3 test suite, you need to define a type system first, matching the data structure of message flows or API calls you would like to test. This type system should either depend on an existing type structure like BNF or ASN.1 or is structured on protocol message requirements. In TTCN-3, there are several structured types available (**record**, **set**, etc.) to put data in ordered or unordered sequences.

Here we define an ordered structure type called **Entry** including seven fields. Two of these fields are predefined TTCN-3 base types (**charstring**, **octetstring**) while the others are self defined types (SurName, Gender, HexData, BitData) shown in following subtype declaration.

```
type record Entry {
    SurName surName,
    charstring firstName,
    Gender gender,
    Contact contact,
    octetstring data1 optional,
    HexData data2,
    BitData data3 optional
}
```

STEP 2: Running Through TTCN-3 Source Code

To open specific declarations, press **Control** and hold your mouse on the **SurName** statement type. Click left mouse button.



The **SurName** type is a restricted charstring type with a length restriction from 1 up to 20 characters.

```
type charstring SurName length (1 .. nameLength)
```

The **nameLength** constant is declared in a separate module (**TypeAndConst.ttcn3**) and known in current **AddressBookExample.ttcn3** module because of the import mechanism in the first line of the code **import from TypesAndConst all**.

Gender is a limited value list (enumeration) with values **male** and **female**. The represented integer values in round brackets after male and female (see below) can be used for ordering purposes.

```
type enumerated Gender {
    male(1),
    female(2)
}
```

Contact is a choice of **PhoneNumber** or **email** address (union type).

```
type union Contact {
    PhoneNumber number,
    charstring email
}
```

HexData is a restricted hexstring type with a length of exactly five hexadecimal values.

```
type hexstring HexData length (5)
```

BitDate is a restricted bitstring with a length of exactly ten bits.

```
type bitstring BitData length (10)
```

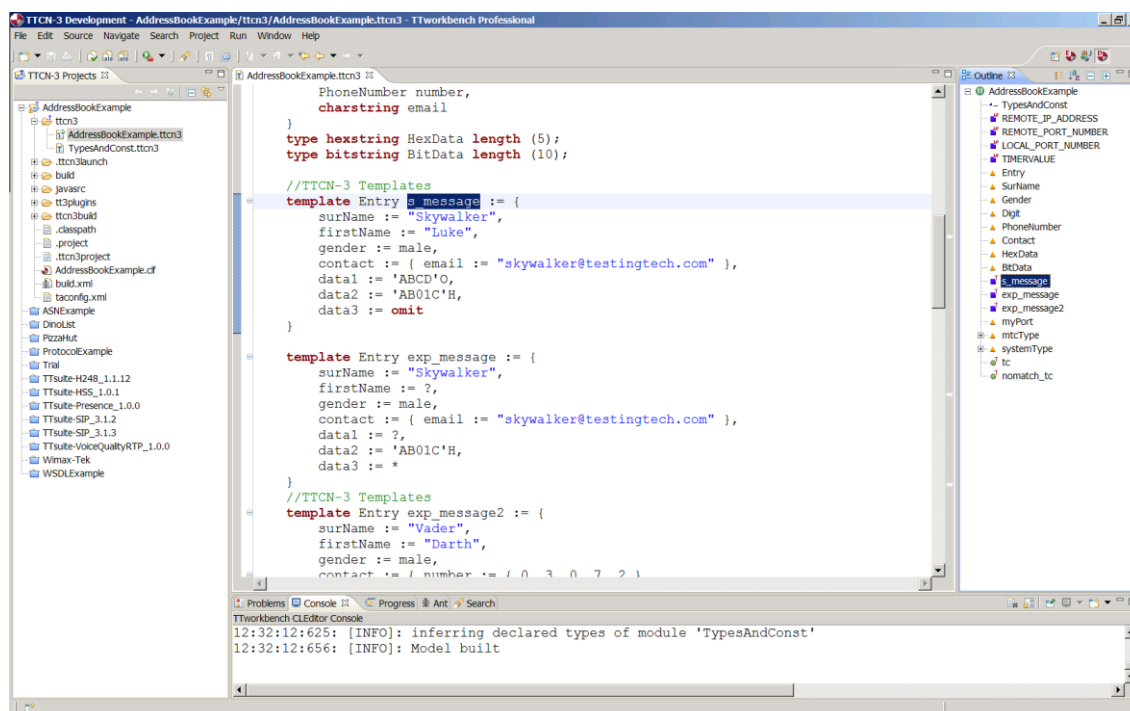
All other subtypes can also be accessed. Click on *Declaration* and press the button **F3**.

