The Open Group

VSORB 2.3

Verification Suite

User Guide

April, 2001

For Details of the Support Service please contact your nearest Open Group Sales Office
See http://www.opengroup.org/corval2/

Table of Contents

1 Introduction

This document provides a basic user information to install and run the test suite for CORBA version 2.3 . It provides both, a practical guide to handle the related software package and a short introduction on the features and coverage of the test suite. It is assumed that the user of the document and delivered software is familiar with the principles of conformance testing techniques and the required test tools needed to carry out the execution of the test suite in this version. For basic information please refer to the CORVAL2 project reports and/or related tool documentation.

The test suite is composed of the following tests to address different aspects of CORBA specification:

- IDL syntax tests, to verify the syntax aspects of the code generated by the ORB's IDL compiler (IDL),
- API declaration tests, to verify the declarations of the CORBA APIs provided with the ORB implementation (DECL),
- IDL stub/ skeleton tests, to verify the runtime behaviour of the code generated by the ORB's IDL compiler (SII),
- API behaviour tests, to verify the behaviour of the CORBA APIs provided with the ORB implementation (API), and
- GIOP/IIOP tests, to verify the ORB's capability in terms of the syntax of GIOP messages and their exchange over IIOP (GIOP).

The relationship of these test categories and the involved test tools is presented in Table 1 on page 2. Please note that three test categories need to be performed in the TETware environment (IDL, DECL, API) and on the other side the SII and GIOP tests will use TTCN and optionally TTman for the "new" test suite ("old" refers to exiting tests for CORBA 2.1 upgraded to CORBA 2.3, "new" refers to additional tests for new features introduced in CORBA 2.3).

Table 1: Relationship of test categories and test tools

	IDL	DECL	SII	API	GIOP
TETware	+	+	+(old)	+	+(old)
ADL				+	
make	+	+	+(old)	+	+(old)
TTman/ TTCN			+(new)		+new

The test campaign has been performed under the following conditions and with the tools listed below:

- Compiler: gcc 2.95.2

- Platform: Solaris 2.6, Linux (RedHat 6.2)
- ORB: ORBacus 4.0.3/4.0.4 (also in combination with JTC 1.0.12)
- Tools: TETware 3.4, ADL1.1, Make, TTman, Tau 4.0 TTCN compiler and code generator

Section 2 focuses on the installation of the required tools and software. The description of their usage is subject of section 3 and section 4, while section 3 is restricted to a very short introduction on how to start and run the test suite, section 4 provides more details on the options and different possibilities to perform the test execution. Both chapters are organized to reflect the fact that each test category requires one test management environment, either TETware or TTman/TTCN. Instead of TTman, a TTCN command line can also be used.

Section 5 provides an insight to the test suite structure, tested features, and the particular prerequisites and conditions to run the tests. It has been organized according to the different test categories.

The current version of the CORBA 2.3 test suite is restricted for checking C++ implementations of the OMG's CORBA 2.3 standard. The test suite for Java based implementations is subject of a future release. This release of the test suite covers version 2.3.1 of the CORBA specification and is an extension of a previous test suite version, which was restricted to CORBA version 2.1 only.

The sources of the test suite are delivered in one single software package. After storage of the software in a target file system the directory name for this package has to be used as the root for the test suite. At the top level of this root the following subdirectories and files are provided with the following major contents:

- `results/` for test results,
- `runtest/c++/` including the control scripts to control the test campaign,
- `tset/c++/` including the sources of the test programs,
- `scenarios/c++/` contains lists of tests to be performed (to be selected),
- `vsorb_conf/` contains example config files for ORBs

According to the test categories the `runtest/` and `tset/` subdirectories have been subdivided in `api`, `hdr`, `idl`, `iiop`, and `sii`. In addition the test suite root directory contains further test configuration and control files for test execution. The usage of these scripts is explained in the chapters below.

2 Installation

2.1 Tests with TET

2.1.1 General

The Test Environment Toolkit (TET) is a universal management and reporting framework for automated testing provided by the Open Group.

In order to run the CORBA 2.3 test suite, Distributed TETware (i.e The version of TET suitable for the execution of tests in a distributed network architecture) has to be installed first. Also, the Assertion Definition Language is used to express the assertions for the API-test cases. Therefore the ADL translation system (also provided by the Open Group) is also needed for those tests to be executed

2.1.2 TETware/TET

TETware is a supported version, current versions are 3.4 and 3.5. TET is a public (free) version, current version is 3.3 and can be loaded from http://tetworks.opengroup.org.

Installing Distributed TETware can be achieved by following one of the two procedures below, depending on which TETware version to be installed:

2.1.2.1 TETware version 3.4/TET3.3:

1. Load the TETware distribution on to every machine on which you want to run local, remote or distributed test: The TETware distribution has to be loaded in a directory, which will be further referred to as the *tet-root* directory in this document. After this operation the tet-root directory contains the source code for TETware to be configured and built.
2. Take decisions as to...

- which network interface you want to use: Two alternatives:
  - BSD-style network calls (i.e the unix socket interface)
  - The X/Open Transport Interface (XTI)
- How you want to start the Test Case Controller daemon: Three alternatives
  - At system boot time from an entry in the system initialisation file /etc/rc.
  - At system boot time from an entry in the init configuration file /etc/inittab.
  - On demand, from an entry in the inetd configuration file /etc/inetd.conf.

Example:

```
tcc stream tcp nowait tet <tet-root>/bin/in.tccd in.tccd [<options>
...]
```

Notes:

a)`<tet-root>` is the full pathname of the directory where Distributed TETware is installed.

b) If you are NOT going to run tccd as the user tet, replace the 5th field with the required user name and add `-u<username>` to the `<options>`.

c) If you want other users to be able to run tcc:

add the tet group to each of the other users' supplementary group lists
add -m2 to the <options>
tell each of the other users to work with a umask of 2 (or less)

d) On Solaris, you will probably need to specify the value of the PATH environment variable, so that build tools are able to find cc, make, ar, etc.
For example, if you are using the Sun Workshop compiler, you might add:
`-ePATH=:/usr/bin:/usr/css/bin:/opt/SUNWspro/bin` to the `<options>`.

e) Send a SIGHUP signal to inetd, to make it re-read the /etc/inetd.conf file.

For example: `kill -1 <inetd-pid>`

- Create a test suite user (e.g tet) on each TETware machine and install a .profile for it.
  Example:
  ```
  TET_ROOT=/home/tet
   PATH=$PATH:$TET_ROOT/bin
   export TET_ROOT PATH
  ```
- Create an entry in the services database for the tcc service.
  (The services database might be in the file /etc/services, or it might be provided by NIS.)
  The port number can be any available non-privileged port, but it should be the same on all the machines where Distributed TETware will run.

  For example, to specify port 1234 you would say:

  ```
  tcc/tcp 1234
  ```

3. Create a file called systems.equiv in the tet user's home directory.
Each line in this file specifies the hostname of a machine that may connect to tccd (e.g. the file can contain the entry: `localhost`).

Whether or not each hostname in this file should be fully qualified depends on what kind of hosts database is in use.
If you are running DNS, each hostname should be fully qualified.
If the hosts database is in the /etc/hosts file or NIS, hostnames may or may not need to fully qualified, depending on what kind of name is returned by an address-to-name lookup operation.
The systems.equiv file should always contain at least an entry for this machine's hostname, and one for "localhost".

