IBM® Rational® Rhapsody® TestConductor Add On

Testing on an Integrity Target

Rhapsody®

IBM® Rational® Rhapsody® TestConductor Add On

Testing on an Integrity Target

Release 2.8.0

Contents

Content

Contents..4

Contacting IBM® Rational® Software Support..5

Introduction..6

Execution of TestCases on the Integrity Target (animation based testing mode)...............7

Preparing the Code Generation Configuration..7

Settings of the Code Generation Component..7

Settings of the Code Generation Configuration..7

Properties of the Code Generation Configuration..7

Preparing the Test Architecture..8

Preparing the TestPackage..8

Properties of the TestPackage..8

Executing a TestCase..8

Execution of TestCases on the Integrity Target (assertion based testing mode)...............10

File IO needed for assertion based testing..10

Environment INTEGRITY5: Preparing the Code Generation Configuration..11

Settings of the Code Generation Component..11

Settings of the Code Generation Configuration..11

Properties of the Code Generation Configuration..11

Tags of the Code Generation Configuration:..13

Environment INTEGRITY5: Executing a TestCase..14

Computation of code coverage..14

Target configuration..14

Options file for computation of code coverage..15

Batch files for annotating the source code file during build..17

Building and executing tests with computation of code coverage..18

Contacting IBM® Rational® Software Support

IBM Rational Software Support provides you with technical assistance. The IBM Rational Software Support Home page for Rational products can be found at http://www.ibm.com/software/rational/support/.

For contact information and guidelines or reference materials that you need for support, read the IBM Software Support Handbook.

For Rational software product news, events, and other information, visit the IBM Rational Software Web site.

Voice support is available to all current contract holders by dialing a telephone number in your country (where available). For specific country phone numbers, go to http://www.ibm.com/planetwide.

Before you contact IBM Rational Software Support, gather the background information that you will need to describe your problem. When describing a problem to an IBM software support specialist, be as specific as possible and include all relevant background information so that the specialist can help you solve the problem efficiently. To save time, know the answers to these questions:

What software versions were you running when the problem occurred?

Do you have logs, traces, or messages that are related to the problem?

Can you reproduce the problem? If so, what steps do you take to reproduce it?

Is there a workaround for the problem? If so, be prepared to describe the workaround.

Introduction

This document describes how TestCases can be executed with IBM® Rational® Rhapsody® TestConductor Add On on an Integrity target, while Rhapsody is running on a Windows or Linux host. We assume the basic installation is already done: The tools needed to develop software for an Integrity target are installed (for example Greenhills multi IDE, compiler, Integrity simulator, etc.). Rhapsody is installed on the Windows or Linux host and the Rhapsody adapter for development of applications for an Integrity target is installed and prepared. We will describe the execution of TestCases with the Integrity simulator using an example.

In the first section of this document, testing on an Integrity target using the animation based testing mode is described. For this testing mode, also a TCP/IP connection between Rhapsody and the tested application is needed. In the second section, testing on an Integrity target using the assertion based testing mode is described. This new testing mode is available since Rhapsody 7.6.

Execution of TestCases on the Integrity Target (animation based testing mode)

Please follow the steps as described in the sections below.

Preparing the Code Generation Configuration

Settings of the Code Generation Component

Add some settings in the Settings dialog of the CG Component:

- General::Include Path
  <RhapsodyShare>/../TestConductor
  Enter the full path to the Rhapsody Share folder instead of the variable $OMROOT because the $OMROOT variable is not expanded during the build.

Settings of the Code Generation Configuration

Add some settings in the Settings dialog of the CG Configuration:

- Settings::Libraries
  libivfs.a

- Settings::CompilerSwitches
  -DTC_INTEGRITY
  Please note the CompilerSwitches are added to a Makefile so each line must start with a tab.

Properties of the Code Generation Configuration

Adjust some properties in the CG and in the CPP_CG subject (when testing a C application use the corresponding properties of the subject C_CG instead); example using environment INTEGRITY5:

- CPP_CG::INTEGRITY5::RemoteHost
  Set the value to the IP of the Integrity simulator or target. This might be needed for the Rhapsody Animation.