STEP 3: Template Definition

After defining your type system in TTCN-3 you need to declare the required data structure for send messages and expected receive messages.

This will be done by a so called TTCN-3 template definition filling up the types with values. There are two different templates available in the example test suite – send and receive templates.

On the right hand side of your screen you find an outline with a list of specific criteria to ease up the search within TTCN-3 source code. Start searching for the blue square with a red T (**s_message**). Click on **s_message** in the outline to jump to the template definition.



For defining send templates only concrete values are feasible:

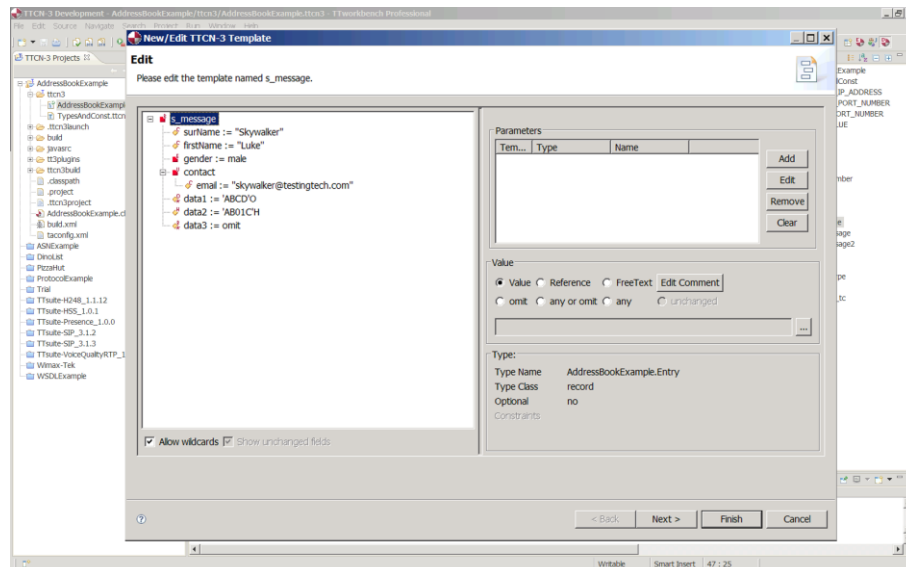
```
template Entry s_message := {
  surName := "Skywalker",
  firstName := "Luke",
  gender := male,
  contact := { email := "skywalker@testingtech.com" },
  data1 := 'ABCD'O,
  data2 := 'AB01C'H,
  data3 := omit
}
```

For defining expected data in receive templates concrete values as well as wildcards are allowed:

```
template Entry exp_message := {
  surName := "Skywalker",
  firstName := "Luke",
  gender := male,
  contact := { email := "skywalker@testingtech.com" },
  data1 := ?,
  data2 := 'AB01C'H,
  data3 := *
}
```

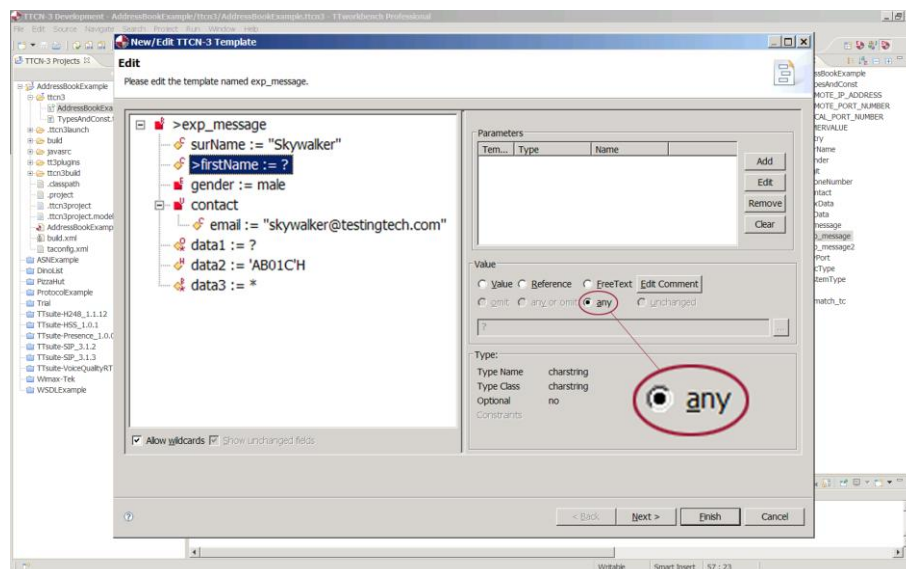
STEP 4: Message Building System (MBS)