4. Create file systems in TET install directory (<tet-root>). Example content of the file:

```
000 localhost
001 localhost
...
005 localhost
```

5. Configure and install TETware 3.4 (to install the sources, you can skip this step if you have binaries):

-> type

```
cd $TET_ROOT
```

-> then, type either

```
sh configure -t inet
# for Distributed TETware with BSD socket interface
```

or

```
sh tetconfig
# then enter inet or xti, depending on whether you will be using the BSD
socket  interface (inet) or the X/open Transport Interface (xti).
```

-> type

```
cd src
make install
make compat  # for backwards compatibility for include directories
```

Please refer to the TETware Installation Guide for UNIX Operating Systems Rev. 1.5 [1] for more details on the installation of Distribute TETware version 3.4.

2.1.2.2 TETware professional and TET GUI

The installation of TETware professional and TETGUI can be done automatically, using the java installation binaries provided by the Open Group. These are the steps to follow to install TETware professional (note: you need a java 1.2 environment):

1. If you are not already there, change directory to the directory containing the binaries

2. Type:

```
java -cp ./ <installation-binary>
```

(in case the installation binary is a <installation-binary>.class file) or

```
java -jar -cp ./ <installation-binary>.jar
```

(in case the installation binary is a <installation-binary>.jar file), replacing <installation-binary> with the name of the installation binary file.

3. Follow the instructions from the installation program.

For more details on how to install TETware Professional and TET GUI, please refer to the TETware Installation Guide for unix [1], available at the Open Group's TET web site (http://tet-works.opengroup.org).

2.1.3 ADL Translation System 1.1

- Make sure that your system fulfils the requirements for installing the ADL Translation System 1.1. Check the ADL TRANSLATION SYSTEM 1.1 RELEASE NOTES [2] for details of the hardware requirements, as well as a description of the Applications and Libraries needed for the installation.
- Compiling ADLT

The ADL Translation System is released as a compressed tar file called adl1.1.tar.Z

You will need approximately 35 megabytes of free disk space to hold all the sources and objects.

After acquiring the tar file and creating a directory to hold the system, you can build 'adlt' by changing to the directory under which you want to unpack the ADL system and executing the following shell commands:

Note: the following steps assume you are using the source only release directory adl1.1.tar.Z, if your are using one of the precompiled directories the installed directory name will be different.

1. Unpack the archive, creating a directory named "adl1.1":

```
uncompress adl1.1.tar.Z
tar xvf adl1.1.tar
```

2. Change to the installation directory:

```
cd adl1.1
```

3. Before compiling 'adlt', you must set the environment variable ADLHOME.

ADLHOME should be set to the directory in which you will be installing the ADLT system:

```
setenv ADLHOME ‘pwd‘
```

4. properly set the variables in the make file $ADLHOME/Makefile.vars.
5. To build and install the ADL system enter the following command in the adl1.1 directory:

```
make all
```

2.1.4 TET-based tests

To install the tests to be executed with TET, you have to take the following steps:

- Take examples in `$VS_ROOT/vsorb_conf/orbacus` or vsorb/vsorb_conf/mico and edit for your ORB
- Use an install script to copy the files
- Go to `$VS_ROOT` and edit the files `config.env` and `*.cfg` for path settings
- Now you can source `config.env`

  ```
  source config.env
  ```
- Start the Interface Repository server, e.g. for orbacus:

  ```
  $ORBHOME/bin/irserv --ior > /tmp/vsir.ior
  ```
- Build libraries and some binaries

  ```
  vsorb_config c++
  ```

Note: $`VS_ROOT` is the location of the root of your vsorb tree (`vsOrb` directory).

To clean the configuration use: `vsorb_config clean_c++`. To feed the Interface Repository server only type: `vsorb_config ir`.

2.1.5 TTCN-based tests

The installation of tests using TTCN is described in section 5.4 and in section 5.5.

2.2 Tests with TTCN

2.2.1 TTCN

The Tree and Tabular Combined Notation (TTCN) is defined in the well established ‘OSI Conformance Testing Methodology and Framework’ (CTMF), which has been developed and standardized by ISO and ITU-T. TTCN is a notation for the specification of abstract test suites

for OSI conformance testing.

To run the test suites using TTCN no additional installation is required, because the compiler generates code, which is included in the ETS package.

2.2.2 TTman

TTman is a test management tool with a Java-based GUI.

After loading the TTman package files into a directory on the installation machine, the user has to go through following steps, to be able to run the test cases using TTman:

Note: We assume here that TTman was loaded in the $(my_test_suite) directory, i.e the directory in which the test suite has been installed.

1. Set the TMHOME variable in $(my_testsuite_dir)/include/Make.rules to point to the TTman installation directory (which contains the bin, include and lib sub-directories).
2. Customize the $(my_testsuite_dir)/WorkDir/gui.sh, that will be used to start the GUI so that the path of the java binary (.jar) file would be correct for your system.
3. create symbolic links (or copy) for the Test Manager daemon $(TMHOME)/bin/mngrd and for the Front-End daemon $(TMHOME)/bin/frntendd
4. create symbolic links (or copy) for $(my_testsuite_dir)/bin/ttcnclt

3 Quick start

3.1 TET command line

A simple way to start the Test Environment Toolkit TETware is possible in the command line operation mode. After running the configuration script (to set TET environment variables) a user has to start the TET Test Case Controller `tcc`. Test results are collected in a journal file.

1. customise and run the configuration script `config.env` to set environment variables,
2. start the Test Case Controller tcc, and
3. check the journal file for verdicts.

The Test Case Controller needs parameter values about

- the location of a scenario file (list of test programs to be executed)
- the location of a journal file (for test execution logs and results)

The tcc standard call to build and execute the tests specified by a scenario file is:

```
tcc -be -s $VS_ROOT/scenarios/c++/CPPSc_all -i $CUR_DIR $TS_NAME
```

where `$CUR_DIR` should represent the file name of the journal file and `$TS_NAME` identifies the name of the test suite (to be defined in the configuration script). To call the TCC you can also use a set of scripts, which are defined in `$VS_ROOT/runtest/c++` directory.

After completion of the test execution process the journal file has to be inspected. Each test program concludes with "PASS" or "FAIL" (other verdicts like INCONC, NORESULT are in principle also possible). The corresponding lines for passed tests which you will find in the journal file look like e.g.

```
220|1 1 0 10:13:27|PASS
```

The outlook of failed test could be very different. You may find lines like e.g.

```
520|385 1 00023974 1 13|ERROR: Expected CORBA::Short to be signed, is
unsigned.
```

or simply an error message e.g. from your compiler or linker.

3.2 TET GUI

How to start TET GUI:

Go to the directory, where you have installed test suites (`$VS_ROOT`). In the directory you have a file `config.env`. The file sets variables used by TET to corresponding paths. Source the file:

```
source config.env
```

Go to the directory, where you have installed TET GUI and call the GUI (note: you need a java environment)

```
java -jar TETware-professional-1-2.jar
```

Note: Use your version of TETware-professional.jar

Getting started with TET GUI

After the GUI has finished loading you can create a new test run or open an existing test run.