- CPP_CG::INTEGRITY5::IDEInterfaceDLL
  <MultiInstallPath>/rhapsody_multi_ide.dll
  Enter the full installation path of the installed multi IDE tool (for example: C:\ghs\multi_614\rhapsody_multi_ide.dll).

- CPP_CG::INTEGRITY5::IntegrityRoot
  <IntegrityInstallPath>
  Enter the full installation path of the installed integrity sources (for example: C:\ghs\int1104).

Preparing the Test Architecture

- The TestConfiguration of the TestContext shall depend on the CG Configuration with the proper settings for Integrity. This way the application is started in the Integrity simulator if the TestCase execution is activated.

Preparing the TestPackage

Properties of the TestPackage

- TestConductor::TestCase::ResetAppBeforeStartTest
  "unchecked" - This means that TestConductor does not reset/start an already running (e.g. started by Rhapsody) application again, if the user presses "Execute Test Case". So the test case execution is simple attached to the already running application.

Executing a TestCase

- Start the Integrity MULTI Project Manager and load a kernel project (this kernel project must contain: Debugging, Dynamic Load, Resource Manager, File System Client, Core File Collection)
- Start the MULTI Debugger/Simulator via "Debug <kernel>" menue entry point
- Connect <kernel> with the simulator (for example "INTEGRITY Simulator for PowerPC (isimppc)")
- In the MULTI console, enter command: set_runmode_partner -auto
- In the MULTI console, enter command: c (this loads the kernel into the debugger / simulator)
- Connect Rhapsody with the Integrity simulator: In Rhapsody, menu Code->Target->Connect
- Update the TestCase (from the context menu of the TestCase)
- Build the TestCase (from the context menu of the TestCase). It generates code, compiles and builds the application.
- Download the built application to the simulator: In Rhapsody, menu Code->Target-> Download

- Press “Go” within the simulator, in order to launch the downloaded application within the simulator. Note, that the Animation Toolbar within Rhapsody gets active, after the user has started the application in the simulator.
- Execute the TestCase (from the context menu of the TestCase). Then the test case execution is simple attached to the already running application, while TestConductor is driving and monitoring the TestCase execution on the host machine. TestConductor shows the status of the test execution in the test execution window.
- Inspect the result of the TestCase execution: TestConductor automatically adds the detailed html result for the TestCase execution to the Rhapsody model.

Also, execution of several TestCases in a row is possible by invoking “Execute TestContext” on a TestContext or by invoking “Execute TestPackage” on a TestPackage.

Execution of TestCases on the Integrity Target (assertion based testing mode)

Please follow the steps as described in the sections below. Depending on the used environment (INTEGRITY5, Integrity5ESTL) some properties or tags must be set differently.

This table shows the Integrity environments supported by TestConductor. For all supported environments, some properties and tags need to be adjusted manually (see details in the next sections).

Environment	C++	C
INTEGRITY5	Supported	Supported
Integrity5ESTL		

File IO needed for assertion based testing

In assertion based testing mode, the tested application must be able to read and write files. Files are used to pass arguments to the tested application and the application writes information about the results of assertions to a file. When computing code coverage, also information used to compute the code coverage is written to a file.

For Integrity TestConductor uses per default the function hostio_fopen to open a file directly on the host. If another function should be used instead some files in the TestConductor installation need to be modified to support this: TestConductor.h, TestConductor_C.c, TestConductor_C.h, TCCoverage.h. These files can be found at the location <RhapsodyShare>/../TestConductor/
Replace all occurrences of hostio_fopen with the function which should be used instead. Also when modifying the properties of the Code Generation Configuration (see below) the other function should be used instead of hostio_fopen.

Check the Integrity documentation for information which folder of the host can be accessed by the target.

If the target being used does not support file IO at all a different setup for assertion based testing has to be applied. See document "Testing with TestConductor on a small target.pdf".

Environment INTEGRITY5: Preparing the Code Generation Configuration

Settings of the Code Generation Component

Add some settings in the Settings dialog of the CG Component:

- General::Include Path
  <RhapsodyShare>/../TestConductor
  Enter the full path to the Rhapsody Share folder instead of the variable $OMROOT because the $OMROOT variable is not expanded during the build.