Changing templates can be done directly in the code or much easier via the message building system (MBS) by just filling out values instead of dealing with TTCN-3 syntax.



To change values, click on the template name **exp_message** and press **F4**.

After the MBS Editor popped up, click in the tree structure on **firstName := Luke** and change the value **Luke** to **?** by choosing the radio button **any**.



Press **Next** and **Finish** to get a modified template like below.

```
template Entry exp_message := {
  surName := "Skywalker",
  firstName := ?,
  gender := male,
  contact := { email := "skywalker@testingtech.com" },
  data1 := ?,
  data2 := 'AB01C'H,
  data3 := *
}
```

The expected template changes in case you do not expect the exact value **Luke** as first name but you are expecting any (?) kind of name. The asterisk (*) assigned in field data3 stands for any or none value. In this case, it does not matter if there is a value given in the expected message or not.

To define complete new templates you can use the MBS too. Click on a type definition and press *Shift F4*. The MBS will pop up and you can fill in individual values for every type and subtype.

STEP 5: Special Test Types

In addition to regular type system definitions you use for data declaration there are special test types like ports and components given in TTCN-3. You see the declared interfaces connecting to the SUT via ports.

Here, the interface is described as definition of message-based ports which allow exchanging type data without a restriction. All message types can be sent (**out**) and received (**in**) by using **inout** via this port.

```
type port myPort message {
    inout all;
}
```

Components are used as means to run test case behavior on. It is required to define at least one test component to emulate the test system entity.

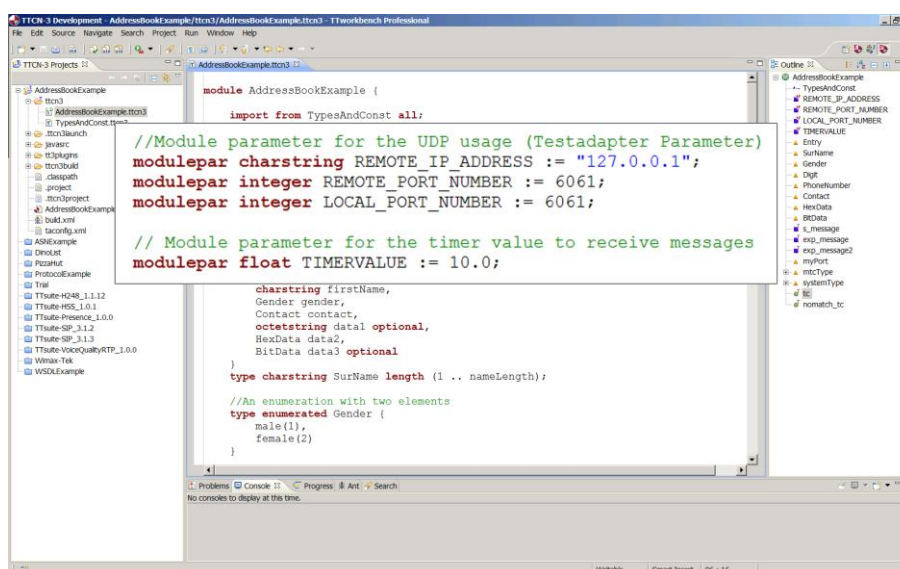
```
type component mtcType {
    port myPort mtcPort;
    timer localtimer := TIMERVALUE;
}
```

The test system entity defines a component type to be used as main test component (**mtc**) later on, providing a port and a timer able to execute the test behavior. The system component to emulate the SUT entity defines a component type that will be used as SUT representation later. Here, the same port type as on **mtc** will be used as **systemPort**.