Note: Press the button Help and choose the Browse Help pages. Read the Getting Started from the help pages.

Create a new test run

To create a new test run use the "New Test Run" item in the File menu. You can also right click on the "Test Runs" tree node and choose "New Test Run" or use a corresponding icon in the toolbar. ("Test Runs" tree node is in the left part of the screen).

In the dialogue box which you will see, specify the name of the test run (e.g. "demo"). A new test run is created and added to the "Test Runs" tree node.

Note: if the name of the test run already exists "New Test Run" name will appear.

Now start to set the configuration. Left click on the icon which is on the left side of the created test run ("demo"): new menu will appear which contains the following items:

- Configuration
- Execution Output
- Journal
- Report

Choose "Configuration". If you click on the icon of the left side of "Configuration" you will get addition item in menu for configuration: "local system". If you click on "Configuration" you will get a window with a menu at the top, which contains:

- Main,
- Test Cases,
- Custom Result Codes,
- Alternate Config files,
- User Defined Variables,

- Advanced.

After choosing "Configuration" the window for "Main" item is shown. You can put your description in the "Description of the Test Run". You have to choose the "Modes of Operation": Build, Execute or Clean, e.g. you can choose Build and Execute.

There is a default configuration file for each of the tcc running-modi.The `tetbuild.cfg` file is for the build modus, `tetexec.cfg` is for the execute modus and `tet_clean.cfg` is for the clean modus. Also the user might specify alternate configuration files to be used in place of those, by entering their absolute path In the "Alternate Config files" menu of the GUI.

Click "local system" from the menu under "Configuration" and make the configuration of the "Local Environment". You must specify the TET_ROOT and TET_SUITE_ROOT variables. TET_ROOT variable is to specify the location of the TET software on your system. TEST_SUITE_ROOT specify the test suite you are using for this test run (corresponds to $VS_ROOT). Any scenario file you are specifying for this test run must lie on the same path as this variable.

As next step add a scenario file to the test run. Before you choose the scenario file you have to prepare a scenario file: e.g. demo.scen in the $TEST_SUITE_ROOT directory (or subdirectory of $TEST_SUITE_ROOT).

You can add scenario when you right click the test run "demo" and choose "Add Scenario File", or use the "Add Scenario File" in the "Config" menu.

You will get a window where you can search the paths and put the name of the scenario file ("demo.scen"). Under your Test Run "demo" a new icon will appear "demo.scen" with the scenario file. Now you can run your tests.

Running the tests

Click on the scenario file icon in your test run. You will get a list of the test groups you can run (corresponding to the structure of your scenario file). Click on one, some or all.

To run the tests choose the "Execute Test Run" from the "Run" menu. You will get the window to confirm the execution and the execution will start according to your mode specified in the "Modes of Operation" (build, execute, clean). After the execution has finished you can click on the "Journal" to see the output of the tests - journal file. To do it you can take the "Journal" icon under the test run "demo" or the corresponding icon in the toolbar.

If you created a new test run you can save it for further use.

Save a new test run

Right click on the test run "demo" and choose from the menu "Save As Test Run". You can do

it also from main menu "File" or using a corresponding icon from the toolbar. You will get a window where you can choose the path, where you want to save your test run and specify a file name (e.g. demo.rtf).

Note: Test run files are saved with the .rtf suffix.

Now you can exit the TET GUI.

Exit the TET GUI

Click "Exit" in the "File" menu. You will have to confirm, whether test runs have to be saved before exit or not. When you have created a new test run and you have not saved it before, you will lost the information about the test run, if you answer "No".

3.3 TTCN command line

The execution of test suite, which using TTCN, depends on the test configuration, therefore the following steps are abstract. For detailed information, please look the particular test suites in section 5.

- Start the CORBA-based test component.

- Execute theTTCN-based test component. A list of the available test cases appears.

- Select the desired test case, one of the following results is reported:

  - PASS,
  - FAIL
  - INCONC
  - NONE

3.4 TTCN with TTman GUI

TTCN tests are executed with TTman. There are two groups of tests with TTCN, sii and giop, which have to be run separately.

TTman needs IP multicast to realise communication between its components. It uses IP group address 225.0.0.1. If your host is not configured to route multicast packets, add a corresponding entry to the routing table.

under Linux:

```
route add 225.0.0.1 gw localhost
```

3.4.1 sii:

Go to the directory where you have sii-tests installed: <sii>.

To make the test components refer to the section 5.4. Now you can start all the programs to start the tests. You have to test stub and skeleton separately. For example you can start test of skeleton. Go to the directory `<sii>/WorkDir` which contains you tests (tests are in *.tsl). You can use scripts to call the programs or do it manually.

Note: Following examples are for orbacus. Please edith path and options for your ORB.

Start all processes

Start a repository

```
ir.sh
```

or

```
rm -f Repository.ref
${ORBHOME}/bin/irserv --ior ../IDL/obv_test.idl > Repository.ref &
```

where ${ORBHOME} has to be set to the path of the tested ORB.

Start a gateway

```
gw.sh
```

or

```
../bin/gw -vvv -ORBservice InterfaceRepository ‘cat Repository.ref‘ -
OAthread_per_request &
```

Start a server emulation

Here you are testing skeleton produced be an IDL compiler.

```
emsrv.sh
```

or

```
../bin/emsrv -ORBservice InterfaceRepository ‘cat Repository.ref‘ &
```

Start a manager

```
mngrd
```

Start a frontend

```
frntendd
```

Start TTman GUI

```
gui.sh
```

or

```
java -classpath <your path>/ttman/gui/ttman1_1_0.jar:$CLASSPATH
fokus.ttman.TTman $1 &
```

where `<your path>` is the path of your TTman location.

Now you can start working with TTman GUI.

Run the tests with TTman

Load a test suite loader file

Choose "Open" item from the "File" menu. In the dialogue box choose a *.tsl file - skel.tsl for the skeleton test. You can also open a file using an icon in the toolbar.

Check the configuration

Choose "Configuration" item from the "Configuration" menu. In the dispay the "active" should be marked. The "MASTER" should be: "localhost" and "7000", i.e. you are using localhost interface with port 7000. This address is used by the frontend (frntendd) to communicate with the TTman GUI.

Open a session

Choose "OpenSession" item from the "Execution" menu or use a corresponding icon in the toolbar.

Select the tests

Now you can select/deselect the tests listed in the left part of the screen with the left mouse click in the "Select"-column. Through the left click on the test case name will the information about the test case will be shown.

Run the selected test

After you have selected the test cases, you can run the test. To do so, choose the "Run" item from the "Execution" menu or corresponding icon in the toolbar. You can see that the test cases are executed.

Look at the verdicts

Scroll with the mouse the test case list on the left part of the screen and have a look at the column verdict.

Close a session

After running tests you can close the session using the "CloseSession" item from the "Execution" menu or using a corresponding icon in the toolbar. You can open the session again or you can close the test suite loader file.

Close a test suite loader file

Click on "Close" item in "File" menu. You will get a dialogue box with the question, whether the changes should be saved to ttcnclt. If you answer yes, next time you load the test suit, the verdicts will be available. If you answer no, the information about the run results will not be saved.

Now you can open another test suite loader file or exit the tool and stop all processes.

