Rhapsody®

IBM Engineering Systems Design Rhapsody - TestConductor Add On

Testing on a Linux Target

Release 2.8.4

License Agreement

Trademarks

Contents

Content

Introduction

This document describes how TestCases can be executed with IBM® Engineering Systems Design Rhapsody® TestConductor Add On on a Linux target, while Rhapsody is running on a Windows or Linux host. We assume that the basic installation is already done: Rhapsody is installed on the Windows or Linux host, a Linux distribution supported by Rhapsody is installed on the target.

There are different ways to set up the test process on the Linux targets, depending on the used work flow to build the code for the target, start the execution of the tested application and access the collected results some details might differ from the example shown in this document.

In this example we also assume a TCP/IP connection between the host and the Linux target machine is available. Rhapsody running on the host will invoke the tested application on the target via this TCP/IP connection, and during test execution the communication between Rhapsody animation and the application uses TCP/IP. Also needed is a network drive or a folder which is accessible both from the host and the Linux target machine, for example by using Samba: When generating code for the modeled application Rhapsody writes the files into this folder, and the code is compiled on the connected Linux machine. In this document we will further refer to this shared folder as “/mnt/winlinux”. We will describe the execution of TestCases on a Linux target using an example. There are two different modes to execute tests with TestConductor, animation based mode and assertion based mode. Assertion based mode is available since Rhapsody 7.6, it is the default mode for newly created TestArchitectures. Depending on the used testing mode, there are some differences when testing on a Linux target: The example will show the needed settings for both modes.

Preparing the Execution

Rhapsody Share for Linux Targets

In order to compile and link the tested application the Rhapsody framework for Linux is needed. The Linux framework has to be deployed on the Linux target. A tar archive with this framework can be found in the folder \Share\LangCpp (or \Share\LangC for C) of the Rhapsody installation: Copy the file linuxShare.tar to /mnt/winlinux and untar it there. The archive contains header and library files and a folder Share/etc with some scripts and helper tools.

In addition, to compile the tested application some additional header and source files provided by TestConductor are needed: Per default the code generation configurations of a test architecture have the additional include path "$OMROOT/../TestConductor", OMROOT usually points to the Rhapsody Share folder. If you create a folder "TestConductor" on the Linux machine in the same folder where the Rhapsody Share folder is located and copy the needed TestConductor header files into the TestConductor folder the default additional include path matches to this folder.

In this example, create a folder "TestConductor" in the folder /mnt/winlinux and copy the three files TestConductor.h, TestConductor_C.c and TestConductor_C.h from the folder Rhapsody/TestConductor/ of your host Rhapsody installation to /mnt/winlinux/TestConductor. For computation of model or code coverage results also the files TC_ModCov.h and TCCoverage.h should be copied to the TestConductor folder on the Linux machine. These header files are also included in the archive "tc_codecov_tools_linux.tgz" located in the folder "LinuxTools" of the TestConductor installation for Windows, see below.

If the Linux Share is installed somewhere else on the Linux machine you should make sure the folders "Share" and "TestConductor" have the same parent folder otherwise the include path of the code generation configuration must be adjusted.

Compilation of code coverage

When testing with assertion based testing mode computation of code coverage is supported also when executing tests on a Linux target. To be able to collect the code coverage information the source code of the SUT needs to be instrumented ("annotated") by tools which are part of the TestConductor installation. Annotating the code can either be done on the Windows host (using the Windows variants of these tools, installed in the TestConductor folder) or during a compilation on the Linux target using native Linux variants of these tools.

The Linux tools are provided in the archive "tc_codecov_tools_linux.tgz" located in the folder "LinuxTools" of the TestConductor installation for Windows and need to be deployed on the Linux machine. This archive contains a folder "TestConductor" with some binaries needed for computation of code coverage, some header files needed to build the test application (see section above) and some additional files for generation of the html code coverage results. It is recommended to deploy the folder TestConductor into the same folder where the Rhapsody

Share folder is located: This way the include paths of the code generation configurations don't have to be adjusted.

Preparing the Rhapsody model

Folder for generated code

Start Rhapsody and open the model you want to test. It is not needed that the whole Rhapsody model is stored in the folder /mnt/winlinux, the files with the data of the Rhapsody project can be stored on the Windows host. Only the generated code for the tested application has to be generated into this folder: The code needs to be compiled on the target. If the model is not stored in /mnt/winlinux, open the feature tab of the Linux CG configuration, go to the "Settings" tab, uncheck the check box "Directory – Use Default", and enter a path in /mnt/winlinux (in Windows path notation).

If you now invoke the code generation for the Linux CG configuration the code for the application will be generated into /mnt/winlinux and is ready to be compiled on the Linux machine.

IP address of target

If the application is compiled with animation instrumentation, a TCP/IP connection between the running application and Rhapsody is needed. For this connection, the IP address of the host machine must be generated into the code before the application is compiled and executed. The IP address of the host should be entered in the property CPP_CG::Linux::RemoteHost (for C++) or C_CG::Linux::RemoteHost (for C).