In this specific case we just got one port type. For dynamic configurations, different declared ports can be mapped to different Test System Interface (TSI) ports (see 5.2.1 Map Event).

```
type component systemType {
    port myPort systemPort;
}
```

STEP 6: Module Parameters



```
modulepar charstring REMOTE_IP_ADDRESS := "127.0.0.1";
modulepar integer REMOTE_PORT_NUMBER := 6061;
modulepar integer LOCAL_PORT_NUMBER := 6061;
modulepar float TIMERVALUE := 10.0;
```

These parameters are used to change configurations outside of TTCN-3. They will be shown later with their default value in the Management Perspective of the execution management and can be modified by the user. Here, they are used for defining the local and the remote port number which need to be the same to enqueue the sent message directly back in.

IP address is the loopback address valid on every machine.

In case you are using a real SUT (e.g. an echo server not included in this product) which runs on another machine you should change the IP address and the port numbers accordingly.

5.2. Test Case Definition Analysis

The test case starts with

```
testcase tc() runs on mtcType system systemType {
... ..
}
```

After the keyword **runs on** stands the main component type reference. After keyword **system**, the system component **systemType** reference is given. It declares which test component you are using to run the test case on and which component represents the SUT interfaces. Within the curly brackets you find the test behavior itself, showing several test statements.

5.2.1. Test Case TC1 Details

Map Event

Using the map event is required to assure that the **mtcPort** (for **mtc**) and the **systemPort** (for **system**) are connected to each other and are able to send and receive messages to and from the SUT (**system**).

```
map (mtc: mtcPort, system: systemPort)
```

Sending a Message

Sending a message is done by using the send keyword together with the template reference. The s_message template is sent from **mtcPort** via the **systemPort** to SUT.

```
mtcPort.send (s_message)
```

Starting a Timer

To prevent blocking TTCN-3 code you have to use a timer which is already declared within the **mtc** instance and can be used directly. Default assignment of the module parameter is 10 seconds. To start a timer, just use the **start** keyword.

```
localtimer.start
```

Defining Alternative Expected Behavior

It is possible to create a set of expected messages by defining alternative (**alt**) branches. The example shows three branches of possible reactions. Every incoming event will be checked against the branches from top to bottom until it matches.

```
alt {
(1) [ ] mtcPort.receive (exp_message) {
    localtimer.stop;
    setverdict (pass);
}
```

```
(2)    [ ] mtcPort.receive {
        localtimer.stop;
        setverdict (fail);
    }

(3)    [ ] localtimer.timeout {
        setverdict (fail);
    }
}
```

1st match: If the expected message is received, the verdict is set to **pass**.

2nd match: If any kind of message is received except the required first message, the verdict is set to **fail**.

3rd match: If a timeout occurs and no message is received within 10s, the verdict is set to **fail**.

Unmap Event

The disconnection of ports is handled in the unmap event at the end of the test case. Like in the map event, the required test component (**mtcPort**) and test system interface port (**systemPort**) are given pairwise.

```
unmap (mtc: mtcPort, system: systemPort)
```

5.2.2. Test Case TC2 Details

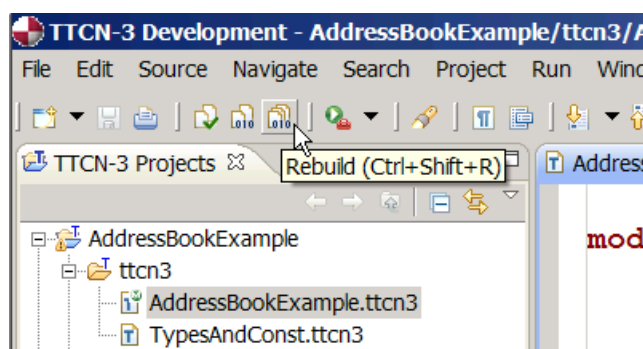
In TC2 we are expecting a different answer (**exp_message2**) than in TC1 (**exp_message**).

In TC1 the message sent out matches against the expected message. TC2 differs relating the expected values. For instance surname **Skywalker** was received but surname **Vader** expected.