Exit a session

Choose "Exit" from the "File" menu to exit the TTman GUI. All other running processes you have to stop entering "ctrl-c" (running in foreground like frontend and manager) or "kill" (running in background like interface repository, gateway and server emulation).

3.4.2 giop:

Go to the directory where you have giop-tests installed: `<giop>/WorkDir` which contains you tests (tests are in *.tsl).

Now you can start all the programs to start the tests. You have to test the ORB in the client and server role separately. For example you can start to test the server role. You can use scripts to call the programs or do it manually.

Start all processes

Start a server emulation

Here you are testing server side of your ORB.

```
emsrv.sh
```

or

```
../bin/emsrv &
```

Start a manager

```
mngrd
```

Start a frontend

```
frntendd
```

Start TTman GUI

```
gui.sh
```

or

```
java -classpath <your path>/ttman/gui/ttman1_1_0.jar:$CLASSPATH
fokus.ttman.TTman $1 &
```

where `<your path>` is the path of your TTman location.

Run the tests with TTman

The tests are run with TTman GUI similar to the sii-test. You load a test suite loader file (a *.tsl file) for giop test (e.g. giop_all.tsl).

4 Details of test execution

4.1 TET command line

In addition to the instructions given in the previous chapter TETware has a significant flexibility to modify and control the test execution. This section is restricted to the possibilities which relate to the CORBA test suite execution. Further details have to be checked from the TET user manual.

Overview on involved files:

The user of the TET command line mode should be aware of the different files which participate in the test campaign. It is the scenario file which has the overall control on the selection of tests. In particular this file indicates the actions defined in a series of makefiles. Each of these makefiles specifies the tools and commands to be executed, e.g. a compiler with options and parameters. Therefore they include e.g. names of the source files of the test programs (e.g. idl, c++, java).

Depending on the test methods the tests may be restricted to the actions specified in the makefiles (build mode) or continues with the execution of compiled and linked test programs (execution mode). The outcome of both phases is collected in the journal file. The test logs resulting from the execution of one scenario file will be collected in one journal file. An overview on the dependencies of the involved files is given in Figure 1.

tcc -be -s <scenariofile> -i <journaldir> <resultdir>

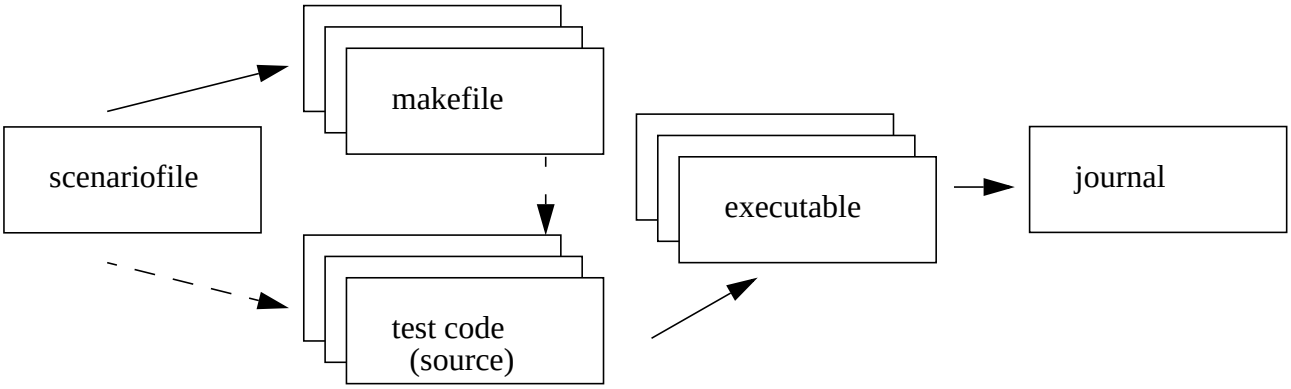

Figure 1: Overview on dependent files

Modification of the list of tests to be executed:

Since the scenario file includes the total list of tests to be executed, a selection of scenario files is available to restrict the test to the specific predefined features a test should cover. The CORBA 2.3 test suite has scenario files for each of the testing categories. In particular an user may start the tcc with the following options for the "-s" parameter:

- CPPSc_all, for the complete test suite,
- CPPSc_api, for CORBA API behaviour tests,
- CPPSc_hdr, for CORBA API declaration tests,
- CPPSc_idl, for IDL compiler tests,
- CPPSc_sii, for IDL stub and skeleton tests, and
- CPPSc_iiop, for GIOP/IIOP tests.

Calling the tcc:

The command line mode of TETware allows the explicit specification of the location of the major test (control) programs. We distinguish three main options available by tcc. The "-be" option forces both the actions of the makefile and (if appropriate) the execution of the compiled test programs. A test operator may omit one of these activities by calling tcc with "-b" (build mode) or "-e" (execution mode) only. A third option "-c" (clean mode) leads to the deletion of the executables produced during the build mode.

For further information please refer to the TET user manual.

4.2 TET GUI

Open an existing test run

If you have a prepared test run, which has been created in one of the previous session, you have to open it. You can do it when you:

- right click on the Test Run and choose from menu "Open Test Run" or
- choose "Open Test Run" from the "File" menu or
- choose corresponding icon from the toolbar.

You will get the window, where you can specify you file with a test run. The test run will be loaded. you have to click on the test run to get it active before running. Now you can change the configuration of the test run or start running the tests.

Note: If you change the configuration you have to save your test run before you exit the TET GUI. Otherwise the old configuration will be taken by the next session.

You can have open more than one test run within the TET GUI. Therefore you have to click a test run which have to become active before taking any action.

Save test run

To save a test run right click on the test run and choose from the menu "Save Test Run". You can also do it from main menu "File" or using a corresponding icon from the toolbar. See description of "Save a new test run".

Remove test run

From "File" menu choose the "Remove Test Run" item. After your confirmation a test run will be removed from the list of test runs.

Remove scenario file

You can remove a scenario file from the existing test run. You can do it using "Remove Scenario File" item from the "Config" menu.

Observe test execution

If you want to observe the execution of the tests before the journal file is produced click on "Execution Output" under you test run before starting the execution. You will get on the screen a trace of TCC execution/error output.

Set report file

With the "Report" in your test run menu you can see, where you journal file is located (journal file is an ASCII file). You can also specify the journal file that should be used for building the report.

Note: For more information about the TET GUI look in the "Help menu".

4.3 TTCN command line

For test execution with TTCN command line see section 5.4 for sii-tests and section 5.5 for giop-tests.

4.4 TTCN with TTman GUI

Observe test execution

Under `<sii>/WorkDir` or `<giop>/WorkDir` you will find a file: ETS.log. The file is a log-file of the test run by TTman. You can also do it partly during a test execution with TTman.

Run tests

To run the sii stub test you have to proceed like for skeleton tests with following changes:

- start client emulation instead of server emulation
- load stub.tsl instead of skel.tsl

Fo more information see sections: section 5.4, section 5.5, section 3.4.1, section 3.4.2

Note: giop client side test are not available at the moment.

Work with TTman

Save a test suite loader file

Using a "Save As" item from the TTman "File" menu you can save a test suite loader file under a new name. You can also save with the same name using "Save" from the "File" manu or corresponding item. After save the selection of test cases and results from the run will be kept.