Settings of the Code Generation Configuration

Add some settings in the Settings dialog of the CG Configuration:

- Settings::Libraries
  libivfs.a

- Settings::CompilerSwitches
  -DTC_INTEGRITY
  Please note the CompilerSwitches are added to a Makefile so each line must start with a tab.

Properties of the Code Generation Configuration

Adjust some properties in the CG and in the CPP_CG subject (when testing a C application use the corresponding properties of the subject C_CG instead); example using environment INTEGRITY5:

- CPP_CG::INTEGRITY5::BLDTarget
  sim800

- CPP_CG::INTEGRITY5::RemoteHost
  Set the value to the IP of the Integrity simulator or target. This might be needed when using instrumentation mode Animation.

- CPP_CG::INTEGRITY5::IDEInterfaceDLL
  <MultiInstallPath>/rhapsody_multi_ide.dll
  Enter the full installation path of the installed multi IDE tool (for example: C:\ghs\multi_614\rhapsody_multi_ide.dll).

- CPP_CG::INTEGRITY5::IntegrityRoot
  <IntegrityInstallPath>
  Enter the full installation path of the installed integrity sources (for example: C:\ghs\int1104).

- CPP_CG::INTEGRITY5::RTC_DownloadApplication
  Checked, if TestConductor should automatically download the application to the target.

- CPP_CG::Configuration::MainFunctionArgList
  void

- For C++ only:
  CPP_CG::Configuration::ImplementationProlog
  extern “C” FILE* hostio_fopen(const char* filename, const char* mode);

  Alternatively:

- For C only:
  C_CG::Configuration::ImplementationProlog
  extern FILE* hostio_fopen(const char* filename, const char* mode);

- For C++ only:
  CG::Configuration::PreFrameworkInitCode

```
char c1[200];
char c2[200];
FILE *f;
int argc = 8;
char* argv[]={"-resultfile","rtcresult.rst","-logfile","rtclog.txt","-tcontext",c1,"-tcase",c2};
f = hostio_fopen("<CGPath>/test_args.txt", "r");
if (f != NULL) {
        if(fgets(c1, 200, f)) {
                int i = strlen(c1);
                if (c1[i - 1] == '\n')
                        c1[i - 1] = '\0';
                if (c1[i - 2] == '\r')
                        c1[i - 2] = '\0';
        }
        if(fgets(c2, 200, f))  {
                int i = strlen(c2);
                if (c2[i - 1] == '\n')
                        c2[i - 1] = '\0';
                if (c2[i - 2] == '\r')
                        c2[i - 2] = '\0';
        }
        fclose(f);
}
```

  <CGPath> Replace this with the path to the CG folder on your host. Use forward slashes for this path (like “C:/model/MyComponent/DefaultConfiguration”).

  Alternatively:

- For C only:
  CG::Configuration::PreFrameworkInitCode

```
char c1[200];
char c2[200];
FILE *f;
```

```
int argc;
char* argv[];
argc = 8;
f = hostio_fopen("<CGPath>/test_args.txt", "r");
if (f != NULL) {
        if(fgets(c1, 200, f)) {
                int i = strlen(c1);
                if (c1[i - 1] == '\n')
                        c1[i - 1] = '\0';
                if (c1[i - 2] == '\r')
                        c1[i - 2] = '\0';
        }
        if(fgets(c2, 200, f))  {
                int i = strlen(c2);
                if (c2[i - 1] == '\n')
                        c2[i - 1] = '\0';
                if (c2[i - 2] == '\r')
                        c2[i - 2] = '\0';
        }
        fclose(f);
}
argv[0]="-resultfile";
argv[1]="rtcresult.rst";
argv[2]="-logfile";
argv[3]="rtclog.txt";
argv[4]="-tcontext";
argv[5]=c1;
argv[6]="-tcase";
argv[7]=c2;
```

<CGPath> Replace this with the path to the CG folder on your host. Use forward slashes for this path (like "C:/model/MyComponent/DefaultConfiguration").