5.3. TTCN-3 Compiler (TTthree)

STEP 7: Compilation of Source Code

After defining the source code you need to compile the TTCN-3 file to make it executable. Please use the *Rebuild Compiler* button above the Management View while the TTCN-3 source is displayed in the Core Language Editor Perspective. The compiled sources will be generated in the *ttn3build* folder.



6. TTCN-3 Execution Perspective

6.1. Campaign Loader File Generation

STEP 8

There is a predefined campaign loader file (***.clf**) available for running the two test cases. This ***.clf** contains information about TTCN-3 executables being used (compiled source code) and test cases and module

parameters being available. Double click on this file to open the Execution Management Perspective showing the loaded, ready to run test cases.

6.2. Using Module Parameters

STEP 9

Click on *AddressBookExample* root tree element to show the parameters in the Parameter View (below Management View).

REMOTE_IP_ADDRESS: IP Address of SUT.

This IP Address will be used as receiver address in the IP packet, i.e. the IP address of the server.

REMOTE_PORT_NUMBER: Port used as receiver port in the IP packet.

LOCAL_PORT_NUMBER: Port used as sender port in the IP packet.

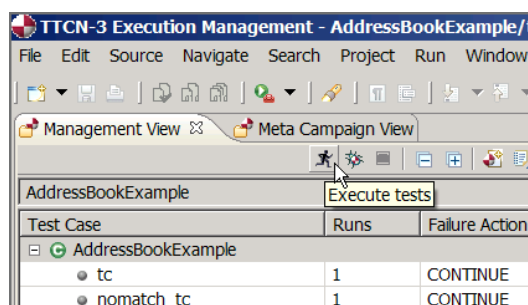
TIMER_VALUE: Configurable time frame to receive messages.

Parameters are configured by default to be used within our example scenario: The test suite sends out the message and reacts as built in SUT by receiving it internally. Loopback IP address 127.0.0.1 and the same port 6061 are used. The timer value stands for the time frame within the message should be received from the SUT.

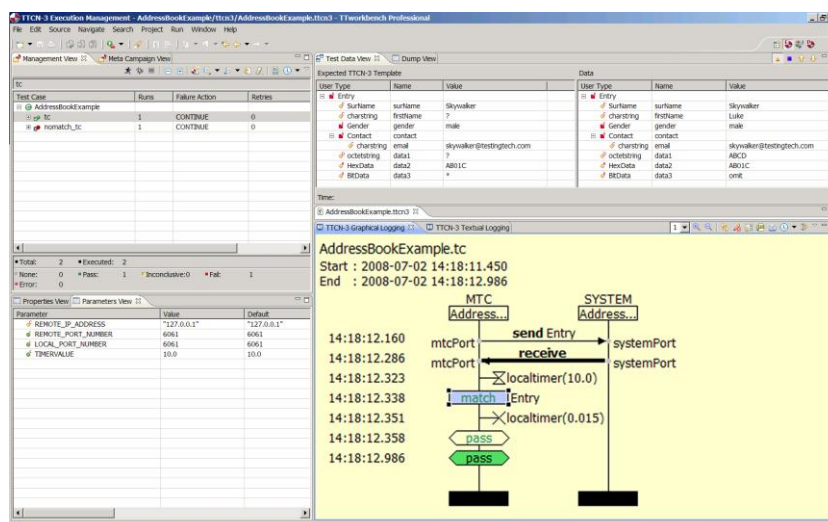
6.3. Running Test Cases

STEP 10

There is no need to modify any parameter before running the test cases. Everything is already configured in this example. The available test cases are loaded and ready to run within the TTman. In the Management View, click on the root leave *AddressBookExample* and press the *Execute tests* button.



Test cases will run one after the other automatically. All logging results you can find in the TTCN-3 Graphical Logging tab online.



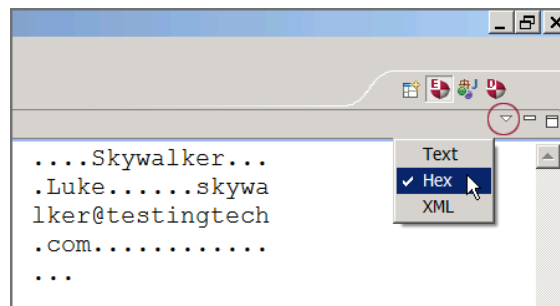
6.4. Checking Test Cases with Graphical Logging and Test Data View

STEP 11

Click on test case name *tc* in the Management View. Click on *TTCN-3 Graphical Logging* tab.