Reset verdicts

Use "ResetVerdict" from the "Edit" menu to reset all the verdict to NONE.

Test Suite Parameters

You can read/change test suite parameters choosing the item "Parameters" from the "Configuration" menu.

Observe test execution

After loading your *.tsl file on the left side you will see the test cases. The verdict column will be NONE, if the test have not be executed before. The last column is to select/deselect a test case for the next run. You can also select/deselect from the "Edit" menu. You don’t need to select each test case. They have a structure, which is shown on the screen. Selecting a test group will select all test cases which are in the group.

With a left click on the test case you will see on the right side the information about the test case. The information includes: test case name, selected or not, verdict, status (market for execution or not), test configuration, test purpose description and a line with summary of test verdicts (number of test cases, number of verdict PASS, FAIL and INCONCLUSIVE).

After you open a session and click on "Run" you can observe on the screen how the TTman executes the test cases one after another and changes the verdicts from NONE to PASS, FAIL or INCONCLUSIVE. The line with summary of test verdicts will show you the number of test cases, number of verdict PASS, FAIL and INCONCLUSIVE of the whole run.

TTman help

TTman provides help pages, which you can find in "Help" menu of TTman GUI.

5 Test Suites

5.1 IDL

Structure and Tested features

The purpose of the IDL-Tests in the CORBA 2.3 Test Suite is to check whether the IUT's IDL compiler maps IDL code to target language code (in this case C++-Code) correctly according to the corresponding CORBA 2.3 Mapping Standard. The issue here is on statical aspects of the mapping, i.e on the types and functions that, according to the specification, should be defined in the generated target language source code.

Given the fact that the mapping specification for already existing IDL constructs did not undergo any major modifications moving from version 2.1 to version 2.3, the CORBA 2.3 Test Suite will focus on the new features of the IDL language, i.e Valuetype, Abstract Valuetype and Abstract Interface. The CORBA 2.1 tests upgraded to CORBA 2.3 are not structured, i.e. each test case forms a test group and is coded in one directory together with an IDL example. Each test case (test001 - test054) contains an implementation of a client, server and some files with testing functions, which include files generated by an IDL compiler. Most of the test cases are based on C++ compiler check: if a C++ compiler compiles all the files correctly a l program will be executed with the only task to write PASS-verdict to a journal file; otherwise compiler error will be written to a journal file. Only some tests are based additionally on an execution of a compiled program: test003, test004, test005, test006, test007, test011, test013, test014, test015 (there are no client-server calls in IDL tests).

The new tests are based on compiler error check: For each test subgroup an IDL-file will be written to be used as input to the IUT's IDL compiler. The generated code is then included in a codelet-file, where the mapping elements are referenced (in case of mapping types) or called (in case of required mapping functions). The codelet file is then compiled using a target-language compiler (e.g. C++-compiler). Which verdict is attributed to the test depends on whether compilation was OK or not. Most of the tests rely on positive checking, i.e they return a PASS verdict if the codelet file compiles normally and a FAIL verdict if it provokes a compiler error, but some test cases also rely on negative checking, i.e when their expected behaviour is to provoke a compiler error. Therefore they will return a PASS verdict in case they cause a compiler error.

The test suite reflects the same logical structure as the mapping standard, with test groups built around the new constructs and test subgroups featuring their different possible mapping situations. A set of test cases has been written for each required mapping type or function. TET enables the tester to execute either the whole subgroup or a selection of test cases in the subgroup. Therefore the test cases for the new features have been organized in 6 main groups. The tests are illustrated in table 1 below:

Table 2: Tested features on the IDL-part of the CORBA 2.3 conformance Test Suite

Test Group	Tested construct	Subgroup	Description
test001	identifier		long identifiers and all alphanumeric characters
test002	names		names which are C++ keywords
test003	literals		literals for constants
test004	characters		characters and their possible values
test005	float, double		float and double value check
test006	string		string values check
test007	different		- any and structured types (struct, union, enum, sequence) defined in a module, - boolean constants - interface with exception, attribute and operations - any as operation parameters
test008	macros		preprocessor and macros
test009	comments		comments
test010	scopes of names		names in different scopes, continuation of module definition
test011	interface def.		possible constructs in an interface definition: constant, typedef, exception, attribute and operation
test012	interface		interface used in different constructs, as members of other types, operation parameters etc. (e.g. member of a struct); multiple inheritance of interface
test013	interface inheritance		interface inheritance: check the redefinitions and scope of types
test014	interface inheritance		interface inheritance (multiple inheritance): check the redefinitions and scope of types
test015	scopes of constants		constants in different scopes with the same name
test016	operation		operation and parameters (parameters of basic types)
test017	TypeCode		check CORBA::TypeCode and user defined type TypeCode; one example of a string as attribute
test018	any		'any' in different construct
test019	exception		exceptions
test020	attribute		attributes of different types
test021	scopes of names		names and scoping (nested modules)

Test Group	Tested construct	Subgroup	Description
test022,23,24	sequences		sequences and array of sequences used in attribute definition, operation parameters and exceptions
test025,26,27,28	sequences		sequences, sequences of sequences and array of sequences used in attribute definition, operation parameters and exceptions
test029,30,31	sequences		sequences, sequences of sequences, sequences of sequences of sequences, etc. and array of sequences used in attribute definition, operation parameters and exceptions
test032	struct with sequence		struct and struct with sequence members used in attribute definition, operation parameters and exceptions
test033	types with sequences		enum, sequence of enum, struct of struct with sequence member; the types used in: attribute definition, operation parameters and exceptions
test034, 35	types with sequences		struct of struct with sequence members used in: attribute definition, operation parameters and exceptions
test036,37,38,39	types with sequences		struct of struct of struct (of struct) with sequence members used in: attribute definition, operation parameters and exceptions
test040	recursive type		recursive type: a member type as a sequence of an enclosed struct, the type used in attribute definition, operation parameters and exceptions
test041	sequences		sequence of float and array of sequence of float used in: attribute definition, operation parameters and exceptions
test042	struct		struct definition with float members and array of float members used in: attribute definition, operation parameters and exceptions
test043	struct		struct with members of basic types, and arrays of these types used in: attribute definition, operation parameters and exceptions
test044	struct		like the test042, but new types are defined as typedefs of struct definition with float members and array of float members
test045	struct		struct with members of sequences of basic types and arrays of these types used in attribute definition, operation parameters and exceptions
test046	struct		structs nested in structs (many levels of nesting) used in: attribute definition, operation parameters and exceptions