Tags of the Code Generation Configuration:

- rtc_testexecution_script_content
  ```
  C:
  cd $CONFIGDIR
  echo $tcontext>test_args.txt
  echo $tcase>>test_args.txt
  "$executable" -resultfile "$rtc_resultfile" -logfile "$rtc_logfile" -tcontext $tcontext -tcase
  $tcase
  TIMEOUT 30
  ```
- rtc_result_filename
  <TargetDirOnHost>/rtcresult.rst
  <TargetDirOnHost> Replace this with the folder on the host machine which can be accessed by the application running on the target (like "C:/ghs/int1104/sim800").

After applying these changes, TestConductor tests can be executed on the Integrity target using the standard TestConductor work flow: Create and specify tests, update, build and execute tests.

Environment INTEGRITY5: Executing a TestCase

- Start the Integrity MULTI Project Manager and load a kernel project (this kernel project must contain: Debugging, Dynamic Load, Resource Manager, File System Client, Core File Collection)
- Start the MULTI Debugger/Simulator via “Debug <kernel>” menue entry point
- Connect <kernel> with the simulator/target
- In the MULTI console, enter command: set_runmode_partner -auto
- In the MULTI console, enter command: c (this loads the kernel into the debugger / simulator)
- Connect Rhapsody with the Integrity simulator: In Rhapsody, menu Code->Target->Connect
- Update the TestCase (from the context menu of the TestCase)
- Build the TestCase (from the context menu of the TestCase). It generates code, compiles and builds the application.
- Download the built application to the simulator: In Rhapsody, menu Code->Target-> Download (not needed if property CPP_CG::INTEGRITY5::RTC_Download Application is checked)
- Execute the TestCase (from the context menu of the TestCase). The application is downloaded on the target (if property CPP_CG::INTEGRITY5::RTC_Download Application is checked). Then the application is launched on the target (by the user who must press “Go” within the simulator), while TestConductor is driving and monitoring the TestCase execution on the host machine. TestConductor shows the status of the test execution in the test execution window.
- Inspect the result of the TestCase execution: TestConductor automatically adds the detailed html result for the TestCase execution to the Rhapsody model.

Also, execution of several TestCases in a row is possible by invoking “Execute TestContext” on a TestContext or by invoking “Execute TestPackage” on a TestPackage.

Computation of code coverage

Computation of code coverage using the gnu compiler is supported for the Rhapsody environment Integrity5 for C++ and C.

Target configuration

For computation of code coverage, the source code needs to be instrumented (annotated) with some macros which are used to track the executed functions, statements, decision branches. For a correct annotation TestConductor needs some information about the compiler and the target system, the so called target configuration: The size and sign of some types, the endian of the target, the compiler family and some more. To collect this information a small program must be compiled (with the same compiler and compiler options which are used to compile the tested application) and executed on the same target the tested application will be executed on. The target configuration tool will collect the needed information and write it into an xml file, this file can be used from then on for the instrumentation for code coverage until the target configuration (compiler, compiler options, target) changes.

To build and execute the target configuration tool a Rhapsody model can be used which is part of the TestConductor installation, in folder TestConductor/CodeCoverage/TargetConfiguration. Copy the folder TargetConfiguration to a folder which can be written to and open the project in Rhapsody in C++ or C. The project contains one Code Generation Component for C++ and one for C, each with several configurations for different environments. Set the Code Generation Configuration predefined for the environment to be used as active configuration and adjust the settings in the properties according to your environment: Compiler options, information about the Integrity installation (like IntegrityRoot property). Generate code and build the tool.

To build and execute the target configuration tool a Rhapsody model can be used which is part of the TestConductor installation, in folder TestConductor/CodeCoverage/TargetConfiguration. Copy the folder TargetConfiguration to a folder which can be written to and open the project in Rhapsody in C++ or C. The project contains one Code Generation Component for C++ and one for C, each with several configurations for different environments. Set the predefined Code Generation Configuration ConfigIntegrity5 as active configuration and adjust the settings in the properties according to your environment: Compiler options, information about the Integrity installation (like IntegrityRoot property). Generate code, copy the file targetconf.cpp (or targetconf.c) from the folder TargetConfiguration/src to the code generation folder and build the tool.