Click once on the *Send Entry* arrow to see the TTCN-3 test data representation in the Test Data View.

Double click on the message to get the encoded message.
Use the *Menu* button (white triangle) to change to HexView.



Single click on a *match* (green box) to see the template representation of received messages in the Test Data View. Click on test case name **nomatch_tc** in the Management View.

Click on *TTCN-3 Graphical Logging* tab. Single click on the *mismatch* box to see WHY the test case fails.

All differences are shown in the Test Data View tab after execution. On the right hand side you see the received message structure, on the left hand side the expected template message structure.

Details are given as red stripes for every single mismatched value.

The screenshot shows the 'TTCN-3 Execution Management' window. The 'Management View' tab is active, showing a table of test cases. The 'Test Case' column lists 'nomatch_tc'. The 'Runs' column shows 1 run. The 'Failure Action' column shows 'CONTINUE'. The 'Retries' column shows 0. Below the table, the 'Total' row shows 2 runs, 1 passed, 1 failed, and 0 inconclusive or errors.

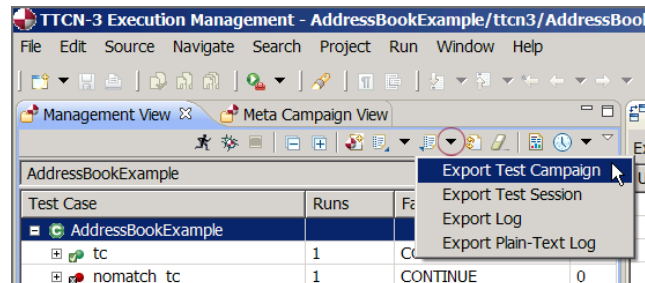
The 'Test Data View' tab is also visible, showing a table of test data. The 'Expected TTCN-3 Template' and 'Data' columns are shown. The 'Data' column contains the received message structure, which is highlighted with red stripes to indicate mismatches.

The 'TTCN-3 Graphical Logging' tab is active, showing a diagram of the test execution. The diagram includes a 'send Entry' block, a 'receive' block, and a 'mismatch' block. The 'mismatch' block is highlighted with a red stripe, indicating a failure. The diagram also shows a 'fail' block and a 'pass' block.

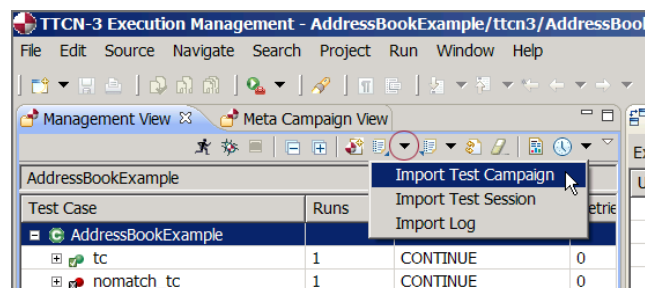
6.5. Loading and Saving of Test Campaigns and Logging

STEP 12

Current configurations (module parameter values, selected test cases) can be saved individually in the *.clf file by pressing the *Export* button. Select *Export Test Campaign* in the Management View.



Use the *Import* button and select *Import Test Campaign* to load an existing test campaign. You may also load one by double clicking on the file in the Development Perspective (TTCN-3 projects tab).



The current executed run of test cases can be saved in a *.tlz file by pressing the *Export* button. Select *Export Log* in the Management View to get a complete save of current log events.

The generated *.tlz file contains current configurations (*.clf) and the executed log traces (*.log).

Double click on the existing [AddressBookExample.tlz](#) in the root folder of your project to load an existing executed test run.

6.6. Generate Test Reports

STEP 13

After execution you can generate a test report in HTML, XML (Excel, Word) or PDF format.

Just click the *Generate Test Report* button, choose the test report format and fill in some tester specific properties. The generated test report will be stored in the specified destination directory.

7. Test Adapter Code

The test adapter (TA) provides real connectivity, e.g. to an IP based SUT with UDP, TCP or SCTP via the standardized interface TRI.