Test Group	Tested construct	Subgroup	Description
test047	union		union with a default, discriminator of type long, basic types as union types, union used in attribute definition, operation parameters and exceptions
test048	union		union with a default at the beginning of the definition, discriminator of type short, union used in attribute definition, operation parameters and exceptions
test049	union		union with a default, unsigned short discriminator, used in attributes, operations and exceptions
test050	union		union with char discriminator, default in the middle of the union definition, used in attributes, operations and exceptions
test051	union		union with boolean discriminator, without a default but all cases are listed, used in attributes, operations and exceptions
test052	union		union with enumeration discriminator without a default, but not all cases listed; both types (enum and union) used in attributes, operations and exceptions
test053	union		union with a default, with long discriminator; used in attributes, operations and exceptions
test054	union		union without a default with not all cases listed, long discriminator; struct as union members; usage of structs defined inside an union - defined types used in attributes, operations and exceptions
aif	Abstract Interface	aif_b01 (/tset/c++/idl/aib_b01/tin)	Basic features of Abstract Interface: existence and features of mapping class
		aif_p01	Abstract interface as operation parameter
avt	Abstract Valuetype	avt_b01	Basic features of the class mapping for an Abstract valuetype
		avt_p01	Mapping of Abstract valuetype as operation parameter
vt	Valuetype	vt_b01	Basic features of the valuetype
		vt_e01	Mapping of an exception declared in a valuetype
		vt_i01	Valuetype as member
		vt_m01	Value type with members of basic type
		vt_m02	Valuetype with members of structured types
		vt_o01	Operation defined in a valuetype
		vt_p01	Valuetype as operation-parameter

Test Group	Tested construct	Subgroup	Description
vb	Valuebox	vb_b01	Basic features of the class mapping for a valuebox
		vb_d01	Valuebox for basic types and enums
		vb_d02	Valuebox for struct types
		vb_d03	Valuebox for Array
		vb_d04	Valuebox for Strings and WStrings
		vb_d05	Valuebox for Unions, Sequence and Any
		vb_e01	Exception in a Valuebox
		vb_p01	Value Box of Short as attribute and as operation parameter
		vb_p02	Value Box as attribute and as operation parameter
		vb_p03	Valuebox as attribute and as operation parameter
vti	Valuetype and Inheritance	vti_avb01	Valuetype inheriting from abstract valuetype
		vti_avb02	Valuetype inheriting from abstract valuetype
		vti_si01	Valuetype supporting an Interface
		vti_si02	Valuetype supporting an Interface
		vti_si03	Valuetype supporting an Abstract Interface Interface derived from Abstract Interface
		vti_vb01	Valuetype inheriting from another Valuetype
va	Advanced features of valuetype (Scoping, forward declaration etc...)	va_f01	Valuetype forward declaration
		va_f02	Valuetype forward declaration and inheritance
		va_t01	Valuetype Inheritance with typedef
		va_t02	Valuetype Inheritance with typedef
		va_t03	Valuetype Inheritance and support for Interface with typedef
		va_t04	Valuetype Inheritance with modules in typedef

Prerequisites

The current version of the test suite uses the GNU C++ compiler for the checks, although any other compiler would also fit. The make-tool also has to be available on the system for the checks to run properly.

Run the tests and assessment:

After installation of one of the TETware versions presented in the previous chapter there are no

specific conditions to run the IDL tests.

The actual list of IDL tests to be executed is given with the scenario file to be used. If you like to restrict the test campaign to IDL tests please run `tcc` with `CPPSc_idl` which is located in the scenario file directory (see introduction). Output of the tests is written into a journal file.

All the tests should return PASS verdict. The old tests provide no verdict, if a compilation error (IDL, C++ compiler) occurs. The error message will be written into journal file but there is no message about the tested feature. The old test are compiled during a TET build mode, executed in a TET execute mode.

The new tests are organised differently. During a TET build mode only a controller is compiled together with files produced by an IDL compiler, no test programs. There is no verdict in two cases:

- IDL compiler produces an error, the error message is written to the journal file,
- C++ compiler produces an error, the error message is written the journal file. The error means that: there is a general problem with a C++ compiler and a controller program or that the code produced by IDL compiler cannot compile itself (e.g. differences between .h and .cpp files).

In the TET execution mode the test programs are compiled and a verdict is written together with a test purpose to the journal file. By most of the tests if the compilation produces no error, a test returns PASS; and in a case of an error the test returns FAIL together with the error message (so called positive check). There are some tests marked: NEGATIVE CHECK. This tests return compiler error message and PASS verdict if the tested feature is supported by the tested ORB. If a required feature is not supported, no compiler error is produced and a verdict is FAIL. These negative tests are often preceded by a corresponding positive tests - only when a positive check returns PASS the following negative check has any value for the testing.

5.2 DECL

Structure and Tested features:

Testing CORBA version 2.3 implies mainly series of new tests on the new CORBA features ValueType semantics, abstract interface semantics, dynamic management of Any values and the portable object adaptor (POA). Additionally these new features imply a series of extensions and modifications of CORBA features which are existing in the previous version.

The resulting structure of the new test suite is due to the fact that the test suite for the previous version is still available and has to be maintained w.r.t. the changes in the CORBA 2.3 standard. The current test suite is structured according to:

1. BasicTypes

2. Any
3. AnyVar
4. StringVar
5. WStringVar
6. Object
7. ORB
8. Environment
9. TypeCode
10. Exceptions
11. Service
12. DomainPolicy
13. Current
14. DII
15. IR

Subgroup 1 checks the availability of CORBA types like boolean, char, etc. The implementation of the Any and AnyVar classes (constructors and exchange with the basic types) is subject of subgroups 2 - 3. String and WString class construction and operation tests are in subgroups 4 - 5. The tests of the subgroups 1 - 5 are located in subdirectory /tset/c++/hdr/basicTypes.

Subdirectory /tset/c++/hdr/pObjects contain the tests of subgroups 6 - 9. They address the implementation of the C++ classes Object_var, ORB_var, TypeCode_var and Environment_var including their standardized members and related type definitions (e.g. ORBid).

The Exceptions subgroup (10) is restricted to the type CompletionStatus and the standardized system exceptions. Test subgroup 11 focuses on the Service type, related structures, constants or operations. Subgroup 12 address the Policy related types and functions. In Subgroup 13 there is only the test on the existence of class CORBA::Current. For each of these four subgroups there is a subdirectory with all corresponding tests (identified by the subgroup name, e.g. /tset/c++/hdr/Exceptions).

The tests on the Dynamic Invocation Interface (subgroup 14) are located in /tset/c++/hdr/DII. Since they cover operations for different purposes a substructure has been introduce to support the overview in this testgroup. The tests are located in the subdirectories Object, ORB, Context, ContextList, ExceptionList, Request, NVList and NamedValue. The DII Flag types could be found in /tset/c++/hdr/DII.

The Interface Repository test subgroup (15) requires a substructure, too. The tests are included in a subdirectory of /tset/c++/hdr/IR (except the tests on IR types which are not in a subdirectory). The following IR test subgroups are available and are related to an own subdirectory: ORB, AliasDef, ArrayDef, AttributeDef, ConstantDef, Contained, Container, EnumDef, ExceptionDef, IDLType, IRObject, InterfaceDef, ModuleDef, OperationDef, PrimitiveDef, Repository, SequenceDef, StringDef, StructDef, TypeDef and UnionDef. Due to CORBA 2.3 there are also the new groups NativeDef, ValueMemberDef, ValueDef and ValueBoxDef.

The following parts are new in the test suite. They have been substructured according to a new scheme which divides the test group into further subgroups, each addresses one of the following items: constants, types, exceptions, interfaces and C++ specific mapping (Cpp_specific). Test subgroups addressing interfaces are subdivided into subgroups whereas each comprise the tests at one single interface only.