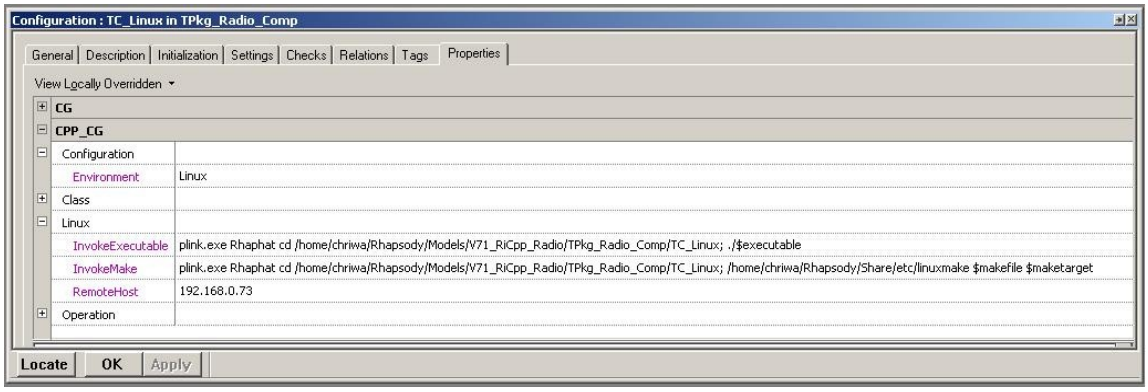
Note: To execute tests in animation based mode, the tested application must be compiled with animation instrumentation. For assertion based mode, animation instrumentation is not needed (but testing of applications with animation instrumentation is supported, too).

Invoke make to build the application

The property CPP_CG:Linux:InvokeMake (for C++) or CPP_CG:Linux:InvokeMake (for C) of the CG configuration can be used to specify the necessary command(s) to compile the application on the Linux target. The command entered in this property is performed when the user invokes the menu Code->Build.

The shell script *linuxmake* in the folder /mnt/winlinux/Share/etc can be used to invoke the compiler. The script has to be invoked with at least two arguments: the first argument is the path to the Makefile generated for the CG configuration. The second argument has to be one of the make targets *build*, *rebuild* or *clean*. This script uses two other scripts (*removeCR.sh* to remove *carriage return* characters from the Makefile and *changeOMROOT.sh* to set the correct path for the header and library files of the Linux framework) to prepare the Makefile, and then invokes the make command.
SSH can be used log in onto the Linux target and to invoke the linuxmake script from the host.

Example



The screenshot shows an example for the settings of the properties. In this example the program “plink.exe” is used to establish the connection between the host and the Linux target. plink is part of the “PuTTY” toolset; PuTTY is a free SSH implementation for Windows and Unix.

The values of the changed properties are:

- InvokeExecutable (only needed for animation based mode): “*plink.exe Rhaphat cd /home/chriwa/Rhapsody/Models/V71_RiCpp_Radio/TPkg_Radio_Comp/TC_Linux; ./$executable*”
- InvokeMake: “*plink.exe Rhaphat cd /home/chriwa/Rhapsody/Models/V71_RiCpp_Radio/TPkg_Radio_Comp/TC_Linux; /home/chriwa/Rhapsody/Share/etc/linuxmake $makefile $maketarget*”
- RemoteHost: This should be set to the IP address of the host machine

PuTTY allows to save the settings of a session (login machine, login name, etc.) and re-load a session. In this example a session named “Rhaphat” is used.

If the build of the application is invoked the program plink.exe loads the session “Rhaphat” and logs in on the Linux target. Then the directory is changed to the directory containing the generated code and the script linuxmake is invoked to compile the application.

If the application is launched (in animation based mode), the program plink.exe loads the session “Rhaphat” and logs in on the Linux target. Then the directory is changed to the directory with the application binary and the application is launched.

Invoke the application (animation based mode)

When executing tests in animation based mode, TestConductor is starting the tested application by calling a Rhapsody API. The command used by Rhapsody to start the application is defined in the property CPP_CG::Linux::InvokeExecutable (for C++) or C_CG::Linux::InvokeExecutable (for C). This property should be modified only for animation based testing mode.
Again SSH can be used to login onto the Linux machine and to launch the application. For a better automation of the TestCase execution (especially if multiple TestCases of a TestContext/TestPackage shall be executed) the SSH login on the Linux target can be done using an *authorized key*: This way the user does not have to enter a password each time the

application is launched. A series of TestCases can be fully automatically executed on the Linux target.
If the TestCase execution is activated TestConductor starts the application of the used TestConfiguration (if the TestContext of the TestCase does not have a TestConfiguration, then the currently active CG Configuration is used). Currently, TestConductor does not support having more than one TestConfiguration for a TestContext. If there are multiple CG Configurations (for example for different environments, Windows host and Linux target), and the user wants to perform tests in animation based mode with a different CG Configuration, the existing TestConfiguration has to be deleted and a new TestConfiguration pointing to another CG Configuration has to be added to the TestContext.

Invoke the application (assertion based mode)

For assertion based testing mode, there are some tags to configure the code generation configuration used for testing. Some of these tags need to be modified to automate execution of tests on the Linux target.

When executing tests in animation based mode, TestConductor directly starting the tested application by calling a batch file on the host. The content of this batch file is automatically generated by TestConductor and can be configured by the user in the tag rtc_testexecution_script_content of the code generation configuration. An example for the content of this script is (when using plink to log in to the target):

plink.exe Rhaphat "/mnt/winlinux/<CG path>/binary -resultfile /mnt/winlinux/<CG path>/rtcresult.rst -logfile /mnt/winlinux/<CG path>/rtclog.txt -tcontext $tcontext -tcase $tcase

<CG path> is the path to the generated code, *binary* is the name of the tested application. *$tcontext* and *$tcase* will be provided by TestConductor depending on which element is executed. When executing tests, TestConductor invokes the batch file on the host and the batch file does the remote invocation of the application on the target. After the test has been executed, another batch file is called to evaluate the outcome of the test and to generate an html report.

When starting the execution of a TestCase in assertion based mode, TestConductor the application of the used TestConfiguration. If the currently active CG Configuration is not the TestingConfiguration but a Configuration in the same CG Component, then the active CG Configuration is started.