In this example, the TA is a simple UDP test adapter. It sends messages over a given UDP port to the SUT and enqueues messages received by the SUT. Within the TA there are standardized interface methods available (TRI Standard Part 5) being called by specific TTCN-3 statements like **map**, **send**, **unmap**, etc.

The source code of the example is given in the file [UDPTTestAdapter.java](#) (folder: javasrc/com/testingtech/ttcn/tri).

The **triMap** method instantiates a datagram socket class and starts a receiver thread to fetch incoming UDP packages from the SUT.

The **triSend** method encapsulates the message into a UDP package and sends it to the specified IP address.

triUnmap closes the sender/receiver socket.

8. Codec Code

There are two standardized interface methods available which are always called if there is a message to be encoded or decoded. They are specified within the TCI Standard (Part 6 of the TTCN-3 Standard).

Encoding means the transformation of the TTCN-3 template representation into a bytestream or bitstream.

Decoding is the transformation of an incoming bytestream or bitstream into a template representation. This specified template type will be taken from the TTCN-3 receive statement template as a hypothesis to start decoding.

The [AddressBookCodec.java](#) file (folder: javasrc/com/testingtech/ttcn/tci/codec) includes the complete encoding and decoding of the entry message.

For the **encode** method, the abstract value of the template is available as parameter. Here we check if the type name is **Entry** and if so, we call the **encodeEntry** method.

```
public TriMessage encode(Value template) {
    if (template.getType().getName().equals("Entry")) {
        return encodeEntry((RecordValue) template);
    }
    return super.encode(template);
}
```

The **encodeEntry** method encodes the header fields of the assumed record value.

Within the **encodeCharstring** the surName is encoded in Hex ASCII representation.

The **encodeGender** method encodes the chosen gender enum as one byte.

```
private TriMessage encodeEntry(RecordValue value) {
    bitpos = 0; // reset bit position
    ByteArrayOutputStream out = new ByteArrayOutputStream();

    encodeCharstring(value, "surName", out);
    ...
    encodeGender((EnumeratedValue) value.getField("gender"), out);
    ...

    private void encodeGender(EnumeratedValue value, ByteArrayOutputStream out) {
        // Enumeration implements also IntegerValue, so we can cast to IntegerValue
        // and encode the enum as one byte
        out.write(((IntegerValue) value).getInt());
    }
}
```

The **TriMessage**, a container for the received bytestream/bitstream and the type hypothesis, is available as parameter for the decoding method. Here we build a new instance of the type (**RecordValue**) and push the required values step by step to this result.

For example: The gender field will be decoded at a specific bit position as integer and will be set as **IntegerValue**. After that, the field 'gender' will be set to the **result** which is of type **RecordValue**.

```
public Value decode(TriMessage message, Type decodingHypothesis) {
    // reset bit position
    bitpos = 0;
    if (decodingHypothesis.getName().equals("Entry")) {
        RecordValue result = (RecordValue) decodingHypothesis.newInstance();
        byte[] encodedMessage = message.getEncodedMessage();
        decodeEntry(encodedMessage, result);
        return result;
    }
    return super.decode(message, decodingHypothesis);
}

private void decodeEntry(byte[] message, RecordValue result) {
    // surName
    int length;
    CharstringValue surName = decodeCharstring(message);
    result.setField("surName", surName);

    // gender
    int gender = message[bitpos / 8];
    bitpos += 8; // skip one decoded byte
    IntegerValue genderValue = (IntegerValue) result.getField("gender");
    genderValue.setInt(gender);
    result.setField("gender", genderValue);
    ...
}
```

9. Plugin Concept**

To enhance and/or to integrate an existing test adapter or codec source code, the port and codec plugin concept was allocated within TTworbench. Both classes, the **UDPTTestAdapter.java** class and the **AddressBookCodec.java** class are integrated via the Runtime Plugin Development Environment (RPDE).

This environment allows you to define your own port and codec plugin with GUI assistance or to enhance the existing one from the example. For further instructions please have a look at the built-in TTworbench Developers' Guide.

