

Rational. Rhapsody

IBM

IBM® Rational® Rhapsody® TestConductor Add On



Code coverage limitations

Rhapsody[®]

**IBM[®] Rational[®] Rhapsody[®]
TestConductor Add On**

Code coverage limitations

Release 2.4.5



License Agreement

No part of this publication may be reproduced, transmitted, stored in a retrieval system, nor translated into any human or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of the copyright owner, BTC Embedded Systems AG.

The information in this publication is subject to change without notice, and BTC Embedded Systems AG assumes no responsibility for any errors which may appear herein. No warranties, either expressed or implied, are made regarding Rhapsody software and its fitness for any particular purpose.

Trademarks

IBM[®] Rational[®] Rhapsody[®], IBM[®] Rational[®] Rhapsody[®] Automatic Test Generation Add On, and IBM[®] Rational[®] Rhapsody[®] TestConductor Add On are registered trademarks of IBM Corporation.

All other product or company names mentioned herein may be trademarks or registered trademarks of their respective owners.

© Copyright 2000-2011 BTC Embedded Systems AG. All rights reserved.

Contents

<u>Contents</u>	<u>6</u>
<u>Contacting IBM® Rational® Software Support</u>	<u>7</u>
<u>General information</u>	<u>8</u>
<u>Limitations regarding C</u>	<u>9</u>
<u>Limitations regarding C++</u>	<u>10</u>
<u>Tested Compilers</u>	<u>13</u>

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.

General information

TestConductor can compute the code coverage achieved by executed test cases. The code coverage feature of TestConductor is based on source code instrumentation. Code coverage computation is restricted to C and C++. Section “Limitations regarding C” contains known limitations regarding C language. Section “Limitations regarding C++” contains known limitations regarding C++ language. Section “Tested Compilers” contains a list of tested compiler versions.

Limitations regarding C

- 1) The C annotation feature works for all valid C language constructs defined in the C90 standard.

Limitations regarding C++

1) The base of the C++ annotation feature is an extension of the already existing C annotation feature (cf. docus and limitations of the C-annotation-feature)

2) Templates are not yet annotated, but may exist in the input-sourcecode

3) Implicit constructs (that is, C++ language constructs which do not literally exist in the input-sourcecode, but exist implicitly as dictated by the C++ standard) are not annotated, that is its coverage could not be measured. Examples:

- Default-Constructor
- implicit copy-constructor
- implicit assignment-operator
- implicit destructor

4) Downcasts appear only if the source- and destination-type of the cast-expression are "integral types" (That is, Integer-, Bool- and Enum-Types). Note that it is possible that the cast-operand could be a call-expression of a class-member-conversion-operator, which converts the class-instance to an integral type. Example:

```
class C {public: operator short() const;};  
  
char c = c_inst; /* Downcast! */
```

5) "Division-By-Zero" is only detected for the builtin '/'-operator (applied on Integral- or floating-point types), but not for user-defined '/'-operators.

Example:

```
class D { public: bool operator/(int i) const; };  
  
int i;  
  
char c = d_inst/i; /* no DBZ-Check */
```

6) Conditions and decisions are only defined based on the predefined logical operators, and by their definition-location (for instance: expression of if-statement), corresponding to the definition in C. Note that suitable conversion operators allow also class-instanced to appear as conditions/decisions, but defining a logical class-member-operation does not yield to "Conditions/Decisions". Example:

```
class C { public: operator bool() const; };  
  
class D { public: bool operator&&(const D& d); };  
  
C c_inst;  
  
D d_inst;
```

```
bool b1 = c_inst && c_inst; /* <-- Two conditions 'c_inst', '&&'-result is decision */
```

```
bool b2 = d_inst && d_inst; /* <-- Contains neither a decision nor a condition */
```

7) A Cast to bool is defined not to be a "Downcast", if it is a decision or condition.

8) A "Downcast" is defined to only occur when a cast happens from an integer/enum/bool to another integer/enum/bool (causing a value-change), but not for instance if a cast happens from e.g. a floating point type to another floating point type involving a value change (e.g. from long to float, or from float to int).