Now load the tool on the target and execute it. It will collect the necessary data and write it to the file targetconf.xml (the location of this file depends on the Integrity environment being used, refer to the Integrity documentation for details). Copy this file to a location which can be accessed during compilation of the application you want to test, for example in the main or code generation folder of the application's project.

Options file for computation of code coverage

To compute code coverage, the user must provide some information about the installation of the Integrity environment and the Integrity version in an xml options file.

A template for an options file with some comments is provided in the TestConductor installation: Copy file **<RhapsodyInstall>**/TestConductor/TCCodeAnnotationOptions.xml to another location (for example, into the main folder of the Rhapsody project). Open the copy in an editor and enter the needed attributes and values in the <Environment> section:

- Attribute <Compiler>
  - name=”INTEGRITY”
  - cppcompiler= When using C++, enter the name of the Integrity C++ compiler (example: cxintppc.exe).

    - ccompiler= When using C, enter the name of the Integrity C compiler (example: ccintppc.exe).

- Attribute <TargetConfigFile>

    - relative_path= Enter the path and file name of the target configuration xml file, relative to the code generation folder.

      Alternatively:

    - absolute_path= Enter the full path of the target configuration xml file.

- Attribute <HostToolsEnvironment>

    - name=”GHS”

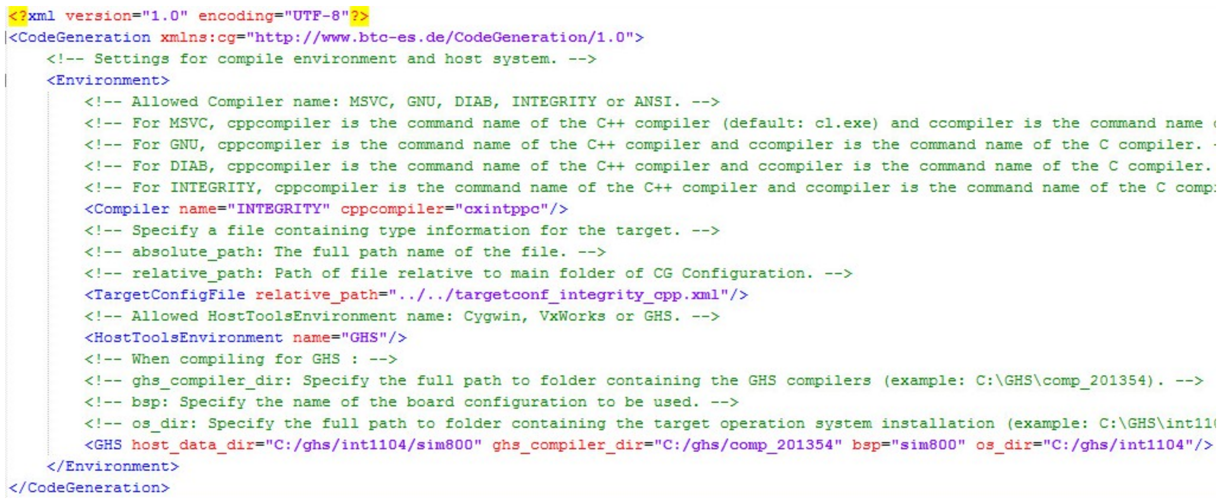
- Attribute <GHS>

    - host_data_dir= Enter the path to a folder on the host which can be used to exchange files between the host and the target/simulator (example: C:/ghs/int1104/sim800).

    - bsp= Enter the Integrity board configuration to be used (example: sim800).

    - os_dir= Enter the full path of the target operation system installation (example: C:/ghs/int1104).