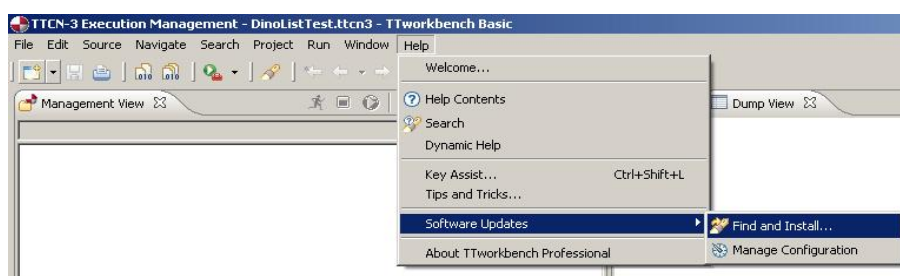
10. Download and Installation of Available Plugins

Step 1

Follow steps 1-3 of chapter "2. Download" to download the respective plugin.

Step 2

In the TTworbench menu, click on menu item *Help* → *Software Updates* → *Find and Install...*. Choose *Search for new features to install* and click on *Next...*

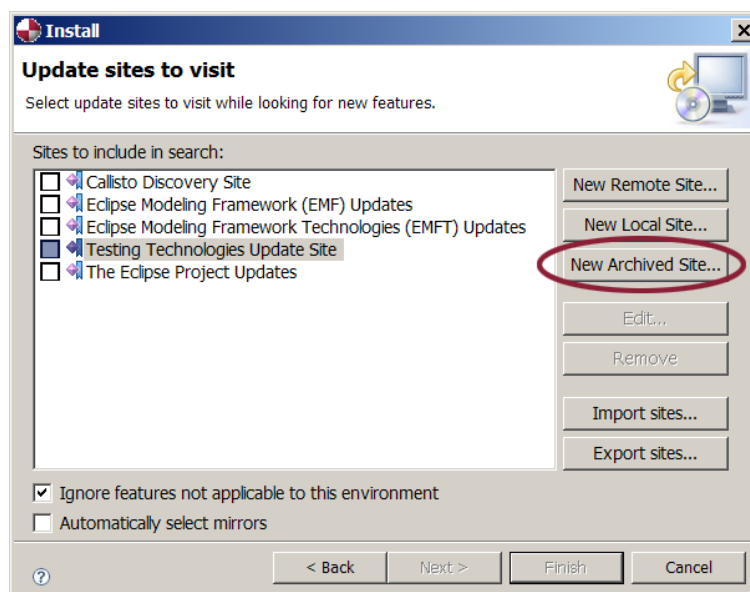


Step 3

Press the button *New Archived Site...*

Browse for the already downloaded *.zip file. Choose *Finish*.

Select the respective feature to install. Confirm installation with *Next* → *Next* → *Finish* → *Install*.



Step 4

Restart TTworkbench on request.

11. Appendix

Acronyms

ASN.1	Abstract Syntax Notation One	SUT	System Under Test
API	Application Programming Interface	TA/TC	Test Adapter / Test Case
BNF	Backus Naur Form	TCI	TTCN-3 Control Interface
CL Editor	Core Language Editor	TCP	Transmission Control Protocol
clf	campaign loader file	TRI	TTCN-3 Runtime Interface
MBS	Message Building System	TSI	Test System Interface
mtc	main test component	TTCN-3	Testing and Test Control Notation
RPDE	Runtime Plugin Development Environment	UDP	User Datagram Protocol
SCTP	Stream Control Transmission Protocol	XML	eXtensible Markup Language

Notes

This document is subject to change without notice.

Testing Technologies IST GmbH
Michaelkirchstraße 17/18
10179 Berlin, Germany

Tel/Fax: +49 30 726 19 19 -0 / -20
Email: info@testingtech.com
Internet: www.testingtech.com

Testing Technologies, the Testing Tech logo, TTworkbench and TTsuite are trademarks or registered trademarks of Testing Technologies IST GmbH. Other terms are used for identification purposes and are trademarks or registered trademarks of their respective companies. Testing Technologies' TTsuite is powered by Eclipse technology and includes Eclipse plug-ins that can be installed and used with other Eclipse 3.1-based offerings. It includes software developed by the Apache Software Foundation (<http://www.apache.org/>), ANTLR (<http://www.antlr.org/>), Tigris.org (<http://gef.tigris.org/>), and L2FProd.com (<http://L2FProd.com/>).