16. ValueType
17. AbstractInterface
18. DynAny
19. POA

According to the test group division the ValueType test group (no. 16) contains tests in subdirectories VT_SeqTypes, VT_ValueTypes and VT_Cpp_specific. The first group verifies data types like BooleanSeq etc., the second contain tests defined for specific value type definitions (e.g. StringValue) and custom marshalling. C++ specific tests are in the last subdirectory (e.g. class ValueBase, ValueFactoryBase).

The AbstractInterface test group (no. 17) includes the C++ specific test on the AbstractBase class implementation in the subdirectory AI_C++specific.

In the test group DynAny the two subgroups DA_Types and DA_Interfaces are included. The first subgroup comprises tests on type definitions (e.g. FieldName), while the latter subgroup has been further subdivided due to the individual interfaces DynAny, DynFixed, DynEnum, DynStruct, DynUnion, DynSeq, DynArray, DynValue and DynAnyFactory. Each of this subgroups is provides corresponding tests on constructors, functions etc. in a single subdirectory (e.g, /hdr/c++/DynAny/DA_Interfaces/DA_I_DynAny).

Similar to DynAny the POA tests have been ordered. The tests are provided in the subdirectories PA_Constants, PA_Types, PA_Exceptions, PA_Interfaces, and PA_Cpp_specific. PA_Interfaces is subdivided to PA_I_ServantManager, PA_I_POA, PA_I_POAManager, PA_I_Current and PA_I_others. In PA_I_others there are especially the tests on interfaces which are derived from CORBA::Policy.

Prerequisites:

The DECL test group has been implemented for the usage with TETware. Most of the test require a compiler run only (TET option "-b"). The use of the "-e" option is needed for testing some of the basic types only, e.g. to validate type length by the shift function. Nevertheless the "-e" option will provide within the journal file a short information on test purposes and linkage errors.

The test execution requires the include library of the ORB under test. It is the $ORBHOME environment variable which is used in the TET configuration script to identify the selected ORB. The additional environment variable $JTCHOME is an ORB specific information and shows the idea how specific requirements could be satisfied for the configuration script.

Run the tests and assessment:

After installation of one of the TETware versions presented in the previous chapter and after providing the required information as mentioned above in the prerequisites there are no specific conditions to run the DECL tests.

Please remember that the actual list of DECL tests to be executed is given with the scenario file to be used. The DECL tests are part of the complete version of the test campaign but there is also a scenario file available which has to be used in case of running DECL tests only. If you like to restrict the test campaign to DECL tests please run `tcc` with `CPPSc_hdr` which is located in the scenario file directory (see introduction). Further restrictions to parts of the DECL tests is possible, too. In this case the tests to be excluded have to be mark as comments (use "#" at the beginning of the column) in the selected scenario file.

5.3 API

Structure and Tested features:

The purpose of the API-Tests in the CORBA 2.3 Test Suite is to check whether the semantics of the operations correctly accords with CORBA 2.3 standard specification. The issue here is on dynamical aspects tests.

Testing CORBA version 2.3 implies mainly series of new tests on the new CORBA features. e.g. the chapter portable object adapter(POA), dynamic management of any value etc.

The resulting structure of the new test suit is divided in chapter:

1. POA (Portable Object Adapter)
   - POAInterface
   - POAManager

2. DAny (Dynamic Management of Any Value)
   - DynAny
   - DynEnum
   - DynStruct
   - DynUnion
   - DynSequence
   - DynArray
   - DynValue

The API-Tests is Operation testing with ADL version 1.0. ADL provide constructs to specify describing the input-output behaviour. The test precondition for API-Tests is the test input data to build. the testServant and testDAny provide the input data.
Use scripts in /runtest/c++/api if you are going to run the API tests only.

5.4 SII

The IDL stub and skeleton (SII) tests are composed of TET-based tests and TTCN-based tests. The TET-based tests cover earlier CORBA specification, whereas the TTCN-based SII tests address in particular the value semantics definition introduced in CORBA 2.3. For building and execution of the TET-based tests please refer to the previous sections. In the following, details of the TTCN-based SII tests are introduced.

The value semantics tests focus on substitution of value parameters and passing of parameters of different IDL types in value boxed as defined in Section 5 of CORBA 2.3 specification.

The tests are separated into two test groups:

- A skeleton side test group (abbr. skel) to check mainly the functionality of skeletons generated by the IDL compiler under consideration, and
- A stub side test group (abbr. stub) to verify the IDL stubs.

Each test group uses two test components corresponding to the client-server paradigm. The skel test group has the following configuration:

- A server application based on the ORB under test, including the IDL skeletons.
- A client application based on TTCN and TCgate. TCgate provides the communication between the client and the ORB.

Analogously, the stub test group uses a TTCN-based server test component and a CORBA-client.

Sources of the TTCN-based SII tests are structured into the following sub-directories (`/ttcnTS/sii` is the parent directory):

- `ATS` contains the abstract test suite developed using TTCN, among them `skel.pdf` and `stub.pdf`.
- `ETS` contains the implementations of the TTCN-based test client and test server, separate in the `skel` and `stub` sub-directories.
- `Gateway` contains the IDL definition and implementation of TCgate.
- `IDL` contains the IDL definition and implementation of the CORBA-based server and client applications.
- `WorkDir` is the working directory for test execution.
- `bin` will contain test relevant executables.
- `include` contains some configuration and header files.

Some environment variables need to be set before building and executing the tests. An example can be found in: `/ttcnTS/sii/include/config.env.`

To build the executables:

```
cd /ttcnTS/sii
make
```

To execute the tests, some scripts under the WorkDir directory can be adapted and used. For the `skel` tests, the executables are started in the following order, in order to launch the Interface Repository server, the TCgate, the CORBA-based server and the TTCN-based client:

```
0.    cd ttcnTS/sii/WorkDir
1.    ./ir.sh
2.    ./gw.sh
3.    ./emsrv.sh
4.    ./tnclt.sh
```

A list of test case names will be printed for selection. Either type "all" for the execution of all available tests, or select an individual test case by typing the appropriate test case name, or type "q" to terminate the execution.

To execute the `stub` tests, repeat step 0-2 if the Interface Repository server and TCgate have not already been activated, and then start the CORBA-based client and the TTCN-based server:

```
5.    ./emclt.sh
6.    ./tnsrv.sh
```

Results of the command-line execution are written into a report file named `ETS.log` under the

`WorkDir` directory.

For test execution with TTman see section 3.4.

5.5 GIOP

Structure and Tested features:

GIOP tests are to verify, weather GIOP messages are generated with correct syntax and the message exchange over IIOP works properly. The test suite allies to GIOP version 1.0, 1.1 and 1.2. There are two test suites one to test the server side ORB and another one to test the client side ORB. The test suites are developed using TTCN and divided into three main groups.

- **Message Ordering:** The sub groups of Message Ordering of the server side tests differ from those of the client side tests, in order to differentiate the behaviour of the server and the client side ORB. According to the test configurations, when testing the server side ORB, the behaviour of a client is emulated by the tester, by sending Request, LocateRequest and CancelRequest messages. Reply, LocateReply, CloseConnection messages are expected from the server ORB under test. In order to test the client side ORB, the tester acts in the opposite way.
- **CDR:** covers the test of the primitive types and the constructed types.
- **pseudo object types:** covers the test of the pseudo object types.