See figure 1 below for an example of an options file for computation of code coverage.



```
<?xml version="1.0" encoding="UTF-8"?>
<CodeGeneration xmlns:cg="http://www.btc-es.de/CodeGeneration/1.0">
    <!-- Settings for compile environment and host system. -->
    <Environment>
        <!-- Allowed Compiler name: MSVC, GNU, DIAB, INTEGRITY or ANSI. -->
        <!-- For MSVC, cppcompiler is the command name of the C++ compiler (default: cl.exe) and ccompiler is the command name
        <!-- For GNU, cppcompiler is the command name of the C++ compiler and ccompiler is the command name of the C compiler.
        <!-- For DIAB, cppcompiler is the command name of the C++ compiler and ccompiler is the command name of the C compiler.
        <!-- For INTEGRITY, cppcompiler is the command name of the C++ compiler and ccompiler is the command name of the C comp
        <Compiler name="INTEGRITY" cppcompiler="cxintppc"/>
        <!-- Specify a file containing type information for the target. -->
        <!-- absolute_path: The full path name of the file. -->
        <!-- relative_path: Path of file relative to main folder of CG Configuration. -->
        <TargetConfigFile relative_path="../../targetconf_integrity_cpp.xml"/>
        <!-- Allowed HostToolsEnvironment name: Cygwin, VxWorks or GHS. -->
        <HostToolsEnvironment name="GHS"/>
        <!-- When compiling for GHS : -->
        <!-- ghs_compiler_dir: Specify the full path to folder containing the GHS compilers (example: C:\GHS\comp_201354). -->
        <!-- bsp: Specify the name of the board configuration to be used. -->
        <!-- os_dir: Specify the full path to folder containing the target operation system installation (example: C:\GHS\int11
        <GHS host_data_dir="C:/ghs/int1104/sim800" ghs_compiler_dir="C:/ghs/comp_201354" bsp="sim800" os_dir="C:/ghs/int1104"/>
    </Environment>
</CodeGeneration>
```

Figure 1: Code coverage options to provide information about the Integrity environment (example for C++)

In the Rhapsody model, use a tag of the Code Generation Configuration to specify the path to the options file: Open the feature dialog of the Code Generation Configuration and go to the Tags section. Then enter the path (including name and extension) to the options file into the tag CodeCoverageOptionsFilename. You can use an absolute path or a path relative to the code generation main folder (location of the Makefile).

Batch files for annotating the source code file during build

To be able to collect code coverage information during test execution the source code needs to be annotated with macros collecting the coverage information. This is done by a tool which is part of the TestConductor installation. For parsing and annotating the code the tool must be provided with some information like the list of defines and include paths (the same which are passed to the compiler). By creating two batch files the code annotation tool (TCCodeAnnotation.exe) can be called including this information during the build of the tested application.

1. Create a file “annotate.bat” in the code generation folder with this content (in one line):
   <RhapsodyInstall>/TestConductor/TCCodeAnnotation.exe **<CGPath> <Includes> <Defines>** %1

   <RhapsodyInstall> The Rhapsody installation path.
   <CGPath> The absolute path to the generated code (location of the Makefile).
   <Includes> The list of all include directives from the Makefile for the program.
   <Defines> The list of all defined macros from the Makefile for the program.

   This batch file is used to call the code annotation tool before compilation of the implementation file.

2. Create a file copyfile.bat in the same folder with this content (in two lines):
   if exist "%1.bak" copy "%1" "%1.annotated"
   if exist "%1.bak" move "%1.bak" "%1"

   This batch file is used after compilation of the annotated implementation file to rename it and to move the backup of the original implementation to it's original name.

When performing “Update TestCase” (or “Update TestContext” or “Update TestPackage”), TestConductor will modify the property “CPP_CG::INTEGRITY5::CPPCompileCommand” (C_CG::INTEGRITY5::CPPCompileCommand when using C) of the code generation configuration with the necessary calls of the two batch files. When generating code Rhapsody generates the content of this property into the Makefile.

When modifying the CPPCompileCommand (for example if the batch files should be located somewhere else in the file system), TestConductor will overwrite these modifications automatically when performing “Update Test...”. To avoid this, uncheck tag “PopulateCompileCommandForCodeCoverage” on the code generation configuration.

Building and executing tests with computation of code coverage

After building the application the tests can be executed on the target to compute code coverage information. After the execution of the tests has finished, TestConductor automatically adds a detailed code coverage report to the Rhapsody model.