11) 'long long' is supported as type in C++, as type in the Microsoft C-dialect (for Visual Studio greater than 6) and as type in the GNU C-dialect. '__int64' is supported for the Microsoft dialects.

12) 'Designating Initializers' (a C99-extension, e.g. implemented by gcc) are supported if the input-code is C (C++ does not define 'designating initializers').

13) 'Compound literals' (a C99-extension, e.g. implemented by gcc) are supported if the input-code is C (C++ does not define 'compound literals').

14) Metaprogramming by 'type traits' (A C++-extension, introduced by ISO/IEC TR 19768, and implemented by e.g. the GCC- and Microsoft-Compilers) is supported, but corresponding pseudo-functions (which are evaluated at compile-time) do not induce a decision (e.g. the expression '__is_base_of(A,B)' - where A and B are classnames - is not a "decision").

15) __try...__finally- and __try...__except-statements as well as __leave-statements (all three statements are Microsoft-C extensions realizing a (limited) kind of exception mechanism similar to the C++ try...catch-statement) are supported.

16) Statement-expressions (a gcc extension of C/C++, e.g. 'e=({if (x) x=y; x;});') are supported. The surrounding '{ ... }' is not counted as a statement (because it's a special kind of expression), but each statement within the curly brackets is taken into account for the statement-coverage-calculation.

17) The following GNU-extensions are not supported (they are not only unsupported by the backend [in particular: The downcast of "complex integer types" (GNU extension) is not defined], in particular the EDG-frontend does not support the following extensions:

a) The forward declaration of function parameters, e.g. "f(int x; char str[x], int x);"

b) Complex integral types (that is, complex numbers with real- and imaginary-part being integral numbers) - note that complex float types are supported.

c) Nested functions (GCC-extension of C, e.g. "int f(){int g(){return 42;} return g();};")

d) Local structure types with variable length array fields (GCC-extension of C). For this construct, the array bound is treated as zero and a warning is printed by the EDG-frontend. EDG states that this is a workaround covering common cases where variable length array fields are used. EDG states: "GNU C allows variable-length array fields in local classes. Supporting that requires operations like sizeof and field offset to be computed at run time. However, this extension is often only used in situations where the size does not matter. As a compatibility work-around, we therefore warn about the construct and treat the resulting array as having bound zero.". Concerning the coverage calculation, the consequence is that constructs in the (non-constant) index-expression of the array-declaration are not annotated (and thus are not taken into account for the coverage calculation).

18) Microsoft SAL-annotations may occur in the input-source, but they are not existant any more in the code annotated by TestConductors CodeCoverage-functionality. This should not be a problem, because SAL-annotations have no execution semantics, they are just meta-information. In addition, the macro '`_USE_ATTRIBUTES_FOR_SAL`' is set to '0' for each sourcefile which is passed to the annotation tools by TestConductor (which is also the default-setting for Visual Studio 2010, but which is not the default-setting for Visual Studio 2008).

19) For each Translation Unit passed to the annotation tools by TestConductor, a line '`#define __declspec(X)`' is defined at the beginning of the translation unit, if the target is GNU/Cygwin. As a consequence, an annotated translation unit may not rely on this Windows-specific GCC-extension, respectively this extension is not supported by the annotation.

20) Code, i.e statements, decisions etc inside header files are not yet annotated, but may occur in the input-sourcecode.

Tested Compilers

The following compilers were used to test the C and C++ code coverage feature of TestConductor:

- Microsoft Visual Studio 6.0 Service Pack 5 on Windows XP Service Pack 3 (32 bit)
- Microsoft Visual Studio 2008 on Windows XP Service Pack 3 (32 bit)
- Microsoft Visual Studio 2008 on Windows Vista Service Pack 2 (32 bit)
- Microsoft Visual Studio 2008 on Windows 7 Service Pack 1 (32 bit)
- Cygwin 1.5.25 with gcc/g++ 4.3.2 on Windows XP Service Pack 3 (32 bit)