Note: Currently, the client side ORB test suite is not available.

All relevant sources of the tests are located under `ttcnTS/giop`. It has the following sub-directories:

- `ATS` contains the abstract test suite in different forms, among them `GIOP_Srv.pdf`.
- `ETS` contains the implementation of the test client.
- `Srv` contains the IDL definition and implementation of the server application.
- `WorkDir` is the working directory for test execution.
- `bin` will contain test relevant executables.
- `include` contains some configuration and header files.

Some environment variables need to be set before building and executing the tests. An example can be found in: `ttcnTS/giop/include/config.env`

To build the executables:

```
cd ttcnTS/giop
make
```

Prerequisites:

The test execution requires the include library of the ORB under test.

Run the tests and assessment:

You can execute the currently available test suite (server side ORB) either with TTCN command line or TTman. To start the testsuite with TTCN command line, the following steps are are required

- Start the server based on the ORB under test, with *emsrv*
- Start the TTCN-Client(tester) *tnclt*. A list of the available test cases appears
- Select the desired test case
- Type the IOR-File name *Hello.ref,* if required
- The end result, as one from the available results (PASS, FAIL, INCONC or NONE) is reported

Note: giop client side tests are not available at the moment.

To execute the test suite using TTman, please refer to the section 4.4.

6 References

[1] The Open Group: TETware Installation Guide for UNIX Operating Systems Revision 1.5 (TET3-IGU-1.5) Nov. 1999

[2] The Open Group, Sun Microsystems Laboratory: ADL TRANSLATION SYSTEM 1.1 RELEASE NOTES 1994-1996

VSORB v2.3.1.1: C++ Tests for CORBA v2.3.1.1 ORBs End User Licence

Preamble

Testing is essential for proper development and maintenance of standards-based products.

For buyers: adequate conformance testing leads to reduced integration costs and protection of investments in applications, software and people.

For software developers: conformance testing of platforms and middleware greatly reduces the cost of developing and maintaining multi-platform application software.

For suppliers: In-depth testing increases customer satisfaction and keeps development and support costs in check. API conformance is highly measurable and suppliers who claim it must be able to substantiate that claim.

As such, since these are benchmark measures of conformance, we feel the integrity of test tools is of importance. In order to preserve the integrity of the existing conformance modes of this test package and to permit recipients of modified versions of this package to run the original test modes, this license requires that the original test modes be preserved.

If you find a bug in one of the standards mode test cases, please let us know so we can feed this back into the original, and also raise any specification issues with the appropriate bodies (for example the Corval2 partners).

BY RETRIEVING THIS DISTRIBUTION OF VSORB V2.3.1.1: C++ TESTS FOR CORBA V2.3.1.1 ORBS, YOU ARE CONSENTING TO BE BOUND BY THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL OF THE TERMS OF THIS AGREEMENT, DO NOT INSTALL THE PRODUCT, AND DESTROY YOUR COPY.

End User Licence

This program is free software; you can redistribute it and/or modify it under the terms of the "Artistic License" which comes with this Kit, with the following modification:

a) "executable(s)" should be interpreted to include "test case(s)"
b) if you wish to make changes as defined in clause 2 and 3, and distribute a modified version of this package, then clauses 3c and 4c are required
c) clause 7 is rephrased as follows: "Subroutines supplied by you and linked into this Package shall not be considered part of this Package".
d) clause 6 is rephrased as follows: "The scripts and library files supplied as input to or produced as output from the programs of this Package do not automatically fall under the copyright of this Package, but belong to whomever generated them, and may be sold commercially, and may be aggregated with this Package, with the exception of test results from the Standard Version of this Package. Disclosure of test results, except for the purposes of reporting a suspected problem in the execution of the tests, and claims of "passing the tests" are not permitted without the previous written consent of X/Open (The Open Group)."

X/OPEN, TRADING AS THE OPEN GROUP, DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL X/OPEN BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

The "Artistic License"

Preamble

The intent of this document is to state the conditions under which a Package may be copied, such that the Copyright Holder maintains some semblance of artistic control over the development of the package, while giving the users of the package the right to use and distribute the Package in a more-or-less customary fashion, plus the right to make reasonable modifications.

Definitions:

1. "Package" refers to the collection of files distributed by the Copyright Holder, and derivatives of that collection of files created through textual modification.

2. "Standard Version" refers to such a Package if it has not been modified, or has been modified in accordance with the wishes of the Copyright holder.

3. "Copyright Holder" is whoever is named in the copyright or copyrights for the package.

4. "You" is you, if you're thinking about copying or distributing this Package.

5. "Reasonable copying fee" is whatever you can justify on the basis of media cost, duplication charges, time of people involved, and so on. (You will not be required to justify it to the Copyright Holder, but only to the computing community at large as a market that must bear the fee.)

6. "Freely Available" means that no fee is charged for the item itself, though there may be fees involved in handling the item. It also means that recipients of the item may redistribute it under the same conditions they received it.

    1. You may make and give away verbatim copies of the source form of the Standard Version of this Package without restriction, provided that you duplicate all of the original copyright notices and associated disclaimers.

    2. You may apply bug fixes, portability fixes and other modifications derived from the Public Domain or from the Copyright Holder. A Package modified in such a way shall still be considered the Standard Version.

    3. You may otherwise modify your copy of this Package in any way, provided that you insert a prominent notice in each changed file stating how and when you changed that file, and provided that you do at least ONE of the following:

        (a) place your modifications in the Public Domain or otherwise make them Freely Available, such as by posting said modifications to Usenet or an equivalent medium, or placing the modifications on a major archive site such as ftp.uu.net, or by allowing the Copyright Holder to include your modifications in the Standard Version of the Package.
        (b) use the modified Package only within your corporation or organization.
        (c) rename any non-standard executables so the names do not conflict with standard executables, which must also be provided, and provide a separate manual page for each non-standard executable that clearly documents how it differs from the Standard Version.
        (d) make other distribution arrangements with the Copyright Holder.

    4. You may distribute the programs of this Package in object code or executable form, provided that you do at least ONE of the following:

        (a) distribute a Standard Version of the executables and library files, together with instructions (in the manual page or equivalent) on where to get the Standard Version.

(b) accompany the distribution with the machine-readable source of the Package with your modifications.

(c) accompany any non-standard executables with their corresponding Standard Version executables, giving the non-standard executables non-standard names, and clearly documenting the differences in manual pages (or equivalent), together with instructions on where to get the Standard Version.

(d) make other distribution arrangements with the Copyright Holder.

5. You may charge a reasonable copying fee for any distribution of this Package. You may charge any fee you choose for support of this Package. You may not charge a fee for this Package itself. However, you may distribute this Package in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial)software distribution provided that you do not advertise this Package as a product of your own.
6. The scripts and library files supplied as input to or produced as output from the programs of this Package do not automatically fall under the copyright of this Package, but belong to whomever generated them, and may be sold commercially, and may be aggregated with this Package.
7. C or perl subroutines supplied by you and linked into this Package shall not be considered part of this Package.
8. The name of the Copyright Holder may not be used to endorse or promote products derived from this software without specific prior written permission.
9. THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The End

VSORB v2.3.1.1: C++ Tests for CORBA v2.3.1.1 ORBs End User Licence